

HỌC VIỆN CÔNG NGHỆ Bưu Chính Viễn Thông

-----  
PGS. TS. Phạm Văn Cường (chủ biên)

GIÁO TRÌNH  
KIẾN TRÚC MÁY TÍNH

HÀ NỘI 2024

## LỜI NÓI ĐẦU

Kiến trúc máy tính là một trong các lĩnh vực khoa học cơ sở của ngành Khoa học máy tính nói riêng và Công nghệ thông tin nói chung. Kiến trúc máy tính là khoa học về lựa chọn và ghép nối các thành phần phần cứng của máy tính nhằm đạt được các mục tiêu về hiệu năng cao, tính năng đa dạng và giá thành thấp.

Môn học Kiến trúc máy tính là môn học cơ sở chuyên ngành trong chương trình đào tạo công nghệ thông tin hệ đại học và cao đẳng. Mục tiêu của môn học là cung cấp cho sinh viên các kiến thức cơ sở của kiến trúc máy tính, bao gồm kiến trúc máy tính tổng quát, kiến trúc bộ xử lý trung tâm và các thành phần của bộ xử lý trung tâm, kiến trúc tập lệnh máy tính, cơ chế xử lý xen kẽ dòng mã lệnh; hệ thống phân cấp của bộ nhớ, bộ nhớ trong, bộ nhớ cache và các loại bộ nhớ ngoài; hệ thống bus và các thiết bị vào ra.

Kiến trúc máy tính là một lĩnh vực đã được phát triển trong một thời gian tương đối dài với lượng kiến thức đồ sộ, nhưng do khuôn khổ của tài liệu có tính chất là bài giảng môn học, Tác giả cố gắng trình bày những vấn đề cơ bản nhất nhằm phục vụ mục tiêu môn học. Nội dung của tài liệu được biên soạn thành sáu chương:

Chương 1 là phần giới thiệu các khái niệm cơ sở của kiến trúc máy tính, như khái niệm kiến trúc và tổ chức máy tính; cấu trúc và chức năng các thành phần của máy tính; và kiến trúc Harvard. Chương này cũng giới thiệu về các hệ đệm và cách tổ chức dữ liệu trong máy tính.

Chương 2 giới thiệu về khói xử lý trung tâm, nguyên tắc hoạt động và các thành phần của nó. Khối xử lý trung tâm là thành phần quan trọng và phức tạp nhất trong máy tính, đóng vai trò là bộ não của máy tính. Thông qua việc thực hiện các lệnh của chương trình bởi khói xử lý trung tâm, máy tính có thể thực thi các yêu cầu của người sử dụng.

Chương 3 trình bày về xen kẽ dòng lệnh và bộ nhớ đệm cache. Bộ nhớ đệm là một bộ nhớ đặc biệt có khả năng giúp tăng tốc hệ thống nhớ nói riêng và cả hệ thống máy tính nói chung.

Chương 4 trình bày về lập trình hợp ngữ với vi xử lý 8086/8088. Hợp ngữ là một trong những ngôn ngữ gần với mã máy nhất. Ngôn ngữ hợp ngữ được thiết kế để giúp cho các lập trình viên dễ dàng hơn trong việc lập trình cho các dòng vi xử lý. Chương này cũng trình bày về tập lệnh hợp ngữ của vi xử lý 8086/8088 với các ví dụ của từng nhóm lệnh.

Chương 5 trình bày về phối ghép giữa 8086/8088 với các thiết bị ngoại vi. Chương này đồng thời cũng đưa ra các ví dụ về lập trình cho 8086/8088 để điều khiển trong quá trình phối ghép với thiết bị ngoại vi.

Chương 6 thảo luận về các kiến trúc máy tính tiên tiến, bao gồm kiến trúc IA-64 với công nghệ EPIC, bộ xử lý mảng, kiến trúc SIMD/MIMD, GPU hiệu suất cao và siêu máy tính. Chương này cung cấp cái nhìn tổng quan về các công nghệ hiện đại đang được sử dụng để tối ưu hóa hiệu suất tính toán trong các ứng dụng đòi hỏi xử lý song song và tốc độ cao.

Tài liệu được biên soạn dựa trên kinh nghiệm giảng dạy môn học Kiến trúc máy tính trong nhiều năm của tác giả tại Học viện Công nghệ Bưu chính Viễn thông, kết hợp tiếp thu các đóng góp của đồng nghiệp và phản hồi từ sinh viên. Tài liệu có thể được sử dụng làm tài liệu học tập cho sinh viên hệ đại học và cao đẳng các ngành công nghệ thông tin và điện tử viễn thông. Đồng thời, tác giả xin gửi lời cảm ơn sâu sắc tới PGS. TS. Hoàng Xuân Dậu và TS. Phạm Hoàng Duy, khoa An toàn thông tin đã có những đóng góp quan trọng cho giáo trình này. Trong quá trình biên soạn, mặc dù tác giả đã rất cố gắng song khó có thể tránh khỏi những thiếu sót. Tác giả rất mong muốn nhận được ý kiến phản hồi để giáo trình được hoàn thiện hơn nữa.

Hà Nội, tháng 12 năm 2024

Tác giả

# MỤC LỤC

CHƯƠNG 1 TỔNG QUAN VỀ KIẾN TRÚC MÁY TÍNH.....	13
1.1 Khái niệm về kiến trúc và tổ chức máy tính .....	13
1.1.1 Một số khái niệm .....	13
1.1.2 Sơ đồ khối chức năng .....	14
1.1.3 Các thành phần của máy tính .....	14
1.2 Lịch sử phát triển máy tính .....	16
1.3 Kiến trúc máy tính Harvard .....	18
1.4 Các hệ số đếm và tổ chức dữ liệu trên máy tính.....	18
1.4.1 Các hệ số đếm .....	18
1.4.2 Tổ chức dữ liệu trên máy tính .....	20
1.4.3 Số có dấu và số không dấu .....	21
1.4.4 Bảng mã ASCII.....	22
1.5 Kết luận chương.....	23
1.6 Câu hỏi ôn tập .....	23
CHƯƠNG 2 KHỐI XỬ LÝ TRUNG TÂM.....	24
2.1 Sơ đồ khối tổng quát và chu trình xử lý lệnh.....	24
2.1.1 Sơ đồ khối tổng quát của CPU .....	24
2.1.2 Chu trình xử lý lệnh.....	25
2.2 Các thanh ghi .....	25
2.2.1 Giới thiệu về thanh ghi .....	25
2.3 Khối điều khiển.....	28
2.4 Khối số học và logic .....	29
2.5 Bus trong cpu .....	30
2.6 Kết luận chương.....	30
2.7 Câu hỏi ôn tập .....	31
CHƯƠNG 3 XỬ LÝ XEN KẼ DÒNG MÃ LỆNH VÀ BỘ NHỚ CACHE .....	32
3.1 Cơ chế xử lý xen kẽ dòng mã lệnh (pipeline).....	32
3.1.1 Giới thiệu cơ chế xử lý xen kẽ dòng mã lệnh .....	32
3.1.2 Các vấn đề của cơ chế xử lý xen kẽ dòng mã lệnh và hướng giải quyết.....	33
3.2 Bộ nhớ cache.....	37
3.2.1 Phân cấp hệ thống nhớ .....	37
3.2.2 Cache là gì? .....	39
3.2.3 Vai trò và nguyên lý hoạt động .....	40
3.2.4 Các dạng kiến trúc cache .....	42
3.2.5 Tổ chức/ánh xạ cache.....	44

3.2.6 Phương pháp đọc ghi và các chính sách thay thế dòng cache .....	49
3.2.7 Hiệu năng bộ nhớ cache .....	51
3.2.8 Các phương pháp giảm miss cho cache.....	53
3.3 Một số công nghệ lưu trữ dữ liệu lớn .....	54
3.3.1 Công nghệ RAID .....	54
3.3.2 Công nghệ NAS .....	57
3.3.3 Công nghệ SAN .....	58
3.4 Kết luận chương.....	59
3.5 Câu hỏi ôn tập .....	59
<b>CHƯƠNG 4 LẬP TRÌNH HỢP NGỮ VỚI BỘ VI XỬ LÝ 8086/8088.....</b>	<b>61</b>
4.1 Kiến trúc và các thành phần bộ vi xử lý 8086/8088 .....	61
4.1.1 Kiến trúc bộ xử lý 8086/8088.....	61
4.1.2 Các thành phần bộ xử lý 8086/8088 .....	62
4.2 Mã hóa lệnh và các chế độ địa chỉ .....	66
4.2.1 Mã hóa lệnh.....	66
4.2.2 Khái niệm về chế độ địa chỉ .....	67
4.2.3 Các chế độ địa chỉ của bộ xử lý 8086/8088 .....	68
4.3 Tập lệnh và công cụ emu8086 .....	70
4.3.1 Tập lệnh 8086/8088.....	70
4.3.2 Công cụ Emu8086 .....	76
4.4 Lập trình hợp ngữ .....	78
4.4.1 Chương trình hợp ngữ.....	78
4.4.2 Lập trình hợp ngữ .....	83
4.5 Kết luận chương.....	102
4.6 Câu hỏi ôn tập .....	103
<b>CHƯƠNG 5 PHỐI GHÉP VÀ LẬP TRÌNH ĐIỀU KHIỂN THIẾT BỊ.....</b>	<b>104</b>
5.1 Phối ghép CPU với bộ nhớ .....	104
5.1.1 Các tín hiệu của CPU 8086/8088 .....	104
5.1.2 Phối ghép CPU với bộ nhớ .....	106
5.1.3 Giải mã địa chỉ cho bộ nhớ.....	107
5.1.4 Giải mã địa chỉ cho bộ nhớ sử dụng mạch lô gic cơ bản .....	109
5.1.5 Giải mã địa chỉ cho bộ nhớ sử dụng mạch tích hợp 74LS138.....	110
5.1.6 Giải mã địa chỉ cho bộ nhớ sử dụng mạch PROM .....	111
5.2 Phối ghép cpu với thiết bị ngoại vi .....	112
5.2.1 Phối ghép CPU với thiết bị ngoại vi .....	112
5.2.2 Phối ghép CPU với thiết bị ngoại vi .....	113
5.3 Lập trình điều khiển bàn phím.....	112
5.3.1 Cơ bản về bàn phím .....	114

5.3.2 Dịch vụ của DOS và BIOS phục vụ bàn phím .....	118
5.3.3 Một số chương trình ví dụ về lập trình bàn phím .....	119
5.4 Lập trình điều khiển đèn LED .....	123
5.4.1 Cơ bản về đèn LED và vi mạch 7447.....	123
5.4.2 Một số chương trình ví dụ về lập trình bàn phím .....	126
5.5 Lập trình điều khiển nhiệt độ .....	127
5.5.1 Cơ bản về điều khiển nhiệt độ.....	127
5.5.2 Mạch đo và điều khiển nhiệt độ .....	128
5.5.3 Một số chương trình ví dụ về lập trình điều khiển nhiệt độ .....	130
5.6 Lập trình điều khiển hệ thống đèn giao thông .....	131
5.6.1 Mô tả hệ thống đèn giao thông .....	131
5.6.2 Nguyên lý hoạt động .....	131
5.6.3 Một số chương trình ví dụ về lập trình điều khiển đèn giao thông .....	132
5.7 Lập trình điều khiển rô bốt đơn giản và một số ví dụ.....	133
5.7.1 Cơ bản về rô bót.....	133
5.7.2 Một số ví dụ về lập trình điều khiển robot đơn giản .....	134
5.8 Các phương pháp vào ra dữ liệu .....	137
5.8.1 Giới thiệu.....	137
5.8.2 Vào/ra bằng phương pháp thăm dò .....	138
5.8.3 Vào/ra bằng ngắn.....	139
5.8.4 Vào/ra bằng truy nhập trực tiếp bộ nhớ (Direct memory Access) .....	151
5.8.5 Các phương pháp trao đổi dữ liệu .....	153
5.8.6 Bộ điều khiển truy nhập trực tiếp vào bộ nhớ Intel 8237A .....	154
5.9 Kết luận chương.....	164
5.10 Câu hỏi ôn tập .....	165
<b>CHƯƠNG 6 : Kiến trúc máy tính tiên tiến.....</b>	<b>166</b>
6.1 Kiến trúc Intel IA32/64.....	166
6.1.1 Động lực và tổng quan .....	166
6.1.2 Tổ chức Tổng Quát của IA-64 .....	169
6.2 Kiến trúc bộ xử lý mảng và song song .....	171
6.2.1 Tổng Quan về Kiến Trúc Bộ Xử Lý Mảng .....	171
6.2.2 Các Thành Phần Cơ Bản của Bộ Xử Lý Mảng .....	171
6.2.3 Nguyên Tắc Hoạt Động của Bộ Xử Lý Mảng .....	172
6.2.4 Ứng Dụng Phổ Biến của Bộ Xử Lý Mảng .....	172
6.3 Kiến trúc SIMD và MIMD .....	173
6.3.1 Kiến Trúc SIMD .....	173
6.3.2 Kiến Trúc MIMD .....	176
6.4 Bộ xử lý đồ họa (GPU) và siêu máy tính.....	178

6.4.1 Kiến Trúc GPU Hiện Đại.....	178
6.4.2 Tại Sao Cần Tăng Tốc Độ Hoặc Song Song Hóa? .....	181
6.4.3 Siêu máy tính? .....	181
6.5 Kết luận chương.....	182
6.6 Câu hỏi ôn tập .....	183
TÀI LIỆU THAM KHẢO .....	184

## DANH MỤC CÁC THUẬT NGỮ TIẾNG ANH VÀ VIỆT TẮT

<b>Thuật ngữ tiếng Anh</b>	<b>Từ viết tắt</b>	<b>Thuật ngữ tiếng Việt/Giải thích</b>
Central Processing Unit	CPU	Bộ/Đơn vị xử lý trung tâm
Read Only Memory	ROM	Bộ nhớ chỉ đọc
Random Access Memory	RAM	Bộ nhớ truy cập ngẫu nhiên
Control Unit	CU	Bộ/Đơn vị điều khiển
Arithmetic and Logic Unit	ALU	Bộ/Đơn vị tính toán số học và logic
Program Counter	PC	Bộ đếm chương trình
System Bus		Buýt hệ thống
Memory		Bộ nhớ
Cache		Bộ nhớ đệm / bộ nhớ kết
High-Order Nibble	H.O Nibble	Nibble bậc cao
Low-Order Nibble	L.O Nibble	Nibble bậc thấp
High-Order Byte	H.O Byte	Byte bậc cao
Low-Order Byte	L.O Byte	Byte bậc thấp
High-Order Word	H.O Word	Word bậc cao
Low-Order Word	L.O Word	Word bậc thấp
American Standard Code for Information Interchange	ASCII	Mã chuẩn Mỹ để trao đổi thông tin
Instruction Register	IR	Thanh ghi lệnh
Memory Address Register	MAR	Thanh ghi địa chỉ bộ nhớ
Memory Buffer Register	MBR	Thanh ghi đệm dữ liệu
Flag Register	FR	Thanh ghi cờ
Operating System	OS	Hệ điều hành
Memory Management Unit	MMU	Đơn vị quản lý bộ nhớ
Instruction pointer	IP	Con trỏ lệnh
Out Of Order execution	OOO	Thực thi không theo thứ tự
Stack Pointer	SP	Con trỏ ngăn xếp
Status Register	SR	Thanh ghi trạng thái
Arithmetic and Logic Unit	ALU	Khối số học và logic

Instruction Fetch	IF	Đọc lệnh từ bộ nhớ
Instruction Decode	ID	Giải mã lệnh
Execute	EX	Thực hiện lệnh
Memory Access	MEM	Đọc/ghi bộ nhớ
Write Back	WB	Ghi kết quả
Read After Write	RAW	Đọc sau khi ghi
Advanced Technology Attachments	ATA	Chuẩn ghép nối đĩa cứng ATA
Parallel Advanced Technology Attachments	PATA	Chuẩn ghép nối đĩa cứng PATA – hay ATA song song
Integrated Drive Electronics	IDE	Chuẩn ghép nối đĩa cứng IDE
Serial ATA	SATA	Chuẩn ghép nối đĩa cứng SATA – hay ATA nối tiếp
Small Computer System Interface	SCSI	Chuẩn ghép nối đĩa cứng SCSI
Redundant Array of Independent Disks	RAID	Công nghệ lưu trữ RAID – tạo thành từ một mảng liên kết các đĩa cứng vật lý
Network Attached Storage	NAS	Hệ thống lưu trữ gắn vào mạng
Basic Input Output System	BIOS	Hệ thống vào ra cơ sở
Pipeline		Cơ chế xử lý xen kẽ dòng mã lệnh hay cơ chế xử lý xen kẽ các lệnh
Hit		Đoán trúng – là sự kiện CPU truy tìm một mục tin và tìm thấy trong cache.
Miss		Đoán trượt – là sự kiện CPU truy tìm một mục tin và không tìm thấy trong cache.
Bus Interface Unit	BIU	Khối phổi ghép buýt
Execution Unit	EU	Khối thực hiện lệnh
Storage Area Network	SAN	Mạng lưu trữ
Industrial Standard Architecture	ISA	Buýt theo chuẩn công nghiệp ISA
Extended ISA	EISA	Buýt theo chuẩn công nghiệp mở

		rộng EISA
Peripheral Component Interconnect	PCI	Bus PCI
Accelerated Graphic Port	AGP	Cổng tăng tốc đồ họa AGP
PCI Express	PCIe	Buýt PCIe
Cathode Ray Tube	CRT	Màn hình óng điện tử âm cực
Liquid Crystal Display	LCD	Màn hình tinh thể lỏng
Intel Itanium	IA-64	
Explicitly Parallel Instruction Computing	EPIC	Tính toán lệnh song song rõ ràng
Reduced Instruction Set Computing	RISC	Máy tính tập lệnh rút gọn
Complex Instruction Set Computing	CISC	Máy tính tập lệnh phức tạp
Single Instruction, Multiple Data	SIMD	Một lệnh, nhiều dữ liệu
Processing Elements	PEs	Mảng Các Đơn Vị Xử Lý
Multiple-Instruction Multiple-Data	MIMD	Nhiều lệnh, nhiều dữ liệu
Symmetric Multiprocessor	SMP	Máy tính đa xử lý đối xứng
Distributed Shared Memory	DSM	Kiến Trúc Bộ Nhớ Chia Sẻ Phân Tán

## DANH MỤC HÌNH VẼ

Hình 1. 1. Sơ đồ khái niệm của hệ thống máy tính.....	14
Hình 1. 2. CPU của hãng Intel: 8086 và Core 2 Duo. ....	15
Hình 1. 3. Bộ nhớ ROM và RAM .....	16
Hình 1. 4. Kiến trúc máy tính Harvard.....	18
Hình 1. 5. Chuyển đổi số hệ thập phân sang số hệ nhị phân .....	19
Hình 1. 6. Giá trị các số thập lục phân theo hệ thập phân và nhị phân .....	19
Hình 1. 7. Đơn vị biểu diễn dữ liệu Byte .....	20
Hình 1. 8. Đơn vị biểu diễn dữ liệu Word .....	20
Hình 1. 9. Đơn vị biểu diễn dữ liệu Double word .....	21
Hình 1. 10. Bảng mã ASCII - Một số ký tự điều khiển.....	22
Hình 1. 11. Bảng mã ASCII - Các ký tự in được .....	22
Hình 2. 1. Sơ đồ khái tóm tắt của CPU .....	24
Hình 2. 2. Con trỏ ngăn xếp SP .....	27
Hình 2. 3. Các bit của thanh ghi cờ FR 8 bit .....	28
Hình 2. 4. Khối điều khiển CU và các tín hiệu. ....	29
Hình 2. 5. Bộ tính toán ALU .....	30
Hình 3. 1. Thực hiện lệnh (a) không pipeline và (b) có pipeline .....	32
Hình 3. 2. Thực hiện lệnh theo cơ chế pipeline với các đơn vị chức năng của CPU....	33
Hình 3. 3. Tranh chấp dữ liệu kiểu RAW.....	34
Hình 3. 4. Khắc phục tranh chấp RAW bằng chèn thêm NO-OP .....	35
Hình 3. 5. Khắc phục tranh chấp RAW bằng chèn các lệnh độc lập .....	36
Hình 3. 6. Vấn đề này sinh do lệnh rẽ nhánh .....	36
Hình 3. 7. Khắc phục vấn đề lệnh rẽ nhánh bằng cách chèn NO-OP hoặc lệnh độc lập .....	37
Hình 3. 8. Cấu trúc phân cấp hệ thống nhớ .....	38
Hình 3. 9. Dung lượng, thời gian truy cập và giá thành các loại bộ nhớ .....	38
Hình 3. 10. Vị trí của bộ nhớ cache trong hệ thống nhớ .....	39
Hình 3. 11. Lập cận về không gian trong không gian chương trình.....	40
Hình 3. 12. Lập cận về thời gian với việc thực hiện vòng lặp .....	41
Hình 3. 13. Trao đổi dữ liệu giữa CPU với cache và bộ nhớ chính .....	41
Hình 3. 14. Kiến trúc cache kiểu Look Aside .....	43
Hình 3. 15. Kiến trúc cache kiểu Look Through.....	43

Hình 3. 16. Quan hệ giữa các khối của bộ nhớ chính và dòng của cache .....	44
Hình 3. 17. Phương pháp ánh xạ trực tiếp bộ nhớ - cache .....	45
Hình 3. 18. Địa chỉ ô nhớ trong ánh xạ trực tiếp .....	45
Hình 3. 19. Phương pháp ánh xạ kết hợp đầy đủ bộ nhớ - cache.....	47
Hình 3. 20. Địa chỉ ô nhớ trong ánh xạ kết hợp đầy đủ .....	47
Hình 3. 21. Phương pháp ánh xạ tập kết hợp bộ nhớ - cache.....	48
Hình 3. 22. Địa chỉ ô nhớ trong ánh xạ tập kết hợp kết hợp .....	49
Hình 3. 23. RAID - Kỹ thuật tạo lát đĩa .....	55
Hình 3. 24. RAID – Kỹ thuật soi gương đĩa.....	55
Hình 3. 25. Cấu hình RAID 0.....	56
Hình 3. 26. Cấu hình RAID 1 .....	56
Hình 3. 27. Cấu hình RAID 10.....	57
Hình 3. 28. NAS trong một mạng LAN .....	57
Hình 3. 29. NAS của hãng D-Link và mô hình sử dụng trong mạng LAN.....	58
Hình 3. 30. Mô hình mạng lưu trữ SAN và ứng dụng.....	58
Hình 4. 1. Sơ đồ khói của bộ vi xử lý 8086/8088.....	61
Hình 4. 2. Thanh ghi cờ .....	65
Hình 4. 3. Cấu trúc mã hóa lệnh MOV trong vi xử lý 8086/8088 .....	66
Hình 4. 4. Màn hình soạn thảo chương trình nguồn của Emu806.....	77
Hình 4. 5. Màn hình mô phỏng trạng thái chương trình.....	78
Hình 4. 6. Cấu trúc IF THEN và IF THEN ELSE.....	89
Hình 4. 7. Cấu trúc lệnh CASE .....	90
Hình 4. 8. Cấu trúc lặp FOR – DO .....	91
Hình 4. 9. Cấu trúc WHILE - DO và REPEAT-UNTIL .....	92
Hình 5. 1. Chân tín hiệu vi xử lý 8086 .....	104
Hình 5. 2. Sơ đồ khói tổng quát mạch .....	106
Hình 5. 3. Mạch giải mã địa chỉ tổng quát .....	108
Hình 5. 4. Mạch giải mã dùng mạch lô-gic .....	109
Hình 5. 5. 74LS138 và bảng trạng thái .....	110
Hình 5. 6. Giải mã dùng ROM .....	112
Hình 5. 7. Không gian nhớ của thiết bị vào/ra và bộ nhớ chính.....	113
Hình 5. 8. Giải mã thiết bị dùng cổng lô-gíc .....	113
Hình 5. 9. Giải mã địa chỉ cổng dùng 74LS138 .....	114
Hình 5. 10. Mạch tạo phím .....	114

Hình 5. 11. Ma trận phím và phát hiện các phím được nhấn .....	114
Hình 5. 12. Biểu diễn ghép nối giữa 8086 với bàn phím 16 số dạng tiếp điểm.....	115
Hình 5. 13. Sơ đồ mạch kết nối IC 7447A với LED 7 đoạn .....	124
Hình 5. 14. Biểu diễn một mạch hiển thị số sử dụng vi mạch 7447 và LED bảy đoạn .....	125
Hình 5. 15. Cảm biến nhiệt độ LM35.....	129
Hình 5. 16. cảm biến nhiệt độ DS18B20.....	130
Hình 5. 17. Vào/ra lập trình với nhiều thiết bị .....	139
Hình 5. 18. Sơ đồ khối 8259.....	140
Hình 5. 19. Ghép nối 8259 với buýt 8086/8088.....	142
Hình 5. 20. Trình tự sử dụng các thanh ghi khởi đầu.....	143
Hình 5. 21. Dạng thức của ICW1 .....	143
Hình 5. 22. Dạng thức của ICW2 .....	144
Hình 5. 23. Dạng thức của ICW3 cho mạch chủ và thợ.....	145
Hình 5. 24. Dạng thức của ICW4 .....	146
Hình 5. 25. OCW1 Trạng thái yêu cầu ngắn .....	147
Hình 5. 26. Trạng thái ngắn và chế độ quay mức ưu tiên .....	148
Hình 5. 27. OCW2 xác định xử lý các yêu cầu ngắn .....	149
Hình 5. 28. OCW3 .....	150
Hình 5. 29. Thanh ghi IRR và ISR .....	150
Hình 5. 30. Dạng thức từ thăm dò trạng thái.....	150
Hình 5. 31. Hệ vi xử lý với DMA.....	152
Hình 5. 32. Sơ đồ khối 8237A.....	155
Hình 5. 33. Ghép nối 8237 với buýt hệ vi xử lý.....	157
Hình 5. 34. Cấu trúc thanh ghi lệnh của DMA.....	159
Hình 5. 35. Cấu trúc thanh ghi chế độ của DMA .....	160
Hình 5. 36. Thanh ghi yêu cầu cho từng kênh của DMA .....	161
Hình 5. 37. Thanh ghi yêu cầu và thanh ghi mặt nạ riêng cho từng kênh của DMA .....	161
Hình 5. 38. Ghép nối 8237A với 8088 ở chế độ MIN.....	164
 Hình 6. 1. Tổ chức tổng quan của kiến trúc IA-64.....	169
Hình 6. 2 Kiến trúc bộ xử lý mảng .....	171
Hình 6. 3. Kiến trúc mô hình SIMD .....	174
Hình 6. 4. Hai sơ đồ SIMD.....	175
Hình 6. 5. Kiến trúc bộ nhớ chung so với kiến trúc truyền thông điệp.....	176

Hình 6. 6. Kiến trúc của GPU hỗ trợ CUDA.....	179
Hình 6. 7. Phạm vi của các phần ứng dụng tuần tự và song song.....	180

## **DANH MỤC BẢNG**

Bảng 4. 1. Mã hóa các thanh ghi trong vi xử lý 8086/8088 theo giá trị W .....	66
Bảng 4. 2. Mã hóa chế độ địa chỉ và thanh ghi trong vi xử lý 8086/8088 .....	67
Bảng 4. 3. Tóm tắt các chế độ địa chỉ .....	69
Bảng 4. 4. Các lệnh trao đổi dữ liệu .....	70
Bảng 4. 5. Các lệnh số học và lô gíc .....	72
Bảng 4. 6. Các lệnh rẽ nhánh và lắp tiêu biểu .....	75
Bảng 4. 7. Các kiểu kích thước bộ nhớ cho chương trình hợp ngữ.....	84
Bảng 4. 8. Một số dịch vụ ngắn DOS.....	94
Bảng 5. 1. Các bit trạng thái và việc truy nhập các thanh ghi đoạn. ....	105
Bảng 5. 2. Dải tín hiệu của các mạch nhớ 2732 .....	110
Bảng 5. 3. Mẫu dữ liệu ghi vào ROM .....	111
Bảng 5. 4. Địa chỉ bộ nhớ và nội dung liên quan đến thiết bị ngoại vi và bàn phím. .	117
Bảng 5. 5. Ý nghĩa cờ bàn phím .....	118
Bảng 5. 6. Địa chỉ các thanh ghi 8237A.....	157
Bảng 5. 7. Địa chỉ các thanh ghi trong dùng cho các kênh .....	158
Bảng 5. 8. Các thanh ghi điều khiển và trạng thái.....	158
Bảng 6. 1. Kiến trúc Superscalar truyền thống so với IA-64 .....	168
Bảng 6. 2 Mối quan hệ giữa loại lệnh và loại đơn vị thực thi .....	170

# CHƯƠNG 1 TỔNG QUAN VỀ KIẾN TRÚC MÁY TÍNH

Trong thời đại công nghệ hiện nay, máy tính đóng vai trò quan trọng trong mọi lĩnh vực của cuộc sống. Để hiểu rõ cách máy tính hoạt động, chúng ta cần nắm vững kiến thức về kiến trúc và tổ chức máy tính. Chương này sẽ cung cấp cái nhìn tổng quan về các khái niệm cơ bản, thành phần chính của hệ thống máy tính, lịch sử phát triển, và các kiến trúc tiêu biểu như Harvard. Đây là nền tảng quan trọng để nghiên cứu sâu hơn về khoa học máy tính và công nghệ thông tin.

## 1.1 Khái niệm về kiến trúc và tổ chức máy tính

### 1.1.1 Một số khái niệm

Kiến trúc máy tính (Computer Architecture) và Tổ chức máy tính (Computer Organization) là hai trong số các khái niệm cơ bản của ngành Công nghệ máy tính (Computer Engineering). Có thể nói kiến trúc máy tính là bức tranh toàn cảnh về hệ thống máy tính, còn tổ chức máy tính là bức tranh cụ thể về các thành phần phần cứng của hệ thống máy tính.

Kiến trúc máy tính là khoa học về việc lựa chọn và kết nối các thành phần phần cứng để tạo ra các máy tính đạt được các yêu cầu về chức năng (functionality), hiệu năng (performance) và giá thành (cost). Yêu cầu chức năng đòi hỏi máy tính phải có thêm nhiều tính năng phong phú và hữu ích; yêu cầu hiệu năng đòi hỏi máy tính phải đạt tốc độ xử lý cao hơn và yêu cầu giá thành đòi hỏi máy tính phải càng ngày càng rẻ hơn. Để đạt được cả ba yêu cầu về chức năng, hiệu năng và giá thành là rất khó khăn. Tuy nhiên, nhờ có sự phát triển rất mạnh mẽ của công nghệ vi xử lý, các máy tính ngày nay có tính năng phong phú, nhanh hơn và rẻ hơn so với máy tính các thế hệ trước.

Kiến trúc máy tính được cấu thành từ 3 thành phần con: (i) Kiến trúc tập lệnh (Instruction Set Architecture), (ii) Vi kiến trúc (Micro Architecture) và (iii) Thiết kế hệ thống (System Design).

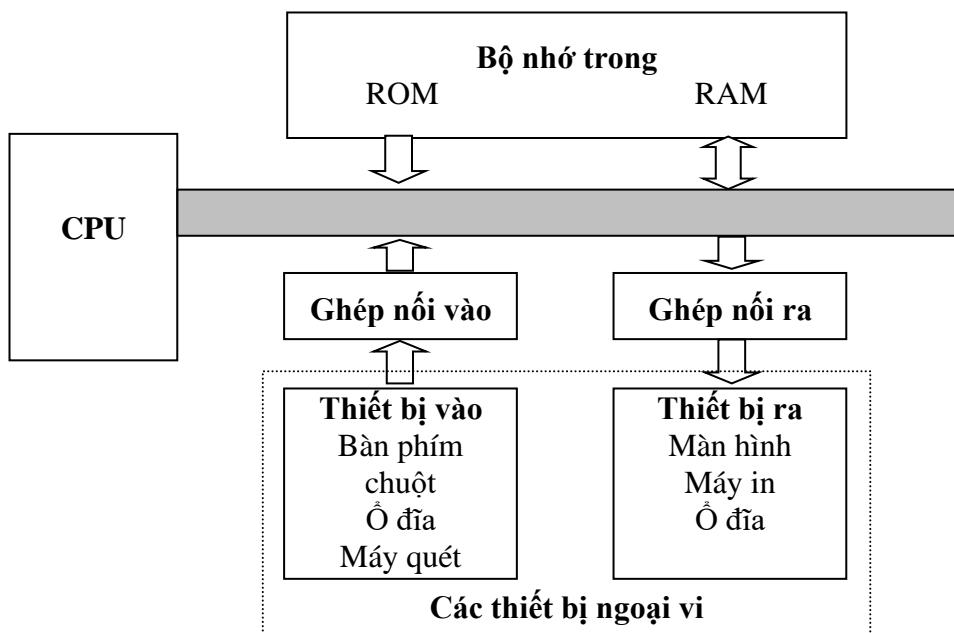
- Kiến trúc tập lệnh là hình ảnh của một hệ thống máy tính ở mức ngôn ngữ máy. Kiến trúc tập lệnh bao gồm các thành phần: tập lệnh, các chế độ địa chỉ, các thanh ghi, khuôn dạng địa chỉ và dữ liệu.
- Vi kiến trúc là mô tả mức thấp về các thành phần của hệ thống máy tính, phối ghép và việc trao đổi thông tin giữa chúng. Vi kiến trúc giúp trả lời hai câu hỏi (1) Các thành phần phần cứng của máy tính kết nối với nhau như thế nào? và (2) Các thành phần phần cứng của máy tính tương tác với nhau như thế nào để thực thi tập lệnh?
- Thiết kế hệ thống: bao gồm tất cả các thành phần phần cứng của hệ thống máy tính, bao gồm: Hệ thống phối ghép (các bus và các chuyển mạch), Hệ thống bộ nhớ, Các cơ chế giảm tải cho CPU (như truy nhập trực tiếp bộ nhớ) và Các vấn đề khác (như đa xử lý và xử lý song song).

Tổ chức máy tính hay cấu trúc máy tính là khoa học nghiên cứu về các bộ phận của máy tính và phương thức làm việc của chúng. Với định nghĩa như vậy, tổ chức

máy tính khá gần gũi với vi kiến trúc – một thành phần của kiến trúc máy tính. Như vậy, có thể thấy rằng, kiến trúc máy tính và khái niệm rộng hơn, nó bao hàm cả tổ chức hay cấu trúc máy tính.

### 1.1.2 Sơ đồ khối chức năng

**Error! Reference source not found.** minh họa sơ đồ khối chức năng của một hệ thống máy tính. Theo đó, hệ thống máy tính gồm bốn thành phần chính: (1) CPU – Khối xử lý trung tâm, (2) Bộ nhớ trong, gồm bộ nhớ ROM và bộ nhớ RAM, (3) Các thiết bị ngoại vi, gồm các thiết bị vào và các thiết bị ra và (4) Bus hệ thống, là hệ thống khen dẫn tín hiệu ghép nối các thành phần kể trên. Ngoài ra, còn có các giao diện ghép nối vào và ghép nối ra dùng để ghép nối các thiết bị ngoại vi vào bus hệ thống. Mục 1.1.3 tiếp theo sẽ mô tả chi tiết chức năng của từng khối.



Hình 1. 1. Sơ đồ khối chức năng của hệ thống máy tính.

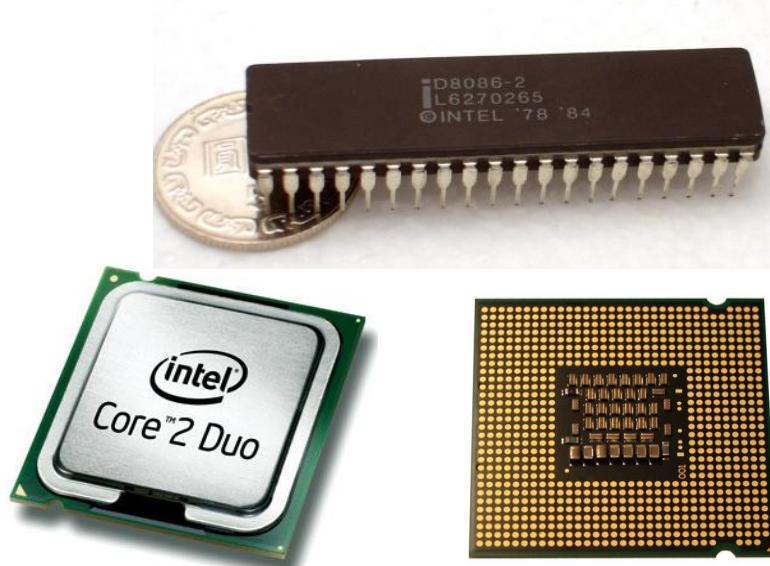
### 1.1.3 Các thành phần của máy tính

#### 1.1.3.1 Khối xử lý trung tâm

Khối xử lý trung tâm (Central Processing Unit - CPU) là thành phần quan trọng nhất - được xem là bộ não của máy tính. Các yêu cầu của hệ thống và của người sử dụng thường được biểu diễn thành các chương trình máy tính, trong đó mỗi chương trình thường được tạo thành từ nhiều lệnh của CPU. CPU đảm nhiệm việc đọc các lệnh của chương trình từ bộ nhớ, giải mã và thực hiện lệnh. Thông qua việc CPU thực hiện các lệnh của chương trình, máy tính có khả năng cung cấp các tính năng hữu ích cho người sử dụng.

CPU là vi mạch tích hợp với mật độ rất cao, được cấu thành từ bốn thành phần con: (1) Bộ điều khiển (Control Unit - CU), (2) Bộ tính toán số học và logic (Arithmetic and Logic Unit - ALU), (3) Các thanh ghi (Registers) và bus trong CPU (Internal Bus). Bộ điều khiển có nhiệm vụ đọc, giải mã và điều khiển quá trình thực

hiện lệnh. Bộ tính toán số học và logic chuyên thực hiện các phép toán số học như cộng trừ, nhân, chia, và các phép toán logic như và, hoặc, phủ định và các phép dịch, quay. Các thanh ghi là kho chứa lệnh và dữ liệu tạm thời cho CPU xử lý. Bus trong CPU có nhiệm vụ truyền dẫn các tín hiệu giữa các bộ phận trong CPU và kết nối với hệ thống bus ngoài. Hình 1. 2 minh họa hai CPU của hãng Intel là 8086 ra đời năm 1978 và Core 2 Duo ra đời năm 2006.



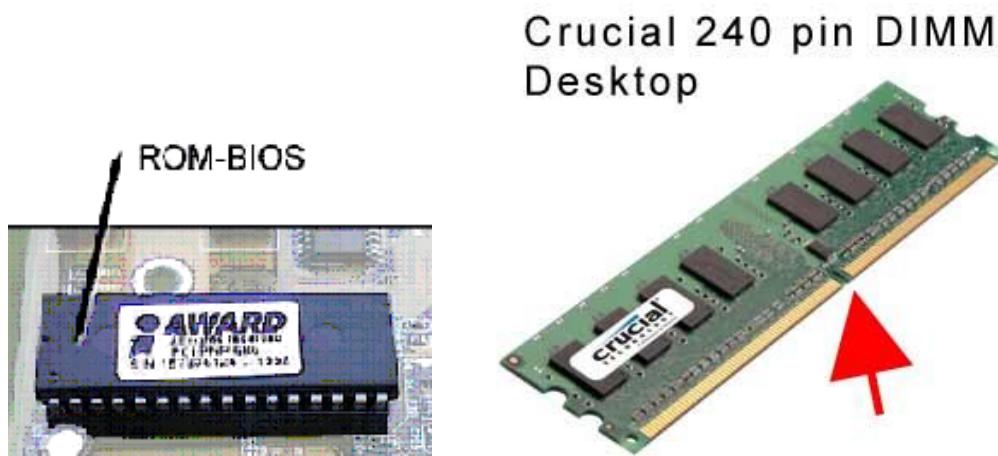
Hình 1. 2. CPU của hãng Intel: 8086 và Core 2 Duo.

#### 1.1.3.2 Bộ nhớ trong

Bộ nhớ trong, còn gọi là bộ nhớ chính (Internal Memory hay Main Memory) là kho chứa lệnh và dữ liệu của hệ thống và của người dùng phục vụ CPU xử lý. Bộ nhớ trong thường là bộ nhớ bán dẫn, bao gồm hai loại: (1) Bộ nhớ chỉ đọc (Read Only Memory – ROM) và (2) Bộ nhớ truy cập ngẫu nhiên (Random Access Memory – RAM). ROM thường được sử dụng để lưu lệnh và dữ liệu của hệ thống. Thông tin trong ROM được nạp từ khi sản xuất và thường chỉ có thể đọc ra trong quá trình sử dụng. Hơn nữa thông tin trong ROM luôn tồn tại kể cả khi không có nguồn điện nuôi.

Khác với bộ nhớ ROM, bộ nhớ RAM thường được sử dụng để lưu lệnh và dữ liệu của cả hệ thống và của người dùng. RAM thường có dung lượng lớn hơn nhiều so với ROM. Tuy nhiên, thông tin trong RAM chỉ tồn tại khi có nguồn điện nuôi.

Hình 1. 3. minh họa vi mạch bộ nhớ ROM và các vi mạch nhớ RAM gắn trên một thanh nhớ RAM.



Hình 1. 3. Bộ nhớ ROM và RAM

#### 1.1.3.3 Các thiết bị vào ra

Các thiết bị vào ra (Input – Output devices), hay còn gọi là các thiết bị ngoại vi (Peripheral devices) đảm nhiệm việc nhập dữ liệu vào, điều khiển hệ thống và kết xuất dữ liệu ra. Có hai nhóm thiết bị ngoại vi: (1) Các thiết bị vào (Input devices) và (2) Các thiết bị ra (Output devices). Các thiết bị vào dùng để nhập dữ liệu vào và điều khiển hệ thống, gồm: bàn phím (keyboard), chuột (mouse), ổ đĩa (Disk Drives), máy quét ảnh (Scanners),... Các thiết bị ra dùng để xuất dữ liệu ra, gồm: màn hình (Screen), máy in (Printers), ổ đĩa (Disk Drives), máy vẽ (Plotters),...

#### 1.1.3.4 Bus hệ thống

Bus hệ thống (System Bus) là một tập các đường dây kết nối CPU với các thành phần khác của máy tính. Bus hệ thống thường gồm ba bus con: Bus địa chỉ – Bus A (Address bus), Bus dữ liệu – Bus D (Data bus), Bus điều khiển - Bus C (Control bus). Bus địa chỉ có nhiệm vụ truyền tín hiệu địa chỉ từ CPU đến bộ nhớ và các thiết bị ngoại vi; Bus dữ liệu vận chuyển các tín hiệu dữ liệu theo hai chiều đi và đến CPU; Bus điều khiển truyền tín hiệu điều khiển từ CPU đến các thành phần khác, đồng thời truyền tín hiệu trạng thái của các thành phần khác đến CPU.

## 1.2 Lịch sử phát triển máy tính

Lịch sử phát triển máy tính có thể được chia thành 5 thế hệ chính, phụ thuộc vào sự phát triển của công nghệ mạch điện tử và các tiến bộ trong kiến trúc máy tính. Dưới đây là chi tiết về các thế hệ máy tính, bao gồm cả các kiến trúc mới và các bộ vi xử lý tiêu biểu như 8086/8088, vốn đóng vai trò quan trọng trong lịch sử phát triển của máy tính.

### Thế hệ 1 (1944-1959)

Máy tính thế hệ 1 sử dụng đèn điện tử làm linh kiện chính và băng từ làm thiết bị vào ra. Mật độ tích hợp linh kiện vào khoảng 1.000 linh kiện/foot<sup>3</sup> (1 foot = 30.48 cm). Các máy tính thế hệ này có kích thước rất lớn, tiêu thụ nhiều điện năng và có tốc độ xử lý chậm. Đại diện tiêu biểu là siêu máy tính ENIAC (Electronic Numerical

Integrator and Computer), trị giá 500.000 USD, được sử dụng chủ yếu cho các tính toán quân sự và khoa học.

### **Thế hệ 2 (1960-1964)**

Máy tính thế hệ 2 sử dụng bóng bán dẫn (transistor) làm linh kiện chính, thay thế cho đèn điện tử. Mật độ tích hợp linh kiện tăng lên khoảng 100.000 linh kiện/foot<sup>3</sup>. Các máy tính thế hệ này nhỏ gọn hơn, tiêu thụ ít điện năng hơn và có tốc độ xử lý nhanh hơn. Đại diện tiêu biểu là UNIVAC 1107, UNIVAC III, IBM 7070, IBM 7080, IBM 7090, và các dòng máy thuộc series 1400 và 1600. Giá thành của UNIVAC đầu tiên ra đời năm 1951 có giá khởi điểm 159.000 USD, trong khi các phiên bản sau có giá lên đến 1.250.000 – 1.500.000 USD

### **Thế hệ 3 (1964-1975)**

Máy tính thế hệ 3 sử dụng mạch tích hợp (IC – Integrated Circuit) làm linh kiện chính. Mật độ tích hợp linh kiện tăng vọt lên khoảng 10.000.000 linh kiện/foot<sup>3</sup>. Các máy tính thế hệ này có kích thước nhỏ hơn, hiệu suất cao hơn và giá thành giảm đáng kể. Đại diện tiêu biểu là UNIVAC 9000 series, IBM System/360, IBM System 3, và IBM System 7. Sự ra đời của các hệ thống máy tính đa năng này, có thể chạy nhiều ứng dụng khác nhau.

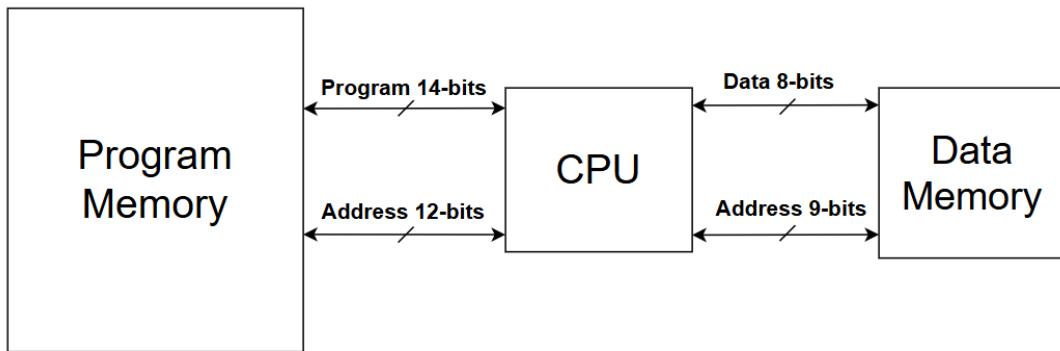
### **Thế hệ 4 (1975-1989)**

Máy tính thế hệ 4 sử dụng mạch tích hợp loại lớn (LSI – Large Scale Integrated Circuit) làm linh kiện chính, với mật độ tích hợp đạt khoảng 1 tỷ linh kiện/foot<sup>3</sup>. Những máy tính này có hiệu năng vượt trội, kích thước nhỏ gọn và giá thành tiếp tục giảm. Đại diện tiêu biểu gồm IBM System 3090, IBM RISC 6000, IBM RT và siêu máy tính Cray 2 XMP. Đặc biệt, sự xuất hiện của các bộ vi xử lý như Intel 8086/8088 (1978-1979) đánh dấu kỷ nguyên máy tính cá nhân (PC). Intel 8086 là bộ vi xử lý 16-bit, được sử dụng rộng rãi trong các máy tính cá nhân đầu tiên, trong khi Intel 8088 – phiên bản rút gọn của 8086 – được chọn làm bộ xử lý trung tâm cho IBM PC đầu tiên (1981).

### **Thế hệ 5 (1990 - nay)**

Máy tính thế hệ 5 sử dụng mạch tích hợp loại siêu lớn (VLSI – Very Large Scale Integrated Circuit) với mật độ tích hợp rất cao, khi kích thước transistor giảm xuống còn 180 – 45 nanomet. Những máy tính này có hiệu năng xử lý cực cao và tích hợp nhiều tính năng tiên tiến như xử lý song song, xử lý đồ họa và âm thanh. Đại diện tiêu biểu là các bộ vi xử lý của Intel như Pentium II, Pentium III, Pentium IV, Core Duo, Core 2 Duo, Core i-series và các bộ vi xử lý đa nhân hiện đại. Kiến trúc mới bao gồm kiến trúc đa nhân (Multi-core), giúp tích hợp nhiều lõi xử lý trên một chip để tăng hiệu suất xử lý đa nhiệm; kiến trúc ARM, phổ biến trong thiết bị di động và IoT nhờ khả năng tiết kiệm năng lượng; kiến trúc GPU (Graphics Processing Unit), tối ưu hóa cho xử lý đồ họa và tính toán song song, đặc biệt hữu ích trong AI và machine learning; cùng với kiến trúc lượng tử (Quantum Computing), hiện đang trong giai đoạn nghiên cứu và phát triển, hứa hẹn sẽ cách mạng hóa khả năng tính toán trong tương lai.

### 1.3 Kiến trúc máy tính Harvard



Hình 1. 4. Kiến trúc máy tính Harvard

Kiến trúc máy tính Harvard là một kiến trúc tiên tiến như minh họa trên Hình 1. 4.. Kiến trúc máy tính Harvard chia bộ nhớ trong thành hai phần riêng rẽ: Bộ nhớ lưu chương trình (Program Memory) và Bộ nhớ lưu dữ liệu (Data Memory). Hai hệ thống bus riêng được sử dụng để kết nối CPU với bộ nhớ lưu chương trình và bộ nhớ lưu dữ liệu. Mỗi hệ thống bus đều có đầy đủ ba thành phần để truyền dẫn các tín hiệu địa chỉ, dữ liệu và điều khiển.

Máy tính dựa trên kiến trúc Harvard có khả năng đạt được tốc độ xử lý cao hơn máy tính dựa trên kiến trúc von-Neumann do kiến trúc Harvard hỗ trợ hai hệ thống bus độc lập với băng thông lớn hơn. Ngoài ra, nhờ có hai hệ thống bus độc lập, hệ thống nhớ trong kiến trúc Harvard hỗ trợ nhiều lệnh truy nhập bộ nhớ tại một thời điểm, giúp giảm xung đột truy nhập bộ nhớ, đặc biệt khi CPU sử dụng kỹ thuật đường ống (pipeline).

### 1.4 Các hệ số đếm và tổ chức dữ liệu trên máy tính

#### 1.4.1 Các hệ số đếm

Trong đời sống hàng ngày, hệ đếm thập phân (Decimal Numbering System) là hệ đếm thông dụng nhất do phù hợp với cách tính toán và giao tiếp của con người. Tuy nhiên, trong hầu hết các hệ thống tính toán, hệ đếm nhị phân (Binary Numbering System) lại được sử dụng để biểu diễn dữ liệu. Lý do là vì máy tính hoạt động dựa trên các thiết bị điện tử chỉ có thể nhận biết hai trạng thái: có điện (1) và không có điện (0). Trong hệ đếm nhị phân, chỉ 2 chữ số 0 và 1 được sử dụng: 0 biểu diễn giá trị Sai (False) và 1 biểu diễn giá trị Đúng (True). Điều này giúp máy tính xử lý thông tin một cách chính xác và hiệu quả. Ngoài ra, hệ đếm thập lục phân (Hexadecimal Numbering System) cũng được sử dụng rộng rãi. Hệ thập lục phân sử dụng 16 chữ số: 0-9, A, B, C, D, E, F, giúp đơn giản hóa việc biểu diễn dữ liệu nhị phân một cách ngắn gọn và dễ hiểu hơn, đồng thời hỗ trợ việc tổ chức bộ nhớ và truyền tải thông tin trong các kiến trúc máy tính. Việc sử dụng các hệ số đếm này là nền tảng để máy tính có thể lưu trữ, xử lý và truyền đạt dữ liệu một cách hiệu quả, tạo nên cốt lõi của các hệ thống máy tính hiện đại.

#### *1.4.1.1 Hệ đếm thập phân*

Hệ đếm thập phân là hệ đếm cơ số 10, sử dụng 10 chữ số: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. Mỗi số trong hệ 10 có thể được biểu diễn thành một đa thức:

$$a_n a_{n-1} \cdots a_1 = a_n * 10^{n-1} a_{n-1} * 10^{n-2} * \cdots * a_1 * 10^0 \quad (1.1)$$

## Ví dụ:

$$123 = 1*10^2 + 2 * 10^1 + 3*10^0 = 100 + 20 + 3$$

$$\begin{aligned}123,456 &= 1*10^2 + 2 * 10^1 + 3*10^0 + 4*10^{-1} + 5*10^{-2} + 6*10^{-3} \\&= 100 + 20 + 3 + 0.4 + 0.05 + 0.006\end{aligned}$$

#### **1.4.1.2 Hệ đếm nhị phân**

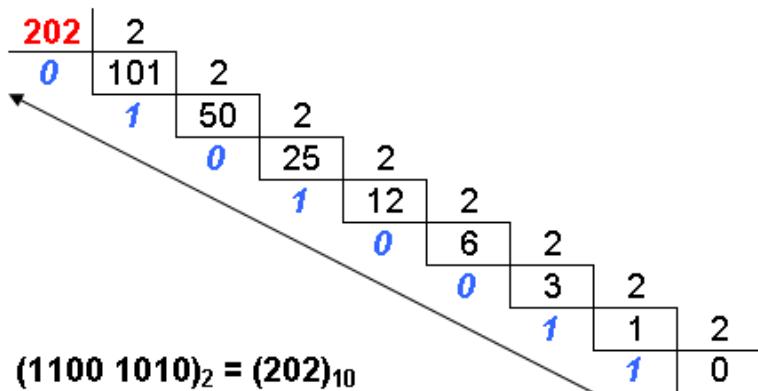
Hệ đếm nhị phân là hệ đếm cơ số 2, chỉ sử dụng 2 chữ số: 0 và 1. Mỗi số trong hệ 2 cũng có thể được biểu diễn thành một đa thức:

$$(a_n a_{n-1} \cdots a_1)_2 = a_n * 2^{n-1} a_{n-1} * 2^{n-2} * \cdots * a_1 2^0 \quad (1.2)$$

## Ví dụ:

$$(11001010)_2 = 1*2^7 + 1*2^6 + 0*2^5 + 0*2^4 + 1*2^3 + 0*2^2 + 1*2^1 + 0*2^0 \\ = 128 + 64 + 8 + 2 = (202)_{10}$$

Việc chuyển đổi số hệ thập phân sang số hệ nhị phân có thể được thực hiện theo thuật toán đơn giản như minh họa trên Hình 1. 5.



Hình 1.5. Chuyển đổi số hệ thập phân sang số hệ nhị phân

#### *1.4.1.3 Hệ đếm thập lục phân*

Hệ đếm thập lục phân là hệ đếm cơ số 16, sử dụng 16 chữ số: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F. Mỗi số trong hệ 16 được biểu diễn bởi 4 chữ số trong hệ nhị phân như minh họa trên Hình 1. **6**. Ưu điểm của hệ thập lục phân là số thập lục phân có thể chuyển đổi sang số hệ nhị phân và ngược lại một cách dễ dàng và cần ít chữ số hơn hệ nhị phân để biểu diễn cùng một đơn vị dữ liệu.

Hexa	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Decimal	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Binary	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111

Hình 1. 6. Giá trị các số thập lục phân theo hệ thập phân và nhị phân

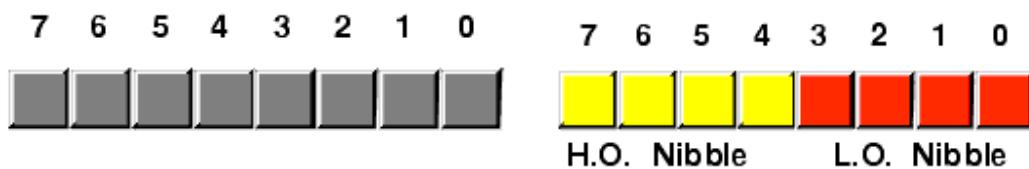
#### 1.4.2 Tổ chức dữ liệu trên máy tính

Dữ liệu trên máy tính được biểu diễn theo các đơn vị (unit). Các đơn vị biểu diễn dữ liệu cơ sở bao gồm: bit, nibble, byte, word và double-word.

- Bit là đơn vị dữ liệu nhỏ nhất, mỗi bit chỉ lưu được tối đa 2 giá trị: 0 hoặc 1, hay được hiểu là đúng hoặc sai.
- Nibble là đơn vị kế tiếp của bit. Một nibble là một nhóm gồm 4 bits. Mỗi nibble có thể lưu tối đa 16 giá trị, từ  $(0000)_2$  đến  $(1111)_2$  hoặc dưới dạng một chữ số thập lục phân.

Byte là đơn vị dữ liệu tiếp theo của nibble. Một byte bao gồm 8 bits hoặc 2 nibbles. Một byte có thể lưu được 256 giá trị, từ  $(0000\ 0000)_2$  đến  $(1111\ 1111)_2$  hoặc từ  $(00)_{16}$  đến  $(FF)_{16}$ . Hình 1. 6 sau cho thấy cách biểu diễn dữ liệu của một byte với hai nibble. Trong một byte, 2 nibble sẽ được phân chia như sau:

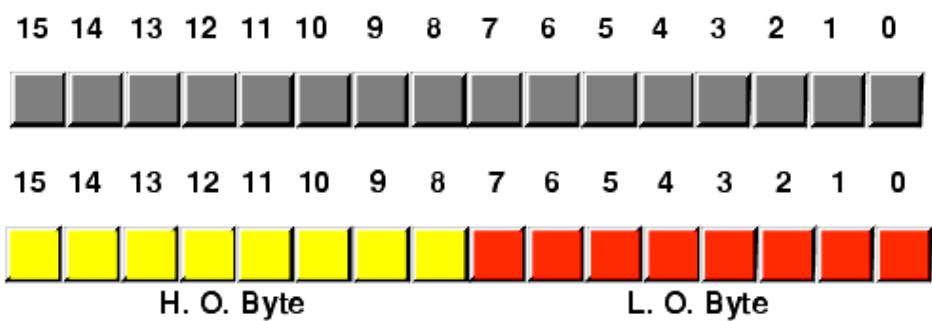
- H.O Nibble (High-Order Nibble): Là 4 bits cao (bên trái).
- L.O Nibble (Low-Order Nibble): Là 4 bits thấp (bên phải).



Hình 1. 7. Đơn vị biểu diễn dữ liệu Byte

Word (từ) là đơn vị dữ liệu kế tiếp byte. Một word là một nhóm gồm 16 bits hoặc 2 bytes. Một word có thể lưu được  $2^{16}$  (65536) giá trị, từ  $(0000)_{16}$  đến  $(FFFF)_{16}$ . Hình 1. 6 minh họa cách tổ chức dữ liệu của một word với hai byte. Trong một word, dữ liệu được chia thành hai phần chính:

- H.O Byte (High-Order Byte): Là byte cao (bên trái), chứa 8 bits đầu tiên.
- L.O Byte (Low-Order Byte): Là byte thấp (bên phải), chứa 8 bits cuối cùng.

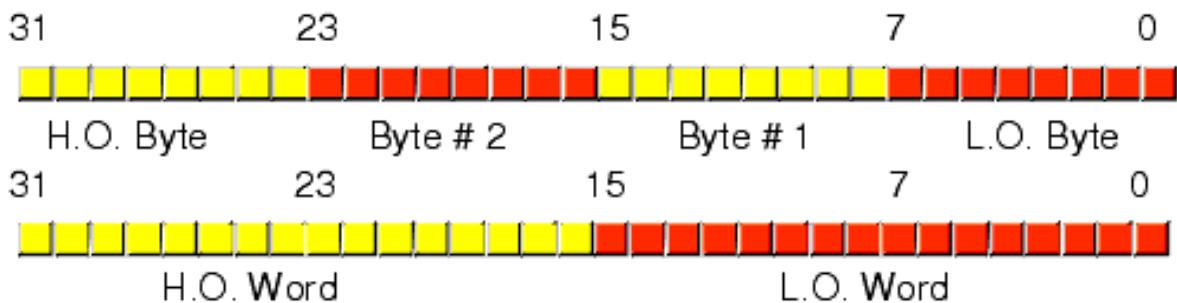


Hình 1. 8. Đơn vị biểu diễn dữ liệu Word

Double Word (từ kép) là đơn vị biểu diễn dữ liệu cơ sở lớn nhất. Một double word là một nhóm gồm 32 bits, tương đương 4 bytes hoặc 2 words. Một double word có thể lưu được  $2^{32}$  giá trị, từ  $(0000\ 0000)_{16}$  đến  $(FFFF\ FFFF)_{16}$ . Hình 1. 6 minh họa

cách tổ chức dữ liệu của một double word. Trong một double word, dữ liệu được tổ chức và chia thành các phần như sau:

- H.O Word (High-Order Word): Là từ cao, bao gồm 16 bits đầu tiên (2-byte đầu).
  - H.O Byte (High-Order Byte): Byte cao nhất trong H.O Word (byte đầu tiên).
  - L.O Byte (Low-Order Byte): Byte thấp nhất trong H.O Word (byte thứ hai).
- L.O Word (Low-Order Word): Là từ thấp, bao gồm 16 bits còn lại (2-byte cuối).
  - H.O Byte (High-Order Byte): Byte cao nhất trong L.O Word (byte thứ ba).
  - L.O Byte (Low-Order Byte): Byte thấp nhất trong L.O Word (byte thứ tư).



Hình 1. 9. Đơn vị biểu diễn dữ liệu Double word

#### 1.4.3 Số có dấu và số không dấu

Trong các hệ thống tính toán, với cùng một số bit có thể biểu diễn các giá trị khác nhau nếu số được biểu diễn là có dấu hoặc không dấu. Để biểu diễn số có dấu, người ta sử dụng bit cao nhất (bên trái nhất) để biểu diễn dấu của số - gọi là bit dấu, chẳng hạn bit dấu có giá trị 0 là số dương và bit dấu có giá trị 1 là số âm. Với số không dấu, tất cả các bit được sử dụng để biểu diễn giá trị của số. Như vậy, miền giá trị có thể biểu diễn của một số gồm n bit như sau:

- **Số có dấu:** miền biểu diễn từ  $-2^{n-1}$  đến  $+2^{n-1} - 1$ 
  - 8 bits: từ -128 đến +127
  - 16 bits: từ -32.768 đến +32.767
  - 32 bits: từ -2.147.483.648 đến +2.147.483.647
- **Số không dấu:** từ 0 đến  $2^n - 1$ 
  - 8 bits: từ 0 đến 255
  - 16 bits: từ 0 đến 65.535
  - 32 bits: từ 0 đến 4.294.967.295

#### 1.4.4 Bảng mã ASCII

Bảng mã ASCII (American Standard Code for Information Interchange) là bảng mã các ký tự chuẩn tiếng Anh dùng cho trao đổi dữ liệu giữa các hệ thống tính toán. Bảng mã ASCII sử dụng 8 bit để biểu diễn một ký tự, cho phép định nghĩa tổng số 256 ký tự, đánh số từ 0 đến 255. 32 ký tự đầu tiên và ký tự số 127 là các ký tự điều khiển (không in ra được). Các ký tự từ số 32 đến 126 là các ký tự có thể in được (gồm cả dấu trắng). Các vị trí còn lại trong bảng (128-255) để dành cho sử dụng trong tương lai. Hình 1. 10. và Hình 1. 11 lần lượt là minh họa các ký tự điều khiển và các ký tự in được của bảng mã ASCII.

Binary	Oct	Dec	Hex	Abbr	PR <sup>[t 1]</sup>	CS <sup>[t 2]</sup>	CEC <sup>[t 3]</sup>	Description
000 0000	000	0	00	NUL	NUL	^@	\0	Null character
000 0001	001	1	01	SOH	SOH	^A		Start of Header
000 0010	002	2	02	STX	STX	^B		Start of Text
000 0011	003	3	03	ETX	ETX	^C		End of Text
000 0100	004	4	04	EOT	EOT	^D		End of Transmission
000 0101	005	5	05	ENQ	ENQ	^E		Enquiry
000 0110	006	6	06	ACK	ACK	^F		Acknowledgment
000 0111	007	7	07	BEL	BEL	^G	\a	Bell
000 1000	010	8	08	BS	BS	^H	\b	Backspace <sup>[t 4][t 5]</sup>
000 1001	011	9	09	HT	HT	^I	\t	Horizontal Tab
000 1010	012	10	0A	LF	LF	^J	\n	Line feed

Hình 1. 10. Bảng mã ASCII - Một số ký tự điều khiển

Binary	Oct	Dec	Hex	Glyph	Binary	Oct	Dec	Hex	Glyph	Binary	Oct	Dec	Hex	Glyph
010 0000	040	32	20		100 0000	100	64	40	@	110 0000	140	96	60	`
010 0001	041	33	21	!	100 0001	101	65	41	A	110 0001	141	97	61	a
010 0010	042	34	22	"	100 0010	102	66	42	B	110 0010	142	98	62	b
010 0011	043	35	23	#	100 0011	103	67	43	C	110 0011	143	99	63	c
010 0100	044	36	24	\$	100 0100	104	68	44	D	110 0100	144	100	64	d
010 0101	045	37	25	%	100 0101	105	69	45	E	110 0101	145	101	65	e
010 0110	046	38	26	&	100 0110	106	70	46	F	110 0110	146	102	66	f
010 0111	047	39	27	'	100 0111	107	71	47	G	110 0111	147	103	67	g
010 1000	050	40	28	(	100 1000	110	72	48	H	110 1000	150	104	68	h
010 1001	051	41	29	)	100 1001	111	73	49	I	110 1001	151	105	69	i
010 1010	052	42	2A	*	100 1010	112	74	4A	J	110 1010	152	106	6A	j
010 1011	053	43	2B	+	100 1011	113	75	4B	K	110 1011	153	107	6B	k
010 1100	054	44	2C	,	100 1100	114	76	4C	L	110 1100	154	108	6C	l

Hình 1. 11. Bảng mã ASCII - Các ký tự in được

## 1.5 Kết luận chương

Trong chương này, chúng ta đã khám phá những khái niệm cơ bản nhất về kiến trúc và tổ chức máy tính. Kiến trúc máy tính được coi là bức tranh tổng quan về hệ thống máy tính, bao gồm các khía cạnh về tập lệnh, vi kiến trúc và thiết kế hệ thống. Tổ chức máy tính tập trung vào việc nghiên cứu các bộ phận cụ thể và cách chúng hoạt động. Ngoài ra, chúng ta đã xem xét sơ đồ khối chức năng, các thành phần chính trong một hệ thống máy tính bao gồm cpu, bộ nhớ trong, các thiết bị ngoại vi và bus hệ thống. Từ CPU được ví như "bộ não" của máy tính đến bộ nhớ trong là kho chứa lệnh và dữ liệu, tất cả đã được phân tích chi tiết. Chương cũng đề cập đến lịch sử phát triển của máy tính qua 5 thế hệ, từ các máy sử dụng đèn điện tử, bóng bán dẫn đến các vi mạch tích hợp siêu lớn VLSI. Chúng ta cũng đã hiểu rõ những đóng góp của mỗi giai đoạn trong việc nâng cao tính năng, hiệu năng và giá thành máy tính. Phần cuối chương đã xem xét các kiến trúc máy tính điển hình như kiến trúc Harvard, cũng như các hệ số đếm và tổ chức dữ liệu trên máy tính. Việc nắm bắt các khái niệm và nguyên lý này đặt nền tảng cho việc nghiên cứu sâu hơn về khoa học máy tính. Những nội dung trong chương này sẽ làm tiền đề quan trọng cho các chương sau, nơi chúng ta sẽ đào sâu hơn vào tập lệnh, các công nghệ vi xử lý và các kỹ thuật tối ưu hóa hiệu năng hệ thống.

## 1.6 Câu hỏi ôn tập

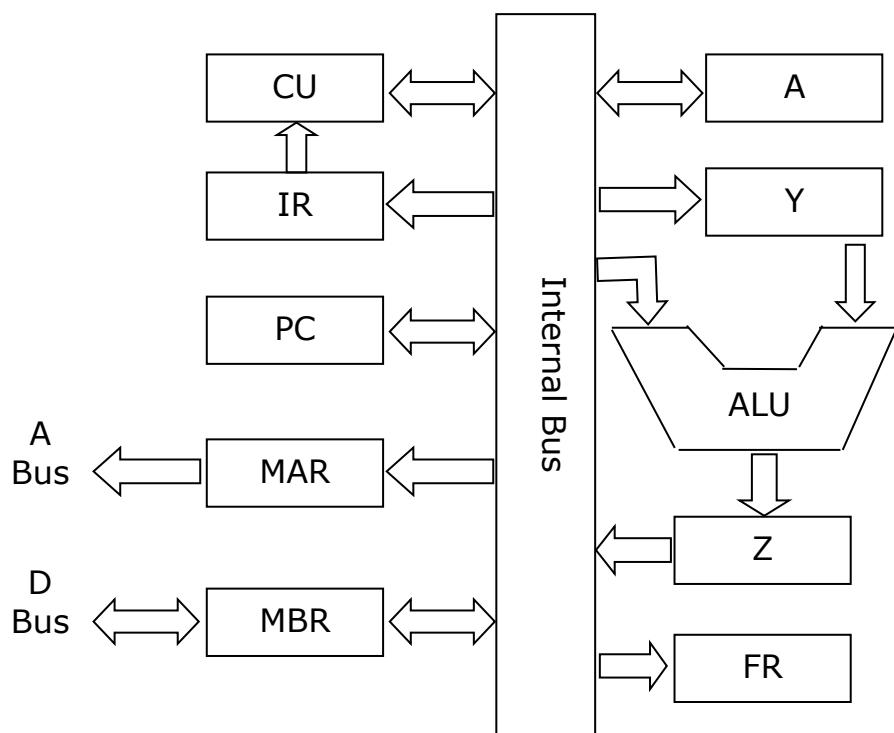
1. Phân biệt khái niệm kiến trúc máy tính và tổ chức máy tính.
2. Nêu sơ đồ khối và mô tả chức năng từng khối của hệ thống máy tính.
3. So sánh các đặc điểm chính giữa kiến trúc máy tính von-Neumann và Harvard.
4. Giải thích các hệ đếm nhị phân (hệ 2), thập phân (hệ 10), và thập lục phân (hệ 16).
5. Chuyển đổi số từ hệ thập phân sang hệ nhị phân và ngược lại.
6. Mô tả các đơn vị lưu trữ dữ liệu trên máy tính, bao gồm: bit, byte, word, và double word.
7. So sánh số có dấu và số không dấu trên máy tính.
8. Liệt kê và mô tả vai trò của các thành phần chính trong CPU: Bộ điều khiển, ALU, thanh ghi, và bus nội bộ.
9. Giải thích các đặc điểm của bộ nhớ ROM và RAM. So sánh chúng về chức năng và đặc tính.
10. (\*) Phân tích ưu nhược điểm của kiến trúc Harvard so với kiến trúc von-Neumann.
11. (\*) Phân tích ưu điểm của kỹ thuật đường ống (pipeline) trong kiến trúc Harvard.
12. (\*) Mô tả quá trình CPU thực hiện lệnh theo kiến trúc von-Neumann.

## CHƯƠNG 2 KHỐI XỬ LÝ TRUNG TÂM

Bộ xử lý trung tâm (CPU) là “bộ não” của máy tính, đảm nhiệm việc đọc, giải mã và thực thi lệnh. Chương này giới thiệu cấu trúc tổng quát của CPU, bao gồm bộ điều khiển (CU), bộ tính toán số học và logic (ALU), hệ thống thanh ghi và bus trong. Đồng thời, chu trình xử lý lệnh cũng được phân tích để làm rõ cách CPU vận hành. Hiểu rõ những thành phần này giúp ta nắm vững nguyên lý hoạt động của máy tính và tối ưu hiệu suất xử lý.

### 2.1 Sơ đồ khối tổng quát và chu trình xử lý lệnh

#### 2.1.1 Sơ đồ khối tổng quát của CPU



Hình 2. 1. Sơ đồ khối tổng quát của CPU

Hình 2. 1. trình bày sơ đồ khối nguyên lý tổng quát của CPU. Các thành phần của CPU theo sơ đồ này gồm:

- Bộ điều khiển (Control Unit – CU)
- Bộ tính toán số học và logic (Arithmetic and Logic Unit)
- Bus trong CPU (CPU Internal Bus)
- Các thanh ghi của CPU:
  - Thanh ghi tích luỹ A (Accumulator)
  - Bộ đếm chương trình PC (Program Counter)
  - Thanh ghi lệnh IR (Instruction Register)
  - Thanh ghi địa chỉ bộ nhớ MAR (Memory Address Register)
  - Thanh ghi đệm dữ liệu MBR (Memory Buffer Register)
  - Các thanh ghi tạm thời Y và Z

- Thanh ghi cờ FR (Flag Register)

### **2.1.2 Chu trình xử lý lệnh**

Như đã trình bày trong chương 1, nhiệm vụ chủ yếu của CPU là đọc lệnh từ bộ nhớ, giải mã và thực hiện lệnh của chương trình. Khoảng thời gian để CPU thực hiện xong một lệnh kể từ khi CPU cấp phát tín hiệu địa chỉ ô nhớ chứa lệnh đến khi nó hoàn tất việc thực hiện lệnh được gọi là *chu kỳ lệnh* (Instruction Cycle). Mỗi chu kỳ lệnh của CPU được mô tả theo các bước sau:

1. Khi một chương trình được kích hoạt, hệ điều hành (OS - Operating System) nạp mã chương trình vào bộ nhớ trong;
2. Địa chỉ của ô nhớ chứa lệnh đầu tiên của chương trình được nạp vào bộ đếm chương trình PC;
3. Địa chỉ ô nhớ chứa lệnh từ PC được chuyển đến bus địa chỉ thông qua thanh ghi MAR;
4. Bus địa chỉ chuyển địa chỉ ô nhớ đến đơn vị quản lý bộ nhớ (MMU - Memory Management Unit);
5. MMU chọn ra ô nhớ và thực hiện lệnh đọc nội dung ô nhớ;
6. Lệnh (chứa trong ô nhớ) được chuyển ra bus dữ liệu và tiếp theo được chuyển tiếp đến thanh ghi MBR;
7. MBR chuyển lệnh đến thanh ghi lệnh IR; IR chuyển lệnh vào bộ điều khiển CU;
8. CU giải mã lệnh và sinh các tín hiệu điều khiển cần thiết, yêu cầu các bộ phận chức năng của CPU, như ALU thực hiện lệnh;
9. Giá trị địa chỉ trong bộ đếm PC được tăng lên 1 đơn vị lệnh và nó trả đến địa chỉ của ô nhớ chứa lệnh tiếp theo;
10. Các bước từ 3-9 được lặp lại với tất cả các lệnh của chương trình.

## **2.2 Các thanh ghi**

### **2.2.1 Giới thiệu về thanh ghi**

Thanh ghi (registers) là các ô nhớ bên trong CPU, có nhiệm vụ lưu trữ tạm thời lệnh và dữ liệu cho CPU xử lý. Thanh ghi thường có kích thước nhỏ, nhưng tốc độ làm việc rất cao - bằng tốc độ CPU. Các CPU cũ (80x86) có khoảng 16-32 thanh ghi. Các CPU hiện đại (như Pentium 4 và Core Duo) có thể có đến hàng trăm thanh ghi. Kích thước thanh ghi phụ thuộc vào thiết kế CPU. Các kích thước thông dụng của thanh ghi là 8, 16, 32, 64, 128 và 256 bit. CPU Intel 8086 và 80286 có các thanh ghi 8 bit và 16 bit. CPU Intel 80386 và Pentium II có các thanh ghi 16 bit và 32 bit. Các CPU Pentium 4 và Core Duo có các thanh ghi 32 bit, 64 bit và 128 bit.

#### **2.2.1.1 Thanh tích luỹ A**

Thanh tích luỹ A (Accumulator) là một trong các thanh ghi quan trọng nhất của CPU. Thanh ghi A không những được sử dụng để lưu toán hạng vào mà còn dùng để chứa kết quả ra. Ngoài ra, thanh ghi A còn thường được dùng trong các lệnh trao đổi

dữ liệu với các thiết bị vào ra. Kích thước của thanh ghi A bằng kích thước từ xử lý của CPU: 8 bit, 16 bit, 32 bit hoặc 64 bit.

Ví dụ về việc sử dụng thanh ghi A trong phép toán:  $x + y \rightarrow s$

- Nạp toán hạng x vào thanh ghi A
- Nạp toán hạng y vào thanh ghi tạm thời Y
- ALU thực hiện phép cộng A + Y và lưu kết quả vào thanh ghi Z
- Kết quả phép tính từ Z được chuyển về thanh ghi A.
- Kết quả trong thanh ghi A được lưu vào ô nhớ s.

#### 2.2.1.2 Bộ đếm chương trình PC

Bộ đếm chương trình PC (Program Counter) hoặc con trỏ lệnh (IP – Instruction pointer) luôn chứa địa chỉ của ô nhớ chứa lệnh kế tiếp được thực hiện. Đặc biệt, PC chứa địa chỉ của ô nhớ chứa lệnh đầu tiên của chương trình khi chương trình được kích hoạt và được hệ điều hành nạp vào bộ nhớ. Khi CPU thực hiện xong một lệnh, địa chỉ của ô nhớ chứa lệnh tiếp theo được nạp vào PC. Kích thước của PC phụ thuộc vào thiết kế CPU. Các kích thước thông dụng của PC là 8 bit, 16 bit, 32 bit và 64 bit.

#### 2.2.1.3 Thanh ghi lệnh IR

Thanh ghi lệnh IR (Instruction register) lưu lệnh đang thực hiện. IR nhận lệnh từ MBR và chuyển tiếp lệnh đến CU giải mã và thực hiện. Lệnh được giữ trong thanh ghi IR trong suốt quá trình thực thi để đảm bảo rằng tất cả các bước của quá trình thực thi được hoàn thành chính xác. Thanh ghi IR đóng vai trò như một cầu nối quan trọng giữa bộ nhớ và bộ giải mã, đảm bảo rằng các lệnh được thực thi một cách chính xác và hiệu quả.

#### 2.2.1.4 Các thanh ghi MAR và MBR

MAR là thanh ghi địa chỉ bộ nhớ (Memory address register) - giao diện giữa CPU và bus địa chỉ. MAR nhận địa chỉ ô nhớ chứa lệnh tiếp theo từ PC và chuyển tiếp ra bus địa chỉ. MAR hoạt động như một cầu nối giữa CPU và bộ nhớ, cho phép CPU chỉ định chính xác vị trí dữ liệu cần truy cập trong bộ nhớ.

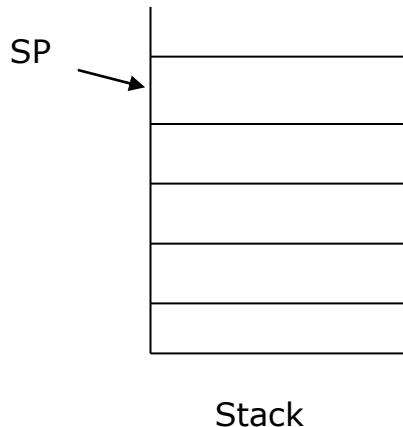
MBR là thanh ghi đệm dữ liệu (Memory buffer register) - giao diện giữa CPU và bus địa chỉ. MBR nhận lệnh từ bus địa chỉ và chuyển tiếp lệnh đến IR thông qua bus trong CPU. MBR hoạt động như một điểm trung chuyển dữ liệu giữa CPU và bộ nhớ, đảm bảo rằng dữ liệu được đọc ghi một cách chính xác và hiệu quả.

#### 2.2.1.5 Các thanh ghi tạm thời

CPU thường sử dụng một số thanh ghi tạm thời để chứa toán hạng đầu vào và kết quả đầu ra, như các thanh ghi tạm thời X, Y và Z. Ngoài ra, các thanh ghi tạm thời còn tham gia trong việc hỗ trợ xử lý song song (thực hiện nhiều lệnh cùng một thời điểm) và hỗ trợ thực hiện lệnh theo cơ chế thực hiện tiên tiến kiểu không theo trật tự (OOO – Out Of Order execution). Sử dụng thanh ghi tạm thời có thể giúp CPU tăng tốc độ xử lý bằng cách giảm các truy cập vào bộ nhớ chính như RAM, ROM. Thanh ghi tạm thời cũng hỗ trợ các tính toán phức tạp bao gồm nhiều bước trung gian. Do

tính đa dụng, các thanh ghi tạm thời có thể được sử dụng cho nhiều mục đích khác nhau tùy thuộc vào yêu cầu của từng lệnh hoặc từng chương trình. Số lượng các thanh ghi tạm thời có thể khác nhau tùy thuộc vào kiến trúc của từng CPU cụ thể. Một số kiến trúc CPU cho phép sử dụng các thanh ghi chuyên dụng như thanh ghi tạm thời trong một số trường hợp nhất định. Các thanh ghi tạm thời đóng vai trò quan trọng trong việc tối ưu hóa hiệu suất của CPU bằng cách cung cấp không gian lưu trữ tốc độ cao và linh hoạt cho dữ liệu tạm thời trong quá trình tính toán.

#### 2.2.1.6 Con trỏ ngăn xếp SP



Hình 2. 2. Con trỏ ngăn xếp SP

Hình 2. 1. minh họa cấu trúc ngăn xếp (Stack) và con trỏ ngăn xếp SP (Stack Pointer). Ngăn xếp là một vùng bộ nhớ đặc biệt hoạt động theo nguyên lý vào sau ra trước (LIFO - Last In, First Out). Con trỏ ngăn xếp SP là một thanh ghi quan trọng, luôn chứa địa chỉ của đỉnh ngăn xếp hiện tại. Có hai thao tác chính liên quan đến ngăn xếp:

- Push - đẩy dữ liệu vào ngăn xếp:

$SP \leftarrow SP + 1$  ; tăng địa chỉ đỉnh ngăn xếp  
 $\{SP\} \leftarrow Dữ\ liêu$  ; nạp dữ liệu vào ngăn xếp

- Pop - lấy dữ liệu ra khỏi ngăn xếp

$Thanh\ ghi \leftarrow \{SP\}$  ; chuyển dữ liệu từ đỉnh ngăn xếp vào thanh ghi  
 $SP \leftarrow SP - 1$  ; giảm địa chỉ đỉnh ngăn xếp

#### 2.2.1.7 Các thanh ghi đa năng

Các thanh ghi đa năng (General Purpose Registers) là các thanh ghi có thể được sử dụng cho nhiều mục đích: để chứa toán hạng đầu vào hoặc chứa kết quả đầu ra. Chẳng hạn, CPU Intel 8086 có 4 thanh ghi tổng quát: AX - Thanh tích luỹ, BX - thanh ghi cơ sở, CX - thanh đếm và DX - thanh ghi dữ liệu.

#### 2.2.1.8 Thanh ghi trạng thái FR

Thanh ghi trạng thái (SR - Status Register) hoặc thanh ghi cờ (FR – Flag Register) là một thanh ghi đặc biệt của CPU: mỗi bit của thanh ghi cờ lưu trạng thái của kết quả của phép tính ALU thực hiện. Có hai loại bit cờ: cờ trạng thái (CF, OF,

AF, ZF, PF, SF) và cờ điều khiển (IF, TF, DF). Các bit cờ thường được sử dụng như là các điều kiện trong các lệnh rẽ nhánh để tạo logic chương trình. Kích thước của thanh ghi FR phụ thuộc thiết kế CPU.

Flag	ZF	SF	CF	AF	IF	OF	PF	1
Bit No	7	6	5	4	3	2	1	0

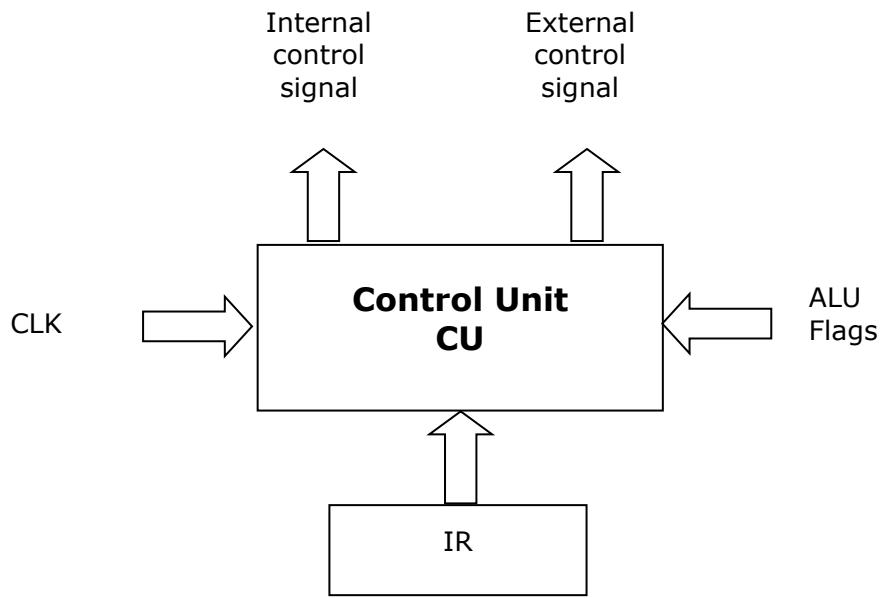
Hình 2. 3. Các bit của thanh ghi cờ FR 8 bit

Hình 2. 3. biểu diễn các bit của thanh ghi cờ FR. Ý nghĩa cụ thể của các bit như sau:

- ZF: Cờ Zero, ZF=1 nếu kết quả=0 và ZF=0 nếu kết quả khác 0.
- SF: Cờ dấu, SF=1 nếu kết quả âm và SF=0 nếu kết quả dương.
- CF: Cờ nhó, CF=1 nếu có nhó/mượn, CF=0 trong trường hợp khác.
- AF: Cờ nhó phụ, AF=1 nếu có nhó/mượn ở nửa thấp của toán hạng.
- OF: Cờ tràn, OF=1 nếu xảy ra tràn, OF=0 trong trường hợp khác.
- PF: Cờ chẵn lẻ, PF=1 nếu tổng số bit 1 trong kết quả là lẻ và PF=0 nếu tổng số bit 1 trong kết quả là chẵn.
- IF: Cờ ngắt, IF=1: cho phép ngắt, IF=0: cấm ngắt.

### 2.3 Khối điều khiển

Khối điều khiển (Control Unit – CU) là một trong các khối quan trọng nhất của CPU. CU đảm nhiệm việc điều khiển toàn bộ các hoạt động của CPU theo xung nhịp đồng hồ. CU sử dụng nhịp đồng hồ để đồng bộ các đơn vị chức năng trong CPU và giữa CPU với các bộ phận bên ngoài. Hình 2. 4. minh họa phương thức làm việc của khối điều khiển CU. Khối điều khiển CU nhận ba tín hiệu đầu vào: (1) Lệnh từ thanh ghi lệnh IR, (2) Giá trị các cờ trạng thái của ALU và (3) Xung nhịp đồng hồ CLK và CU sản sinh hai nhóm tín hiệu đầu ra: (1) Nhóm tín hiệu điều khiển các bộ phận bên trong CPU (Internal control signal) và (2) Nhóm tín hiệu điều khiển các bộ phận bên ngoài CPU (External control signal).



Hình 2. 4. Khối điều khiển CU và các tín hiệu.

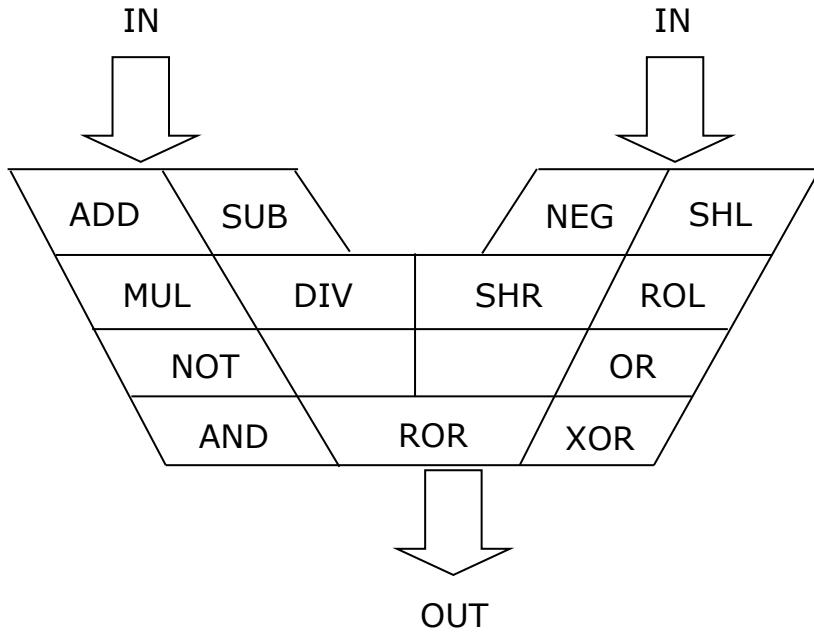
Như vậy khói điều khiển có 2 chức năng chính: giải mã lệnh (decoding) và tạo ra các xung điều khiển để gửi đến các bộ phận bên trong và bên ngoài CPU để kích hoạt các bộ phận khác phục vụ cho thực hiện lệnh. CU là thành phần quan trọng nhất trong CPU, quyết định hiệu suất và khả năng xử lý của CPU. Một CU được thiết kế tốt sẽ giúp cho CPU hoạt động hiệu quả hơn, thực thi các lệnh nhanh hơn và xử lý được nhiều tác vụ cùng lúc. Một số công nghệ liên quan đến CU có thể được liệt kê ra như kỹ thuật đường ống lệnh (Pipelining), kỹ thuật tính toán song song, kỹ thuật thực thi lệnh bất thứ tự.

#### 2.4 Khối số học và logic

Khối số học và logic (Arithmetic and Logic Unit – ALU) đảm nhiệm chức năng tính toán trong CPU. ALU bao gồm một loạt các đơn vị chức năng con để thực hiện các phép toán số học trên số nguyên và logic:

- Bộ cộng (ADD), bộ trừ (SUB), bộ nhân (MUL), bộ chia (DIV), ....
- Các bộ dịch (SHIFT) và quay (ROTATE)
- Bộ phủ định (NOT), bộ và (AND), bộ hoặc (OR) và bộ hoặc loại trừ (XOR)

Hình 2. 5. minh họa các khói con của ALU cũng như các cổng vào và cổng ra của ALU. Hai cổng vào IN nhận các toán hạng đầu vào từ các thanh ghi và một cổng OUT kết nối với bus trong để chuyển kết quả tính toán đến thanh ghi.



Hình 2. 5. Bộ tính toán ALU

ALU hoạt động theo các bước như sau:

- ALU nhận dữ liệu đầu vào từ các thanh ghi trong CPU.
- Dựa trên lệnh từ khối điều khiển (CU), ALU thực hiện phép toán tương ứng trên dữ liệu đầu vào.
- Kết quả của phép toán được lưu trữ lại vào một thanh ghi hoặc trực tiếp vào bộ nhớ.
- ALU cũng có thể cập nhật các bit cờ trong thanh ghi cờ để biểu thị kết quả của phép toán (Ví dụ: cờ Zero cho biết kết quả bằng 0, cờ carry cho biết có sự nhô trong phép cộng,...).

## 2.5 Bus trong cpu

Bus trong CPU (Internal bus) là kênh giao tiếp giữa các bộ phận bên trong CPU, cụ thể giữa bộ điều khiển CU với các thanh ghi và bộ tính toán ALU. Bus trong hỗ trợ kênh giao tiếp song công (full duplex) và cung cấp giao diện để kết nối với bus ngoài (bus hệ thống). So với bus ngoài, bus trong thường có băng thông lớn hơn và có tốc độ nhanh hơn. Bus trong của CPU có một số đặc điểm như tốc độ cao, độ rộng lớn, phức tạp. Hệ thống bus trong của CPU ảnh hưởng trực tiếp đến tốc độ xử lý của CPU. Bus trong có băng thông cao và độ trễ thấp giúp tăng tốc độ truyền dữ liệu giữa các thành phần bên trong CPU, từ đó cải thiện hiệu suất tổng thể của CPU. Bus trong cũng ảnh hưởng đến khả năng mở rộng của CPU. Một bus trong được thiết kế tốt sẽ cho phép dễ dàng thêm các thành phần mới vào CPU trong tương lai, giúp nâng cao hiệu suất và khả năng xử lý của CPU. Bus trong là một thành phần quan trọng của CPU, đảm bảo việc kết nối và truyền tải dữ liệu giữa các thành phần bên trong CPU một cách nhanh chóng và hiệu quả. Hệ thống bus trong kết hợp với các hệ thống bus ngoài hình thành mạng lưới tổng thể cho toàn bộ hệ thống máy tính.

## 2.6 Kết luận chương

Chương 2 đã cung cấp cái nhìn tổng quan và chi tiết về các thành phần và chức năng của khối xử lý trung tâm (cpu), được coi là bộ não của hệ thống máy tính. Thông qua các nội dung trình bày, chúng ta đã hiểu rõ hơn về cấu trúc tổng quát của cpu, bao gồm các khối chức năng quan trọng như bộ điều khiển (cu), bộ tính toán số học và logic (alu), cùng hệ thống bus và các thanh ghi. Chu trình xử lý lệnh đã được phân tích kỹ lưỡng, giúp làm rõ cách cpu thực hiện các lệnh từ việc đọc, giải mã cho đến thực thi. Chức năng của từng loại thanh ghi trong cpu, bao gồm thanh tích luỹ, bộ đếm chương trình, thanh ghi lệnh và các thanh ghi trạng thái, đã được giải thích cụ thể, làm nổi bật vai trò của chúng trong việc đảm bảo hoạt động chính xác và hiệu quả của cpu. Khối điều khiển (cu) được xác định là trung tâm điều hành toàn bộ các hoạt động của cpu, quyết định hiệu năng và khả năng xử lý của hệ thống. Trong khi đó, khối tính toán số học và logic (alu) đóng vai trò thực hiện các phép toán phức tạp, hỗ trợ tối ưu hóa hiệu suất xử lý. Cuối cùng, hệ thống bus trong cpu, với khả năng kết nối nhanh và hiệu quả giữa các thành phần nội bộ, cũng được làm rõ, cho thấy tầm quan trọng của nó trong việc đảm bảo luồng dữ liệu thông suốt và tăng cường hiệu suất của cpu. Những kiến thức trong chương này là nền tảng quan trọng để tiếp tục tìm hiểu các khía cạnh nâng cao hơn trong thiết kế và tối ưu hóa cpu, cũng như các ứng dụng trong công nghệ máy tính hiện đại.

## 2.7 Câu hỏi ôn tập

1. Nêu sơ đồ khái niệm tổng quát và các thành phần chính của CPU?
2. (\*) Nêu chu trình xử lý lệnh của CPU?
3. Nêu vai trò và chức năng của các thanh ghi của CPU?
4. (\*) Nêu sơ đồ và chức năng của CU và ALU?
5. Phân biệt chức năng của các thanh ghi MAR và MBR?
6. Vai trò của thanh ghi tích lũy A trong các phép tính số học?
7. Mô tả nguyên lý hoạt động của ngăn xếp và vai trò của con trỏ ngăn xếp SP?
8. Nêu chức năng và ý nghĩa của các bit cờ trong thanh ghi trạng thái FR?
9. (\*) Giải thích sự khác biệt giữa bus trong và bus ngoài của CPU?
10. (\*) Tại sao khái niệm CU được coi là trung tâm của CPU?

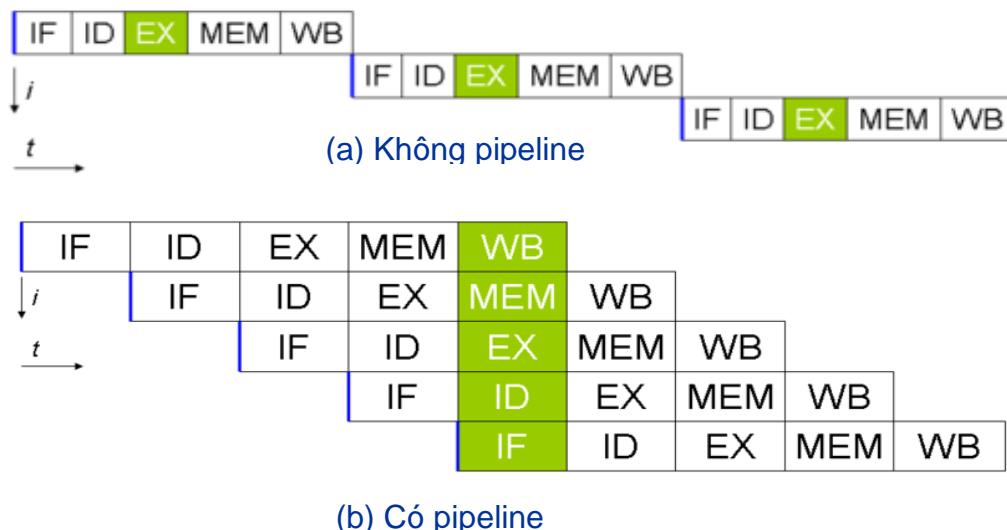
## CHƯƠNG 3 XỬ LÝ XEN KẼ DÒNG MÃ LỆNH VÀ BỘ NHỚ CACHE

Hiệu suất xử lý của hệ thống máy tính phụ thuộc lớn vào khả năng tối ưu hóa luồng thực thi lệnh và truy xuất dữ liệu. Cơ chế xử lý xen kẽ dòng mã lệnh (pipeline) giúp tăng tốc độ xử lý bằng cách thực hiện song song nhiều giai đoạn lệnh, nhưng cũng gặp các vấn đề như xung đột tài nguyên và tranh chấp dữ liệu. Trong khi đó, bộ nhớ cache đóng vai trò trung gian giữa CPU và bộ nhớ chính, giúp giảm thời gian truy cập dữ liệu. Chương này sẽ trình bày chi tiết về các kỹ thuật tối ưu pipeline, cache và các công nghệ lưu trữ hiện đại như RAID, NAS và SAN, nhằm nâng cao hiệu suất hệ thống.

### 3.1 Cơ chế xử lý xen kẽ dòng mã lệnh (pipeline)

#### 3.1.1 Giới thiệu cơ chế xử lý xen kẽ dòng mã lệnh

Cơ chế xử lý xen kẽ dòng mã lệnh (pipeline) hay còn gọi là cơ chế thực hiện xen kẽ các lệnh của chương trình là một phương pháp thực hiện lệnh tiên tiến, cho phép đồng thời thực hiện nhiều lệnh, giảm thời gian trung bình thực hiện mỗi lệnh và như vậy tăng được hiệu năng xử lý lệnh của CPU. Việc thực hiện lệnh được chia thành một số giai đoạn và mỗi giai đoạn được thực thi bởi một đơn vị chức năng khác nhau của CPU. Nhờ vậy CPU có thể tận dụng tối đa năng lực xử lý của các đơn vị chức năng của mình, giảm thời gian chờ cho từng đơn vị chức năng.



Hình 3. 1. Thực hiện lệnh (a) không pipeline và (b) có pipeline

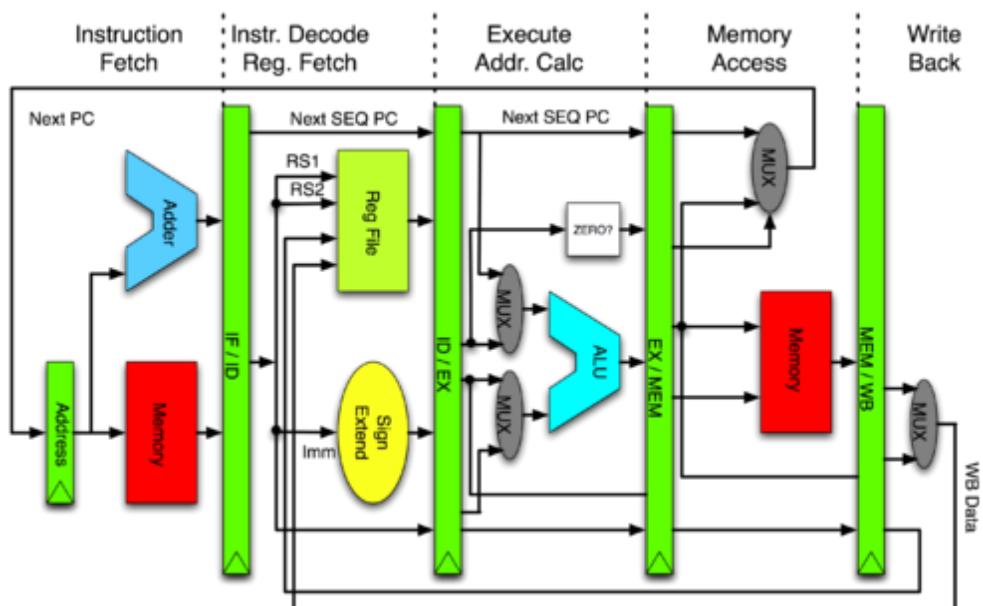
**Error! Reference source not found.** minh họa cơ chế thực hiện lệnh (a) không pipeline và (b) có pipeline của một hệ thống load-store đơn giản. Trong đó, việc thực hiện lệnh được chia thành 5 giai đoạn:

- Instruction Fetch - IF: Đọc lệnh từ bộ nhớ (hoặc cache).
- Instruction Decode - ID: giải mã lệnh và đọc các toán hạng.
- Execute - EX: thực hiện lệnh; nếu là lệnh truy nhập bộ nhớ: tính toán địa chỉ bộ nhớ.

- Memory Access - MEM: Đọc/ghi bộ nhớ; no-op nếu không truy nhập bộ nhớ; no-op là giai đoạn chờ, tiêu tốn thời gian CPU, nhưng không thực hiện thao tác có nghĩa.
- Write Back - WB: Ghi kết quả vào các thanh ghi.

Có thể thấy, với cơ chế thực hiện không pipeline, tại mỗi thời điểm chỉ có một lệnh được thực hiện và chỉ có một đơn vị chức năng của CPU làm việc, các đơn vị chức năng khác trong trạng thái chờ. Ngược lại, với cơ chế thực hiện có pipeline, có nhiều lệnh đồng thời được thực hiện gối nhau trong CPU và hầu hết các đơn vị chức năng của CPU liên tục tham gia vào quá trình xử lý lệnh. Số lượng lệnh được xử lý đồng thời đúng bằng số giai đoạn thực hiện lệnh. Với 5 giai đoạn thực hiện lệnh, để xử lý 5 lệnh, CPU cần 9 nhịp đồng hồ với cơ chế thực hiện có pipeline, trong khi CPU cần đến 25 nhịp đồng hồ để thực hiện 5 lệnh với cơ chế thực hiện không pipeline. **Error! Reference source not found.** minh họa việc các đơn vị chức năng của CPU phối hợp thực hiện lệnh trong cơ chế pipeline.

Việc lựa chọn số giai đoạn thực hiện lệnh sao cho phù hợp là một vấn đề quan trọng của cơ chế xử lý xen kẽ dòng mã lệnh. Về mặt lý thuyết, thời gian thực hiện lệnh trung bình sẽ giảm khi tăng số giai đoạn thực hiện lệnh. Cho đến hiện nay, không có câu trả lời chính xác về số giai đoạn thực hiện lệnh tối ưu mà nó phụ thuộc nhiều vào thiết kế của CPU. Với các CPU cũ (họ Intel 80x86 và tương đương) số giai đoạn là 3 đến 5. Với các CPU Intel Pentium III và Pentium M, Core Duo, Core 2 Duo số giai đoạn là khoảng 10 đến 15. Riêng họ Intel Pentium IV có số giai đoạn vào khoảng 20 và cá biệt phiên bản Intel Pentium IV Prescott chia việc thực hiện lệnh thành 31 giai đoạn.



Hình 3. 2. Thực hiện lệnh theo cơ chế pipeline với các đơn vị chức năng của CPU

### 3.1.2 Các vấn đề của cơ chế xử lý xen kẽ dòng mã lệnh và hướng giải quyết

Như đã trình bày, cơ chế xử lý xen kẽ dòng mã lệnh giúp giảm thời gian trung bình thực hiện từng lệnh và tăng đáng kể hiệu suất xử lý lệnh của CPU. Tuy nhiên, cơ

chế xử lý xen kẽ dòng mã lệnh cũng gặp phải một số vấn đề làm giảm hiệu suất thực hiện lệnh. Tựu chung, có ba vấn đề thường gặp với cơ chế xử lý xen kẽ dòng mã lệnh: (1) Vấn đề xung đột tài nguyên (resource conflicts), (2) Vấn đề tranh chấp dữ liệu (Data hazards) và (3) Vấn đề nảy sinh do các lệnh rẽ nhánh (Branch instructions). Trong phạm vi của bài giảng này, hướng giải quyết các vấn đề của cơ chế xử lý xen kẽ dòng mã lệnh chỉ dừng ở mức giới thiệu phương pháp.

### 3.1.2.1 Vấn đề xung đột tài nguyên

Vấn đề xung đột tài nguyên xảy ra khi hệ thống không cung cấp đủ tài nguyên phần cứng phục vụ CPU thực hiện đồng thời nhiều lệnh trong cơ chế xử lý xen kẽ dòng mã lệnh. Hai xung đột tài nguyên thường gặp nhất là xung đột truy cập bộ nhớ và xung đột truy cập các thanh ghi. Giả sử bộ nhớ chỉ hỗ trợ một truy cập tại mỗi thời điểm và nếu tại cùng một thời điểm, có hai yêu cầu truy cập bộ nhớ đồng thời từ 2 lệnh được thực hiện trong ống lệnh (đọc lệnh – tại giai đoạn IF và đọc dữ liệu – tại giai đoạn ID) sẽ nảy sinh xung đột. Điều tương tự cũng có thể xảy ra với các thanh ghi khi có 2 hay nhiều lệnh đang thực hiện đồng yêu cầu đọc/ghi cùng một thanh ghi.

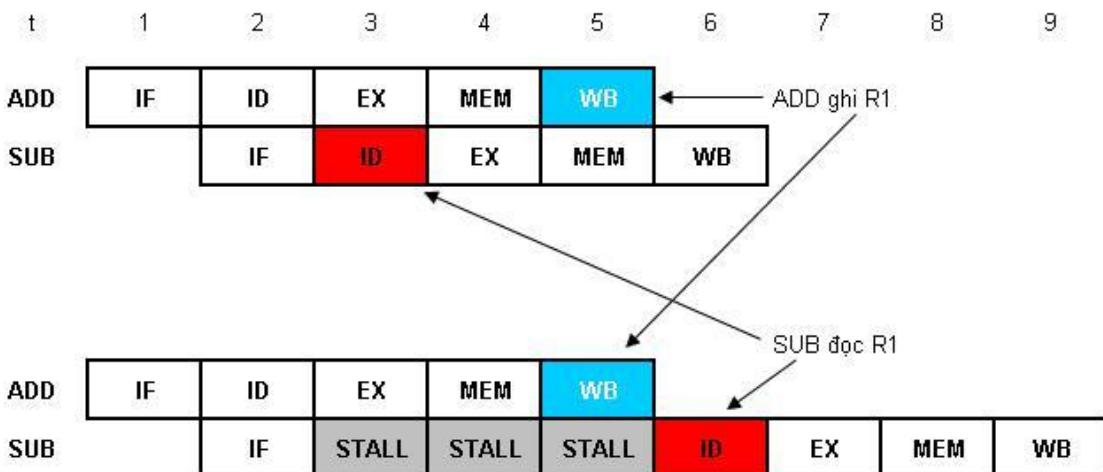
Giải pháp tối ưu cho vấn đề xung đột tài nguyên là nâng cao năng lực phục vụ của các tài nguyên phần cứng. Với xung đột truy cập bộ nhớ có thể sử dụng hệ thống nhớ hỗ trợ nhiều lệnh đọc ghi đồng thời, hoặc sử dụng các bộ nhớ tiên tiến như bộ nhớ cache. Với xung đột truy cập các thanh ghi, giải pháp là tăng số lượng thanh ghi vật lý và có cơ chế cấp phát thanh ghi linh hoạt khi thực hiện các lệnh.

### 3.1.2.2 Vấn đề tranh chấp dữ liệu

Tranh chấp dữ liệu cũng là một trong các vấn đề lớn của cơ chế xử lý xen kẽ dòng mã lệnh và tranh chấp dữ liệu kiểu *đọc sau khi ghi* (RAW – Read After Write) là dạng xung đột dữ liệu hay gặp nhất. Để hiểu rõ tranh chấp dữ liệu kiểu RAW, ta xem xét hai lệnh sau:

$$\text{ADD } R_1, R_2, R_3; \quad R_1 \leftarrow R_2 + R_3 \quad (3.1)$$

$$\text{SUB } R_4, R_1, R_2; \quad R_4 \leftarrow R_1 - R_2 \quad (3.2)$$



Hình 3. 3. Tranh chấp dữ liệu kiểu RAW

**Error! Reference source not found.** minh họa tranh chấp dữ liệu kiểu RAW giữa hai lệnh ADD và SUB được thực hiện kề nhau trong cơ chế xử lý xen kẽ dòng mã lệnh. Có thể thấy lệnh SUB sử dụng kết quả của lệnh ADD (thanh ghi R<sub>1</sub> là kết quả của ADD và là đầu vào cho SUB) và như vậy hai lệnh có sự phụ thuộc dữ liệu. Tuy nhiên, lệnh SUB đọc thanh ghi R<sub>1</sub> tại giai đoạn giải mã (ID), trước khi lệnh ADD ghi kết quả vào thanh ghi R<sub>1</sub> ở giai đoạn lưu kết quả (WB). Như vậy, giá trị SUB đọc được từ thanh ghi R<sub>1</sub> là giá trị cũ, không phải là kết quả tạo ra bởi ADD. Để SUB đọc được giá trị mới nhất của R<sub>1</sub>, giai đoạn ID của SUB phải lùi 3 nhịp, đến vị trí giai đoạn WB của ADD kết thúc.

Có một số giải pháp cho vấn đề tranh chấp dữ liệu kiểu RAW. Cụ thể:

1. Nhận dạng tranh chấp RAW khi nó diễn ra.
2. Khi tranh chấp RAW xảy ra, tạm dừng (stall) ống lệnh cho đến khi lệnh phía trước hoàn tất giai đoạn WB.
3. Có thể sử dụng trình biên dịch (compiler) để nhận dạng tranh chấp RAW và thực hiện:
  - Chèn thêm các lệnh NO-OP vào giữa các lệnh có thể gây ra tranh chấp RAW; NO-OP là lệnh rỗng, không thực hiện tác vụ hữu ích mà chỉ tiêu tốn thời gian CPU.
  - Thay đổi trật tự các lệnh trong chương trình và chèn các lệnh độc lập vào giữa các lệnh có thể gây ra tranh chấp RAW.

Mục đích của cả hai phương pháp kể trên là lùi việc thực hiện lệnh gây tranh chấp dữ liệu cho đến khi lệnh trước nó hoàn tất việc lưu kết quả.

4. Sử dụng phần cứng để nhận dạng tranh chấp RAW và dự đoán trước giá trị dữ liệu phụ thuộc.

**Error! Reference source not found.** minh họa giải pháp khắc phục tranh chấp RAW bằng cách chèn thêm các lệnh NO-OP. **Error! Reference source not found.** minh họa giải pháp khắc phục tranh chấp RAW bằng cách chèn thêm các lệnh độc lập với hai lệnh có tranh chấp. Các lệnh độc lập có thể có được bằng cách thay đổi trật tự thực hiện các lệnh của chương trình mà không thay đổi kết quả thực hiện nó. Cũng có thể sử dụng giải pháp kết hợp chèn NO-OP và lệnh độc lập.

t	1	2	3	4	5	6	7	8	9
ADD	IF	ID	EX	MEM	WB				
NO-OP		NO-OP	NO-OP	NO-OP	NO-OP	NO-OP			
NO-OP			NO-OP	NO-OP	NO-OP	NO-OP	NO-OP		
NO-OP				NO-OP	NO-OP	NO-OP	NO-OP	NO-OP	
SUB					IF	ID	EX	MEM	WB

Hình 3. 4. Khắc phục tranh chấp RAW bằng chèn thêm NO-OP

	t	1	2	3	4	5	6	7	8	9
ADD R1, R1, R3		IF	ID	EX	MEM	WB				
LOAD R4, [1000]			IF	ID	EX	MEM	WB			
ADD R5, 500				IF	ID	EX	MEM	WB		
STORE [2000], R6					IF	ID	EX	MEM	WB	
SUB R4, R1, R2						IF	ID	EX	MEM	WB

Hình 3. 5. Khắc phục tranh chấp RAW bằng chèn các lệnh độc lập

### 3.1.2.3 Vấn đề nảy sinh do các lệnh rẽ nhánh

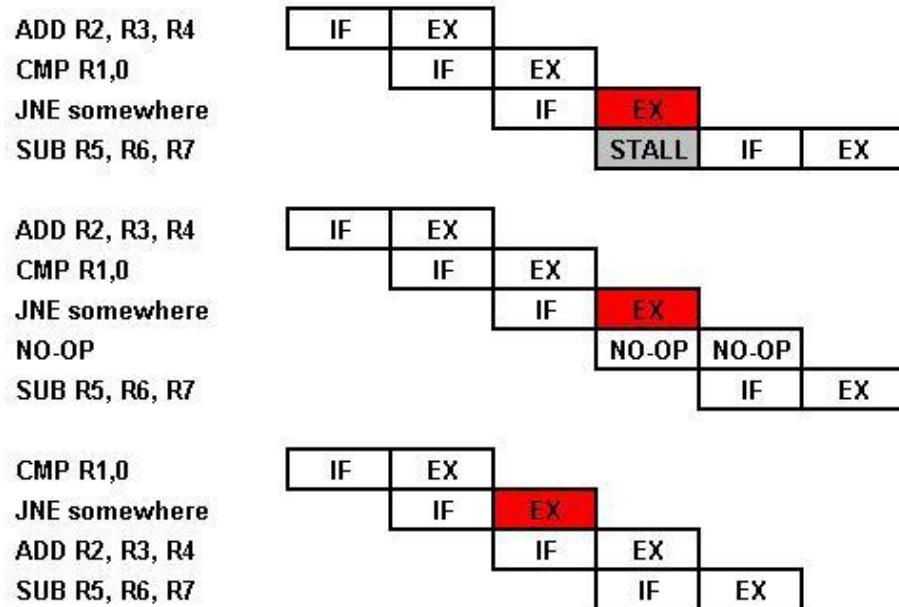
Theo thống kê, tỷ lệ các lệnh rẽ nhánh trong chương trình khoảng 10-30%. Do lệnh rẽ nhánh thay đổi nội dung của bộ đếm chương trình, chúng có thể phá vỡ tiến trình thực hiện tuần tự các lệnh trong ống lệnh vì lệnh được thực hiện sau lệnh rẽ nhánh có thể không phải là lệnh liền sau nó mà là một lệnh ở vị trí khác. Như vậy, do kiểu thực hiện gói đầu, các lệnh liền sau lệnh rẽ nhánh đã được nạp và thực hiện dở dang trong ống lệnh sẽ bị đẩy ra làm cho ống lệnh bị trống rỗng và hệ thống phải bắt đầu nạp mới các lệnh từ địa chỉ đích rẽ nhánh. **Error! Reference source not found.** minh họa vấn đề nảy sinh trong ống lệnh do lệnh rẽ nhánh. Các lệnh sau lệnh rẽ nhánh LOAD và ADD bị đẩy ra và hệ thống nạp mới các lệnh từ địa chỉ đích rẽ nhánh 1000.

	t	1	2	3	4	5	6	7	8	9
JUMP [1000]		IF	ID	EX	MEM	WB				
LOAD R4, [00FF]			IF	ID	EX	MEM	WB			
ADD R5, 500				IF	ID	EX	MEM	WB		
JUMP [1000]		IF	ID	EX	MEM	WB				
1000: ADD R6, 2000					IF	ID	EX	MEM	WB	
LOAD R7, [03FF]						IF	ID	EX	MEM	WB

Hình 3. 6. Vấn đề nảy sinh do lệnh rẽ nhánh

Có nhiều giải pháp khắc phục các vấn đề nảy sinh do các lệnh rẽ nhánh, như sử dụng đích rẽ nhánh (branch targets), làm chậm rẽ nhánh (delayed branching) và dự đoán rẽ nhánh (branch prediction). Tài liệu này chỉ giới thiệu phương pháp làm chậm rẽ nhánh. Ý tưởng chính của phương pháp làm chậm rẽ nhánh là lệnh rẽ nhánh sẽ không gây ra sự rẽ nhánh tức thì mà được làm “trễ” một số chu kỳ, phụ thuộc vào

chiều dài của ống lệnh. Phương pháp này cho hiệu quả khá tốt với các ống lệnh ngắn, thường là 2 giai đoạn và với ràng buộc lệnh ngay sau lệnh rẽ nhánh luôn được thực hiện, không phụ thuộc vào kết quả của lệnh rẽ nhánh. Cách thực hiện của phương pháp chậm rẽ nhánh là chèn thêm một lệnh NO-OP hoặc một lệnh độc lập vào ngay sau lệnh rẽ nhánh. **Error! Reference source not found.** minh họa vấn đề này sinh do lệnh rẽ nhánh có điều kiện JNE (nhảy nếu R1 không bằng 0), giải pháp chèn một lệnh NO-OP hoặc một lệnh độc lập vào sau lệnh nhảy để khắc phục.

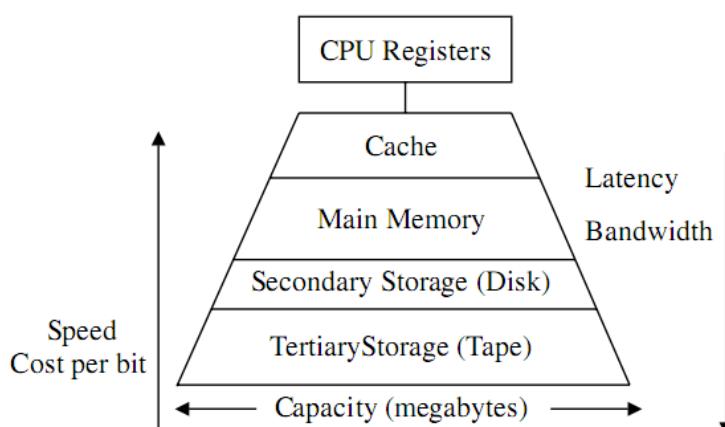


Hình 3.7. Khắc phục vấn đề lệnh rẽ nhánh bằng cách chèn NO-OP hoặc lệnh độc lập

### 3.2 Bộ nhớ cache

#### 3.2.1 Phân cấp hệ thống nhớ

Hầu hết hệ thống nhớ trong các thiết bị tính toán hiện đại được tổ chức theo cấu trúc phân cấp (hierarchical structure). Cấu trúc phân cấp không chỉ được sử dụng trong các hệ thống nhớ mà nó còn sử dụng rộng rãi trong đời sống xã hội, như cấu trúc tổ chức các cơ quan nhà nước, doanh nghiệp và cả các trường học. **Error! Reference source not found.** minh họa cấu trúc phân cấp hệ thống nhớ, gồm các phần chính: các thanh ghi của CPU (CPU Registers), bộ nhớ cache (Cache), bộ nhớ chính (Main Memory) và bộ nhớ ngoài (Secondary / Tertiary Storage).



Hình 3. 8. Cấu trúc phân cấp hệ thống nhớ

	Access type	Capacity	Latency	Bandwidth	Cost/MB
CPU registers	Random	64–1024 bytes	1–10 ns	System clock rate	High
Cache memory	Random	8–512 KB	15–20 ns	10–20 MB/s	\$500
Main memory	Random	16–512 MB	30–50 ns	1–2 MB/s	\$20–50
Disk memory	Direct	1–20 GB	10–30 ms	1–2 MB/s	\$0.25
Tape memory	Sequential	1–20 TB	30–10,000 ms	1–2 MB/s	\$0.025

Hình 3. 9. Dung lượng, thời gian truy cập và giá thành các loại bộ nhớ

**Error! Reference source not found.** minh họa rõ ràng sự cân bằng giữa hiệu năng, dung lượng và chi phí của các loại bộ nhớ, đồng thời thể hiện tầm quan trọng của cấu trúc phân cấp trong hệ thống máy tính. Trong cấu trúc phân cấp hệ thống nhớ, dung lượng các thành phần tăng theo chiều từ các thanh ghi của CPU đến bộ nhớ ngoài. Ngược lại, tốc độ truy nhập hay băng thông và giá thành một đơn vị nhớ tăng theo chiều từ bộ nhớ ngoài đến các thanh ghi của CPU. Như vậy, các thanh ghi của CPU có dung lượng nhỏ nhất nhưng có tốc độ truy cập nhanh nhất và cũng có giá thành cao nhất. Bộ nhớ ngoài có dung lượng lớn nhất, nhưng tốc độ truy cập thấp nhất. Bù lại, bộ nhớ ngoài có giá thành rẻ nên có thể được sử dụng với dung lượng lớn.

Các thanh ghi được tích hợp trong CPU và thường hoạt động theo tần số làm việc của CPU, nên đạt tốc độ truy cập rất cao. Tuy nhiên, do không gian trong CPU rất hạn chế nên tổng dung lượng của các thanh ghi là khá nhỏ, chỉ khoảng vài chục byte đến vài kilobyte. Các thanh ghi thường được sử dụng để lưu toán hạng đầu vào và kết quả đầu ra của các lệnh phục vụ CPU làm việc.

Bộ nhớ cache có dung lượng tương đối nhỏ, khoảng từ vài chục kilobyte đến vài chục megabyte (khoảng 64KB đến 32MB với các máy tính hiện nay). Tốc độ truy cập cache cao, nhưng giá thành còn khá đắt. Cache được coi là bộ nhớ “thông minh” do có khả năng đoán trước được nhu cầu lệnh và dữ liệu của CPU. Cache “đoán” và tải trước các lệnh và dữ liệu CPU cần sử dụng từ bộ nhớ chính, nhờ vậy giúp CPU giảm thời gian truy cập hệ thống nhớ, tăng tốc độ xử lý.

Bộ nhớ chính gồm có bộ nhớ ROM và bộ nhớ RAM, có dung lượng khá lớn (khoảng từ 256MB đến 4GB với các hệ thống 32 bit), nhưng tốc độ truy cập tương đối chậm so với cache. Giá thành bộ nhớ chính tương đối thấp nên có thể sử dụng với dung lượng lớn. Bộ nhớ chính được sử dụng để lưu lệnh và dữ liệu của hệ thống và của người dùng.

Bộ nhớ ngoài hay bộ nhớ thứ cấp, gồm các loại đĩa từ, đĩa quang và băng từ. Bộ nhớ ngoài thường có dung lượng rất lớn, khoảng 20GB đến 1000GB, nhưng tốc độ truy cập rất chậm. Bộ nhớ ngoài có ưu điểm là giá thành rẻ và thường được sử dụng để lưu trữ dữ liệu lâu dài dưới dạng các tệp (files).

Không hoàn toàn giống với vai trò của cấu trúc phân cấp trong các cơ quan và doanh nghiệp là “chia để trị”, cấu trúc phân cấp trong hệ thống nhớ có hai vai trò chính: (1) tăng hiệu năng hệ thống thông qua việc giảm thời gian truy cập các ô nhớ và (2) giảm giá thành sản xuất.

Sở dĩ cấu trúc phân cấp trong hệ thống nhớ có thể giúp tăng hiệu năng hệ thống là do nó giúp dung hoà được CPU có tốc độ cao và phần bộ nhớ chính và bộ nhớ ngoài có tốc độ thấp. CPU sẽ chủ yếu trực tiếp truy cập bộ nhớ cache có tốc độ cao, và cache sẽ có nhiệm vụ chuyển trước các dữ liệu cần thiết về từ bộ nhớ chính. Nhờ vậy, CPU sẽ không phải thường xuyên truy cập trực tiếp bộ nhớ chính và bộ nhớ ngoài để tìm dữ liệu – các thao tác tốn nhiều thời gian do các bộ nhớ này có tốc độ chậm. Như vậy, có thể nói rằng, thời gian trung bình CPU truy cập dữ liệu từ hệ thống nhớ tiệm cận thời gian truy cập bộ nhớ cache.

Cùng với việc có thể giúp cải thiện hiệu năng, cấu trúc phân cấp trong hệ thống nhớ có thể giúp giảm giá thành chế tạo hệ thống. Cơ sở chính là trong hệ thống nhớ phân cấp, các thành phần có tốc độ cao và đắt tiền được sử dụng với dung lượng rất nhỏ, còn các thành phần có tốc độ thấp và rẻ tiền được sử dụng với dung lượng lớn hơn. Nhờ vậy có thể giảm được giá thành chế tạo hệ thống nhớ mà vẫn đảm bảo được tốc độ cao cho cả hệ thống. Có thể nói rằng, nếu ta có hai hệ thống nhớ hoạt động với cùng tốc độ thì hệ thống nhớ phân cấp sẽ có giá thành thấp hơn.

### 3.2.2 Cache là gì?

Cache hay còn gọi là bộ nhớ đệm, bộ nhớ khay là một thành phần của cấu trúc phân cấp của hệ thống bộ nhớ như trình bày trong mục **Error! Reference source not found.**. Cache đóng vai trong trung gian, trung chuyển dữ liệu từ bộ nhớ chính về CPU và ngược lại. **Error! Reference source not found.** minh họa vị trí của bộ nhớ cache trong hệ thống nhớ. Với các hệ thống CPU cũ sử dụng công nghệ tích hợp thấp, bộ nhớ cache thường nằm ngoài CPU; với các CPU mới sử dụng công nghệ tích hợp cao, bộ nhớ cache thường được tích hợp vào trong CPU nhằm nâng cao tốc độ và băng thông trao đổi dữ liệu giữa CPU và cache.



Hình 3. 10. Vị trí của bộ nhớ cache trong hệ thống nhớ

Dung lượng của bộ nhớ cache thường nhỏ so với dung lượng của bộ nhớ chính và bộ nhớ ngoài. Với các hệ thống máy tính cũ, dung lượng cache là khoảng 16KB, 32KB, 64KB, 128KB, 256KB, 512KB; với các hệ thống máy tính gần đây, dung lượng cache lớn hơn, khoảng 1MB, 2MB, 3MB, 4MB, 6MB, 8MB và 16MB. Cache có tốc độ truy cập nhanh hơn nhiều so với bộ nhớ chính, đặc biệt với cache được tích hợp vào CPU. Tuy nhiên, giá thành bộ nhớ cache (tính theo bit) thường đắt hơn nhiều so với bộ nhớ chính. Với các hệ thống CPU mới, cache thường được chia thành hai hay nhiều

mức (levels): mức 1 có dung lượng khoảng 16-32KB có tốc độ truy cập rất cao và mức 2 có dung lượng khoảng 1-16MB có tốc độ truy cập thấp hơn.

### 3.2.3 Vai trò và nguyên lý hoạt động

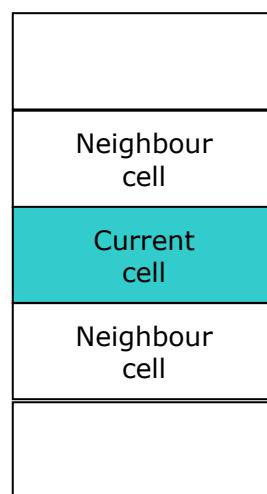
#### 3.2.3.1 Vai trò của cache

Bộ nhớ cache là một thành phần của hệ thống nhớ phân cấp, nên vai trò của cache tương tự như vai trò của cấu trúc phân cấp hệ thống nhớ: tăng hiệu năng hệ thống vào giảm giá thành sản xuất. Sở dĩ cache có thể giúp tăng hiệu năng hệ thống là nhờ cache có khả năng dung hoà được CPU có tốc độ cao và bộ nhớ chính có tốc độ thấp làm cho thời gian trung bình CPU truy nhập dữ liệu từ bộ nhớ chính tiệm cận thời gian truy nhập cache. Ngoài ra, do cache là một loại bộ nhớ “thông minh” có khả năng đoán và chuẩn bị trước các dữ liệu cần thiết cho CPU xử lý nên xác xuất CPU phải trực tiếp truy nhập dữ liệu từ bộ nhớ chính là khá thấp và điều này cũng giúp làm giảm thời gian trung bình CPU truy nhập dữ liệu từ bộ nhớ chính.

Tuy cache có giá thành trên một đơn vị nhớ cao hơn bộ nhớ chính, nhưng do tổng dung lượng cache thường khá nhỏ nên cache không làm tăng giá thành hệ thống nhớ quá mức. Nhờ vậy, cache hoàn toàn phù hợp với cấu trúc phân cấp và có thể giúp làm giảm giá thành sản xuất trong tương quan với tốc độ của cả hệ thống nhớ. Có thể kết luận rằng, nếu hai hệ thống nhớ có cùng giá thành, hệ thống nhớ có cache có tốc độ truy cập nhanh hơn; và nếu hai hệ thống nhớ có cùng tốc độ, hệ thống nhớ có cache sẽ có giá thành rẻ hơn.

#### 3.2.3.2 Nguyên lý hoạt động của cache

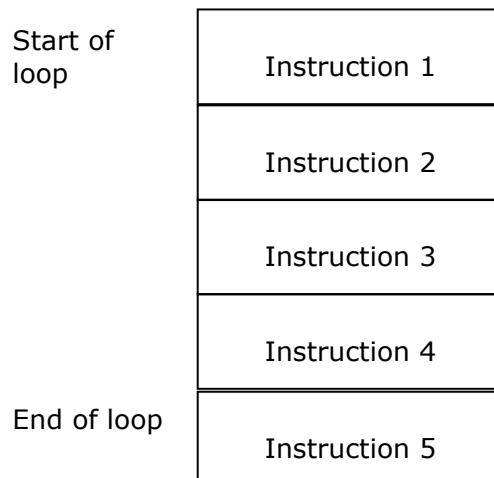
Cache sở dĩ được coi là bộ nhớ “thông minh” là do nó có khả năng đoán trước yêu cầu về dữ liệu và lệnh của CPU. Dữ liệu và lệnh cần thiết được chuyển trước từ bộ nhớ chính về cache và CPU chỉ cần truy nhập cache, giúp giảm thời gian truy nhập hệ thống nhớ. Để có được sự thông minh, cache hoạt động dựa trên hai nguyên lý cơ bản: nguyên lý *lân cận về không gian* (Spatial locality) và nguyên lý *lân cận về thời gian* (Temporal locality).



Hình 3. 11. Lân cận về không gian trong không gian chương trình

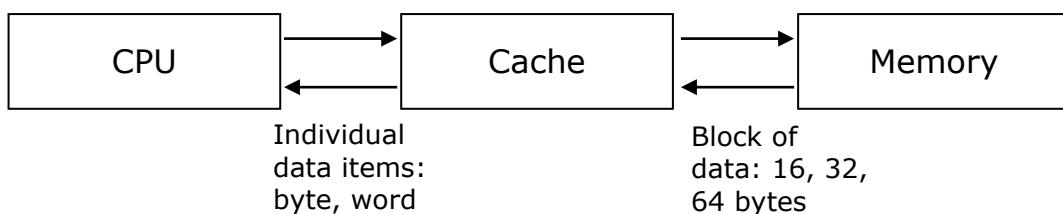
Nguyên lý lân cận về không gian có thể phát biểu như sau: “*Nếu một ô nhớ đang được truy nhập thì xác xuất các ô nhớ liền kề với nó được truy nhập trong tương lai gần là rất cao*”. Lân cận về không gian thường được áp dụng cho nhóm lệnh hoặc dữ liệu có tính tuần tự cao trong không gian chương trình, như minh họa trên **Error! Reference source not found.**. Do các lệnh trong một chương trình thường tuần tự, đặc biệt tính tuần tự thường rất cao trong từng khối lệnh, cache có thể đọc cả khối lệnh từ bộ nhớ chính và khối lệnh đọc được bao phủ cả các ô nhớ lân cận (neighbour cell) của ô nhớ đang được truy nhập (current cell).

Khác với nguyên lý lân cận về không gian, nguyên lý lân cận về thời gian chú trọng hơn đến tính *lặp lại* của việc truy nhập các mẩu thông tin trong một khoảng thời gian tương đối ngắn. Có thể phát biểu nguyên lý này như sau: “*Nếu một ô nhớ đang được truy nhập thì xác xuất nó được truy nhập lại trong tương lai gần là rất cao*”. Lân cận về thời gian được áp dụng cho dữ liệu và nhóm các lệnh trong vòng lặp như minh họa trên **Error! Reference source not found.**. Với các phần tử dữ liệu, chúng được CPU cập nhật thường xuyên trong quá trình thực hiện chương trình nên có tính lân cận cao về thời gian. Với các lệnh trong vòng lặp, chúng thường được CPU thực hiện lặp đi lặp lại nhiều lần nên cũng có tính lân cận cao về thời gian; nếu cache nạp sẵn khối lệnh chứa cả vòng lặp sẽ phủ được tính lân cận về thời gian.



Hình 3. 12. Lân cận về thời gian với việc thực hiện vòng lặp

### 3.2.3.1 Trao đổi dữ liệu giữa CPU – cache – bộ nhớ chính



Hình 3. 13. Trao đổi dữ liệu giữa CPU với cache và bộ nhớ chính

**Error! Reference source not found.** minh họa việc trao đổi dữ liệu giữa CPU với cache và bộ nhớ chính: CPU trao đổi dữ liệu với cache theo các đơn vị cơ sở như byte, từ và từ kép. Còn cache trao đổi dữ liệu với bộ nhớ chính theo các khối, với kích thước 16, 32 hoặc 64 bytes. Sở dĩ CPU trao đổi dữ liệu với cache theo các đơn vị cơ sở mà không theo khối do dữ liệu được lưu trong các thanh ghi của CPU – vốn có dung lượng rất hạn chế. Vì vậy, CPU chỉ trao đổi các phần tử dữ liệu cần thiết theo yêu cầu của các lệnh. Ngược lại, cache trao đổi dữ liệu với bộ nhớ chính theo các khối, mỗi khối gồm nhiều byte kề nhau với mục đích bao phủ các mảng dữ liệu lân cận theo không gian và thời gian. Ngoài ra, trao đổi dữ liệu theo khối (hay mảng) với bộ nhớ chính giúp cache tận dụng tốt hơn băng thông đường truyền và nhờ vậy có thể tăng tốc độ truyền dữ liệu.

### 3.2.3.3 Các hệ số Hit và Miss

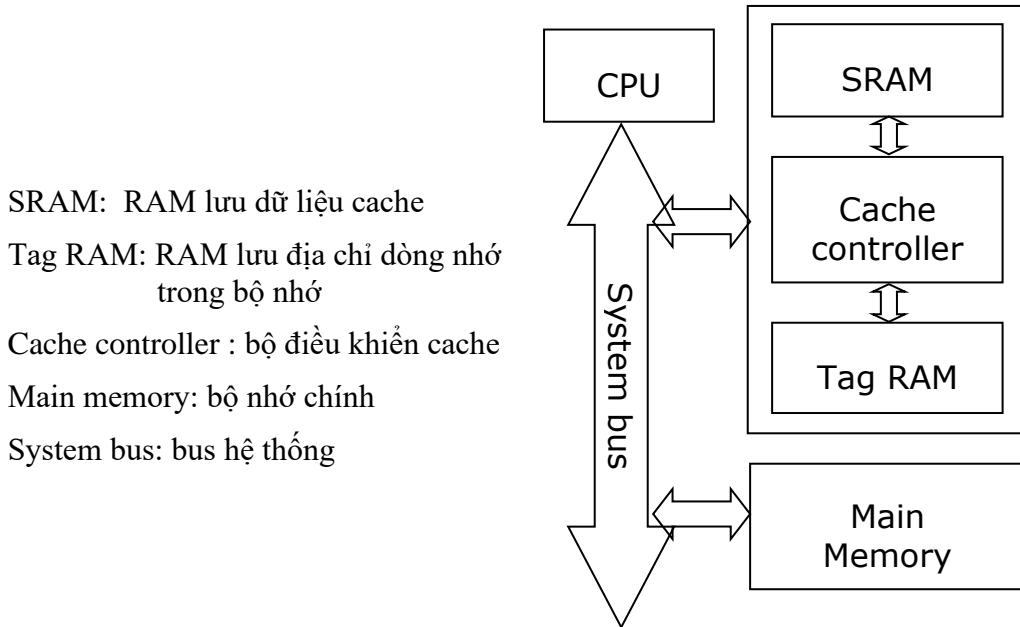
*Hit* (đoán trúng) là một sự kiện mà CPU truy nhập một mục tin và mục tin ấy có ở trong cache. Xác suất để có một hit gọi là hệ số hit, hoặc  $H$ . Để thấy hệ số hit  $H$  thuộc khoảng  $(0, 1)$ . Hệ số hit càng cao thì hiệu quả của cache càng cao. Ngược lại, *Miss* (đoán trượt) là một sự kiện mà CPU truy nhập một mục tin và mục tin ấy không có ở trong cache. Xác suất của một miss gọi là hệ số miss, hoặc  $1-H$ . Cũng có thể thấy hệ số miss  $1-H$  thuộc khoảng  $(0, 1)$ . Hệ số miss càng thấp thì hiệu quả của cache càng cao.

### 3.2.4 Các dạng kiến trúc cache

Kiến trúc cache đề cập đến việc cache được bố trí vào vị trí nào trong quan hệ với CPU và bộ nhớ chính. Có hai loại kiến trúc cache chính: kiến trúc Look Aside (cache được đặt ngang hàng với bộ nhớ chính) và kiến trúc Look Through (cache được đặt giữa CPU và bộ nhớ chính). Mỗi kiến trúc cache kể trên có ưu điểm và nhược điểm riêng.

#### 3.2.4.1 Kiến trúc Look Aside

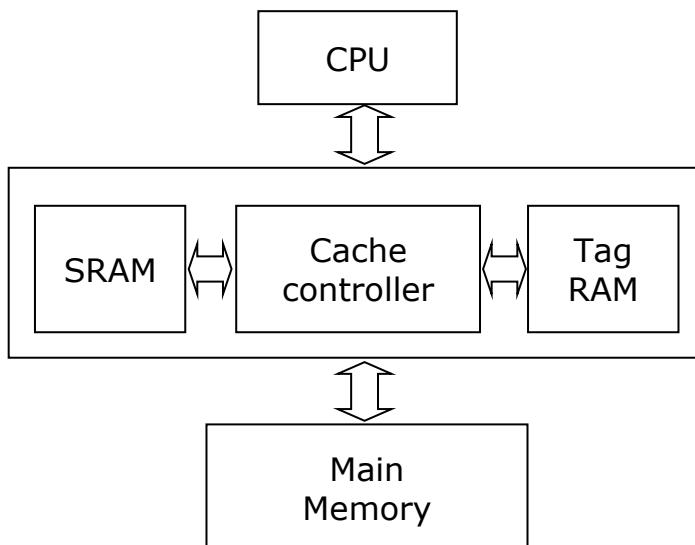
Trong kiến trúc Look Aside, cache và bộ nhớ chính cùng được kết nối vào bus hệ thống. Như vậy, cả cache và bộ nhớ chính đều “thấy” chu kỳ bus của CPU tại cùng một thời điểm. **Error! Reference source not found.** minh họa kiến trúc cache kiểu Look Aside. Kiến trúc Look Aside có thiết kế đơn giản, dễ thực hiện. Tuy nhiên, các sự kiện hit của kiến trúc này thường chậm do cache kết nối với CPU sử dụng bus hệ thống dùng chung – thường có tần số làm việc không cao và băng thông hẹp. Bù lại, các sự kiện miss của kiến trúc Look Aside thường nhanh hơn do khi CPU không tìm thấy mục tin trong cache, nó đồng thời tìm mục tin trong bộ nhớ chính tại cùng một chu kỳ xung nhịp.



Hình 3. 14. Kiến trúc cache kiểu Look Aside

### 3.2.4.2 Kiến trúc Look Through

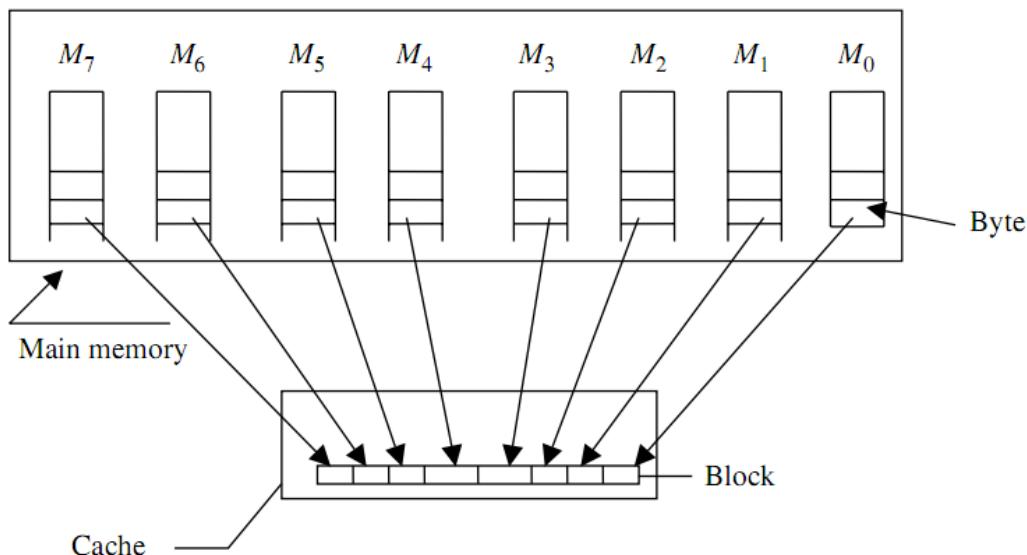
Trong kiến trúc kiểu Look Through, cache được đặt nằm giữa CPU và bộ nhớ chính như minh họa trên **Error! Reference source not found..**. Như vậy cache có thể “thấy” chu kỳ bus của CPU trước, rồi nó mới “truyền” lại cho bộ nhớ chính. Cache kết nối với CPU bằng hệ thống bus riêng tốc độ cao và băng thông lớn, thường là bus mặt sau (BSB – Back Side Bus). Cache kết nối với bộ nhớ chính thông qua bus hệ thống hay bus mặt trước (FSB – Front Side Bus). FSB thường có tần số làm việc và băng thông thấp hơn nhiều so với BSB. Kiến trúc Look Through phức tạp hơn kiến trúc Look Aside. Ưu điểm chính của kiến trúc này là các sự kiện hit của kiến trúc này thường rất nhanh do CPU kết nối với cache bằng kênh riêng có tốc độ cao. Tuy nhiên, các sự kiện miss của kiến trúc Look Through thường chậm hơn do khi CPU không tìm thấy mục tin trong cache, nó cần tìm mục tin đó trong bộ nhớ chính tại một chu kỳ xung nhịp tiếp theo.



Hình 3. 15. Kiến trúc cache kiểu Look Through

### 3.2.5 Tổ chức/ánh xạ cache

#### 3.2.5.1 Giới thiệu tổ chức/ánh xạ cache



Hình 3. 16. Quan hệ giữa các khối của bộ nhớ chính và dòng của cache

**Error! Reference source not found.** minh họa mối quan hệ giữa các khối của bộ nhớ chính (Main Memory) và các dòng của bộ nhớ đệm (Cache). Do kích thước của bộ nhớ cache thường nhỏ hơn nhiều so với bộ nhớ chính, chỉ một phần dữ liệu từ bộ nhớ chính được chuyển vào cache tại mỗi thời điểm. Câu hỏi đặt ra là, phải xây dựng mô hình tổ chức / ánh xạ trao đổi dữ liệu giữa các phần tử nhớ bộ nhớ chính và các phần tử nhớ của cache như thế nào để hệ thống nhớ đạt được tốc độ truy cập tối ưu.

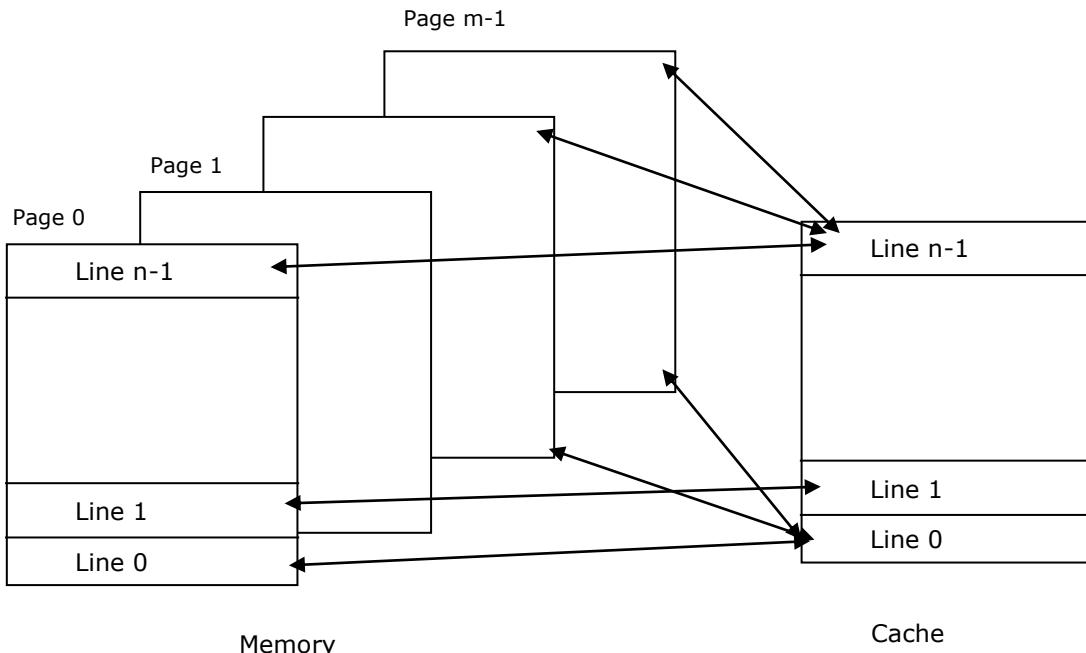
Cho đến hiện nay, có ba phương pháp tổ chức / ánh xạ cache đã được sử dụng, bao gồm: Ánh xạ trực tiếp (Direct mapping), Ánh xạ kết hợp đầy đủ (Fully associative mapping) và Ánh xạ tập kết hợp (Set associative mapping). Phương pháp ánh xạ trực tiếp có ưu điểm là thiết kế đơn giản và nhanh. Tuy nhiên, đây là dạng ánh xạ cố định dễ gây xung đột dẫn đến hiệu quả cache không cao. Phương pháp ánh xạ kết hợp đầy đủ sử dụng ánh xạ mềm, ít gây xung đột và có thể đạt hệ số hit rất cao. Tuy nhiên, phương pháp này có thiết kế phức tạp và có tốc độ chậm. Phương pháp ánh xạ tập kết hợp là sự kết hợp của hai phương pháp ánh xạ trực tiếp và ánh xạ kết hợp đầy đủ, tận dụng được ưu điểm của cả hai phương pháp: nhanh, ít xung đột và không quá phức tạp, cho hiệu quả cao.

#### 3.2.5.2 Ánh xạ trực tiếp

**Error! Reference source not found.** minh họa phương pháp ánh xạ trực tiếp bộ nhớ - cache. Cache được chia thành  $n$  dòng (line) đánh số từ 0 đến  $n-1$ . Bộ nhớ chính được chia thành  $m$  trang (page), đánh số từ 0 đến  $m-1$ . Mỗi trang nhớ lại được chia thành  $n$  dòng (line) đánh số từ 0 đến  $n-1$ . Kích thước mỗi trang của bộ nhớ chính bằng kích thước cache và kích thước một dòng trong trang bộ nhớ cũng bằng kích thước một dòng cache. Ánh xạ từ bộ nhớ chính vào cache được thực hiện theo quy tắc sau:

- $Line_0$  của các trang ( $page_0$  đến  $page_{m-1}$ ) ánh xạ đến  $Line_0$  của cache;

- Line<sub>1</sub> của các trang (page<sub>0</sub> đến page<sub>m-1</sub>) ánh xạ đến Line<sub>1</sub> của cache;
- ....
- Line<sub>n-1</sub> của các trang (page<sub>0</sub> đến page<sub>m-1</sub>) ánh xạ đến Line<sub>n-1</sub> của cache.



Hình 3. 17. Phương pháp ánh xạ trực tiếp bộ nhớ - cache

Có thể thấy với phương pháp ánh xạ trực tiếp, tại mọi thời điểm luôn có cố định  $m$  dòng bộ nhớ cùng cạnh tranh một dòng cache. Khi biết được địa chỉ của dòng trong bộ nhớ, ta biết vị trí của nó trong cache – vì thế phương pháp ánh xạ trực tiếp còn gọi là ánh xạ cứng hay ánh xạ cố định. Để có thể quản lý được các ô nhớ được nạp, cache sử dụng địa chỉ ánh xạ trực tiếp gồm 3 thành phần: *Tag*, *Line* và *Word* như minh họa trên **Error! Reference source not found..** *Tag* (bit) là địa chỉ trang trong bộ nhớ chứa dòng được nạp vào cache, *Line* (bit) là địa chỉ dòng trong cache và *Word* (bit) là địa chỉ của từ trong dòng.

Tag	Line	Word
-----	------	------

Hình 3. 18. Địa chỉ ô nhớ trong ánh xạ trực tiếp

Ví dụ tính các thành phần địa chỉ ô nhớ trong ánh xạ trực tiếp:

- Vào:
  - Dung lượng bộ nhớ = 4GB
  - Dung lượng cache = 1MB
  - Kích thước dòng = 32 byte
- Ra:

- Kích thước dòng Line = 32 byte =  $2^5$ , vậy Word = 5 bit
- Dung lượng Cache = 1MB =  $2^{20}$  → có  $2^{20} / 2^5 = 2^{15}$  dòng, vậy Line = 15 bit
- Địa trang Tag: 4GB =  $2^{32}$ , cần 32-bit địa chỉ tổng cộng để địa chỉ hóa các ô nhớ:  
 $\text{Tag} = 32\text{-bit địa chỉ} - \text{Line} - \text{Word} = 32 - 15 - 5 = 12\text{ bit.}$

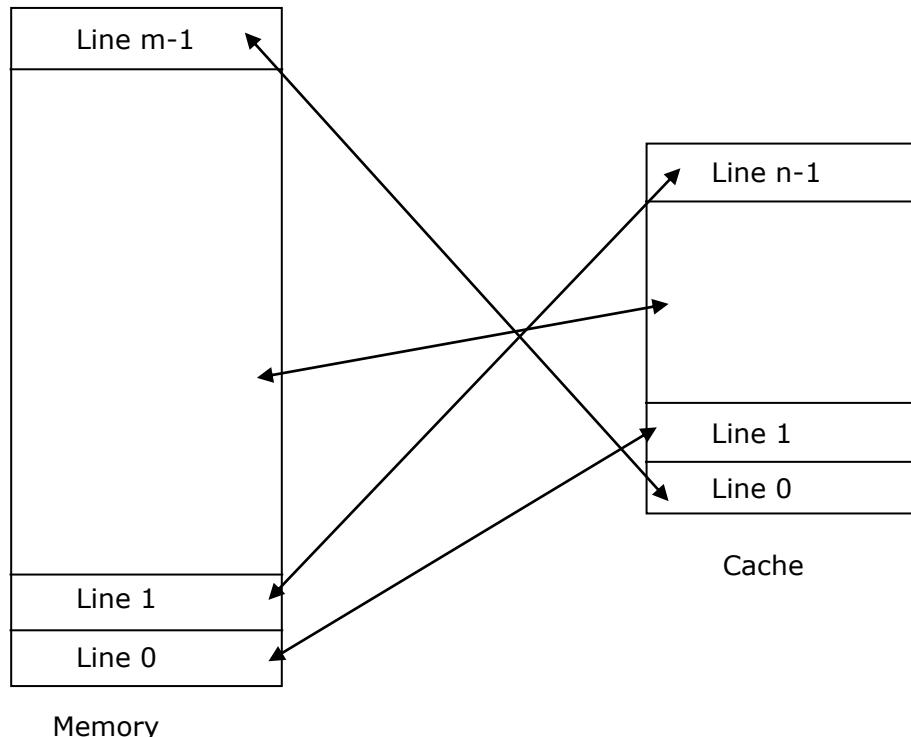
Phương pháp ánh xạ trực tiếp có thiết kế đơn giản và rất nhanh do không tồn nhiều thời gian truy tìm địa chỉ ô nhớ trong cache. Do các ánh xạ là cố định, nên khi biết địa chỉ ô nhớ có thể tìm được vị trí của nó trong cache rất nhanh chóng. Tuy nhiên, cũng do ánh xạ cố định nên phương pháp này dễ gây xung đột vì có thể tạo ra nhiều dòng cache bị nứt cỗ chai trong quá trình hoạt động của cache. Có thể có nhiều dòng cache rảnh rỗi hay ít được sử dụng, nhưng cũng có nhiều dòng cache quá tải do bị nhiều dòng bộ nhớ cùng cạnh tranh. Cũng vì lý do dễ gây xung đột nên hiệu quả tận dụng không gian cache của phương pháp ánh xạ trực tiếp không cao và hệ số hit thấp.

### 3.2.5.3 Ánh xạ kết hợp đầy đủ

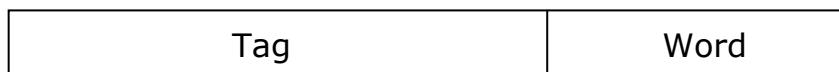
Phương pháp ánh xạ kết hợp đầy đủ hay còn gọi là ánh xạ liên kết đầy đủ được minh họa trên **Error! Reference source not found..** Cache được chia thành n dòng (line) đánh số từ 0 đến n-1. Bộ nhớ chính được chia thành m dòng (line), đánh số từ 0 đến m-1. Kích thước một dòng bộ nhớ bằng kích thước một dòng cache. Do bộ nhớ chính có kích thước lớn hơn nhiều kích thước cache, nên m >> n. Ánh xạ từ bộ nhớ chính vào cache được thực hiện theo quy tắc sau:

- Một dòng trong bộ nhớ chính có thể ánh xạ đến một dòng bất kỳ trong cache, hay
- $\text{Line}_i$  ( $i = 0 \div m-1$ ) của bộ nhớ chính ánh xạ đến  $\text{Line}_j$  ( $j = 0 \div n-1$ ) của cache.

Có thể thấy với phương pháp ánh xạ kết hợp đầy đủ, có n dòng cache để lựa chọn ánh xạ – vì thế phương pháp ánh xạ kết hợp đầy đủ còn gọi là ánh xạ mềm hay ánh xạ không cố định. Ngược lại với phương pháp ánh xạ trực tiếp, khi biết được địa chỉ của dòng trong bộ nhớ, ta chưa biết vị trí của nó trong cache. Để có thể quản lý được các ô nhớ được nạp, cache sử dụng địa chỉ ánh xạ kết hợp đầy đủ chỉ gồm 2 thành phần: *Tag* và *Word* như minh họa trên **Error! Reference source not found..** *Tag* (bit) là địa chỉ dòng trong bộ nhớ được nạp vào cache và *Word* (bit) là địa chỉ của từ trong dòng. Phần địa chỉ *Line* như trong địa chỉ ánh xạ trực tiếp bị bỏ do bộ nhớ chính chỉ còn là một trang duy nhất với m dòng.



Hình 3. 19. Phương pháp ánh xạ kết hợp đầy đủ bộ nhớ - cache



Hình 3. 20. Địa chỉ ô nhớ trong ánh xạ kết hợp đầy đủ

Ví dụ tính các thành phần địa chỉ ô nhớ trong ánh xạ kết hợp đầy đủ:

- Vào:
  - Dung lượng bộ nhớ = 4GB
  - Dung lượng cache = 1MB
  - Kích thước dòng = 32 byte
- Ra:
  - Kích thước dòng Line = 32 byte =  $2^5$ , vậy Word = 5 bit
  - Địa trang Tag:  $4GB = 2^{32}$ , cần 32-bit địa chỉ tổng cộng để địa chỉ hóa các ô nhớ:

$$\text{Tag} = \text{32-bit địa chỉ} - \text{Word} = 32 - 5 = 27 \text{ bit.}$$

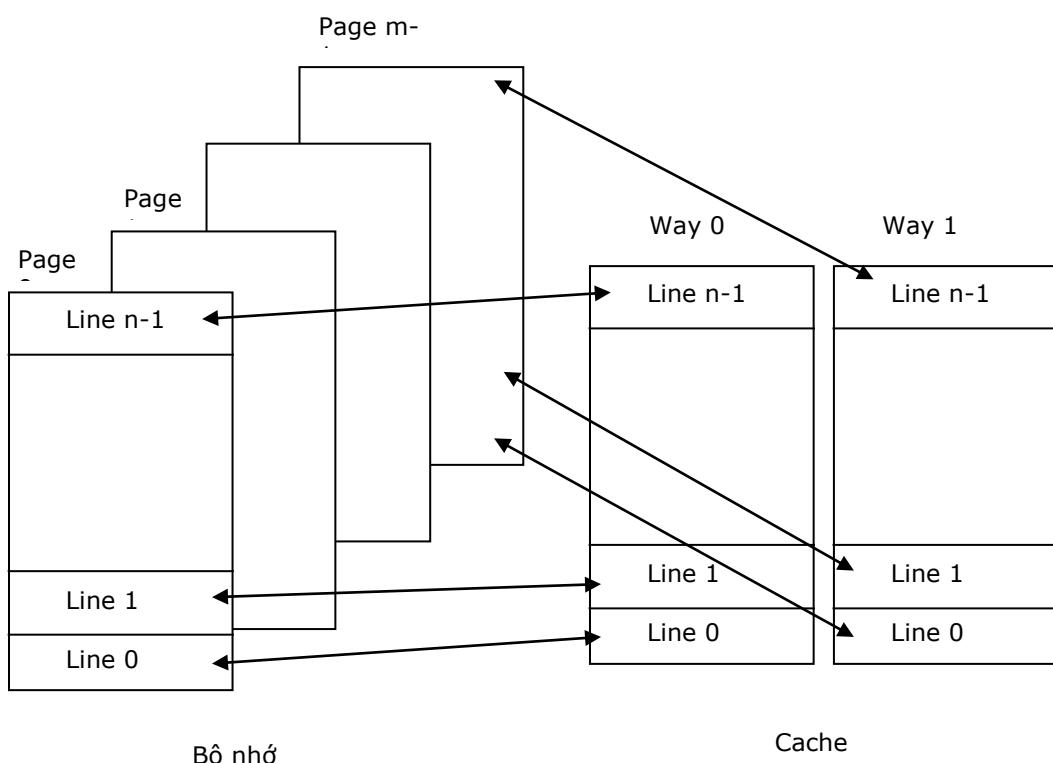
Phương pháp ánh xạ kết hợp đầy đủ sử dụng ánh xạ không cố định nên có ưu điểm là mềm dẻo, giảm được xung đột sử dụng dòng cache. Việc sử dụng các dòng cache có thể được điều phối hướng đến phân bổ hợp lý hơn, giảm hiện tượng tạo các dòng bị nứt cỗ chai với mức độ cạnh tranh lớn. Nhờ vậy, phương pháp này có hiệu suất sử dụng không gian cache cao hơn và có khả năng cho hệ số hit cao. Tuy nhiên, cũng do việc sử dụng ánh xạ không cố định, nên việc truy tìm địa dòng nhớ trong cache tốn nhiều thời gian, gây chậm trễ, đặc biệt với các cache có kích thước lớn. Ngoài ra, phương pháp này cũng có thiết kế phức tạp hơn so với phương pháp ánh xạ

trực tiếp do cần bổ sung thêm các bộ so sánh địa chỉ dòng cache nhằm tăng tốc cho quá trình truy tìm địa chỉ dòng nhớ trong cache. Do vậy, phương pháp ánh xạ kết hợp đầy đủ thường chỉ thích hợp với các cache có dung lượng nhỏ.

#### 3.2.5.4 Ánh xạ tập kết hợp

Phương pháp ánh xạ tập kết hợp hay còn gọi là ánh xạ liên kết nhóm được minh họa trên **Error! Reference source not found..**. Cache được chia thành k đường (way) đ ánh số từ 0 đến k-1. Mỗi đường cache lại được chia thành n dòng (line) đ ánh số từ 0 đến n-1. Bộ nhớ chính được chia thành m trang (page), đ ánh số từ 0 đến m-1. Mỗi trang lại được chia thành n dòng (line) đ ánh số từ 0 đến n-1. Kích thước mỗi trang của bộ nhớ chính bằng kích thước một đường của cache và kích thước một dòng trong trang bộ nhớ cũng bằng kích thước một đường của đường cache. Ánh xạ từ bộ nhớ chính vào cache được thực hiện theo quy tắc sau:

- Ánh xạ trang bộ nhớ đến đường cache (ánh xạ không cố định):
  - Một trang của bộ nhớ có thể ánh xạ đến một đường bất kỳ của cache.
- Ánh xạ dòng của trang đến dòng của đường (ánh xạ cố định):
  - Line<sub>0</sub> của page<sub>i</sub> của bộ nhớ ánh xạ đến Line<sub>0</sub> của way<sub>j</sub> của cache;
  - Line<sub>1</sub> của page<sub>i</sub> của bộ nhớ ánh xạ đến Line<sub>1</sub> của way<sub>j</sub> của cache;
  - ....
  - Line<sub>n-1</sub> của page<sub>i</sub> của bộ nhớ ánh xạ đến Line<sub>n-1</sub> của way<sub>j</sub> của cache.



Hình 3. 21. Phương pháp ánh xạ tập kết hợp bộ nhớ - cache

Có thể thấy phương pháp ánh xạ tập kết hợp đảm bảo được sự kết hợp hài hòa giữa ánh xạ mềm từ trang nhớ đến đường cache và ánh xạ cố định từ dòng của trang nhớ đến dòng của đường cache. Do số đường cache không lớn (thường chỉ khoảng 4, 8, 16, 32 hoặc 64 đường) nên việc tìm kiếm địa chỉ trang nhớ trong các đường cache không ảnh hưởng nhiều đến tốc độ truy cập cache. Hơn nữa, do ánh xạ từ dòng của trang nhớ đến dòng của đường cache là cố định, có thể nhanh chóng xác định được vị trí của dòng nhớ trong đường cache khi biết địa chỉ của nó. Để có thể quản lý được các ô nhớ được nạp, cache sử dụng địa chỉ ánh xạ trực tiếp gồm 3 thành phần: *Tag*, *Set* và *Word* như minh họa trên **Error! Reference source not found..** *Tag* (bit) là địa chỉ trang trong bộ nhớ chia dòng được nạp vào cache, *Set* (bit) là địa chỉ dòng trong đường cache và *Word* (bit) là địa chỉ của từ trong dòng.

Tag	Set	Word
-----	-----	------

Hình 3. 22. Địa chỉ ô nhớ trong ánh xạ tập kết hợp

Ví dụ tính các thành phần địa chỉ ô nhớ trong ánh xạ tập kết hợp:

- Vào:
  - Dung lượng bộ nhớ = 4GB
  - Dung lượng cache = 1MB, 2 đường
  - Kích thước dòng = 32 byte
- Ra:
  - Kích thước dòng Line = 32 byte =  $2^5$ , vậy Word = 5 bit
  - Dung lượng Cache = 1MB =  $2^{20}$  → có  $2^{20} / 2$  đường /  $2^5 = 2^{14}$  dòng / đường, vậy Set = 14 bit
  - Địa trang Tag:  $4GB = 2^{32}$ , cần 32-bit địa chỉ tổng cộng để địa chỉ hóa các ô nhớ:

$$\text{Tag} = \text{32-bit địa chỉ} - \text{Set} - \text{Word} = 32 - 14 - 5 = 13 \text{ bit.}$$

Phương pháp ánh xạ tập kết hợp tận dụng được ưu điểm của cả hai phương pháp ánh xạ trực tiếp và ánh xạ kết hợp đầy đủ: nhanh do ánh xạ trực tiếp được sử dụng cho ánh xạ dòng - chiếm số lớn ánh xạ và mềm dẻo, ít xung đột do ánh xạ từ các trang bộ nhớ đến các đường cache là không cố định. Nhờ vậy, phân bổ sử dụng không gian cache đồng đều hơn và đạt hệ số hit cao hơn. Nhược điểm lớn nhất của phương pháp này là có độ phức tạp thiết kế và điều khiển cao do cache được chia thành một số đường, thay vì chỉ một đường duy nhất.

### 3.2.6 Phương pháp đọc ghi và các chính sách thay thế dòng cache

#### 3.2.6.1 Các phương pháp đọc ghi cache

Việc trao đổi thông tin giữa CPU – cache và giữa cache – bộ nhớ chính là một trong các vấn đề có ảnh hưởng lớn đến hiệu năng cache. Câu hỏi đặt ra là cần có chính

sách trao đổi hay đọc ghi thông tin giữa các thành phần này như thế nào để đạt được hệ số hit cao nhất và giảm thiểu miss.

Xét trường hợp đọc thông tin và nếu đó là trường hợp hit (mẫu tin cần đọc có trong cache): mẫu tin được đọc từ cache vào CPU và bộ nhớ chính không tham gia. Như vậy thời gian CPU truy nhập mẫu tin bằng thời gian CPU truy nhập cache. Ngược lại, nếu đọc thông tin và đó là trường hợp miss (mẫu tin cần đọc không có trong cache): mẫu tin trước hết được chuyển từ bộ nhớ chính vào cache, sau đó nó được đọc từ cache vào CPU. Đây là trường hợp xấu nhất: thời gian CPU truy nhập mẫu tin bằng thời gian truy nhập cache cộng với thời gian cache truy nhập bộ nhớ chính – còn gọi là *miss penalty* (gấp đôi thời gian truy cập khi đoán trượt).

Với trường hợp ghi thông tin và nếu đó là trường hợp hit, có thể áp dụng một trong 2 chính sách ghi: *ghi thẳng* (write through) và *ghi trễ* (write back). Với phương pháp ghi thẳng, mẫu tin cần ghi được lưu đồng thời ra cache và bộ nhớ chính. Phương pháp ghi này luôn đảm bảo tính nhất quán dữ liệu giữa cache và bộ nhớ chính, nhưng có thể gây chậm trễ và tốn nhiều băng thông khi tần suất ghi lớn với nhiều mẫu tin có kích thước nhỏ. Ngược lại, với phương pháp ghi trễ, mẫu tin trước hết được ghi ra cache và dòng cache chứa mẫu tin sẽ được ghi ra bộ nhớ chính khi nó bị thay thế. Như vậy, mẫu tin có thể được ghi ra cache nhiều lần, nhưng chỉ được ghi ra bộ nhớ chính một lần duy nhất, giúp tăng tốc độ và giảm băng thông sử dụng. Phương pháp ghi trễ đang được ứng dụng rộng rãi trong các hệ thống cache hiện nay.

Với trường hợp ghi thông tin và nếu đó là trường hợp miss, cũng có thể áp dụng một trong hai chính sách ghi: *ghi có đọc lại* (write allocate / fetch on write) và *ghi không đọc lại* (write non-allocate). Với phương pháp ghi có đọc lại, mẫu tin trước hết được ghi ra bộ nhớ chính, và sau đó dòng nhớ chứa mẫu tin vừa ghi được đọc vào cache. Việc đọc lại mẫu tin vừa ghi từ bộ nhớ chính vào cache có thể giúp giảm miss đọc kế tiếp áp dụng nguyên lý lân cận theo thời gian: mẫu tin vừa được truy nhập có thể được truy nhập lại trong tương lai gần. Với phương pháp ghi không đọc lại, mẫu tin chỉ được ghi ra bộ nhớ chính. Không có thao tác đọc dòng nhớ chứa mẫu tin vừa ghi vào cache.

### 3.2.6.2 Các chính sách thay thế dòng cache

Như đã đề cập, với cả ba phương pháp ánh xạ bộ nhớ chính – cache, luôn có nhiều dòng nhớ cùng ánh xạ đến một dòng cache. Do có nhiều dòng bộ nhớ chia sẻ một dòng cache, các dòng bộ nhớ được nạp vào cache sử dụng một thời gian và được thay thế bởi dòng nhớ khác theo yêu cầu thông tin phục vụ CPU. Các chính sách thay thế (replacement policies) xác định các dòng cache nào được chọn để thay thế bởi các dòng khác từ bộ nhớ nhằm đạt hệ số hit cao nhất. Có ba chính sách thay thế được sử dụng hiện nay: *thay thế ngẫu nhiên* (Random Replacement), *thay thế kiểu vào trước ra trước* (FIFO – First in First Out) và *thay thế các dòng ít được sử dụng gần đây nhất* (LRU – Least Recently Used).

Thay thế ngẫu nhiên là phương pháp đầu tiên được sử dụng do có thiết kế đơn giản và dễ cài đặt. Các dòng cache được lựa chọn để thay thế một cách ngẫu nhiên, không theo một quy luật nào. Do vậy, phương pháp thay thế ngẫu nhiên thường có hệ số miss cao do phương pháp này không xem xét đến các dòng cache đang thực sự được sử dụng. Nếu một dòng cache đang được sử dụng và bị thay thế sẽ xảy ra miss và nó lại cần được đọc từ bộ nhớ chính vào cache.

Trong phương pháp thay thế kiểu vào trước, các dòng nhớ được nạp vào cache trước sẽ được chọn để thay thế trước. Phương pháp này luôn có khuynh hướng loại bỏ các dòng cache có thời gian sử dụng lâu nhất, hay “già nhất”. Nó có khả năng cho hệ số miss thấp hơn so với thay thế ngẫu nhiên do phương pháp này có xem xét đến yếu tố lân cận theo thời gian – các dòng nhớ có thời gian tồn tại trong cache lâu nhất có xác suất được sử dụng thấp hơn. Tuy nhiên, phương pháp này vẫn chưa thực sự xem xét đến các dòng cache đang thực sự được sử dụng - một dòng cache “già” vẫn có thể đang được sử dụng. Một nhược điểm khác của thay thế kiểu vào trước là thiết kế và cài đặt phức tạp hơn, do cần phải có mạch điện tử chuyên dụng để theo dõi trạng thái nạp các dòng bộ nhớ vào cache.

Phương pháp thay thế các dòng ít được sử dụng gần đây nhất hoạt động theo nguyên tắc: các dòng cache được lựa chọn để thay thế là các dòng ít được sử dụng gần đây nhất. Phương pháp này cho hệ số miss thấp nhất so với thay thế ngẫu nhiên và thay thế FIFO, do thay thế LRU có xem xét đến các dòng đang thực sự được sử dụng – tuân theo yếu tố lân cận theo thời gian một cách chặt chẽ. Nhược điểm duy nhất của phương pháp này là thiết kế và cài đặt phức tạp hơn, do cần phải có mạch điện tử chuyên dụng để theo dõi tần suất sử dụng các dòng cache.

### 3.2.7 Hiệu năng bộ nhớ cache

#### 3.2.7.1 Hiệu năng cache

Hiệu năng của cache (Cache Performance) có thể được đánh giá tổng thể theo thời gian truy nhập trung bình của CPU đến hệ thống nhớ. Thời gian truy nhập trung bình của một hệ thống nhớ có cache được tính như sau:

$$t_{access} = (\text{Hit cost}) + (\text{miss rate}) * (\text{miss penalty})$$

$$t_{access} = t_{cache} + (1 - H) * (t_{memory})$$

trong đó,  $t_{access}$  là thời gian truy nhập trung bình hệ thống nhớ,  $t_{memory}$  là thời gian truy nhập bộ nhớ chính,  $t_{cache}$  là thời gian truy nhập cache và  $H$  là hệ số hit.

Nếu  $t_{cache} = 5\text{ns}$ ,  $t_{memory} = 60\text{ns}$  và  $H=80\%$ , ta có:

$$t_{access} = 5 + (1 - 0.8) * (60) = 5+12 = 17\text{ns}$$

Nếu  $t_{cache} = 5\text{ns}$ ,  $t_{memory} = 60\text{ns}$  và  $H=95\%$ , ta có:

$$t_{access} = 5 + (1 - 0.95) * (60) = 5+3 = 8\text{ns}$$

Như vậy, thời gian truy nhập trung bình hệ thống nhớ tiệm cận thời gian truy nhập cache với trường hợp cache đạt hệ số hit cao.

### 3.2.7.2 Các yếu tố ảnh hưởng

Có nhiều yếu tố ảnh hưởng đến hiệu năng cache, trong đó ba vấn đề (1) kích thước cache, (2) chia tách cache và (3) tạo cache nhiều mức có ảnh hưởng lớn nhất. Chúng ta lần lượt xem xét ảnh hưởng của từng yếu tố đến hiệu năng cache.

#### Vấn đề kích thước cache

Vấn đề kích thước cache liên quan đến việc trả lời câu hỏi: nên lựa chọn kích thước cache lớn hay nhỏ? Nhiều số liệu thống kê cho thấy, kích thước cache không ảnh hưởng nhiều đến hệ số miss và hệ số miss của cache lệnh thấp hơn nhiều so với cache dữ liệu:

- **8KB cache lệnh:** Hệ số miss nhỏ hơn **1%**.
- **256KB cache lệnh:** Hệ số miss nhỏ hơn **0.002%**.

→ **Nhận xét:**

- Tăng kích thước cache lệnh từ 8KB lên 256KB (tăng 32 lần) chỉ làm giảm hệ số miss rất ít.
- Điều này cho thấy tăng kích thước cache lệnh không hiệu quả trong việc giảm tỷ lệ miss.
- **8KB cache dữ liệu:** Hệ số miss nhỏ hơn **4%**.
- **256KB cache dữ liệu:** Hệ số miss nhỏ hơn **3%**.

→ **Nhận xét:**

- Tăng kích thước cache dữ liệu lên 32 lần giúp giảm hệ số miss từ 4% xuống 3%, tức giảm khoảng 25%.
- Điều này cho thấy việc tăng kích thước cache dữ liệu mang lại hiệu quả cao hơn so với cache lệnh.

Trên thực tế, xu hướng chung mong muốn kích thước cache càng lớn trong giới hạn cho phép của giá thành. Với kích thước lớn, có thể tăng được số dòng bộ nhớ lưu trong cache và nhờ vậy giảm tần suất tráo đổi các dòng cache của các chương trình khác nhau với bộ nhớ chính. Đồng thời, cache lớn hỗ trợ đa nhiệm, xử lý song song và các hệ thống CPU nhiều nhân tốt hơn do không gian cache lớn có khả năng chứa đồng thời thông tin của nhiều chương trình. Nhược điểm của cache lớn là chậm, do có không gian tìm kiếm lớn hơn cache nhỏ.

#### Vấn đề chia tách cache

Cache có thể được tách thành cache lệnh (I-Cache) và cache dữ liệu (D-Cache) để cải thiện hiệu năng, do:

- Dữ liệu và lệnh có tính lân cận khác nhau;
- Dữ liệu thường có tính lân cận về thời gian cao hơn lân cận về không gian; lệnh có tính lân cận về không gian cao hơn lân cận về thời gian;
- Cache lệnh chỉ cần hỗ trợ thao tác đọc; cache dữ liệu cần hỗ trợ cả 2 thao tác đọc và ghi và tách cache giúp tối ưu hóa dễ dàng hơn;

- Tách cache hỗ trợ nhiều lệnh truy nhập đồng thời hệ thống nhớ, nhờ vậy giảm xung đột tài nguyên cho CPU pipeline.

## Vấn đề tạo cache nhiều mức

Khi cache được chia thành nhiều mức với kích thước tăng dần và tốc độ truy nhập giảm dần sẽ giúp cải thiện được hiệu năng hệ thống do hệ thống cache nhiều mức có khả năng dung hòa tốt hơn tốc độ của CPU với tốc độ của bộ nhớ chính.

Ví dụ: xem xét 2 hệ thống nhớ có số mức cache khác nhau: hệ thống 3 mức cache (L1, L2 và L3) và hệ thống 1 mức cache (L1). Giả thiết CPU có thời gian truy nhập là 1ns, các mức cache L1, L2, L3 có thời gian truy nhập lần lượt là 5ns, 15ns và 30ns. Bộ nhớ chính có thời gian truy nhập là 60ns.

	CPU	L1	L2	L3	Bộ nhớ chính
Cache 3 mức:	1ns	5ns	15ns	30ns	60ns
Cache 1 mức:	1ns	5ns			60ns

Có thể thấy hệ thống nhớ với nhiều mức cache có khả năng dung hòa tốc độ giữa các thành phần tốt hơn và có thời gian truy nhập trung bình hệ thống nhớ thấp hơn. Trên thực tế, đa số cache được tổ chức thành 2 mức: L1 và L2. Một số cache có 3 mức: L1, L2 và L3. Ngoài ra, nhiều mức cache có thể giúp giảm giá thành hệ thống nhớ.

### 3.2.8 Các phương pháp giảm miss cho cache

#### 3.2.8.1 Các loại miss của cache

Một hệ thống nhớ với cache tốt cần đạt được các yếu tố: (1) hệ số hit cao, (2) hệ số miss thấp và (3) nếu xảy ra miss thì không quá chậm. Để có thể có giải pháp giảm miss hiệu quả, ta cần phân biệt rõ các loại miss. Cụ thể, tồn tại ba loại miss chính: *miss bắt buộc* (Compulsory misses), *miss do dung lượng* (Capacity misses) và *miss do xung đột* (Conflict misses). Miss bắt buộc thường xảy ra tại thời điểm chương trình được kích hoạt, khi mã chương trình đang được tải vào bộ nhớ và chưa được nạp vào cache. Miss do dung lượng lại thường xảy ra do kích thước của cache hạn chế, đặc biệt trong môi trường đa nhiệm. Do kích thước cache nhỏ nên mã của các chương trình thường xuyên bị tráo đổi giữa bộ nhớ và cache. Theo một khía cạnh khác, miss do xung đột xảy ra khi có nhiều dòng bộ nhớ cùng cạnh tranh một dòng cache.

#### 3.2.8.2 Các phương pháp giảm miss cho cache

Trên cơ sở các loại miss đã được đề cập, hai phương pháp giảm miss có thể phối hợp áp dụng nhằm đạt hiệu quả giảm miss tối đa, gồm: *tăng kích thước dòng cache* và *tăng mức độ liên kết cache*. Biện pháp tăng kích thước dòng cache có thể giúp giảm miss bắt buộc do dòng có kích thước lớn sẽ có khả năng bao phủ các mục tin lân cận tốt hơn. Tuy nhiên, biện pháp này sẽ làm tăng miss xung đột, do dòng kích thước lớn sẽ làm giảm số dòng cache, dẫn đến tăng mức độ cạnh tranh của các dòng nhớ đến một dòng cache. Ngoài ra, dòng kích thước lớn có thể gây lãng phí dung

lượng cache do có thể có nhiều phần của dòng cache lớn không bao giờ được sử dụng. Hiện nay, kích thước dòng cache thường dùng hiện nay là 64 bytes.

Biện pháp tăng mức độ liên kết cache hay tăng số đường cache có thể giúp giảm miss xung đột, do tăng số đường cache làm tăng tính mềm dẻo của ánh xạ trang bộ nhớ đến đường cache do có nhiều lựa chọn hơn. Tuy nhiên, nếu tăng số đường cache quá lớn, có thể làm cache chậm do tăng không gian tìm kiếm các đường cache. Hiện nay, số đường cache hợp lý cho miss tối ưu thường dùng là khoảng 8 đường.

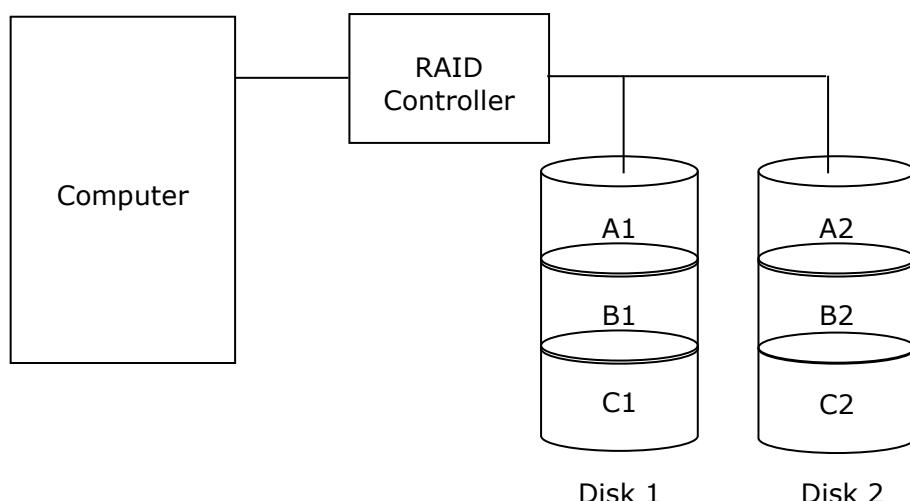
### 3.3 Một số công nghệ lưu trữ dữ liệu lớn

#### 3.3.1 Công nghệ RAID

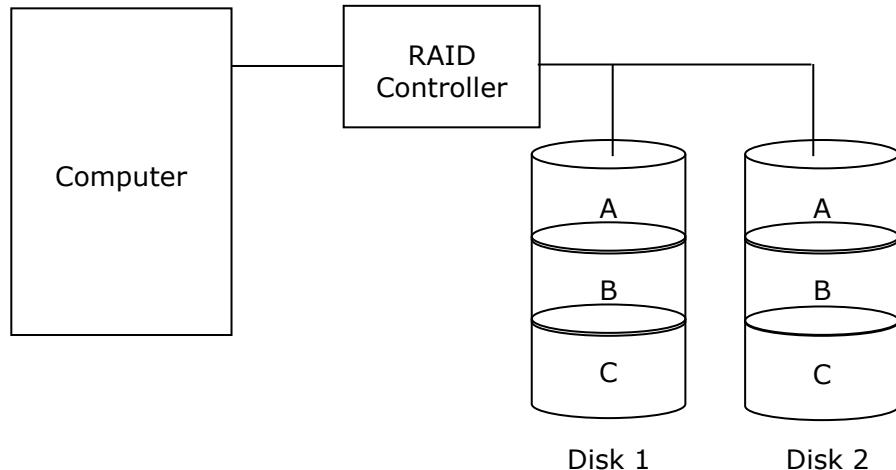
RAID (Redundant Array of Independent Disks) là một công nghệ tạo ra các hệ thống lưu trữ tiên tiến dựa trên các ổ đĩa cứng, nhằm đạt các yêu cầu về tốc độ cao (high performance / speed), tính tin cậy cao (high reliability) và dung lượng lớn (large volume). Mặc dù RAID là một mảng của các ổ đĩa cứng, nhưng không phải mọi loại ổ cứng đều hỗ trợ RAID. Trên thực tế, chỉ có các ổ cứng theo chuẩn SATA, SCSI và tương đương mới hỗ trợ tạo RAID.

Có hai kỹ thuật chính được sử dụng để tạo RAID: kỹ thuật tạo lát đĩa (Disk Striping) và kỹ thuật soi gương đĩa (Disk Mirroring). **Error! Reference source not found.** minh họa kỹ thuật tạo lát đĩa. Điểm mấu chốt của kỹ thuật này là điều khiển RAID cung cấp khả năng ghi và đọc song song các khối của cùng một đơn vị dữ liệu. Nhờ vậy tăng được tốc độ đọc ghi. Theo đó, các dữ liệu cần ghi được chia thành các khối cùng kích thước và được ghi đồng thời vào các ổ đĩa vật lý độc lập. Tương tự, trong quá trình đọc, các khối của dữ liệu cần đọc được đọc đồng thời từ các đĩa cứng độc lập, giúp giảm thời gian đọc.

Trong khi kỹ thuật tạo lát đĩa hướng đến tốc độ cao, kỹ thuật soi gương đĩa nhằm đạt độ tin cậy cao cho hệ thống lưu trữ. **Error! Reference source not found.** minh họa kỹ thuật soi gương đĩa. Theo đó, dữ liệu cũng được chia thành các khối và mỗi khối được ghi đồng thời lên hai hay nhiều ổ đĩa độc lập. Như vậy, tại mọi thời điểm ta đều có nhiều bản sao dữ liệu trên các đĩa cứng độc lập, đảm bảo tính an toàn cao.



Hình 3. 23. RAID - Kỹ thuật tạo lát đĩa



Hình 3. 24. RAID – Kỹ thuật soi gương đĩa

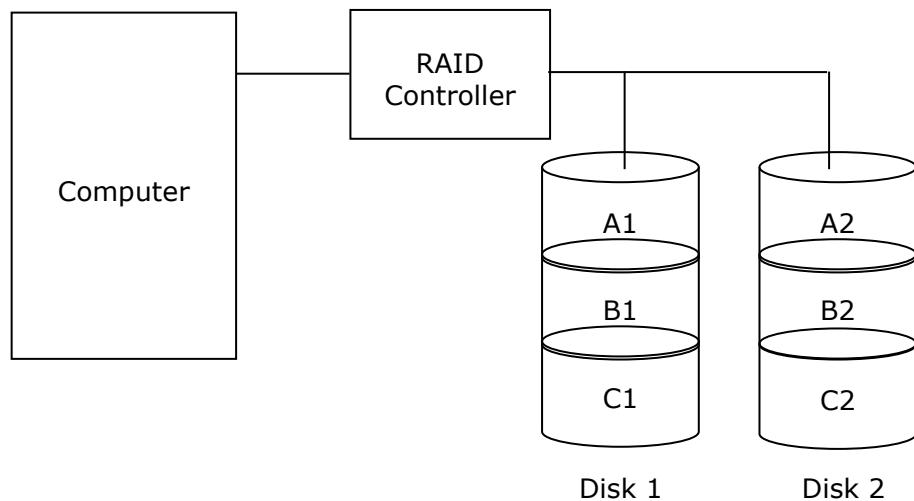
Trong mục này giới thiệu ba dạng RAID được sử dụng phổ biến nhất: RAID 0, RAID 1 và RAID 10. Các dạng RAID khác chẳng hạn RAID 2, 3, 4, 5, 6, 01, độc giả có thể tham khảo thêm trong các tài liệu khác. Một số loại RAID:

### 3.3.1.1 RAID 0

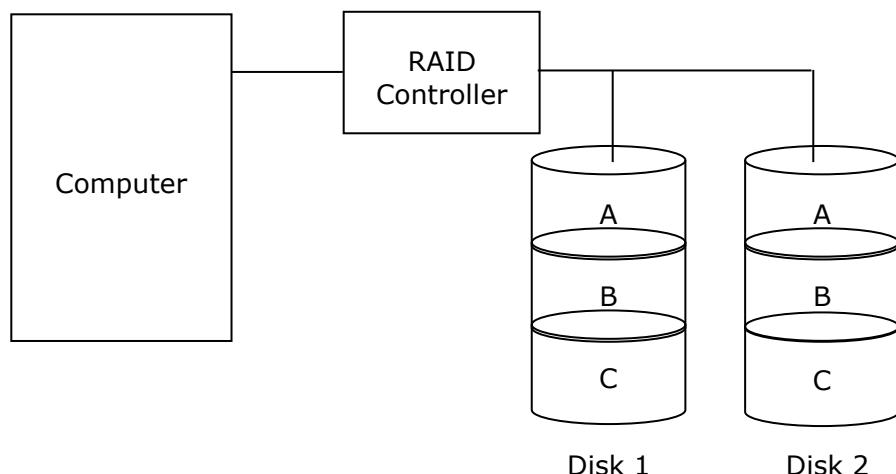
Cấu hình RAID 0 dựa trên kỹ thuật tạo lát đĩa và cần tối thiểu hai ổ đĩa vật lý, như minh họa trên **Error! Reference source not found..** Ưu điểm chính của RAID 0 là đạt tốc độ cao – tốc độ truy nhập RAID tỷ lệ thuận với số lượng đĩa độc lập của RAID. Ngoài ra, RAID 0 có thể giúp tăng dung lượng: dung lượng RAID 0 bằng tổng dung lượng của các đĩa độc lập tham gia. Hạn chế lớn nhất của RAID 0 là tính tin cậy – tính tin cậy của RAID 0 chỉ tương đương tính tin cậy của một ổ đĩa đơn.

### 3.3.1.2 RAID 1

Khác với RAID 0, cấu hình RAID 1 dựa trên kỹ thuật soi gương đĩa và cũng cần tối thiểu hai ổ đĩa vật lý, như minh họa trên **Error! Reference source not found..** Ưu điểm chính của RAID 1 là đạt độ tin cậy cao, do tại mỗi thời điểm luôn có nhiều bản sao lưu dữ liệu trên các đĩa độc lập. Tốc độ truy nhập và dung lượng của RAID 1 đều tương đương với một ổ đĩa đơn.



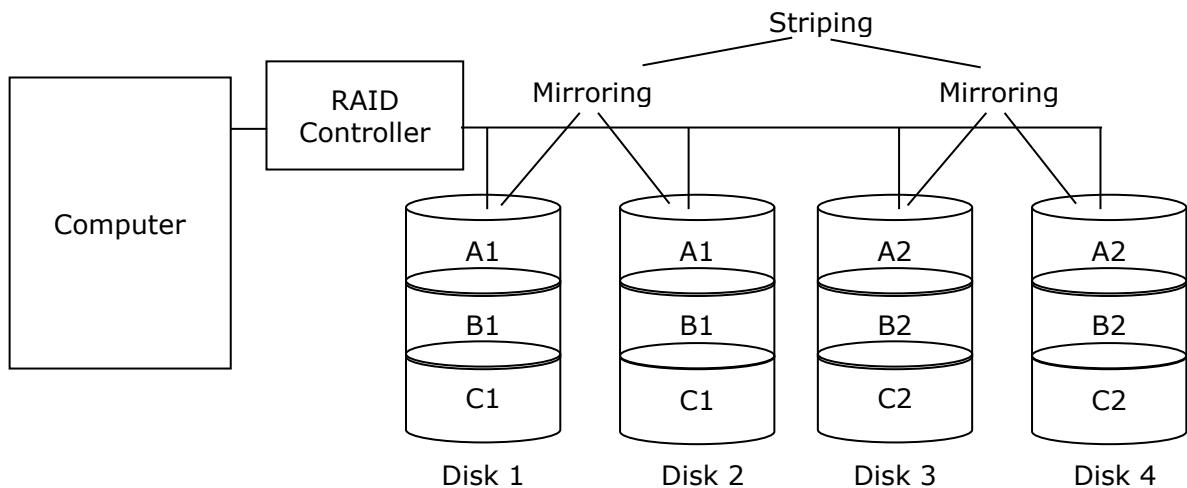
Hình 3. 25. Cấu hình RAID 0



Hình 3. 26. Cấu hình RAID 1

### 3.3.1.3 RAID 10

Cấu hình RAID 10 là sự kết hợp của RAID 1 và RAID 0, kết hợp cả hai kỹ thuật tạo lát đĩa và soi gương đĩa. RAID 10 cần tối thiểu 4 ổ đĩa độc lập như minh họa trên **Error! Reference source not found.**. Ưu điểm của RAID 10 là đạt được cả tốc độ cao và tính tin cậy cao, nên rất phù hợp với các hệ thống máy chủ đòi hỏi tính an toàn cao, hiệu năng lớn như máy chủ cơ sở dữ liệu. Dung lượng RAID 10 bằng một nửa tổng dung lượng các đĩa độc lập tham gia tạo RAID. Nhược điểm duy nhất của RAID 10 là giá thành cao.



Hình 3. 27. Cấu hình RAID 10

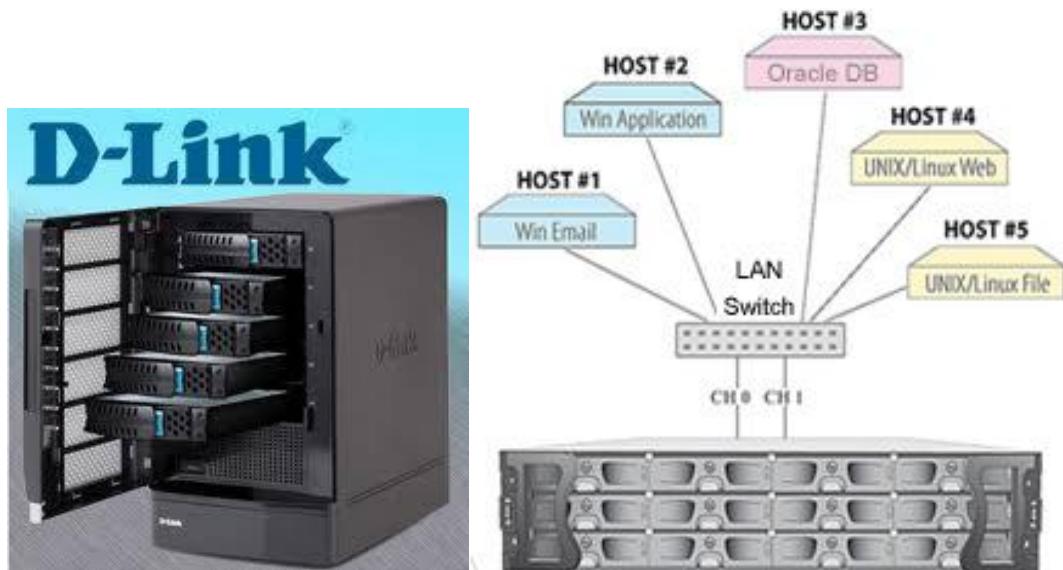
### 3.3.2 Công nghệ NAS

NAS (Network Attached Storage) là một dạng thiết bị lưu trữ được gắn trực tiếp vào mạng, thường là mạng cục bộ LAN. **Error! Reference source not found.** minh họa một thiết bị NAS USR8700 được gắn vào mạng LAN và cung cấp khả năng lưu trữ cho toàn bộ mạng.



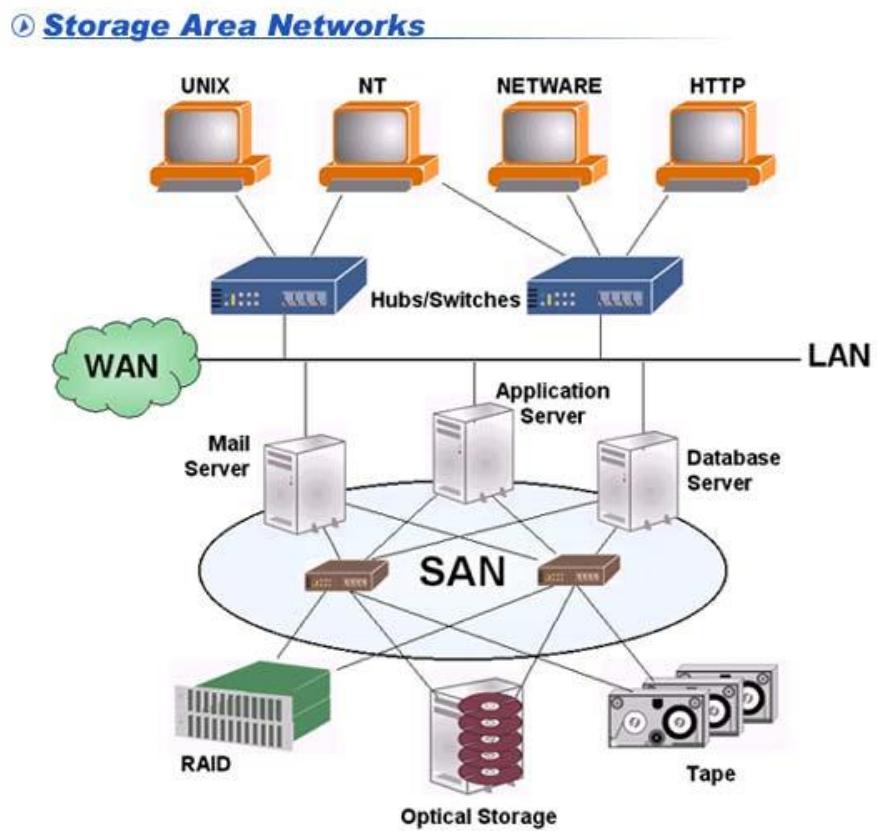
Hình 3. 28. NAS trong một mạng LAN

Trên thực tế, NAS thường là một máy chủ chuyên dùng làm thiết bị lưu trữ, được kết nối vào mạng (thường là LAN tốc độ cao) và cung cấp các dịch vụ lưu trữ thông qua mạng. NAS thường dựa trên nền tảng là một RAID có tốc độ cao, dung lượng lớn và độ tin cậy rất cao. NAS có thể cung cấp dịch vụ lưu trữ cho hầu hết các loại máy chủ có cấu hình phần cứng khác nhau và chạy các hệ điều hành khác nhau, cũng như các phần mềm ứng dụng khác nhau, như minh họa trên **Error! Reference source not found..**



Hình 3. 29. NAS của hãng D-Link và mô hình sử dụng trong mạng LAN

### 3.3.3 Công nghệ SAN



Hình 3. 30. Mô hình mạng lưu trữ SAN và ứng dụng

SAN (Storage Area Network) là một mạng của các máy chủ chuyên dụng cung cấp dịch vụ lưu trữ như minh họa trên **Error! Reference source not found.**, với các đặc điểm:

- Tốc độ truy nhập rất cao;
- Dung lượng cực lớn;
- Độ an toàn rất cao
- An toàn dữ liệu cục bộ
- An toàn dữ liệu với các bản copy được đồng bộ ở khoảng cách xa về địa lý.

Thông thường, SAN thường được tổ chức dưới dạng hệ thống tệp tin phân tán (Distributed File System), phục vụ nhu cầu lưu trữ lượng dữ liệu khổng lồ của các tổ chức và doanh nghiệp lớn.

### 3.4 Kết luận chương

Chương 3 đã trình bày chi tiết về cơ chế xử lý xen kẽ dòng mã lệnh (pipeline) và bộ nhớ cache, cùng các công nghệ lưu trữ dữ liệu lớn. Pipeline giúp tăng hiệu suất CPU thông qua việc thực hiện đồng thời nhiều lệnh, nhưng đối mặt với các vấn đề như xung đột tài nguyên, tranh chấp dữ liệu, và các lệnh rẽ nhánh. Các giải pháp khắc phục bao gồm cải tiến phần cứng, tối ưu hóa thứ tự lệnh và dự đoán trước dữ liệu. Bộ nhớ cache đóng vai trò quan trọng trong việc cân bằng tốc độ giữa CPU và bộ nhớ chính, với các phương pháp tổ chức như ánh xạ trực tiếp, kết hợp đầy đủ và tập kết hợp, đảm bảo hiệu năng tối ưu. Các công nghệ lưu trữ hiện đại như RAID, NAS và SAN cung cấp giải pháp lưu trữ tốc độ cao, dung lượng lớn và độ tin cậy cao, đáp ứng nhu cầu ngày càng tăng về xử lý và lưu trữ dữ liệu. Sự phối hợp giữa các kỹ thuật xử lý lệnh và công nghệ lưu trữ hiện đại không chỉ cải thiện hiệu năng hệ thống mà còn nâng cao khả năng mở rộng và độ tin cậy, phù hợp với các hệ thống tính toán hiện đại.

### 3.5 Câu hỏi ôn tập

1. Nguyên lý hoạt động cơ chế xen kẽ dòng mã lệnh?
2. (\*) Các vấn đề của cơ chế xen kẽ dòng mã lệnh và cách khắc phục?
3. Hệ thống bộ nhớ phân cấp: đặc điểm, vai trò.
4. (\*) Bộ nhớ cache:
  - Cache là gì? vai trò và nguyên lý hoạt động.
  - Kiến trúc cache
  - Tổ chức/ánh xạ cache
  - Đọc ghi thông tin trong cache
  - Các chính sách thay thế dòng cache
  - Hiệu năng cache và các yếu tố ảnh hưởng
  - Các biện pháp giam miss cho cache.
5. (\*) RAID:
 

RAID là gì? Trình bày các kỹ thuật chính để tạo RAID.

So sánh các cấu hình RAID 0, RAID 1 và RAID 10.

6. Trình bày về NAS.

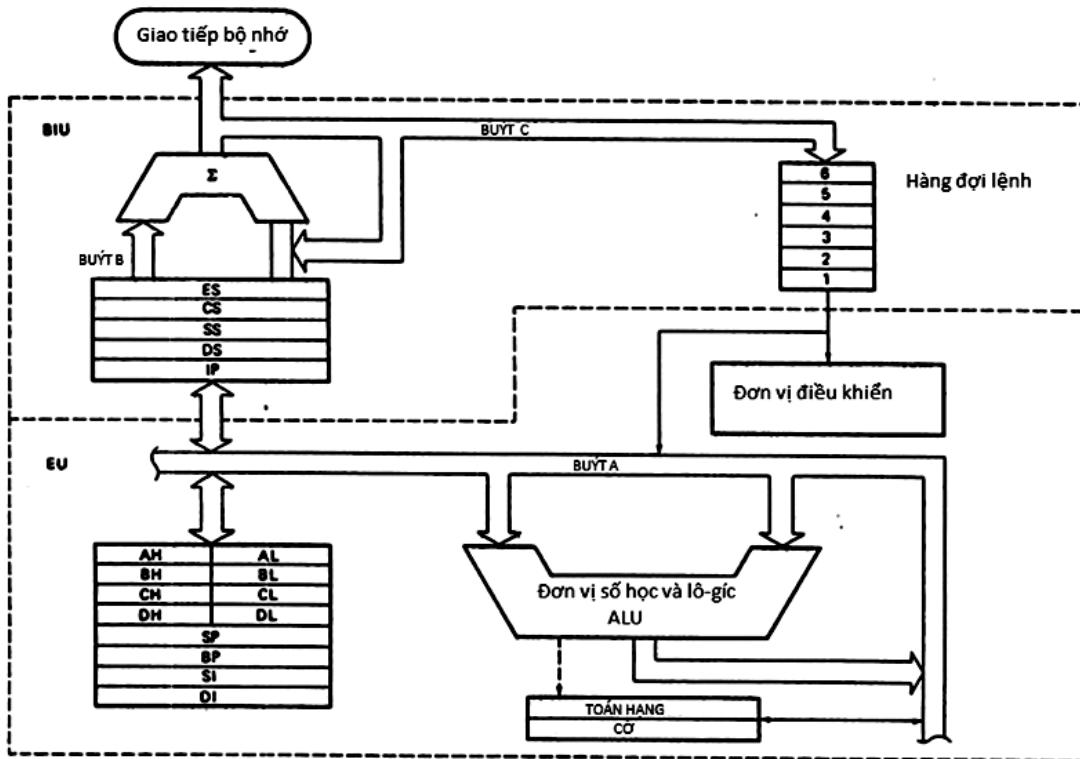
## CHƯƠNG 4 LẬP TRÌNH HỢP NGỮ VỚI BỘ VI XỬ LÝ 8086/8088

Chương 4 giới thiệu về lập trình hợp ngữ trên vi xử lý 8086/8088, giúp người học hiểu rõ kiến trúc, các thanh ghi, mã hóa lệnh và chế độ địa chỉ. Ngoài lý thuyết, chương còn trình bày tập lệnh quan trọng, hướng dẫn lập trình hợp ngữ với các cấu trúc cơ bản như tuần tự, rẽ nhánh và lặp, đồng thời sử dụng công cụ giả lập Emu8086 để thực hành. Qua đó, người học có thể nắm vững cách viết và tối ưu hóa mã hợp ngữ, tạo nền tảng cho lập trình hệ thống và nhúng.

### 4.1 Kiến trúc và các thành phần bộ vi xử lý 8086/8088

#### 4.1.1 Kiến trúc bộ vi xử lý 8086/8088

Intel 8086 là bộ vi xử lý 16 bit đầu tiên của Intel và là vi xử lý đầu tiên hỗ trợ tập lệnh x86. Ngoài ra Intel cũng giới thiệu 8088 tương thích với 8086 nhưng độ rộng buýt dữ liệu bằng một nửa (8 bit). Vi xử lý 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 thời gian tới do tính kế thừa của các sản phẩm trong họ x86. Các chương trình viết cho 8086/8088 vẫn có thể chạy trên các hệ thống tiên tiến sau này



Hình 4. 1. Sơ đồ khái niệm của bộ vi xử lý 8086/8088

Trong sơ đồ khái niệm, vi xử lý 8086 có hai khối chính BIU và EU. Về chi tiết, vi xử lý này bao gồm các đơn vị điều khiển, số học và lô-gíc, hàng đợi lệnh và tập các thanh ghi. Chi tiết các khối và đơn vị chức năng này được trình bày trong phần sau.

Theo sơ đồ khối trên **Error! Reference source not found.** CPU 8086 có 2 k hối chính: *khối phối ghép buýt* BIU (Bus Interface Unit) và *khối thực hiện lệnh* EU (Execution Unit). 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 buýt bên trong CPU có nhiệm vụ chuyển tải tín hiệu giữa các khối. Trong số các buýt đó có buýt dữ liệu 16 bit của ALU, buýt các tín hiệu điều khiển ở EU và buýt trong của hệ thống ở BIU. Trước khi đi ra buýt ngoài hoặc đi vào buýt trong của bộ vi xử lý, các tín hiệu truyền trên buýt thường được cho đi qua các bộ đệm để nâng cao tính tương thích cho nối ghép hoặc nâng cao phổi 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 buýt và trao đổi dữ liệu với buýt.

EU bao gồm một *đơn vị điều khiển*, khối 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 ta thu được các dãy xung khác nhau trên kênh điều khiển (tùy theo mã lệnh) để điều khiển hoạt động của các bộ phận bên trong và bên ngoài CPU. Ngoài ra, EU còn có *khối số học và lôgic* (Arithmetic and 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 6 byte dùng để chứa các mã lệnh để chờ EU xử lý (bộ đệm lệnh này còn được gọi là *hàng đợi lệnh*).

#### 4.1.2 Các thành phần bộ xử lý 8086/8088

##### 4.1.2.1 Các thanh ghi đoạn

Thông thường bộ nhớ của chương trình máy tính được chia làm các đoạn phục vụ các chức năng khác nhau như đoạn chứa các câu lệnh, chứa dữ liệu. Trong thực tế bộ vi xử lý 8086 cung cấp các thanh ghi 16 bit liên quan đến địa chỉ đầu của các đoạn kể trên và chúng được gọi là các thanh ghi đoạn (Segment Registers) cụ thể:

- Thanh ghi đoạn mã CS (Code-Segment),
- Thanh ghi đoạn dữ liệu DS (Data segment).
- Thanh ghi đoạn ngăn xếp SS (Stack segment)
- 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. Để xác định chính xác vị trí một ô nhớ của chương trình các thanh ghi đoạn sẽ phải phối hợp với các thanh ghi đặc biệt khác còn gọi là các thanh ghi lệch hay phân đoạn (offset register).

Khối BIU đưa ra trên buýt địa chỉ 20-bit địa chỉ, như vậy 8086/8088 có khả năng phân biệt ra được  $2^{20} = 1.048.576 = 1M$  ô nhớ hay 1Mbyte, vì các bộ nhớ thường 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 8086/8088 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 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 MByte.

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), do nó ứng với khoảng lệch địa chỉ của 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 được trình bày sau. Cụ thể, để xác định địa chỉ vật lý 20-bit của một ô nhớ nào đó trong một đoạn bất kỳ. CPU 8086/8088 phải dùng đến 2 thanh ghi 16 bit: một thanh ghi để chứa địa chỉ cơ sở, còn thanh kia chứa độ lệch. Từ nội dung của cặp thanh ghi đó tạo ra địa chỉ vật lý theo công thức sau:

$$\text{Địa chỉ vật lý} = \text{Thanh ghi đoạn} \times 16 + \text{Thanh ghi lệch} \quad (4.1)$$

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:

### **Thanh ghi đoạn: Thanh ghi lệch 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 buýt địa chỉ.

#### 4.1.2.2 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. Chính vì vậy các thanh ghi thường được gán cho những cái tên có ý nghĩa. Cụ thể:

- AX (accumulator): thanh chứa. Các kết quả 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 thanh ghi tích luỹ.
- 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.

#### 4.1.2.3 Các thanh ghi con trỏ và chỉ số

Trong 8086 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 ngắn xếp SS. Địa chỉ đầy đủ của một phần tử trong đoạn ngắn xếp ứng với SS : BP và được xác định theo cách đã nói ở trên.
- SP: con trỏ ngắn xếp (Stack Pointer). SP luôn trỏ vào đỉnh hiện thời của ngắn xếp nằm trong đoạn ngắn xếp SS. Địa chỉ đỉnh ngắn xếp ứng với SS: SP và được xác định theo cách đã nói ở trên.
- 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.

- 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 thao 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.

#### 4.1.2.4 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 bit bit nhưng người ta chỉ dùng hết 9 bit của nó để làm các bit cờ như **Error! Reference source not found.** dưới đây.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
U	U	U	U	OF	DF	IF	TF	SF	ZF	U	AF	U	PF	U	CF

Hình 4. 2. Thanh ghi cờ

- U không sử dụng.
- C hoặc CF (Carry Flag): cờ nhớ. CF = 1 khi có nhớ hoặc muộn từ bit có nghĩa lớn nhất MSB (Most Significant Bit).
- P hoặc PF (Parity Flag): cờ parity. PF phản ánh tính chẵn lẻ của tổng số bit 1 có trong kết quả. Cờ PF = 1 khi tổng bit bit 1 trong kết quả là chẵn (even parity).
- A hoặc AF (Auxiliary Carry Flag): cờ nhớ phụ rất có ý nghĩa khi ta làm việc với các số BCD (Binary Coded Decimal). AF = 1 khi có nhớ hoặc muộn từ một số BCD thấp (4-bit thấp) sang một số BCD cao (4-bit cao).
- Z hoặc ZF (Zero Flag): cờ rỗng. ZF = 1 khi kết quả = 0.
- S hoặc SF (sign flag): cờ dấu. SF = 1 khi kết quả âm.
- O hoặc OF (Overflow Flag): cờ tràn. OF = 1 khi kết quả là 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 quả 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 lắp hoặc xoá tùy theo các điều kiện cụ thể sau các thao tác của ALU. Ngoài ra, bộ vi xử lý 8086/8088 còn có các cờ điều khiển sau đây (các cờ này được lắp hoặc xoá bằng các lệnh riêng):

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

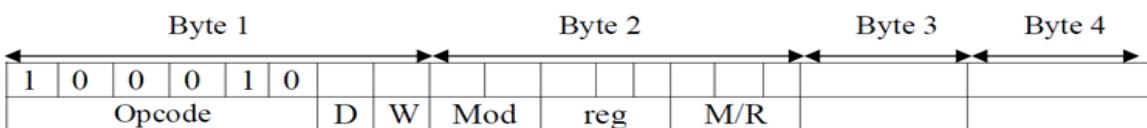
## 4.2 Mã hóa lệnh và các chế độ địa chỉ

### 4.2.1 Mã hóa lệnh

Lệnh của bộ vi xử lý được ghi bằng các ký tự dưới dạng gọi nhớ (mnemonic) để người sử dụng dễ nhận biết. Đối với bản thân bộ vi xử lý thì lệnh cho nó được mã hóa 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 lệnh.

Một lệnh có thể có độ dài một vài byte tùy theo bộ vi xử lý. Số lượng các bit  $n$  dùng để mã hóa vi lệnh (opcode) cho biết số lượng tối đa các lệnh ( $2^n$ ) có trong bộ vi xử lý. Với 1 byte bộ vi xử lý có thể mã hóa được tối đa 256 lệnh. Trong thực tế việc ghi lệnh không phải hoàn toàn đơn giản như vậy. Việc mã hóa lệnh cho bộ vi xử lý là rất phức tạp và bị chi phối bởi nhiều yếu tố khác. Đối với bộ vi xử lý 8086/8088 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 8086/8088.

Lệnh *MOV đích, gốc* dùng để chuyển dữ liệu giữa thanh ghi và ô nhớ. Chỉ riêng với các thanh ghi của 8086/8088, 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 rất nhiều mã lệnh khác nhau để mã hóa tổ hợp các này.



Hình 4. 3. Cấu trúc mã hóa lệnh MOV trong vi xử lý 8086/8088

**Error! Reference source not found.** trên biểu diễn dạng thức các byte dùng để mã hóa lệnh MOV. Từ đây ta thấy rằng để mã hóa 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. Bit W dùng để chỉ ra rằng một byte ( $W = 0$ ) hoặc 1 từ ( $W = 1$ ) sẽ được chuyển. 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 hóa các thanh ghi trong CPU như sau:

Bảng 4. 1. Mã hóa các thanh ghi trong vi xử lý 8086/8088 theo giá trị W

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

Bit D dùng để chỉ hướng đi của dữ liệu. D = 1 thì dữ liệu đi đến thanh ghi cho bởi bit của REG. 2-bit MOD (chế độ) cùng với 3-bit M/R (bộ nhớ/thanh ghi) tạo ra 5-bit dùng để chỉ ra chế độ địa chỉ cho các toán hạng của lệnh. **Error! Reference source not found.** dưới đây cho ta thấy cách mã hóa các chế độ địa chỉ, cách tìm ra các toán hạng bằng các bit này.

Bảng 4. 2. Mã hóa chế độ địa chỉ và thanh ghi trong vi xử lý 8086/8088

MOD R/M \	00	01	10	11	
				W = 1	W = 0
000	[BX]+[SI]	[BX]+[SI]+addr8	[BX]+[SI]+addr16	AX	AL
001	[BX]+[DI]	[BX]+[DI]+addr8	[BX]+[DI]+addr16	CX	CL
010	[BP]+[SI]	[BP]+[SI] +addr8	[BP]+[SI] +addr16	DX	DL
011	[BP]+[DI]	[BP]+[DI] +addr8	[BP]+[DI] +addr16	BX	BL
100	[SI]	[SI] +addr8	[SI] +addr16	SP	AH
101	[DI]	[DI] +addr8	[DI] +addr16	BP	CH
110	addr16	[BP] +addr8	[BP] +addr16	SI	DH
111	[BX]	[BX] +addr8	[BX] +addr16	DI	BH

Ghi chú:

- addr8, addr16 tương ứng với địa chỉ 8 và 16 bit
- 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)

#### 4.2.2 Khái niệm về chế độ địa chỉ

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 vi 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 ghi (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).

### **4.2.3 Các chế độ địa chỉ của bộ xử lý 8086/8088**

#### **4.2.3.1 Chế độ địa chỉ thanh ghi (Register)**

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 hơn so với các lệnh có truy nhập đến 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
```

#### **4.2.3.2 Chế độ địa chỉ tức thì (immediate)**

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 vị trí của toán hạng này ở ngay sau mã lệnh. Chế độ địa chỉ này có thể được dùng để nạp dữ liệu cần thao tác vào bất kỳ thanh ghi nào (ngoại 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, OFF0H    ; chuyển OFF0H 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.
```

#### **4.2.3.3 Chế độ địa chỉ trực tiếp (direct)**

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]   ; chuyển ô nhớ DS:1234 vào AL.
MOV [4320H], CX   ; chuyển CX vào 2 ô nhớ liên tiếp DS:4320 và DS:4321
```

#### **4.2.3.4 Chế độ gián tiếp qua thanh ghi (indirect)**

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 (8086/8088 không cho phép truy chiêu bộ nhớ hai lần đối với một lệnh).

Ví dụ:

```
MOV AL, [BX]      ; chuyển ô nhớ có địa chỉ DS: BX vào AL.
MOV [SI], CL      ; chuyển CL vào ô nhớ có địa chỉ DS:SI.
```

```
MOV [DI], AX ; chuyển AX vào 2 ô nhớ liên tiếp tại DS:DI và DS:(DI + 1).
```

#### 4.2.3.5 Chế độ địa chỉ tương đối cơ sở

Trong chế độ địa chỉ này các thanh ghi cơ sở như BX và BP và các hằng số biểu diễn các giá trị dịch chuyển (*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 của địa chỉ so với địa chỉ cơ sở.

Ví dụ :

```
MOV CX, [BX] +1 ; chuyển 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 ô nhớ SS : [BP+5] vào AL.  
ADD AL, Table [BX] ; cộng AL với ô nhớ do BX chỉ ra trong bảng table  
; (bảng này nằm trong DS), kết quả dựa vào AL.
```

Trong ví dụ trên:

- 10 và 5 là các giá trị cụ thể cho biết mức dịch chuyển của các toán hạng. 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 [BP+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.

#### 4.2.3.6 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 tạp 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 địa chỉ hóa các mảng hai chiều.

Ví dụ:

```
MOV AX, [BX] [SI]+8 ; chuyển 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 ô nhớ SS : [BP+DI+5] vào CL.
```

#### 4.2.3.7 Tổng kết các chế độ địa chỉ

Các chế độ địa chỉ đã trình bày ở trên có thể tóm tắt lại trong **Error! Reference source not found..**

Bảng 4. 3. Tóm tắt các chế độ địa chỉ

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

## 4.3 Tập lệnh và công cụ Emu8086

### 4.3.1 Tập lệnh 8086/8088

Bộ xử lý 8086 có tập lệnh gồm 111 lệnh, chiều dài của lệnh từ 1 byte đến vài byte. Tập lệnh 8086 hỗ trợ các nhóm thao tác căn bản như dưới đây.

#### 4.3.1.1 Các lệnh trao đổi dữ liệu.

**Error! Reference source not found.** liệt kê các lệnh trao đổi dữ liệu trong hệ thống, cho phép di chuyển dữ liệu giữa các thanh ghi, ô nhớ, hoặc thiết bị ngoại vi. Kích cỡ dữ liệu cho phép với các câu lệnh này là byte (8 bit) hoặc word (16 bit). Như các lệnh trao đổi dữ liệu giúp nạp dữ liệu cần thiết cho các thao tác tính toán của vi xử lý. Ngoài ra các lệnh này cho phép lưu các kết quả tính toán ra bộ nhớ hoặc các thiết bị ngoại vi.

Bảng 4. 4. Các lệnh trao đổi dữ liệu

Mã gọi nhó	Chức năng
MOV	Di chuyển byte hay word giữa thanh ghi và ô nhớ
IN, OUT	Đọc, ghi một byte hay word giữa cổng và ô nhớ
LEA	Nạp địa chỉ hiệu dụng
PUSH, POP	Nạp vào, lấy ra một word trong ngăn xếp.
XCHG	Hoán đổi byte hay word

*MOV – Chuyển 1 byte hay word*

*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	; Gán giá trị 74H (hex) vào thanh ghi AL
MOV CL, BL	; Sao chép giá trị của thanh ghi BL vào thanh ghi CL
MOV DL, [SI]	; Sao chép giá trị tại địa chỉ bộ nhớ được trả bởi SI (trong đoạn DS) vào thanh ghi DL
MOV AL, Table[BX]	; Sao chép giá trị tại địa chỉ bộ nhớ (Table + BX) trong đoạn DS vào thanh ghi AL

*LEA - 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 lêch của Gốc, hoặc

Đích ← Địa chỉ hiệu dụng của Gốc

Đây là lệnh để tính địa chỉ lêch 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 lêch 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

*IN- Đọc dữ liệu từ cổng vào thanh ghi 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 dẻo 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 gán từ trước giá trị ứng với địa chỉ cổng. Lệnh này không tác động đến các cờ.

*OUT - Ghi 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 hóa 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.

#### *4.3.1.2 Các lệnh tính toán số học và lô gíc.*

**Error! Reference source not found.** liệt kê các lệnh tính toán số học và logic, là nhóm lệnh cơ bản của vi xử lý 8086/8088, dùng để thực hiện các phép toán và xử lý dữ liệu.

Bảng 4. 5. Các lệnh số học và logic

<b>Mã gọi nhớ</b>	<b>Chức năng</b>
NOT	Đảo (bù một) byte hay word
AND	Phép và byte hoặc word
OR	Phép hoặc byte hoặc word
XOR	Phép hoặc loại trừ byte hoặc word
SHL, SHR	Dịch trái, dịch phải lôgic byte hay word. Số bước 1 hoặc do CL xác định
SAL, SAR	Dịch trái, dịch phải số học byte hay word. Số

	bước 1 hoặc do CL xác định
ROL, ROR	Quay trái, quay phải byte hay word. Số bước 1 hoặc do CL xác định
ADD, SUB	Cộng trừ byte hoặc word
ADC, SBB	Cộng trừ byte hoặc word có nhớ
INC, DEC	Tăng, giảm
NEG	Đảo byte hoặc word (bù 2)
CMP	So sánh hai byte hoặc word
MUL, DIV	Nhân, chia byte hoặc word không dấu
IMUL, IDIV	Nhân chia byte hoặc word có dấu

### *ADD - Cộng 2 toán hạng*

*Cú pháp:* ADD Đích, Gốc.

*Mô tả:*  $\text{Đích} \leftarrow \text{Đích} + \text{Gốc}$ .

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 ADC. Kết quả cập nhật các cờ: AF, CF, PF, SF, ZP.

### *MUL - Nhân số không dấu*

*Cú pháp:* 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:  $AL \times Gốc$ ,
  - số bị nhân phải là số 8 bit đặt trong AL.
  - sau khi nhân:  $AX \leftarrow$  tích,
- Nếu Gốc là số 16 bit:  $AX \times Gốc$ ,
  - số bị nhân phải là số 16 bit đặt trong AX.
  - sau khi nhân:  $DXAX \leftarrow$  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 quả có trong cặp DXAX.

Cập nhật: CF, OF.

Không xác định: AF, PF, SF, ZP.

DIV – Chia 2 số không có dấu

Cú pháp: 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 đặt trong AX.
- Nếu Gốc là số 16 bit: DXAX/Gốc. Số bị chia phải là số không dấu 32 bit đặt 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.

CMP- So sánh 2-byte hay 2 word

Cú pháp: 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 sánh, sau khi so sánh 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 ZF

Đích = Gốc 0 1

Đích < Gốc 0 1

Đích > Gốc 1 0

Cập nhật: AF, CF, OF, PF, SF, ZP.

AND - Phép và 2 toán hạng

Cú pháp: AND 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 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:* AF.

Ví dụ:

```
AND AL, BL      ; AL, AL BL theo từng bit.  
AND BL, 0FH     ; che 4-bit cao của BL.
```

#### 4.3.1.3 Điều khiển, rẽ nhánh và lặp.

Các câu lệnh thuộc nhóm này cho phép thay đổi trạng thái thực hiện các câu lệnh bên trong chương trình. Một số câu lệnh tiêu biểu được liệt kê trong **Error! Reference source not found.** dưới đây.

Bảng 4. 6. Các lệnh rẽ nhánh và lặp tiêu biểu

Mã gọi nhớ	Chức năng
JMP	Nhảy không điều kiện
JA (JNBE)	Nhảy nếu lớn hơn
JAE (JNB)	Nhảy nếu lớn hơn hoặc bằng
JB (JNAE)	Nhảy nếu bé hơn
JBE (JNA)	Nhảy nếu bé hơn hoặc bằng
JE (JZ)	Nhảy nếu bằng
JC, JNC	Nhảy nếu cờ nhớ đặt, xóa
JO, JNO	Nhảy nếu cờ tràn đặt, xóa
JS, JNS	Nhảy nếu cờ dấu đặt, xóa
LOOP	Lặp không điều kiện, số lần lặp do CX xác định
LOOPE (LOOPZ)	Lặp nếu bằng (cờ không) hoặc số lần lặp do CX xác định
LOOPNE (LOOPNZ)	Lặp nếu không bằng (cờ không xóa) hoặc số lần lặp do CX xác định

CALL, RET	Gọi hàm, trả về từ hàm con
INT	Ngắt mềm
IRET	Quay trở về từ đoạn chương trình ngắt

### JMP - Nhảy (vô điều kiện) đến một đích nào đó

Lệnh này khiến cho bộ vi xử lý 8086/8088 bắt đầu thực hiện một lệnh mới tại địa chỉ được mô tả trong lệnh. Lệnh này phân biệt nhảy xa và nhảy gần theo vị trí của câu lệnh mới. Tuỳ thuộc vào độ dài của bước nhảy chúng ta phân biệt các kiểu lệnh nhảy gần và nhảy xa với độ dài lệnh khác nhau. Lệnh nhảy đến nhãn ngắn *shortlabel* là lệnh nhảy tương đối. Nơi đến phải nằm trong phạm vi từ -128 đến +127 so với vị trí của lệnh nhảy. Toán hạng nguồn trong lệnh chỉ là byte độ dời để cộng thêm vào thanh ghi IP. Byte độ dời này được mở rộng dấu trước khi cộng vào thanh ghi IP.

- Ví dụ:

```
JMP SHORT 18h
JMP 0F008h
JMP DWORD PTR [3000h]
```

Lệnh này không tác động đến các cờ.

### LOOP - Lặp lại đoạn chương trình do nhãn chỉ ra cho đến khi CX=0

Cú pháp: *LOOP NHÃN*

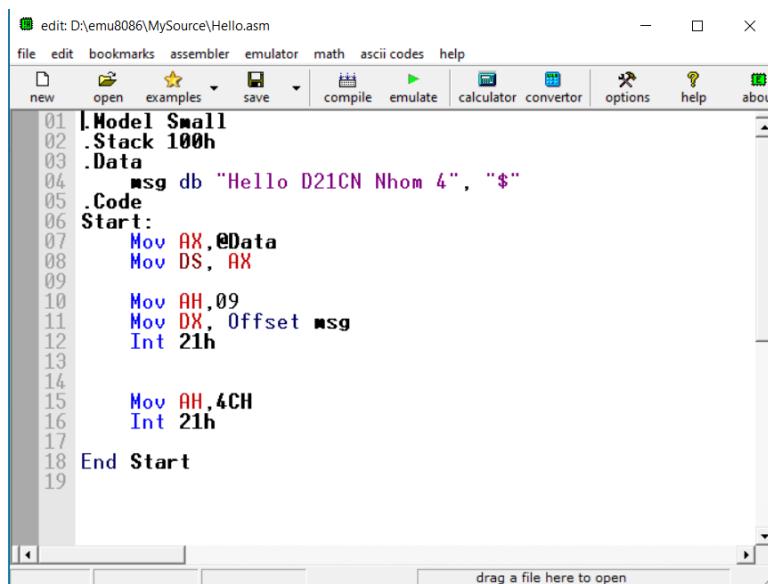
Lệnh này dùng để lặp lại đoạn chương trình (gồm các lệnh nằm trong khoảng từ nhãn NHÃN đến hết lệnh LOOP NHÃN) cho đến khi số lần lặp CX=0. Điều này có nghĩa là trước khi vào vòng lặp ta phải đưa số lần lặp mong muốn vào thanh ghi CX và sau mỗi lần thực hiện lệnh LOOP NHÃN thì đồng thời CX tự động giảm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

#### 4.3.2 Công cụ Emu8086

Hiện nay, đa số các máy tính chạy hệ điều hành 32-bit hoặc 64-bit như người lập trình sẽ gặp khó khăn khi gọi ngắt bằng chương trình người dùng được. Thay vì gọi ngắt trực tiếp, các hệ điều hành cung cấp một tập các hàm giao diện lập trình ứng dụng gọi là API (Application Programming Interface) cho phép người lập trình gọi hàm API. Để người mới học lập trình hệ thống có thể lập trình với các ngắt mà không bị giới hạn bởi phiên bản khác nhau của các hệ điều hành thì cách tốt nhất là học lập trình trên môi trường mô phỏng (ví dụ Emulator 8086). Phần này chúng tôi giới thiệu về phần mềm mô phỏng CPU 8086 của công ty phần mềm Emu8086 Inc., phiên bản 4.08. Các phiên bản mới hơn có thể được download tại địa chỉ của trang web: [www.emu8086.com](http://www.emu8086.com).

- **New:** tạo một chương trình mới, khi đó người dùng sẽ được hỏi xem sẽ tạo file chương trình dạng nào: COM, EXE, BIN hay BOOT.

- **Open:** mở một file chương trình nguồn hợp ngữ.
- **Samples:** Liệt kê các file chương trình mẫu có sẵn do chương trình mô phỏng cung cấp.
- **Save:** Lưu file chương trình nguồn
- **Compile:** dịch file chương trình nguồn
- **Emulate:** cho phép thực hiện chương trình nguồn.
- **Calculator:** người dùng có thể nhập 1 vào một biểu thức với các số là: có dấu, số dạng word hoặc số dạng byte để tính toán. Kết quả tính toán được hiển thị một trong các dạng số thập phân, nhị phân, hexa hoặc số bát phân (cơ số 8).
- **Convertor:** Bộ chuyển đổi giữa các cơ số. Emu8086 hỗ trợ chuyển đổi giữa các cơ số 16 thành cơ số 10 có dấu hoặc không dấu. Chuyển đổi từ cơ số 8 thành thành cơ số 2 (nhị phân). Một mã ASCII gồm 2 số hexa cũng có thể được chuyển đổi thành thập phân hoặc nhị phân.



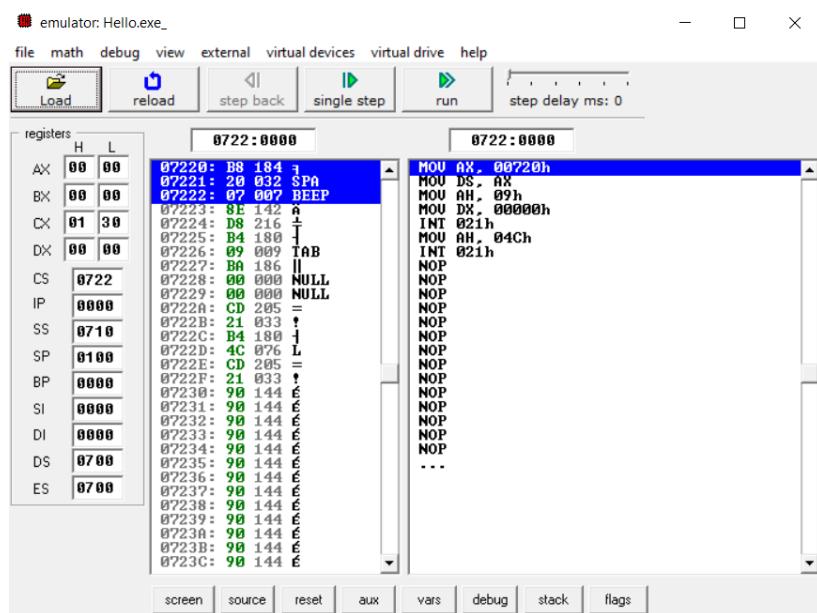
```

edit: D:\emu8086\MySource\Hello.asm
file edit bookmarks assembler emulator math ascii codes help
new open examples save compile emulate calculator convertor options help about
01 .Model Small
02 .Stack 100h
03 .Data
04     msg db "Hello D21CN Nhom 4", "$"
05 .Code
06 Start:
07     Mov AX, @Data
08     Mov DS, AX
09
10    Mov AH, 09
11    Mov DX, Offset msg
12    Int 21h
13
14
15    Mov AH, 4CH
16    Int 21h
17
18 End Start
19

```

Hình 4. 4. Màn hình soạn thảo chương trình nguồn của Emu806

Màn hình mô phỏng trạng thái thực hiện chương trình:



Hình 4. 5. Màn hình mô phỏng trạng thái chương trình

#### Các chức năng chính:

- **Load:** tải chương trình. Trước khi thực hiện thì chương trình sẽ được tải vào trong bộ nhớ. Chương trình có thể ở dạng các file thực hiện được như EXE, COM, BIN, BOOT hoặc dưới dạng file nguồn ASM.
- **Reload:** người dùng có thể tải lại 1 chương trình.
- **Single Step:** chạy chương trình theo chế độ từng lệnh. Với chế độ này, người dùng có thể quan sát trạng thái các thanh ghi, bộ nhớ trong.
- **Run:** chế độ chạy tất cả các lệnh trong chương trình. Trên màn hình, người dùng có thể quan sát trạng thái các thanh ghi và đoạn bộ nhớ sử dụng cho đoạn mã lệnh của chương trình. Phần registers mang nội dung của các thanh ghi trong đó các thanh ghi AX, BX, CX và DX được chia làm 2 nửa phần cao (H) và phần thấp (L). Ngoài ra, ta có thể xem nội dung các thanh ghi đoạn, con trỏ lệnh, ngăn xếp. Phần bộ nhớ lưu trữ đoạn mã chương trình. Địa chỉ đoạn (dạng hexa) được lưu trong thanh ghi CS. Danh sách địa chỉ offset được hiển thị dưới các dạng hexa và thập phân. Ngoài ra, người dùng có thể có thể xem:
  - Kết quả hiển thị lên màn hình (nhấp chuột vào nút User Screen).
  - Mã nguồn của chương trình (nhấp chuột vào nút Actual Source).
  - Trạng thái ALU (nhấp chuột vào nút ALU).
  - Nội dung của ngăn xếp (nhấp chuột vào nút Stack).
  - Nội dung của thanh ghi cờ (nhấp chuột vào nút FLAG)

## 4.4 Lập trình hợp ngữ

### 4.4.1 Chương trình hợp ngữ

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. Các dòng lệnh này có thể được viết 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.

### **Cấu trúc dòng lệnh:**

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 hạng      ;Chú giải

Một ví dụ dòng lệnh gợi nhớ:

TIEP:	MOV	AH, [BX] [SI]	; nạp vào AH 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 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.

#### **a) 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 nhãn 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. Một nhãn thường kết thúc bằng dấu hai chấm (:).

#### **b) 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.

#### **c) 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.

#### d) 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. 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.

#### Khai báo

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. Khi cung cấp số liệu cho chương trình, số cho ở hệ nào phải được kèm đuôi của hệ đó (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
1EF1H	; Số hệ mười sáu.

Nếu dữ liệu là ký tự hoặc chuỗi ký tự thì chúng phải được đóng trong cặp 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 có thể dùng '\$' hoặc 26H hoặc 34...

Biến trong chương trình hợp ngữ có vai trò như nó có ở ngôn ngữ bậc cao. Một biến phải 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

### a) Biến kiểu 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 ký tự như sau:

C1	DB	' \$ '
C2	DB	34

### b) Biến kiểu 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	?
----	----	---

### c) Biến kiểu 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 tổng 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 đặt trong bộ nhớ thì byte thấp của nó sẽ được đặt vào ô nhớ có địa chỉ thấp, byte cao sẽ được đặt vào 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 cắt tại địa chỉ thấp). Cũng nên nói thêm ở đây là các bộ vi xử lý của motorola lại có cách cắt số liệu theo thứ tự ngược lại hay còn được gọi là 'quy ước đầu to' (big endian byte cao được cắt tại địa chỉ thấp).

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

```
WORDA DW 0FEEH
```

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

#### d) 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 gán 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
```

#### e) 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 như sau:

```
CR EQU 0Dh
```

LF EQU 0Ah

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

#### 4.4.2 Lập trình hợp ngữ

##### a) Khung chương trình

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.

Khai báo kích thước 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, tùy theo nhu cầu cụ thể MASM còn cho phép sử dụng các Kiểu\_kích\_thước\_bộ\_nhớ khác như liệt kê trong **Error! Reference source not found..**

Bảng 4. 7. 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 ngắn xếp*

Việc khai báo đoạn ngắn xếp là để dành ra một vùng nhớ đủ lớn dùng làm ngắn 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 ngắn 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 ngắn 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 ngắn xếp và ta có thể khai báo kích thước như sau:

*. Stack 100*

### *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 họa 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.

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 trình EXE được nạp vào bộ nhớ. Hệ điều hành DOS sẽ tạo ra một mảng gồm 256-byte của cái gọi là *đoạn mào đầu chương trình* (Program Segment 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 ES nếu cần):

```

MOV AX, @Data
MOV DS, AX

```

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 8086/8088, 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.

### Ví dụ 1. Chương trình Hello world

```

.Model Small
.Stack 100
.Data
    CRLF        DB            13, 10, ' $ '
    MSG         DB            ' Hello World!$ '
.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
; trả 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 (hàm 4CH) của ngắt INT 21H của DOS để hiện thị xâu ký tự và trả về DOS.

### **b) Các cấu trúc lập trình cơ bản**

Ngày nay, trong khi tiến hành việc thiết kế hệ thống người ta thường dùng phương pháp *thiết kế từ trên xuống dưới*. 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 vụ 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 cài 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,

- + câu trúc lặp kiểu REPEAT-UNTIL và
- + câu trúc lặp kiểu FOR-DO.

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

#### *Câu trúc tuần tự:*

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

Ngữ pháp:

**Lệnh 1**

**Lệnh 2**

**Lệnh n**

**Bài tập 1:** Các thanh ghi CX và BX chứa các giá trị của biến c và b. Hãy tính giá trị của biểu thức  $a = 2 \times (c+b)$  và chép kết quả trong thanh ghi AX.

#### **Giải:**

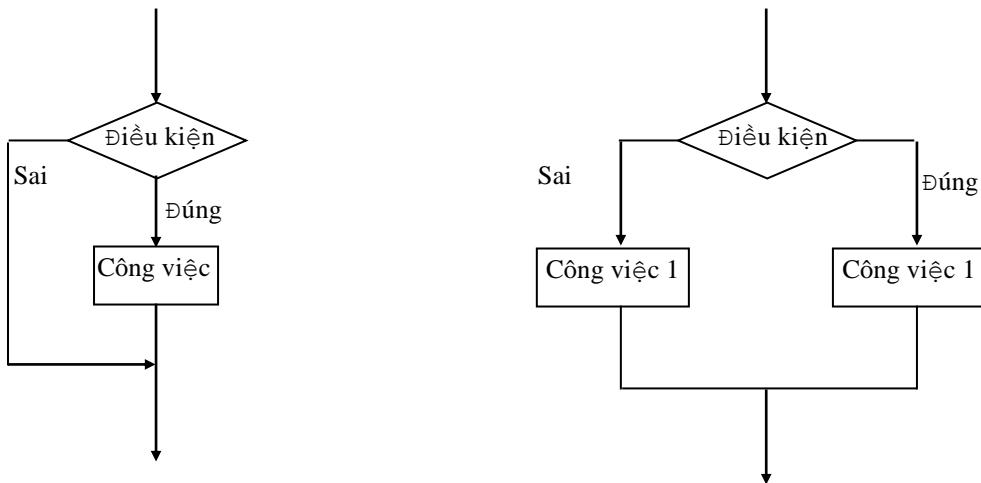
Ta có thể thực hiện công việc trên bằng mẫu chương trình sau:

XOR AX, AX	; tổng tại AX lúc đầu là 0.
ADD AX, BX	; cộng thêm b.
ADD AX, CX	; cộng thêm c.
SHL AX, 1	; nhân đôi kết quả trong AX.
RA:	; lối ra của câu trúc.

#### *Câu trúc IF-THEN:*

#### **IF Điều kiện THEN công việc.**

Từ ngữ pháp của câu trúc IF-THEN ta thấy nếu thỏa 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 nào đó trong chương trình hợp ngữ.



Hình 4.6. Cấu trúc IF THEN và IF THEN ELSE

**Bài tập 2:** Gán cho BX giá trị tuyệt đối của AX.

**Giải:**

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

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

Cấu trúc IF-THEN-ELSE:

**IF ĐiềuKiện THEN CôngViệc1 ELSE CôngViệc2**

Từ ngữ pháp của cấu trúc IF-THEN-ELSE ta thấy nếu thỏa 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.

**Bài tập 3.** Gán cho CL giá trị bit dấu của AX.

**Giải:**

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

CMP AX, 0	; AX>0?
JNS DG	; đúng.
MOV CL, 1	; sai, cho CL $\leftarrow 1$ rồi
JMP RA	; di ra.
DG : XOR CL, CL	; cho CL $\leftarrow 0$ .
RA :	; lối ra của cấu trúc.

Cấu trúc CASE:

## CASE Biểu thức

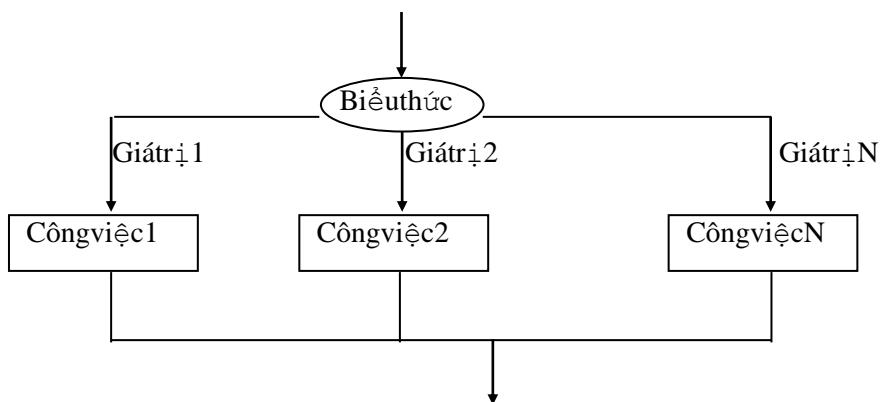
**Giá trị 1: Công việc 1**

**Giá trị 2: Công việc 2**

...

**Giá trị N: Công việc N**

END CASE



Hình 4. 7. 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 cứ tiếp tục cho đến CôngviệcN. Điều này tương đương với việc dùng các lệnh nhảy có điều kiện và nhảy không điều kiện để nhảy các nhãn nào đó trong chương trình hợp ngữ. Cấu trúc CASE có thể thực hiện bằng các cấu trúc lựa chọn lồng nhau.

**Bài tập 4.** Dùng CX để biểu hiện các giá trị khác nhau của AX theo quy tắc sau:

AX < 0 thì CX = -1

AX = 0 thì CX = 0

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:

```

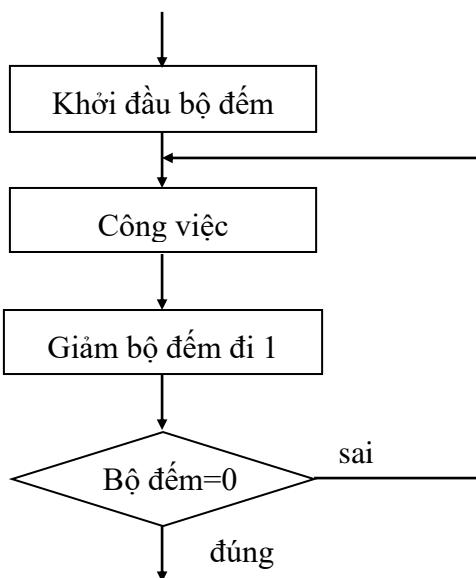
    MOV CX, -1
    JMP RA
    DUONG:
    MOV CX, 1
    JMP RA
    KHONG:
    XOR CX, CX
    RA: ; lối ra của cấu trúc.

```

### Cấu trúc FOR - DO

FOR Số lần lặp DO Công việc

Từ ngữ pháp của 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, trước đó ta phải gán Số lần lặp cho thanh ghi CX.



Hình 4. 8. Cấu trúc lặp FOR – DO.

**Bài tập 5:** 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 lệnh này, ngay từ đầu ta phải nạp vào

thanh ghi CX số lần hiển thị, nội dung của CX được tự động giảm đi 1 do tác động của lệnh LOOP.

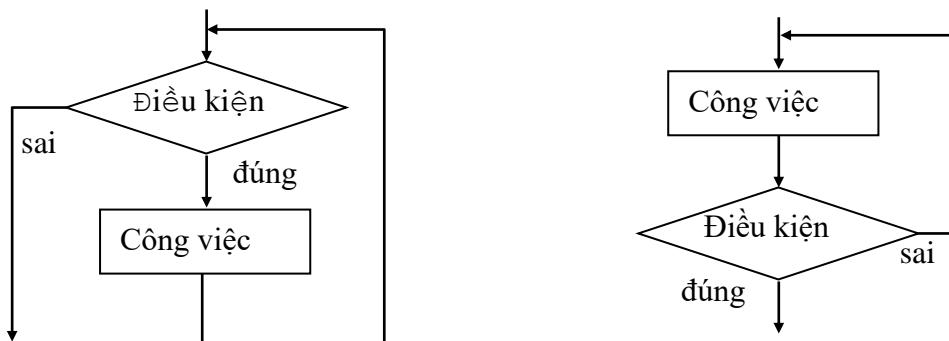
Sau đây là mẫu chương trình thực hiện các công việc trên:

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

### Cấu trúc WHILE - DO

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.



Hình 4. 9. Cấu trúc WHILE - DO và REPEAT-UNTIL

**Bài tập 6:** 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 mẫu 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 REPEAT – UNTIL:

## REPEAT Công việc UNTIL 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 Điều kiện và sau đó dùng lệnh nhảy có điều kiện để thoát khỏi vòng lặp.

**Bài tập 7:** 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 thực hiện công việc trên bằng mẫu chương trình sau:

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

### c) Một số ví dụ

Phần này trình bày một số chương trình ví dụ, 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.

#### Bảng 4. 8. Một số dịch vụ ngắt DOS

**Ngắt INT 20H** dành riêng để kết 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 0Ah của ngắt INT 21H:** nhập vào 1 chuỗi ký tự từ bàn phím

Vào: AH = 0Ah

DX = địa chỉ lệch của buffer chứa chuỗi ký tự

**Hàm 39h của ngắt INT 21H:** tạo một thư mục (folder)

Vào: AH = 39h

DX = địa chỉ lệch của chuỗi ký tự (tên thư mục)

Ra:

CF (Carry Flag) =0: nếu tạo thư mục thành công

CF=1: tạo thư mục không thành công và DX = mã lỗi

**Hàm 41h của ngắt INT 21H:** xóa một thư mục (folder)

#### Ví dụ 1

Trong phần đầu của chương trình hợp ngữ ta có giới thiệu một chương trình hiện lời chào bằng 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
    MOV AX, @ Data           ; khởi đầu thanh ghi DS
    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, CRLF
    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
    ; trả 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 gán giá trị khác cho thanh ghi CX.

### Ví dụ 2

Trên cơ sở ví dụ trước, ta phải viết chương trình sao cho số dòng 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 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 chương trình thực hiện công việc trên:

```

.Model Small
.Stack 100
.Data
    CRLF      DB      13, 10, '$'
    ChaoTay   DB      'Hello!$'
    ChaoTa    DB      'Chao ban!$'

```

```

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            ; cắt đì
    ; 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
    ; hiển thị lời chào dùng hàm 9 của INT 21H
    LEA DX, ChaoTa
    INT 21H
    ; trả về DOS dùng hàm 4CH của INT 21H
    MOV AH, 4CH
    INT 21H
MAIN Endp
END MAIN

```

Trong ví dụ trên có một điều cần chú ý là khi đọc một ký tự từ bàn 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 cắt 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 39H (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) ra giá trị số có thể thực hiện được bằng một trong các lệnh sau:

+ **SUB AL, 30H**

+ **AND AL, 0FH**

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

+ **ADD DL, 30H**

+ **OR DL, 30H**

### 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 bài 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 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?
    JE THOIDOC        ; đúng, thôi đọc
    AND AL, 0FH        ; không, đổi mã ASCII ra số
    SHL BX, 1          ; dịch trái BX 1 bit để lấy chỗ
    OR BL, AL          ; chèn bit vừa đọc vào kết quả
    JMP TIEP           ; đọc tiếp một ký tự
THOIDOC:MOV CX, 16 ; CX chứa số bit của BX
    MOV AH, 2          ; hàm hiện ký tự
```

```

HIEN:XOR DL, DL           ; xoá DL để chuẩn bị đổi
ROL BX, 1                 ; đưa bit MSB của BX sang CF
ADC DL, 30H               ; đổi giá trị bit đó ra ASCII
INT 21H                   ; hiển thị 1 bit của BX
LOOP HIEN                ; lặp lại cho đến hết
MOV AH, 4CH               ; trả 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 bit 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 của 2 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 bit của nó ta không nhất thiết phải gõ đủ 16 bit 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 bit 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 trình nguyên thủy của thanh ghi BX, nhưng sau khi hiện thị xong thì giá trị nguyên thủy 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-bit 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

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 một cách rất đơn giản bằng cách quay trái thanh ghi BX nguyên gốc đi 4-bit rồi sau đó quay tiếp thanh ghi BH đi 4 bit.

Sau đây là chương trình thực hiện công việc trên.

```
.Model Small
.Stack 100
.Code
MAIN Proc
    MOV CL, 4
    ROL BX, CL          ; quay BX đi 4 bit
    MOV CL, 4
    ROR BH, CL          ; tráo 4 bit thấp và cao của BH
    MOV AH, 4CH          ; trở về DOS
    INT 21H
MAIN Endp
END MAIN
```

### Ví dụ 5

Có một chuỗi ký tự thường trong bộ nhớ. Hãy tạo ra một chuỗi ký tự chữ hoa từ chuỗi trên rồi cắt chuỗi đó trong bộ nhớ.

### Giải

Ví dụ 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,
- Cắt 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 họa 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'
```

```

        Tbaο      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
    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                ; cắt vào chuỗi mới
    LOOP LAP                ; làm cho đến hết
    LEA DX, Tbaο             ; chuẩn bị hiện chuỗi mới
    MOV AH, 9
    INT 21H
    MOV AH, 4CH               ; về DOS
    INT 21H

MAIN Endp
END MAIN

```

### Ví dụ 6:

Viết chương trình hợp ngữ nhập vào một xâu kí tự từ bàn phím rồi in ra xâu đó

Giải:

Đây là bài tập yêu cầu người học vận dụng hàm 0Ah của ngắt 21h để nhập một xâu kí tự từ bàn phím và lưu lại xâu này trong một vùng đệm (buffer). Cấu trúc của vùng đệm như sau: byte đầu tiên của vùng đệm chứa độ dài lớn nhất của xâu, byte thứ 2 chứa độ dài thực của xâu. Kí tự đầu tiên của xâu được chứa từ byte thứ 3 trở đi. Tuy nhiên, thông thường thì người dùng kết thúc việc nhập bằng phím Enter có mã ASCII là 13 nhưng trên thực tế thì kí tự kết thúc xâu lại là kí tự “\$” nên chương trình phải xử lý thay thế kí tự Enter bằng kí tự “\$”. Việc này có thể được thực hiện bằng cách sau: lấy địa chỉ offset của xâu đệm cộng với nội dung byte thứ hai rồi cộng với 2 thì sẽ trả đến byte cuối cùng của xâu đang chứa mã ASCII của Enter (13) rồi thay thế mã này bởi kí tự kết thúc xâu là “\$”.

Chương trình hợp ngữ được viết như sau:

```

.Model Small
.Stack 100h
.Data
    XauIn db "Nhập xau: ", "$"
    XauOut db "Xau vua nhap: ", "$"

```

```

Xuongdong db 13,10,24h
Buffer db 100 dup(?) ; Khai bao buffer

.Code
Start:
    Mov AX,@Data
    Mov DS, AX
    Mov AH,9
    Mov DX, offset XauIn
    Int 21h
    Mov AH,0Ah
    Mov DX, offset Buffer
    Mov BX,DX ; BX va DX cung tro den Buffer
    Mov BYTE PTR[BX],100 ; Do dai lon nhat cua
    Int 21h
    Mov AH,9
    Mov DX, offset Xuongdong ; xuong dong va ve dau dong
    Int 21h
    Mov DX, offset XauOut ; Xau kq
    Int 21h
    Mov DX,BX ; BX,DX cung tro den Buffer
    Add BL,[BX+1] ; Cong vao do dai thuc cua xau vao BX
    Add BX,2 ; Tro den byte cuoi cung
    Mov BYTE PTR[BX],"$" ; Thay the byte cuoi cung boi $
    ; in xau len man hinh
    Mov AH,9
    Add DX,2 ; bo qua hai byte dau
    Int 21h
Mov AH,4CH
Int 21h
End Start

```

### Ví dụ 7:

Viết chương trình hợp ngữ tạo một thư mục với tên thư mục được nhập từ bàn phím. In ra thông báo “Tạo thành công” nếu thư mục được tạo thành công, hoặc in ra “Tạo không thành công” nếu thư mục được tạo không thành công.

Giải:

```

.Model Small
.Stack 100h
.Data
    TenThuMuc db "Nhap xau: ","$"
    TaoThanhCong db "Tạo thành công: ","$"
    TaoKhongThanhCong db "Tạo không thành công: ","$"
    Xuongdong db 13,10,24h
    Buffer db 100 dup(?) ; Khai bao buffer

.Code
Start:
    Mov AX,@Data

```

```

Mov DS, AX
Mov AH, 9
Mov DX, offset TenThuMuc
Int 21h
Mov AH, 0Ah
Mov DX, offset Buffer
Mov BX,DX ; BX va DX cung tro den Buffer
Mov BYTE PTR[BX],100 ; Do dai lon nhat cua
Int 21h
Mov AH, 9
Mov DX, offset Xuongdong ; xuong dong va ve dau dong
Int 21h
    Mov DX,BX ; BX,DX cung tro den Buffer
    Add BL,[BX+1] ; Cong vao do dai thuc cua xau vao BX
    Add BX,2 ; Tro den byte cuoi cung
    Mov BYTE PTR[BX],"$" ; Thay the byte cuoi cung boi $
    ; tao thu muc
    Add DX,2
    Mov AH,39H
    Int 21H
    ; in thong bao len man hinh
    Mov AH,9
    JC ThanhCong
    Mov DX, Offset TaoKhongThanhCong
    Int 21h
    Jmp KetThuc

ThanhCong:
    Mov DX, Offset TaoThanhCong
    Int 21h

KetThuc:

Mov AH,4CH
Int 21h
End Start

```

## 4.5 Kết luận chương

Chương 4 đã trình bày chi tiết về lập trình hợp ngữ với bộ vi xử lý 8086/8088, từ kiến trúc, các thành phần chính, mã hóa lệnh, chế độ địa chỉ, tập lệnh đến cách lập trình hợp ngữ cơ bản. Người học được tìm hiểu cấu trúc vi xử lý với hai khối chính BIU và EU, cách hoạt động của các thanh ghi, mã hóa lệnh, cùng 7 chế độ địa chỉ khác nhau, từ đơn giản đến phức tạp. Tập lệnh 8086/8088 đã được minh họa qua các nhóm lệnh cơ bản và ứng dụng, đồng thời công cụ mô phỏng Emu8086 được giới thiệu để hỗ trợ thực hành hiệu quả. Qua đó, chương đã cung cấp kiến thức về cấu trúc tổng quát của một chương trình hợp ngữ và các cấu trúc lập trình cơ bản như tuần tự, rẽ nhánh, và lặp, với các ví dụ minh họa cụ thể để giải quyết bài toán thực tế. Kiến

thức này giúp người học hiểu sâu hơn về hoạt động của vi xử lý, làm nền tảng để nghiên cứu các hệ thống hiện đại, ứng dụng trong thiết kế hệ thống nhúng, lập trình hệ thống và tối ưu hóa phần mềm.

#### 4.6 Câu hỏi ôn tập

Viết các chương trình hợp ngữ giải quyết các vấn đề sau:

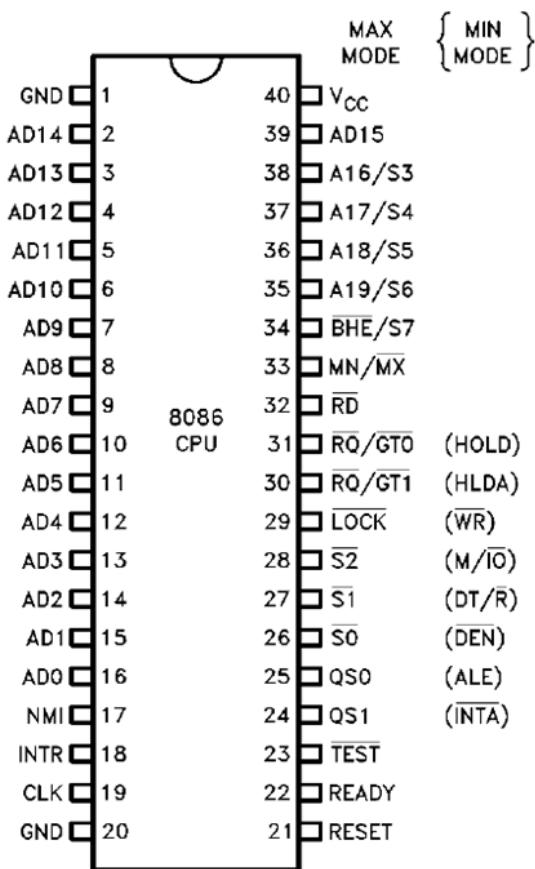
1. In ra 26 ký tự “A”...”Z”.
2. In ra bảng mã ASCII với 2 ký tự liền nhau cách nhau bởi 1 dấu cách
3. Nhập vào 2 số tự nhiên N ( $0 \leq N \leq 9$ ) hãy tính tổng, hiệu, tích, thương và in ra kết quả.
4. Nhập vào số tự nhiên N ( $0 \leq N \leq 99$ ), tính tổng N số tự nhiên đầu tiên rồi in ra kết quả
5. Nhập vào số tự nhiên N ( $0 \leq N \leq 9$ ), tính  $N!$  Và in ra kết quả.
6. Nhập vào 1 xâu ký tự, đổi tất cả các ký tự thành in hoa rồi in ra xâu đó
7. Tạo 1 thư mục với tên thư mục được nhập từ bàn phím.
8. (\*) Đọc một chuỗi ký tự từ bàn phím, kiểm tra xem chuỗi đó có phải là chuỗi palindrome (đối xứng) hay không.
9. (\*) Nhập vào một danh sách 10 số tự nhiên, sắp xếp danh sách đó theo thứ tự tăng dần và in ra kết quả.
10. (\*) Nhập vào một chuỗi ký tự, đếm số lần xuất hiện của mỗi ký tự trong chuỗi và in ra kết quả.
11. (\*) Viết chương trình thực hiện phép nhân hai số lớn (mỗi số có tối đa 10 chữ số) và in ra kết quả.
12. (\*) Viết chương trình thực hiện tính tổng và trung bình của một dãy số nhập từ bàn phím, dừng khi nhập vào ký tự đặc biệt (ví dụ: '#').
13. (\*) Viết chương trình để đảo ngược một chuỗi ký tự nhập vào từ bàn phím và in ra chuỗi đảo ngược.
14. (\*) Viết chương trình tạo một tập tin văn bản và ghi một chuỗi ký tự vào tập tin đó.
15. (\*) Xóa 1 thư mục với tên thư mục được nhập từ bàn phím.

## CHƯƠNG 5 PHỐI GHÉP VÀ LẬP TRÌNH ĐIỀU KHIỂN THIẾT BỊ

Trong hệ thống vi xử lý, việc phối ghép và lập trình điều khiển thiết bị đóng vai trò quan trọng trong thiết kế và vận hành. Chương này giới thiệu nguyên tắc phối ghép vi xử lý 8086/8088 với bộ nhớ và thiết bị ngoại vi, bao gồm giải mã địa chỉ, truy xuất bộ nhớ, và giao tiếp với các thiết bị như bàn phím, đèn LED, cảm biến. Đồng thời, các phương pháp vào/ra dữ liệu, xử lý ngắn và DMA cũng được đề cập, giúp người học nắm vững cách lập trình và ứng dụng trong thực tế.

### 5.1 Phối ghép CPU với bộ nhớ

#### 5.1.1 Các tín hiệu của CPU 8086/8088



Hình 5. 1. Chân tín hiệu vi xử lý 8086

Hình 2. 1. trên cho chúng ta thấy tín hiệu của 8086. Các tín hiệu của 8086 và 8088 chỉ khác ở số lượng kênh dữ liệu. 8086 có 16 đường dữ liệu trong khi 8088 chỉ có 8 đường dữ liệu. Dưới đây trình bày ý nghĩa các tín hiệu của 8086,

Chức năng các tín hiệu tại các chân cụ thể như sau:

- ADO – AD15 [I;O: tín hiệu vào và ra]: Các chân dòn kênh cho các tín hiệu buýt dữ liệu và buýt địa chỉ. Xung ALE sẽ báo cho mạch ngoài biết khi nào trên các đường đó có tín hiệu dữ liệu (ALE = 0) hoặc địa chỉ (ALE = 1). Các chân này ở trạng thái trở kháng cao khi  $\mu P$  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 được truyền trên các chân đó khi ALE = 1 còn khi ALE = 0 thì trên các chân đó có các tín hiệu trạng thái S3 - S6. Các chân này ở trạng thái trở kháng cao khi  $\mu P$  chấp nhận treo. Việc kết hợp S3 và S4 để biểu diễn truy nhập các thanh ghi như trong Hình 2. 1. bảng dưới đây.

Bảng 5. 1. 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

- $\overline{RD}$  [O]: Xung cho phép đọc. Khi  $\overline{RD} = 0$  thì buýt dữ liệu sẵn sàng nhận số liệu từ bộ nhớ hoặc thiết bị ngoại vi. Chân  $\overline{RD}$  ở trạng thái trở kháng cao khi  $\mu P$  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. Lúc này CPU tự kéo dài thời gian thực hiện lệnh ghi/đọc bằng cách chèn thêm các chu kỳ đợi.
- INTR [I]: Tín hiệu yêu cầu ngắt che được. Khi có yêu cầu ngắt mà cờ cho phép ngắt IF = 1 thì CPU kết thúc lệnh đang làm dở, sau đó 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 sùm 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 8086/8088. khi RESET = 1 kéo dài ít nhất trong thời gian 4 chu kỳ đồng hồ thì 8086/8088 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 $\leftarrow 0$  để cảm các yêu cầu ngắt khác tác động vào CPU và cờ TF $\leftarrow 0$  để bộ vi xử lý không - không bị đặt trong chế độ chạy từng lệnh).
- CLK [I]: Tín hiệu đồng hồ (xung nhịp). Xung nhịp có độ rộng là 77% và cung cấp nhịp làm việc cho CPU.
- Vcc [I]: Chân nguồn. Tại đây CPU được cung cấp  $5V \pm 10\%$ . 340mA

- GND [O]: Hai chân nguồn để nối với điểm 0V của nguồn nuôi.
- MN/MX [I]: Chân điều khiển hoạt động của CPU theo chế độ MIN/MAX. Do 8086/8088 có thể làm việc ở 2 chế độ khác nhau nên có một số chân tín hiệu phụ thuộc vào các chế độ đó.

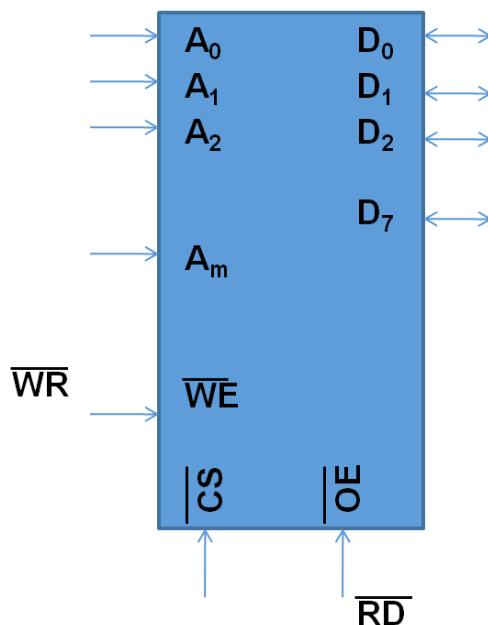
### 5.1.2 Phối ghép CPU với bộ nhớ

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 (Erasable 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 truy nhập ngẫu nhiên) 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 (Static RAM hay SRAM, trong đó mỗi phần tử nhớ là một mạch lật hay trạng thái ổn định) và loại RAM động (Dynamic RAM hay DRAM, trong đó mỗi phần tử nhớ là một tụ điện rất nhỏ được chế tạo bằng công nghệ MOS).

#### a) Cấu trúc vi mạch nhớ

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 Hình 2. 1. sau.



Hình 5. 2. Sơ đồ khối tổng quát mạch

Theo sơ đồ khối này ta thấy một 1 vi mạch nhớ có các nhóm tín hiệu sau:

- a) Nhóm tín hiệu địa chỉ:

Các tín hiệu địa chỉ có tác dụng chọn ra một ô nhớ. Các ô nhớ có độ dài khác nhau (còn gọi là từ nhớ) 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 mạch nhớ đó là  $2^m$  từ nhớ. Ví dụ, với m = 10 ta có dung lượng mạch nhớ là 1K ô nhớ (1 kilô =  $2^{10} = 1024$ ) và với m=20 ta có dung lượng mạch nhớ là 1M ô nhớ (1 Mêga =  $2^{20} = 1048576$ ).

b) 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. Ngoài ra có mạch nhớ RAM với đầu ra và đầu vào dữ liệu riêng biệt. Các mạch nhớ thường có đầu ra dữ liệu kiểu 3 trạng thái. Số đường 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...

c) Nhóm tín hiệu chọn vi mạch (chọn vỏ):

Các tín hiệu chọn vi mạch 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 vi mạch ở các mạch RAM thường là  $\overline{CS}$ , còn ở mạch ROM thường là  $\overline{CE}$ . Các tín hiệu chọn vi mạch thường được nối với đầu ra của bộ giải mã địa chỉ. Khi một mạch nhớ không được chọn thì buýt dữ liệu của nó bị treo (ở trạng thái trở kháng cao)

d) 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 buýt. Một mặt mạch nhớ không được mở bởi  $\overline{OE}$  thì buýt 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 loại trừ lẫn nhau (ngược pha) để điều khiển việc ghi/đọc mạch nhớ.

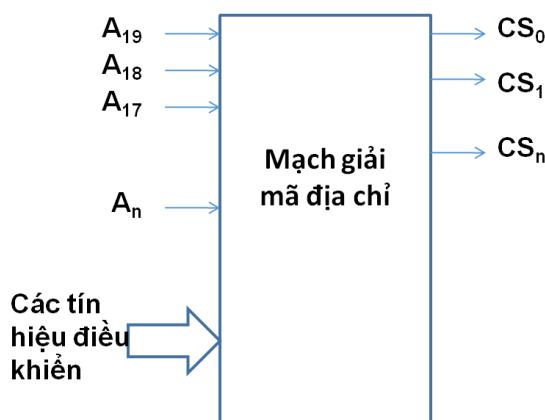
Một thông số đặc trưng khác của bộ nhớ là thời gian truy nhập  $t_{ac}$  được định nghĩa như là thời gian kể từ khi có xung địa chỉ trên buýt địa chỉ cho đến khi có dữ liệu ra ổn định trên buýt dữ liệu. Thời gian truy cập bộ nhớ phụ thuộc rất nhiều vào công nghệ chế tạo nên nó. Các bộ nhớ làm bằng công nghệ lưỡng cực có thời gian truy nhập nhỏ (10 - 30ns) còn các bộ nhớ làm bằng công nghệ MOS có thời gian truy nhập lớn hơn nhiều (cỡ 150ms hoặc hơn nữa).

### 5.1.3 Giải mã địa chỉ cho bộ nhớ

Mỗi mạch nhớ nối ghép với CPU cần phải được CPU tham chiếu chính xác khi thực 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 vi mạch lấy từ mạch giải mã địa chỉ. Việc phân định không gian địa chỉ tổng thể thành các vùng nhớ khác nhau để thực hiện những chức năng nhất định gọi là phân vùng bộ nhớ.

Ví dụ, đối với CPU 8086/8088 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 đây là bảng gồm 256 vectơ ngắt của 8086/8088), tại còn vùng nhớ có chia địa chỉ FFFF0H nhất thiết phải dành cho ROM hoặc EPROM (vì FFFF0H là địa chỉ bắt đầu của đoạn mã khởi động của CPU).

Về nguyên tắc một bộ giải mã địa chỉ khái quát thường có cấu tạo như trên **Error! Reference source not found.** dưới đây. 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 với các tín hiệu chọn vỏ ở đầu ra. Thường là các tín hiệu địa chỉ tương ứng với dải địa chỉ cấp cho vi mạch nhớ sẽ sinh ra tín hiệu chọn vi mạch tương ứng. Tín hiệu điều khiển thường là tín hiệu  $IO/M$  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 tăng thêm trễ thời gian của tín hiệu từ CPU tới bộ nhớ hoặc thiết bị ngoại vi. Tuỳ theo quy mô của mạch giải mã mà ta có thể có ở đầu ra một hay nhiều tín hiệu chọn vỏ.



Hình 5. 3. Mạch giải mã địa chỉ tổng quát

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 đầy 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 chọn ra các mạch nhớ ở các vùng nhớ khác nữa. Nói cách khác, từ một tổ hợp tín hiệu địa chỉ, bộ giải mã sẽ chỉ sinh ra một tín hiệu chọn vỏ duy nhất ứng với không gian địa chỉ cấp cho vi mạch nhớ.

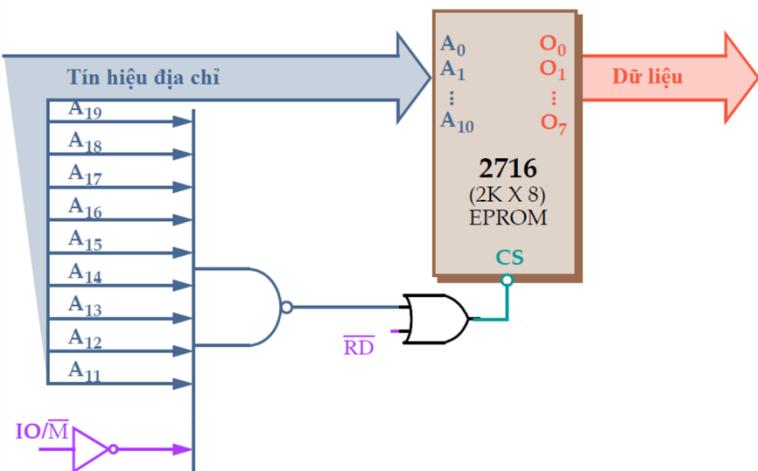
Giải mã địa chỉ thiểu hay giải mã rút gọn thì ta chỉ dùng một nhóm trong số các tín hiệu địa chỉ để sinh ra tín hiệu chọn vỏ cho mạch nhớ. Như vậy, từ một tổ hợp các tín hiệu địa chỉ có thể sinh ra nhiều tín hiệu chọn vỏ khác nhau. Vì sử dụng ít tín hiệu hơn nên mạch giải mã thiểu cần ít linh kiện hơn nhưng lại làm mất tính đơn trị của xung chọn thu được ở đầu ra.

Ví dụ: Chip nhớ C có dung lượng 10000H ô nhớ và được gán cho dải địa chỉ từ 00000H-0FFFFH. Để sinh ra tín hiệu chọn vỏ cho C ta có thể sử dụng tín hiệu địa chỉ A<sub>16</sub> ở mức thấp (A<sub>16</sub>=0) hoặc cả bốn tín hiệu A<sub>16</sub>-A<sub>19</sub> ở mức thấp (A<sub>16</sub>=...=A<sub>19</sub>=0). Với trường hợp thứ nhất ta có giải mã thiếu do A<sub>16</sub>=0 có thể do các yêu cầu truy nhập tới dải địa chỉ 20000H-2FFFFH.

Thông thường khi thiết kế mạch giải mã người ta hay tính đôi ra một chút để dự phòng, để có thể dễ dàng mở rộng dung lượng bộ nhớ sau này thì vẫn có thể sử dụng được mạch giải mã đã được thiết kế. Nói cách khác, hệ thống có thể mở rộng thêm không gian nhớ bằng các bổ sung thêm các vi mạch nhớ. Phần dưới đây sẽ xem xét một số phương pháp thực hiện mạch giải mã địa chỉ bộ nhớ.

#### 5.1.4 Giải mã địa chỉ cho bộ nhớ sử dụng mạch lô gic cơ bản

Các mạch lô-gíc đơn giản bao gồm các mạch AND, OR, NOT hay kết hợp như NAND, NOR. Bằng các mạch kiểu này ta có thể xây dựng được mạch giải mã địa chỉ đơn giản với số đầu ra hạn chế. Các mạch lô-gíc làm nhiệm vụ tổ hợp các tín hiệu địa chỉ và điều khiển đọc/ghi bộ nhớ sao cho với một tổ hợp địa chỉ cho trước sẽ sinh ra tín hiệu chọn vỏ tương ứng.



Hình 5. 4. Mạch giải mã dùng mạch logic

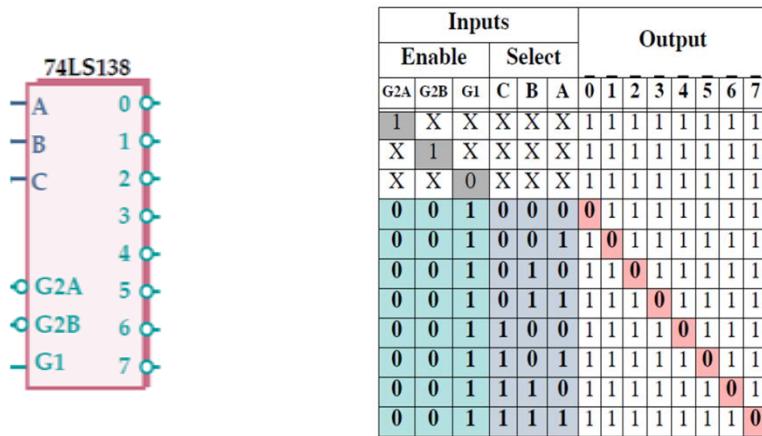
**Error! Reference source not found.** giới thiệu mạch giải mã cho mạch EEPROM 2716 có dung lượng 2K ô nhớ mỗi ô chứa 8 bit, làm việc trong dải địa chỉ FF800H-FFFFFH. Do mạch nhớ có dung lượng 2K tương ứng với dải địa chỉ 0FFH-7FFH (tương ứng với A<sub>0</sub>...A<sub>10</sub>). Như vậy, số lượng các tín hiệu địa chỉ dùng sinh ra tín hiệu kích hoạt chip nhớ này là A<sub>11</sub>-A<sub>19</sub>. Với dải địa chỉ cho trước FF800H-FFFFFH thì tổ hợp A<sub>11</sub>=...=A<sub>19</sub>=1 sẽ sinh ra tín hiệu chọn vỏ cho EEPROM 2716. Bên cạnh đó, ta cần phối hợp với các tín hiệu điều khiển IO/  $\overline{M}$  và RD (ở mức thấp) để tạo ra tín hiệu chọn vỏ.

Như trong hình vẽ, các tín hiệu địa chỉ và tín hiệu đảo của IO/  $\overline{M}$  được liên kết trực tiếp với nhau bằng phép lô-gíc AND rồi đảo. Do tín chất của mạch AND kết quả tổ hợp là duy nhất. Đầu ra sẽ chỉ bằng 1 khi tất cả đầu vào bằng 1. Đầu ra của mạch NAND được OR với RD (mức thấp) để sinh ra tín hiệu chọn vỏ (kích hoạt). Tương tự,

do tính chất của mạch OR đầu ra sẽ chỉ bằng 0 nếu tất cả các đầu vào bằng 0 nên tín hiệu chọn vỏ là tín hiệu duy nhất được sinh ra ứng với thao tác truy nhập tới dải địa chỉ FF800H-FFFFFH. Như vậy, mạch giải mã trên là mạch giải mã đầy đủ.

### 5.1.5 Giải mã địa chỉ cho bộ nhớ sử dụng mạch tích hợp 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 tăng lên. Trong trường hợp như vậy ta thường sử dụng các mạch giải mã tích hợp có sẵn. Một trong các mạch giải mã hay được sử dụng là 74LS138 cho phép giải mã 3 tín hiệu đầu vào thành 8 tín hiệu đầu ra như trong Hình 2. 1. dưới đây.



Hình 5. 5. 74LS138 và bảng trạng thái

Giả sử chúng ta cần xây dựng mạch giải mã cho không gian nhớ 256KB tương ứng với dải địa chỉ F8000H-FFFFFH trong đó mỗi mạch nhớ 2732 có dung lượng  $4K \times 8$ . Từ dải địa chỉ được gán và dung lượng của từng mạch nhớ, có thể thấy rằng từ các tín hiệu địa chỉ  $A_{13}$  tới  $A_{19}$  cần phải sinh ra 8 tín hiệu kích hoạt các vi mạch nhớ ứng với 8 dải địa chỉ như Hình 2. 1. dưới đây:

Bảng 5. 2. Dải tín hiệu của các mạch nhớ 2732

Địa chỉ	$A_{19}-A_{16}$	$A_{15}$	$A_{14}$	$A_{13}$	$A_{12}$
F8	1111	1	0	0	0
F9	1111	1	0	0	1
FA	1111	1	0	1	0
FB	1111	1	0	1	1
FC	1111	1	1	0	0
FD	1111	1	1	0	1
FE	1111	1	1	1	0

FF	1111	1	1	1	1
----	------	---	---	---	---

Qua bảng trên, ta thấy chỉ có các tín hiệu A<sub>12</sub>-A<sub>14</sub> là thay đổi còn A<sub>15</sub>-A<sub>19</sub> bằng 1 và không đổi. Như vậy ta có thể sử dụng mạch giải mã 74LS138 để sinh ra các tín hiệu chọn vỏ cho các mạch nhớ như hình sau:

Trong hình vẽ dưới đây, các tín hiệu A<sub>12</sub>-A<sub>14</sub> được nối trực tiếp vào tín hiệu đầu vào (A-C) của 74LS138. Các tín hiệu địa chỉ còn lại A<sub>15</sub>-A<sub>19</sub> và các tín hiệu điều khiển IO/  $\bar{M}$  được nối vào tín hiệu điều khiển của 74LS138 (G2A, G2B). Tín hiệu G1 luôn ở mức lô-gíc 1. Các đầu ra của 74LS138 được nối lần lượt với các mạch nhớ ứng với dải địa chỉ gán trước.

Tại ví 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 (programmable 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 hiện đại với không gian địa chỉ lớn.

### 5.1.6 Giải mã địa chỉ cho bộ nhớ sử dụng mạch PROM

Việc sử dụng bộ nhớ ROM làm bộ giải mã lợi dụng số lượng lớn các tín hiệu địa chỉ đầu vào, điều khiển và dữ liệu ra của mạch nhớ ROM. Với mỗi tổ hợp tín hiệu địa chỉ và điều khiển đầu vào, mạch nhớ ROM sẽ sinh ra một nhóm tín hiệu trên kênh dữ liệu. Trạng thái của các tín hiệu dữ liệu này tùy thuộc vào giá trị được lưu vào trong ROM trước đó. Nếu các tín hiệu này loại trừ lẫn nhau thì các tín hiệu dữ liệu có thể được dùng làm các tín hiệu chọn vi mạch nhớ.

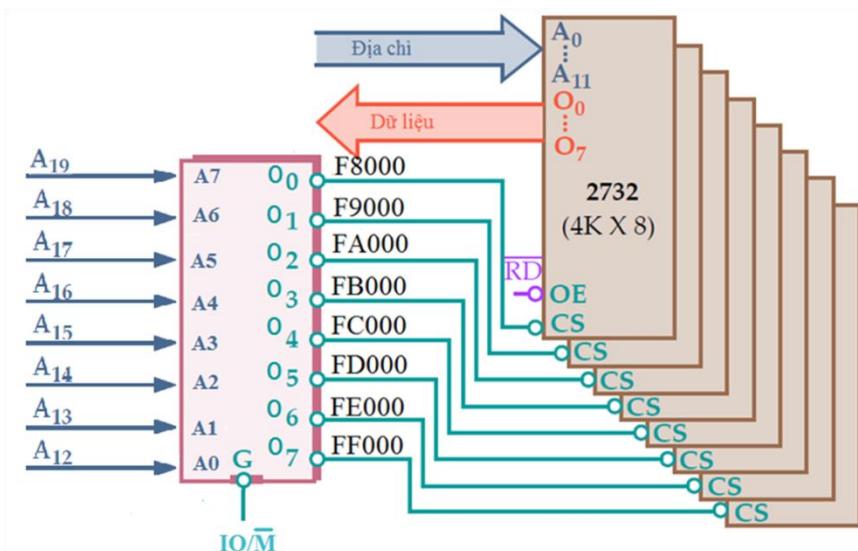
Để trình bày ứng dụng của PROM trong việc thực hiện các bộ giải mã ta lấy lại ví dụ phân vùng bộ nhớ cho ROM trong phần trước. Tại đây ta dùng mạch PROM 256-byte để làm bộ giải mã. Trong Hình 2. 1. dưới đây là mẫu các bit để ghi vào PROM cho trường hợp ứng dụng cụ thể này.

Bảng 5. 3. Mẫu dữ liệu ghi vào ROM

A <sub>7</sub>	A <sub>6</sub>	A <sub>5</sub>	A <sub>4</sub>	A <sub>3</sub>	A <sub>2</sub>	A <sub>1</sub>	A <sub>0</sub>	O <sub>0</sub>	O <sub>1</sub>	O <sub>2</sub>	O <sub>3</sub>	O <sub>4</sub>	O <sub>5</sub>	O <sub>6</sub>	O <sub>7</sub>
1	1	1	1	1	0	0	0	0	1	1	1	1	1	1	1
1	1	1	1	1	0	0	1	1	0	1	1	1	1	1	1
1	1	1	1	1	0	1	0	1	1	0	1	1	1	1	1
1	1	1	1	1	0	1	1	1	1	1	0	1	1	1	1
1	1	1	1	1	1	0	0	1	1	1	1	0	1	1	1
1	1	1	1	1	1	0	1	1	1	1	1	1	0	1	1
1	1	1	1	1	1	1	0	1	1	1	1	1	1	0	1

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Theo bảng trên, trong dải địa chỉ từ F8H-FFH của ROM ta ghi 8 giá trị sao cho tín hiệu dữ liệu đầu ra chỉ có duy nhất một tín hiệu mức thấp còn tất cả các tín hiệu còn lại đều ở mức cao. Ngoài 8 ô nhớ này, tất cả các ô nhớ khác của ROM đều được điền giá trị FFH.



Hình 5. 6. Giải mã dùng ROM

Mạch giải mã cho bộ nhớ PROM được thể hiện trên Hình 2. 1. trên so với cách thực hiện bộ giải mã bằng 74LS138 chúng ta không phải dùng đến các mạch phụ điều này làm giảm đáng kể kích thước vật lý của bộ giải mã. Ngoài ra ta có thể dễ dàng thay đổi địa chỉ của các mạch nhớ bằng cách thay đổi vị trí và giá trị dữ liệu trong mạch nhớ giải mã ROM.

## 5.2 Phối ghép CPU với thiết bị ngoại vi

### 5.2.1 Phối ghép CPU với thiết bị ngoại vi

Đối với 8086/8088 (hay họ 80x86 nói chung) có 2 cách phối ghép CPU với các thiết bị ngoại vi (các cổng vào/ra, I/O):

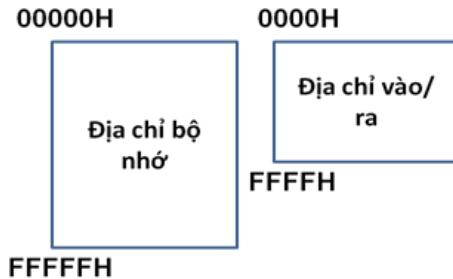
a) Thiết bị vào/ra có không gian địa chỉ tách biệt (xem Hình 2. 1.)

Trong cách phối ghép này, bộ nhớ được dùng toàn bộ 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. Để phân biệt các thao tác truy nhập, ta phải dùng tín hiệu  $IO/\bar{M} = 1$ , và các lệnh trao đổi dữ liệu phù hợp với từng không gian địa chỉ. Với các thiết bị này cần sử dụng các câu lệnh IN, OUT để trao đổi dữ liệu.

b) 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 8086/8088 có khả năng địa chỉ hóa. Các thiết bị ngoại vi sẽ chiếm một phần không gian trong 1MB, phần còn lại là của bộ nhớ. Tất nhiên trong

trường hợp này ta dùng chung tín hiệu  $\overline{M} = 0$  và lệnh trao đổi dữ liệu kiểu lệnh MOV cho cả bộ nhớ và thiết bị ngoại vi

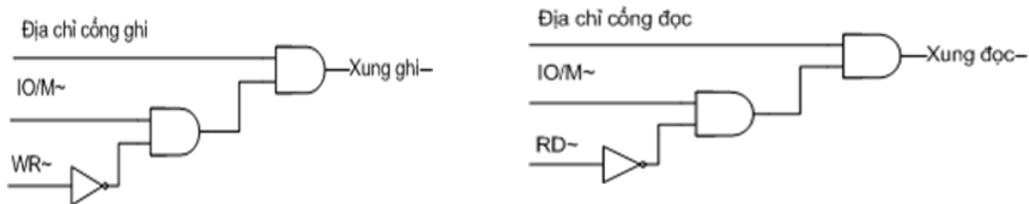


Hình 5. 7. Không gian nhớ của thiết bị vào/ra và bộ nhớ chính

### 5.2.2 Phối ghép CPU với thiết bị ngoại vi

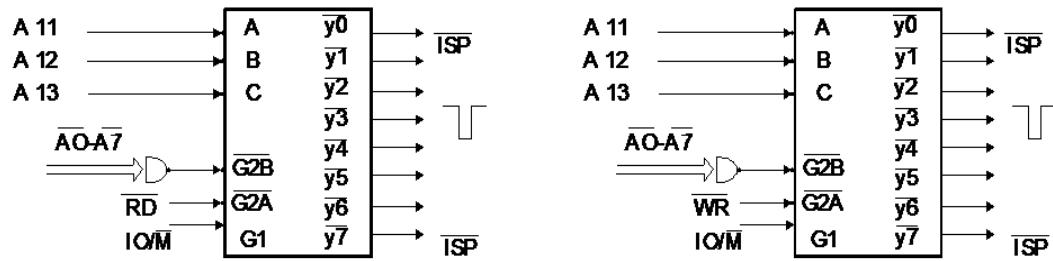
**Error! Reference source not found.** minh họa cách giải mã thiết bị ngoại vi sử dụng mạch logic đơn giản. Việc giải mã địa chỉ cho thiết bị vào/ra cũng gần giống như giải mã địa chỉ cho mạch nhớ. Ta sẽ nhấn mạnh ở đây việc giải mã địa chỉ cho các cổng. Thông thường, các cổng có địa chỉ 8-bit trong khoảng 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 bits ta 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 (như 74LS138) để tạo ra các xung chọn thiết bị.

Các mạch giải mã đơn giản có thể tạo được từ mạch lô-gíc đơn giản như sau:



Hình 5. 8. Giải mã thiết bị dùng cổng lô-gíc

**Error! Reference source not found.** minh họa cách sử dụng mạch giải mã địa chỉ cổng dựa trên IC 74LS138 để tạo tín hiệu chọn cho các cổng vào/ra có địa chỉ liên tiếp. Trong trường hợp cần nhiều 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. Như trên hình dưới đây 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.

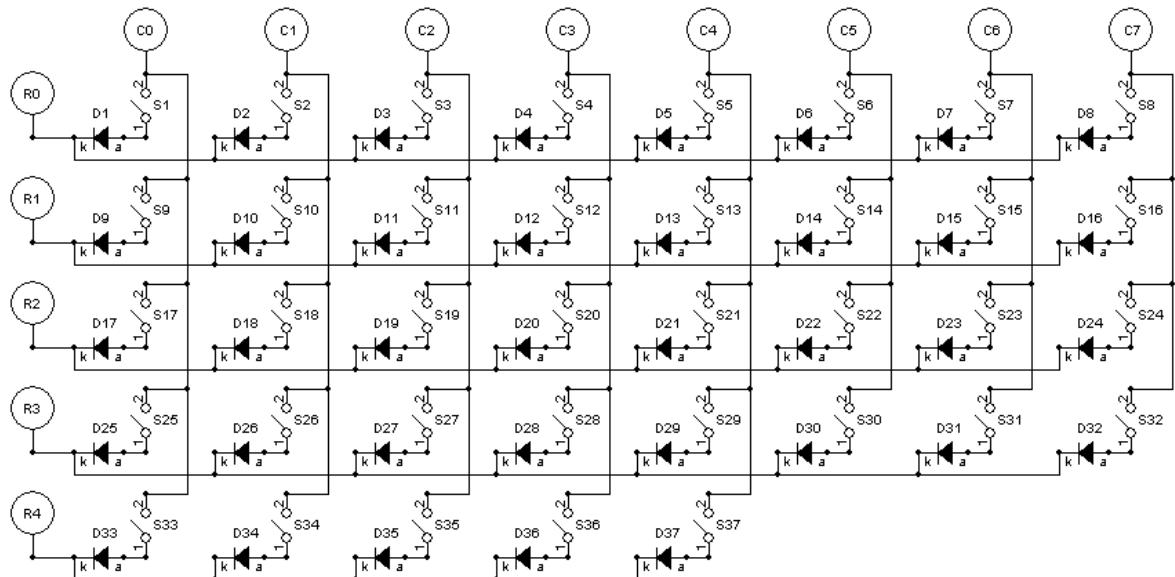


Hình 5. 9. Giải mã địa chỉ cỗng dùng 74LS138

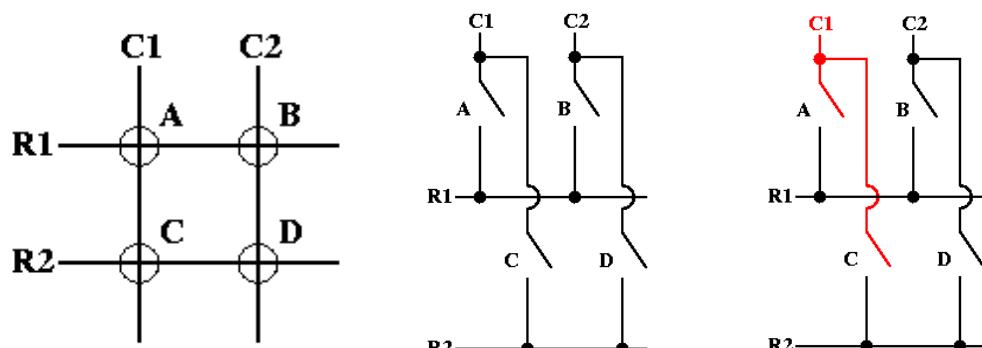
### 5.3 Lập trình điều khiển bàn phím

#### 5.3.1 Cơ bản về bàn phím

Bàn phím (keyboard) là thiết bị vào vào chuẩn của máy tính do bàn phím có thể đảm nhiệm cả chức năng nhập dữ liệu và điều khiển máy tính. Bàn phím tiêu chuẩn có 101 phím: các phím ký tự (a-z), các phím số (0-9), các phím phép toán (+, -, \*, /), các phím chức năng (F1-F12), các phím điều khiển (Ctrl, Alt, Shift, ..) và các phím di chuyển: Home, End, Page Up, Page Down, Up, Down, Left, Right, ...



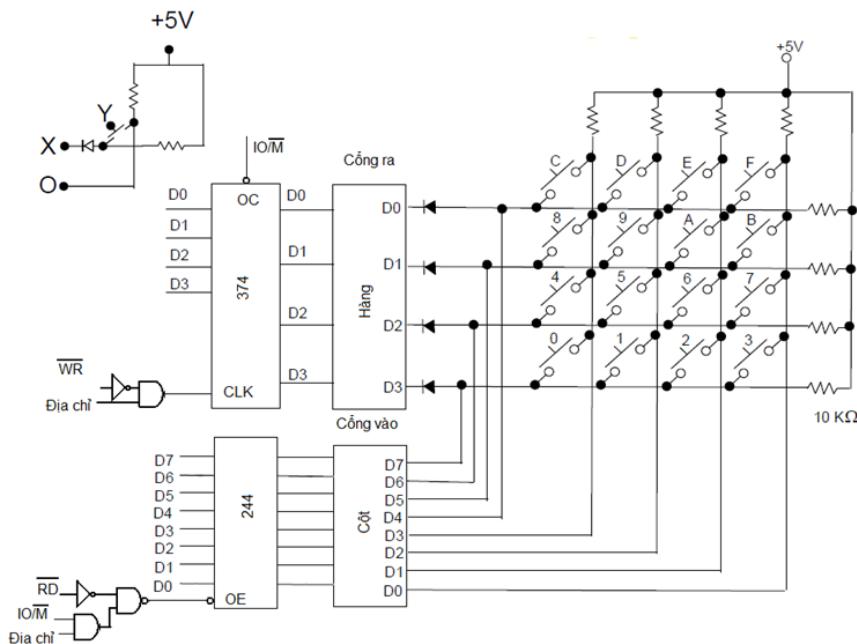
Hình 5. 10. Mạch tạo phím



Hình 5. 11. Ma trận phím và phát hiện các phím được nhấn

Bàn phím sử dụng một ma trận hình thành bởi các dòng và cột dây dẫn, như minh họa trên **Error! Reference source not found.** và **Error! Reference source not found..** Mỗi phím hoạt động như một công tắc điện. Khi phím được ấn, dây dẫn cột được nối với dây dẫn dòng tạo thành một mạch kín. Bộ điều khiển bàn phím liên tục quét ma trận phím để phát hiện mạch kín và ghi nhận phím được ấn. Quá trình xử lý phím ấn và tạo tín hiệu gửi CPU xử lý trong bàn phím có thể được tóm tắt như sau:

- Khi một phím được ấn, bộ điều khiển bàn phím phát hiện và sinh ra một mã quét tương ứng (scan code);
- Một ngắt (interrupt) bàn phím được gửi đến máy tính;
- Khi nhận được tín hiệu ngắt bàn phím:
  - Máy tính thực hiện chương trình điều khiển ngắt bàn phím:
    - Đọc mã quét phím
    - Chuyển mã quét phím thành mã ký tự tương ứng (thông thường là mã ASCII).
  - Một ký tự có thể được hiển thị theo nhiều hình thức khác nhau theo các bộ font.



Hình 5. 12. Biểu diễn ghép nối giữa 8086 với bàn phím 16 số dạng tiếp điểm.

**Error! Reference source not found.** minh họa cách kết nối vi xử lý 8086 với bàn phím 16 phím dạng tiếp điểm, sử dụng các mạch 74LS374 và 74LS244 để điều khiển. Vi mạch 74LS374 được dùng để điều khiển các tín hiệu hàng và 74LS244 dùng để điều khiển các tín hiệu cột của bàn phím. Nguyên tắc hoạt động của một phím như sau. Nếu tín hiệu X ở mức cao (lô-gíc 1) thì đi-ốt sẽ khóa lại, vậy nên tiếp điểm Y có đóng xuống hay không thì tại đầu O ta luôn thu được điện áp 5V (không có dòng điện). Nếu tín hiệu X ở mức thấp (lô-gíc 0), thì đi-ốt mở và khi tiếp điểm Y đóng xuống tại đầu O ta thu được điện áp 0V. Bằng cách quét tuần tự các hàng và đọc trên

các cột ta sẽ xác định được phím bấm. Giả sử tín hiệu địa chỉ giải mã vi mạch đệm công 374 là 0AH còn 244 là 0BH, đoạn mã sau đây cho phép xác định phím C có được bấm hay không:

Hang	EQU 0AH
Cot	EQU 0BH
	MOV AL,11111110b ; Chỉ có D0=0
	OUT Hang, AL
Ktra:	IN AL, Cot ; Đọc tín hiệu cột
	AND AL,00001000b ; Giữ lại bit D3 ứng với phím C
	JNZ Ktra ; Không bấm

Trong trường hợp bàn phím đầy đủ thì bàn phím bao gồm một tập hợp các phím mà mỗi phím hoạt động nhờ một công tắc cảm biến lực chuyển lực nhấn thành một đại lượng điện. Bộ vi điều khiển 8042 của máy tính và bộ vi điều khiển của bàn phím liên lạc với nhau tuần tự và đồng bộ qua hai đầu dây dẫn: dây dữ liệu và dây đồng hồ. Hai vi điều khiển này làm việc với nhau theo nguyên tắc chủ/tớ (master/slaver) trong đó 8042 là chủ còn 8048 là tớ. Khi có một thao tác phím, bàn phím phát ra một tín hiệu điện (IRQ1) gửi đến CPU. CPU sẽ tạm dừng công việc và gọi ngắt Int 9 đọc cổng bàn phím (địa chỉ 60H) để nhận mã Scan của tác động phím vừa xảy ra. Ngoài ra, nó sẽ đọc cờ bàn phím đó là một word ở địa chỉ 40:0017 để kết hợp với mã scan sinh ra mã ASCII tương ứng với tác động phím ở trên. Cặp hai byte gồm mã Scan và mã ASCII sẽ được Int 9h đưa vào vùng đệm bàn phím theo thứ tự mã ASCII đứng trước mã Scan. Chúng nằm ở đó cho tới khi được ngắt 16h phục vụ.

Trong trường hợp ngắt 9h phát hiện thấy từ cổng 60H các mã Scan ứng với tổ hợp phím đặc biệt, ví dụ Ctrl\_Alt\_Del, PrintScreen, Ctrl\_Numlock, Ctrl\_Break, SysReq thì nó sẽ hiểu đây là các lệnh khiến ROM-BIOS phải thực hiện ngay chứ không lưu lại các tổ hợp phím này trong bộ đệm bàn phím.

- Khi gấp Ctrl\_Alt\_Del, ROM BIOS gọi ngắt 19h (Boot strap loader).
- Khi gấp PrtScr, ROM BIOS gọi ngắt 5h (hard copy).
- Khi gấp Ctrl\_Numlock, ROM BIOS thi hành một vòng lặp vô hạn cho đến khi một phím khác được nhấn.
- Khi gấp Ctrl\_Break, ROM BIOS gọi ngắt 1Bh – kết thúc chương trình đang thực hiện, trả điều khiển lại cho DOS.
- Khi gấp SysReq, ROM BIOS gọi ngắt 15h với giá trị 8500H trong AX, khi nhấn phím này int 15h cũng được gọi nhưng với AX=8501. Dịch vụ 85H của ngắt 15H chỉ chứa một lệnh IRET, không gây tác động gì.

*Bộ đệm bàn phím và các thao tác*

Mỗi phím trên bàn phím đợt đọc bộ xử lý của bàn phím gán cho một mã Scan đặc trưng cho vị trí của phím trên bàn phím và cho trạng thái của phím (nhấn, không nhấn, nhấn chưa nhả.). Bộ đệm bàn phím (Keyboard buffer) là một miền nhớ 16-word trong RAM. Thuộc vùng dữ liệu của BIOS. Dưới đây là một phần vùng dữ liệu của BIOS (DTA) liên quan đến bộ đệm và trạng thái bàn phím:

Bảng 5. 4. Địa chỉ bộ nhớ và nội dung liên quan đến thiết bị ngoại vi và bàn phím.

<b>Địa chỉ bộ nhớ</b>	<b>Nội dung</b>
40:0000-40:0006	Địa chỉ các bộ phối ghép RS232 (1-4)
40:0008-40:000F	Địa chỉ các bộ phối ghép máy in (1-4)
40:0010	Cờ thiết bị (do ngắt 11h trả lại)
40:0012	Dấu hiệu kiểm tra của nhà sản xuất
40:0013	Dung lượng nhớ tính theo đơn vị KB
40:0015	Bộ nhớ của kênh vào/ra
40:0017	Cờ bàn phím
40:0019	Các số vào bằng phím alt
40:001A	Vị trí Head của vùng đệm
40:001C	Vị trí Tail của vùng đệm
40:001E-40:003D	Bộ đệm bàn phím

Vùng đệm bàn phím được tổ chức theo một hàng đợi quay vòng (circular queue) trong đó thao tác đọc và ghi vào bộ đệm là độc lập với nhau, điều này làm cho việc ghi và đọc phím vào và đọc ra theo kiểu mã phím (ASCII+Scan) nào được đưa vào bộ đệm trước thì sẽ được lấy ra trước. Con trỏ Head lưu trữ địa chỉ dành cho thao tác đọc, đó là vị trí sẽ đọc ký tự tiếp theo ra khỏi bộ đệm bàn phím. Sau mỗi thao tác đọc Head được tăng lên 1 word. Word tại địa chỉ 40:001A trong vùng DTA chứa địa chỉ của Head. Con trỏ Tail lưu trữ địa chỉ sẽ ghi ký tự tiếp theo vào bộ đệm bàn phím. Sau mỗi thao tác ghi Tail đợt đọc tăng lên 1. Word tại địa chỉ 40:001C chứa địa chỉ của Tail. Khi cả Head và Tail cùng trỏ tới word cuối cùng của bộ đệm và nếu có một tác động phím nữa xảy ra thì cả Head và Tail cùng trỏ tới đầu vùng đệm (40:001E). Khi mỗi phím được đọc ra thì Head lại tiến gần đến Tail, khi tất cả vùng đệm đã được đọc hết thì Head sẽ đuổi kịp Tail và cả hai cùng trỏ tới cùng một địa chỉ, lúc đó bộ đệm là rỗng. Khi có lời gọi ngắt int 9 thì chương trình của ta chiếm quyền điều khiển, nó sẽ gọi đến chương trình xử lý ngắt bàn phím cũ để nhận tác động phím và đặt cặp byte mã ASCII và Scan vào bộ đệm bàn phím. Đồng thời gán cho DS địa chỉ đoạn của vùng dữ liệu BIOS và kiểm tra xem Tail có trong bộ đệm bàn phím

không, word nằm trước Tail sẽ tương ứng với phím vừa mới nhận vào. Đọc byte mã Scan vào thanh ghi DH và byte mã ASCII vào DL. Sau đó, kiểm tra word trong DX có phải là hotkey hay không, nếu không phải sẽ nhảy đến kết thúc.

### 5.3.2 Dịch vụ của DOS và BIOS phục vụ bàn phím

#### ❖ Ngắt 16h của BIOS

##### **Hàm 0h:**

- Ý nghĩa: Chờ đọc một ký tự từ bàn phím (nếu có ký tự trong vùng đệm bàn phím thì sẽ nhận được ký tự đó, còn không thì chờ đến khi bàn phím được nhấn).
- Đầu vào: AH=0

Int 16h

- Đầu ra: Nếu AL $\neq$ 0 thì
  - AL chứa mã ASCII của ký tự
  - AH chứa mã SCAN của ký tự
- Nếu AL= 0 thì
  - AL chứa mã bàn phím mở rộng

##### **Hàm 1h:**

Ý nghĩa: Kiểm tra xem trong vùng đệm của bàn phím có ký tự hay không (không đợi đến khi ký tự có trong vùng đệm mà trả ngay điều khiển lại cho chương trình)?

- Đầu vào: AH=01

Int 16h

- Đầu ra: Nếu ZF=1 không có ký tự trong vùng đệm bàn phím
  - Nếu ZF=0 thì:
  - Nếu AL $\neq$ 0 thì:
    - AL chứa mã ASCII của ký tự
    - AH chứa mã SCAN của ký tự
  - Nếu AL= 0 thì:
    - AL chứa mã bàn phím mở rộng

**Hàm 02h:** Ý nghĩa: Kiểm tra trạng thái một số phím đặc biệt của bàn phím (Insert, CapsLock, NumLock, Scroll Lock).

- Đầu vào: AH=02

Int 16h

- Đầu ra: AL chứa kết quả các trạng thái hay cờ bàn phím, có ý nghĩa như sau:

Bảng 5. 5. Ý nghĩa cờ bàn phím

7	6	5	4	3	2	1	0
---	---	---	---	---	---	---	---

1: ché độ Insert	1: ché độ Cap Lock	1: Num Lock bị nhấn	1: Scroll Lock bị nhấn	1: Alt bị nhấn	1: Ctrl bị nhấn	1: Shift trái bị nhấn	1: Shift phải bị nhấn
------------------------	-----------------------------	---------------------------	------------------------------	-------------------	--------------------	-----------------------------	-----------------------------

### ❖ Một số hàm phục vụ bàn phím của ngắt 21h của DOS

**Hàm 06h:** Ý nghĩa: Đọc một ký tự từ bàn phím hoặc đưa ký tự ra màn hình. Nếu đọc vào một ký tự thì

- Đầu vào: AH=6

Int 21h

DL=0FFh (nếu DL<>0FFh sẽ đưa ra màn hình)

- Đầu ra: Nếu ZF=0 thì có ký tự trong vùng đệm bàn phím và:

- AL chứa mã ASCII của ký tự
- AH chứa mã SCAN của ký tự

Nếu ZF= 1 thì

- Vùng đệm bàn phím rỗng

**Hàm 07h:**

- Ý nghĩa: Chờ đọc một ký tự từ bàn phím
- Đầu vào: AH=07

Int 21h

- Đầu ra: AL chứa mã ASCII của ký tự (AL=0 sẽ không có ký tự nào)  
AH chứa mã SCAN của ký tự

**Hàm 0Bh:**

- Ý nghĩa: Đọc trạng thái bộ đệm bàn phím
- Đầu vào: AH=0Bh

Int 21h

- Đầu ra: AL =0FFh có ký tự trong bộ đệm  
AL =00h không có ký tự trong bộ đệm

**Hàm 0Ch:**

- Ý nghĩa: xóa bộ đệm bàn phím, sau đó gọi hàm vào ký tự có số chức năng đặt trong AL
- Đầu vào: AL = số hàm của ký tự

### 5.3.3 Một số chương trình ví dụ về lập trình bàn phím

Có hai cách lập trình bàn phím:

- Lập trình hệ thống, truy nhập lập trình qua các bộ vi điều khiển 8042, 8044 qua các cổng 60H, 61H và 64H.

- Lập trình ứng dụng, dùng các hàm BIOS (int 9h, Int 16H) hoặc các hàm 0Ah, 0Bh, 0Ch của int 21h.

Cổng 60H là cổng A của 8255A và là nơi để CPU và bàn phím trao đổi thông tin bao gồm các lệnh (thường là của chương trình con ngắn) điều khiển bàn phím hoặc ký tự từ bàn phím vào. Đối với người lập trình thì cổng 60H được coi như thanh ghi mã lệnh của bàn phím. Mã lệnh EDH là mã lệnh tắt, bật đèn

**Ví dụ 0:** Viết chương trình thiết lập mật khẩu là kí tự A thi khởi động máy

```
.MODEL small
.STACK 100h
.DATA
matkhau db „P”,“$”
Saimatkhanh db „Sai mat khau “,“$“
Nhaphmatkhau db „Nhap mat khau: “,“$“
xuongdong db 13,10,“$“
.CODE
Start:
Mov AX,@Data
Mov DS,AX
Lap:
Mov AH,9
Mov DX, offset Nhaphmatkhau
Int 21h ; in lời mời nhập xâu
Mov AH,0 ; Nhập kí tự
Int 16h
Cmp AL,matkhau ; có phải Enter không?
JZ Done ; Nếu là Enter, dừng lại
Mov AH,9
Mov DX, offset Saimatkhanh
Int 21h ; Xuong dong va ve dau dong
Jmp Lap
Done:
Mov AH,4Ch ; Tro ve DOS
Int 21h
End Start
```

**Ví dụ 1:** Viết chương trình tự động bật phím CapLock (sau khi chạy chương trình này phím Caplock sẽ được bật lên)

Cờ bàn phím là byte ở tại địa chỉ 40:0017 trong vùng đệm bàn phím. Để đèn của phím CapLock là ON thì giá trị của cờ bàn phím đọc được đặt bằng: 01000000 =40h. ta chỉ cần gán giá trị 40h vào byte có địa chỉ 40:0017 là xong.

```
.MODEL Tiny
.CODE
Org 100h
Jmp Start
Start:
Mov AX,40h ; AX chứa địa chỉ đoạn dữ liệu DTA
Mov DS, AX ; cho DS trở tới vùng DTA
Mov BX,0017h ; DS:BX chứa địa chỉ của byte chứa cờ
; trạng thái bàn phím
```

```

Mov byte PTR[BX] ,40h ; đặt cờ bàn phím bằng 40h
Mov AH, 4CH
Int 21h ; trả về DOS
End Start

```

**Ví dụ 2:** Viết chương trình tự động bật phím CapLock, NumLock, ScrollLock (sau khi chạy chương trình này phím Caplock, NumLock, ScrollLock sẽ được bật lên). Yêu cầu chương trình thực hiện qua các cổng 60H, 61H và 64H. Chương trình sẽ kiểm tra trạng thái nhấn phím hay chưa.

```

Jmp Start
Start:
    Mov AL,EDHh ; lệnh tắt/bật đèn LED của phím
    Out 60H, AL ; đưa ra cổng bàn phím
Kiemtra:
    In AL,64h ;Kiểm tra trạng thái bàn phím
    Test AL,02h ; Nếu bộ đệm bàn phím bằng đầy
    Jnz Kiemtra ; kiểm tra lại
    Mov Al,07 ; Nếu bộ đệm trống sẽ bật đèn
    Out 60H, AL ; đưa ra mã bật đèn ra cổng bàn phím
    Mov AH, 4CH
    Int 21h ; trả về DOS
End Start

```

**Ví dụ 3:** Viết chương trình cầm bàn phím hoạt động. Biết rằng lệnh khóa ADh là lệnh cầm bàn phím. Kiểm tra trạng thái bàn phím trước khi cầm.

```

.CODE
Jmp Start
Start:
Kiemtra:
    In AL,64H ; đọc trạng thái
    Test AL,02h ; Nếu bộ đệm bàn phím bằng đầy
    Jnz Kiemtra ; kiểm tra lại
    Mov Al,ADh ; Nếu bộ đệm trống sẽ bật đèn
    Out 64H, AL ; đưa ra cầm bàn phím ra cổng 64h
    Mov AH, 4Ch
    Int 21h ; trả về DOS
End Start

```

**Ví dụ 4:** Viết chương trình xử lý bàn phím đơn giản. Chương trình kiểm tra các phím chữ cái và phân biệt chữ hoa, chữ thường

```

.MODEL small
.STACK 100h
.DATA
    Table db 16 dup(0)
    db „qwertyuiop“,0,0,0,0 ; hang tren
    db „asdfghjkl“,0,0,0,0 ; hang giua
    db „zxcvbnm“; hang duoi

```

```

        db 16 dup(0) ; vung danh cho chu hoa
        db „QERTYUIOP“,0,0,0,0 ; chu hoa cho hang tren
        db „ASDFGHJKL“,0,0,0,0,0 ; chu hoa cho hang giua
        db „ZXCVBNM“; chu hoa cho hang duoi

.CODE
Start:
    Mov AX,@Data
    Mov DS,AX
    Cli; xoa ngat
    Push DS ;
    Mov AX, seg TryKB ; DS:AX tro den Checkbanphim
    Mov DS,AX
    Mov DX, offset TryKB ; Checkbanphim
    Mov Al,9
    Mov Ah,25H ; dat lai dia chi vector ngat
    Int 21h
    Pop DS
    Sti ; cho phep ngat
;-----
;----- Chuong trinh xu ly ngat ban phim-----
TryKB proc far
    Push AX
    Push BX
    Push CX
    Push DI
    Push ES
; nhan ma Scan va tra loi ban phim
    In Al,60h ; nhan ma Scan
    Mov AH,AL ; chuyen vao AH
    Push AX ; dua vao stack
    In AL,61h ; doc cong PB cua 8255A
    Or AL,10000000b ; dua bit 7 len bit 1
    Out 61h,AL ; dua ra cong PB
; Cho ES tro den doan du lieu
    Mov AX,40H ; dua AX den cuoi bo nho
    Mov ES,AX
    Pop AX ; AL= ma scan
; kiem tra phim shift
    Cmp Al,42 ; co nhan phim shift ko?
    Jnz checkkey ; neu khong thi kiem tra tiep
    Mov BL,1
    Or ES:[17h],BL dat bit 1 cua byte trang thai =1
    Jmp Thoat ; thoat khoi
PhimKetiep:
    Test AL,10000000b; ma nha phim
    Jnz Thoat
    Mov BL,ES:[17h] ; neu ko doc dc trang thai phim shift
    Test BL,00000011b; nhan phim shift?
    Jz DoiMa ; neu ko doi lai ma
    Add AL,100 ;doi ra ki tu hoa(dia chi 100 byte ke tiep)
DoiMa:
    Mov BX, offset table ;
    Xlat table ; doi ma scan sang ma ASCII
    Cmp Al,0 ; tra ve 0?
    Jz Thoat

```

```

; Kiem tra do dem ban phim da day cua?
Mov BX, 1AH ; nap con tro bo dem ban phim
Mov CX,ES:[BX]
Mov DI,ES:[BX]+2
Cmp CX,60
Jz Tieptuc
Add, CX,2
Cmp CX,DI
Jz Thoat
; Bo dem chua day, nap the ki tu vao
Tieptuc:
Mov ES:[DI],AL
Cmp DI,60H
Jnz Naptiep
Mov DI,28 ; dua dia chi hien tai ve 28+2=30
Naptiep:
Add DI,2
Mov ES:[BX],DI
; ket thuc
Thoat:
Pop ES
Pop DI
Pop CX
Pop BX
Pop AX
Mov AL,20h ; tra lai ngat
Out 20h,AL
IRET
TryKB endp
;-----
End Start

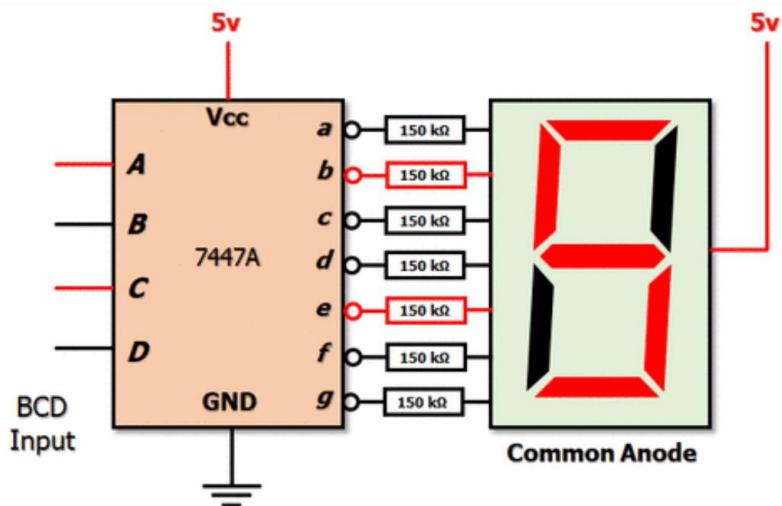
```

## 5.4 Lập trình điều khiển đèn LED

### 5.4.1 Cơ bản về đèn LED và vi mạch 7447

Đèn LED (Light Emitting Diode) là một thiết bị bán dẫn phát ra ánh sáng khi có dòng điện chạy qua. LED được sử dụng rộng rãi trong các hệ thống hiển thị do khả năng tiết kiệm năng lượng, tuổi thọ cao và độ sáng tốt. LED bảy đoạn là một loại LED đặc biệt, bao gồm 7 thanh LED được sắp xếp theo hình dạng số 8. Mỗi thanh LED được đặt tên từ a đến g, tương ứng với các đoạn khác nhau của số 8. Bằng cách bật/tắt các thanh LED này, chúng ta có thể hiển thị các chữ số từ 0 đến 9 và một số ký tự đơn giản như A, b, C, d, E, F, như được minh họa trong **Error! Reference source not found..**

Vi mạch 7447 là một bộ giải mã BCD (Binary-Coded Decimal) sang 7 đoạn. Nó nhận đầu vào là mã BCD 4-bit (từ 0000 đến 1001, tương ứng với các số từ 0 đến 9) và xuất ra 7 tín hiệu (a-g) để điều khiển các thanh LED trong LED bảy đoạn.



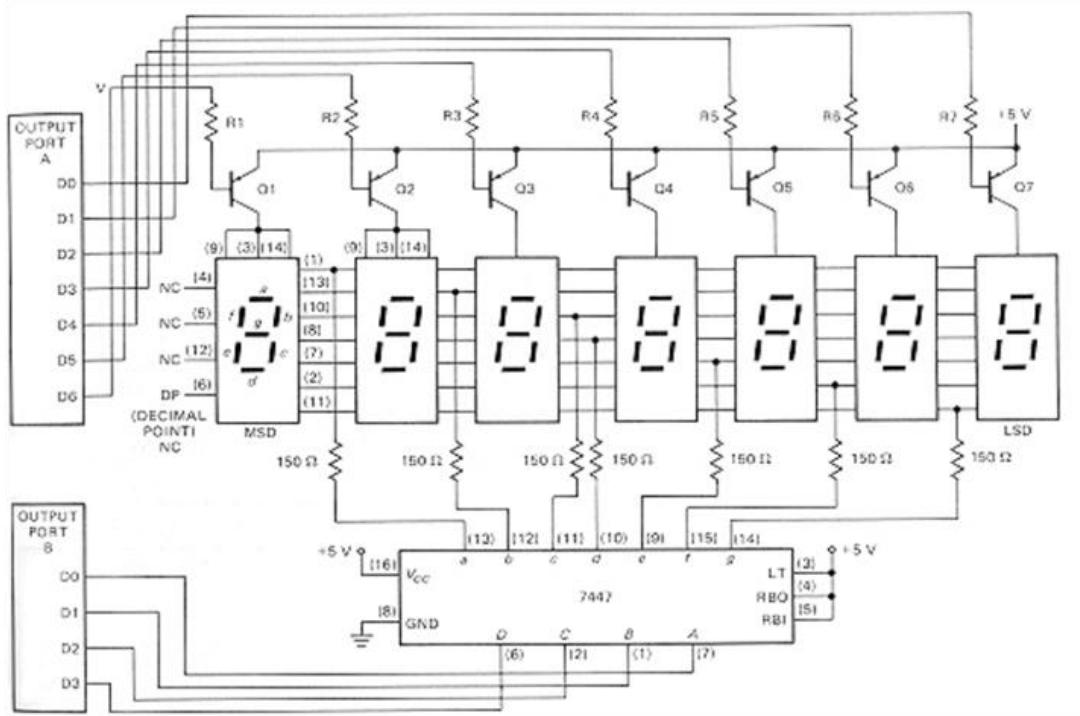
Hình 5. 13. Sơ đồ mạch kết nối IC 7447A với LED 7 đoạn

Nguyên lý hoạt động của 7447:

- Đầu vào BCD (A-D): 4-bit đầu vào, biểu diễn số từ 0 đến 9.
- Đầu ra (a-g): 7 tín hiệu đầu ra, mỗi tín hiệu tương ứng với một thanh LED trong LED bảy đoạn.
- Chức năng: Dựa trên mã BCD đầu vào, 7447 sẽ kích hoạt các thanh LED tương ứng để hiển thị chữ số mong muốn.

Ví dụ:

- Nếu đầu vào BCD là **0000** (số 0), 7447 sẽ kích hoạt các thanh LED **a, b, c, d, e, f** (tất cả các thanh trừ **g**).
- Nếu đầu vào BCD là **1001** (số 9), 7447 sẽ kích hoạt các thanh LED **a, b, c, d, f, g**.



Hình 5. 14. Biểu diễn một mạch hiển thị số sử dụng vi mạch 7447 và LED bảy đoạn

Mạch điện này sử dụng vi mạch 7447 để điều khiển hiển thị trên 7 LED bảy đoạn như **Error! Reference source not found.**. Việc điều khiển này được thực hiện thông qua hai cổng đầu ra (Output Port):

1. Cổng đầu ra A (Output Port A): Chức năng chính của cổng này là chọn LED bảy đoạn nào sẽ được kích hoạt để hiển thị. Mỗi đầu ra (D0 đến D6) của cổng này được kết nối với chân B của một transistor (Q1 đến Q7). Khi một đầu ra của cổng A ở mức thấp (thường là 0V), transistor tương ứng sẽ được kích dẫn. Mỗi transistor này lại được kết nối với cực dương chung (Anode chung) của một LED bảy đoạn. Do đó, chỉ có LED bảy đoạn được kết nối với transistor đang dẫn mới có thể hiển thị.
2. Cổng đầu ra B (Output Port B): Chức năng chính của cổng này là cung cấp mã dữ liệu BCD (Binary-Coded Decimal) cho vi mạch 7447. Các đầu ra D0, D1, D2, D3 của cổng B được kết nối với các chân đầu vào A, B, C, D của vi mạch 7447 tương ứng. Vi mạch 7447 sẽ giải mã BCD này và kích hoạt các chân đầu ra (a đến g) tương ứng để hiển thị chữ số hoặc ký tự trên LED bảy đoạn đang được chọn bởi cổng A.

Sơ đồ kết nối cụ thể:

- Cổng dữ liệu (DL\_LED trong mô tả gốc) tương ứng với Cổng đầu ra B (Output Port B) trong hình vẽ. Nó cung cấp dữ liệu dạng BCD cho IC 7447.
- Cổng điều khiển (DK\_LED trong mô tả gốc) tương ứng với Cổng đầu ra A (Output Port A) trong hình vẽ. Nó điều khiển việc kích hoạt từng LED bảy đoạn thông qua các transistor.

- Transistor điều khiển (Q1-Q7): Mỗi transistor được điều khiển bởi một đầu ra của Cổng A. Khi transistor dẫn, nó cấp nguồn cho cực dương chung của LED bảy đoạn tương ứng.
- Đầu ra 7447 (a-g): Các chân đầu ra này được kết nối với các chân điều khiển các thanh LED (a đến g) của từng LED bảy đoạn thông qua điện trở hạn dòng  $150\Omega$ . Vì mạch 7447 quyết định thanh LED nào sẽ sáng dựa trên mã BCD nhận được từ Cổng B và LED bảy đoạn nào đang được kích hoạt bởi Cổng A.

#### 5.4.2 Một số chương trình ví dụ về lập trình bàn phím

**Ví dụ 0:** Lập trình điều khiển LED bảy đoạn

Để điều khiển LED bảy đoạn, chúng ta cần thực hiện các bước sau:

1. Tắt tất cả các LED: Gửi giá trị 0FFH đến cổng điều khiển (DK\_LED) để tắt tất cả các LED.
2. Gửi mã BCD đến 7447: Gửi mã BCD tương ứng với số cần hiển thị đến cổng dữ liệu (DL\_LED).
3. Bật LED tương ứng: Gửi giá trị 0 đến cổng điều khiển (DK\_LED) để bật LED tương ứng.

```

.MODEL small
.STACK 100h
.DATA
DK_LED EQU 0AH          ; Địa chỉ cổng điều khiển LED
DL_LED EQU 0BH          ; Địa chỉ cổng dữ liệu hiển thị

.CODE
Start:
    MOV AX, @Data
    MOV DS, AX

    MOV AL, 0FFH          ; Tắt tất cả các LED (1 = Tắt LED trong mạch thường)
    OUT DK_LED, AL

    MOV CX, 16             ; Tạo độ trễ

Del:
    NOP                  ; Không làm gì, tạo độ trễ nhỏ
    LOOP Del              ; Giảm CX và lặp lại cho đến khi CX = 0

    MOV AL, 08H            ; Đưa số 8 (BCD) ra 7447 để hiển thị
    OUT DL_LED, AL

    XOR AL, AL            ; Đặt AL = 0
    OUT DK_LED, AL        ; Bật tất cả các LED

    HLT                  ; Dừng chương trình (chờ xử lý tiếp theo)

End Start

```

### Ví dụ 1: Hiển thị số từ 0 đến 9 trên LED bảy đoạn

Để hiển thị số từ 0 đến 9 trên LED bảy đoạn, chúng ta cần thực hiện các bước sau:

1. Tắt tất cả các LED: Gửi giá trị OFFH đến cổng điều khiển (DK\_LED) để tắt tất cả các LED.
2. Hiển thị số từ 0 đến 9: Sử dụng vòng lặp để hiển thị các số từ 0 đến 9 trên LED bảy đoạn.
3. Tạo độ trễ: Sử dụng hàm Delay để tạo độ trễ giữa các lần hiển thị.
4. Kết thúc chương trình: Sử dụng ngắt 21h để kết thúc chương trình.

```
DK_LED EQU 0AH      ; Địa chỉ cổng điều khiển LED
DL_LED EQU 0BH      ; Địa chỉ cổng dữ liệu hiển thị

.MODEL small
.STACK 100h
.CODE
Start:
    MOV AL, 0FFH      ; Tắt tất cả các LED (1 = Tắt LED trong mạch thường)
    OUT DK_LED, AL

    MOV CX, 10         ; Số lần lặp (hiển thị từ 0 đến 9)
    MOV AL, 0           ; Bắt đầu từ số 0

DisplayLoop:
    OUT DL_LED, AL    ; Gửi giá trị hiển thị đến LED bảy đoạn
    XOR AH, AH        ; Đặt AH = 0 (để chắc chắn)
    OUT DK_LED, AH    ; Bật LED

    CALL Delay         ; Gọi hàm trễ để giữ số hiển thị trong một thời gian

    INC AL             ; Tăng giá trị hiển thị (AL = AL + 1)
    LOOP DisplayLoop  ; Giảm CX và lặp lại nếu CX chưa bằng 0

    MOV AH, 4CH         ; Ngắt kết thúc chương trình (DOS interrupt)
    INT 21H

Delay:
    MOV BX, 0FFFFH     ; Giá trị ban đầu của bộ đếm trễ
DelayLoop:
    DEC BX             ; Giảm BX
    JNZ DelayLoop     ; Tiếp tục giảm nếu BX chưa về 0
    RET                ; Quay lại chương trình chính

END Start
```

## 5.5 Lập trình điều khiển nhiệt độ

### 5.5.1 Cơ bản về điều khiển nhiệt độ

Cảm biến nhiệt độ là thiết bị quan trọng trong các hệ thống điều khiển và giám sát. Tùy thuộc vào loại cảm biến, tín hiệu đầu ra có thể là tín hiệu analog hoặc số, với cách xử lý dữ liệu khác nhau:

- Cảm biến nhiệt độ Analog, như LM35, LM34, và TMP36, có đầu ra là điện áp tỷ lệ tuyến tính với nhiệt độ. Ví dụ, LM35 tạo ra tín hiệu điện áp với tỷ lệ  $10 \text{ mV}/\text{°C}$ . Khi nhiệt độ thay đổi, điện áp đầu ra của cảm biến cũng thay đổi tương ứng. Vì điều khiển sẽ sử dụng bộ chuyển đổi ADC (Analog-to-Digital Converter) để đọc giá trị điện áp này, sau đó tính toán và hiển thị nhiệt độ hiện tại.
- Cảm biến nhiệt độ Kỹ thuật số, như DS18B20, DHT11, và DHT22, giao tiếp trực tiếp với vi điều khiển qua các giao thức truyền thông như 1-Wire (DS18B20), I2C, SPI hoặc UART. Các cảm biến này trả dữ liệu nhiệt độ dưới dạng số (digital) với độ phân giải cố định, chẳng hạn như từ 9 đến 12-bit với DS18B20. Việc đọc dữ liệu từ cảm biến được thực hiện bằng cách sử dụng các lệnh, hàm hoặc thư viện giao tiếp được cung cấp bởi nhà sản xuất.

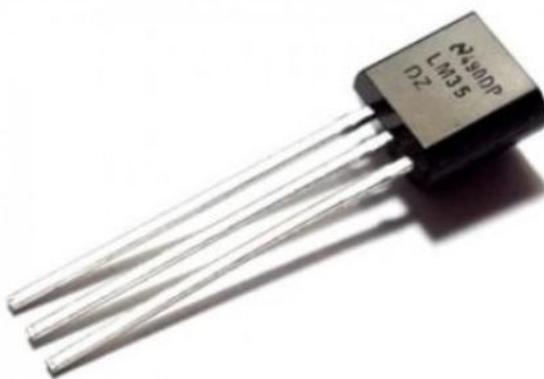
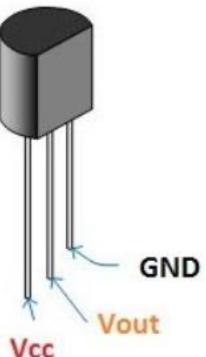
Việc lựa chọn cảm biến nhiệt độ phụ thuộc vào nhiều yếu tố. Đầu tiên là độ chính xác và độ phân giải mong muốn để đáp ứng yêu cầu đo lường. Dải đo nhiệt độ cũng cần phù hợp với môi trường làm việc, ví dụ  $-55\text{°C} \sim 125\text{°C}$  hoặc  $0\text{°C} \sim 100\text{°C}$ . Ngoài ra, cần xem xét phương thức giao tiếp, có thể là Analog (yêu cầu ADC) hoặc Digital (giao thức truyền thông như I2C, SPI). Thời gian đáp ứng của cảm biến cũng quan trọng, đặc biệt trong các ứng dụng yêu cầu phản hồi nhanh. Cuối cùng, cần đảm bảo cảm biến có khả năng hoạt động tốt trong các điều kiện môi trường như độ ẩm cao, rung động mạnh, hoặc nhiễu điện từ.

### 5.5.2 Mạch đo và điều khiển nhiệt độ

Tùy theo loại cảm biến mà ta có các sơ đồ kết nối khác nhau:

- **Mạch đo nhiệt độ dùng cảm biến analog** (ví dụ LM35 trong **Error! Reference source not found.**): Cảm biến nhiệt độ analog LM35 là một thiết bị đo nhiệt độ chính xác với dải đo từ  $0\text{°C}$  đến  $150\text{°C}$ , hoạt động ổn định với điện áp 5V và tiêu thụ điện năng thấp. LM35 có thiết kế 3 chân: chân 1 (VCC) dùng để cấp nguồn 5V, chân 3 (GND) nối đất, và chân 2 (OUT) là chân xuất tín hiệu dạng điện áp analog. Điện áp đầu ra tỷ lệ tuyến tính với nhiệt độ đo được, với tỉ lệ  $10\text{mV}/\text{°C}$  (hoặc  $1\text{mV}/0,1\text{°C}$ ). Để tính giá trị nhiệt độ theo đơn vị  $\text{°C}$ , chỉ cần lấy điện áp đo được tại chân OUT chia cho 10. LM35 là giải pháp lý tưởng trong các ứng dụng đo nhiệt độ nhờ tính đơn giản, độ chính xác cao và khả năng dễ dàng xử lý tín hiệu.

## LM35 Pinout



Hình 5. 15. Cảm biến nhiệt độ LM35

- **Mạch đo nhiệt độ dùng cảm biến kỹ thuật số** (ví dụ DS18B20 trong **Error! Reference source not found.** ): Cảm biến DS18B20 là một thiết bị đo nhiệt độ kỹ thuật số tiên tiến do hãng Maxim phát triển, nổi bật với độ phân giải 12-bit và giao tiếp 1-Wire tiện lợi, dễ dàng lập trình. Cảm biến này có 3 chân: GND, DQ (Data), và VDD, với khả năng cấp nguồn trực tiếp hoặc hoạt động ở chế độ parasite power thông qua chân DQ. Chân DQ được kết nối với nguồn bằng điện trở pull-up  $4.7\text{k}\Omega$  để đảm bảo tín hiệu ổn định. Vì điều khiển giao tiếp với DS18B20 qua giao thức 1-Wire để thực hiện đọc và ghi dữ liệu nhiệt độ với độ chính xác cao. DS18B20 hoạt động ổn định trong dải nhiệt độ từ  $-55^{\circ}\text{C}$  đến  $125^{\circ}\text{C}$ , nhưng nếu sử dụng cáp bọc PVC, nhiệt độ tối đa nên giữ dưới  $100^{\circ}\text{C}$  để đảm bảo độ bền và an toàn. Cảm biến này tích hợp chức năng cảnh báo nhiệt độ vượt ngưỡng và khả năng chống suy giảm tín hiệu khi truyền trên các đường dây dài, làm cho nó trở thành lựa chọn lý tưởng cho nhiều ứng dụng. Một số ứng dụng phổ biến bao gồm: kiểm soát nhiệt độ HVAC, giám sát môi trường, đo nhiệt độ trong các tòa nhà, thiết bị, máy móc, và các hệ thống tự động hóa hiện đại. DS18B20 là giải pháp lý tưởng cho các yêu cầu đo nhiệt độ chính xác, tin cậy và linh hoạt.



Hình 5. 16. cảm biến nhiệt độ DS18B20

### 5.5.3 Một số chương trình ví dụ về lập trình điều khiển nhiệt độ

**Ví dụ 0:** Giả sử sử dụng cảm biến LM35 (analog)

- **Mạch kết nối:**

- Chân Vout của LM35 vào kênh ADC (giả sử cổng 00h).
- Cổng P1.0 (hoặc tương đương trên 8255) điều khiển heater.

- **Yêu cầu:**

- Giữ nhiệt độ trong khoảng  $30^{\circ}\text{C}$  đến  $35^{\circ}\text{C}$ .
- Nếu  $T_{current} < 30^{\circ}\text{C}$  thì bật heater ( $\text{P1.0} = 0$  để kích transistor).
- Nếu  $T_{current} > 35^{\circ}\text{C}$  thì tắt heater ( $\text{P1.0} = 1$ ).

```

;-----
; Giả sử:
; - Cổng ADC được đọc qua địa chỉ 0CH (ví dụ).
; - Cổng điều khiển heater là 0AH (bit D0).
; - Nhiệt độ đọc từ LM35: 10mV/ $^{\circ}\text{C}$  =>  $1^{\circ}\text{C} \sim 1$  đơn vị ADC (với thang đo
; 0~5V/10-bit).
; - Biểu diễn đơn giản, không chuẩn hóa hệ số ADC thực tế.
;-----

ADC      EQU 0CH          ; Địa chỉ đọc ADC
DK_HEAT EQU 0AH          ; Địa chỉ cổng điều khiển Heater

TEMP_LOW EQU 30           ; Nhiệt độ dưới ngưỡng bật heater
TEMP_HIGH EQU 35          ; Nhiệt độ trên ngưỡng tắt heater

Start:
; Khởi động ...
MOV AL, 0FFH
OUT DK_HEAT, AL          ; Mặc định tắt heater (giả sử bit 0=1 -> tắt)

LoopCheck:

```

```

; Đọc giá trị ADC (nhiệt độ)
IN AL, ADC
MOV BL, AL           ; Lưu giá trị đo được vào BL

; So sánh với TEMP_LOW
CMP BL, TEMP_LOW
JB TurnOnHeater      ; Nếu < 30°C -> bật heater

; So sánh với TEMP_HIGH
CMP BL, TEMP_HIGH
JA TurnOffHeater     ; Nếu > 35°C -> tắt heater

JMP LoopCheck         ; Nếu ở giữa 30°C ~ 35°C thì giữ trạng thái hiện tại

TurnOnHeater:
; Bật heater (bit D0 = 0)
IN AL, DK_HEAT
AND AL, 11111110b    ; Clear bit 0
OUT DK_HEAT, AL
JMP LoopCheck

TurnOffHeater:
; Tắt heater (bit D0 = 1)
IN AL, DK_HEAT
OR AL, 00000001b     ; Set bit 0
OUT DK_HEAT, AL
JMP LoopCheck

```

## 5.6 Lập trình điều khiển hệ thống đèn giao thông

### 5.6.1 Mô tả hệ thống đèn giao thông

Một hệ thống đèn giao thông cơ bản thường có ba đèn: đỏ, vàng và xanh. Mỗi đèn sẽ được bật/tắt tuần tự theo chu kỳ để điều khiển luồng giao thông. Thông thường, đèn đỏ sáng một khoảng thời gian (ví dụ 30 giây), sau đó đến đèn xanh (ví dụ 30 giây), rồi đến đèn vàng (ví dụ 3-5 giây) trước khi trở lại đèn đỏ.

Về mặt phần cứng, ta có thể dùng:

- Một vi điều khiển (có thể là 8086, 8051, Arduino, v.v.) hoặc mạch vi xử lý có các cổng đầu ra (như 8255) để điều khiển transistor cấp dòng cho các LED (đỏ, vàng, xanh).
- Mỗi đèn giao thông được cấp nguồn thông qua transistor/bộ khuếch đại (nếu cần) để đảm bảo đủ công suất cho LED hoặc bóng đèn (công suất cao hơn).

### 5.6.2 Nguyên lý hoạt động

Hệ thống đèn giao thông là một hệ thống điều khiển giao thông tại các ngã tư, ngã ba, hoặc các điểm giao cắt khác. Hệ thống này sử dụng các đèn tín hiệu (đỏ, vàng, xanh) để điều khiển luồng giao thông, đảm bảo an toàn và trật tự.

Các thành phần chính của hệ thống đèn giao thông:

1. Đèn tín hiệu: Bao gồm đèn đỏ, đèn vàng, và đèn xanh.
2. Bộ điều khiển: Vi điều khiển hoặc vi xử lý để điều khiển thời gian và trạng thái của các đèn.
3. Cảm biến: Có thể sử dụng cảm biến để phát hiện phương tiện giao thông và điều chỉnh thời gian đèn tín hiệu.

Nguyên lý hoạt động:

- Chu kỳ đèn: Mỗi chu kỳ đèn bao gồm các giai đoạn đèn đỏ, đèn vàng, và đèn xanh.
- Thời gian mỗi giai đoạn: Thời gian của mỗi giai đoạn được cài đặt trước và có thể điều chỉnh tùy theo nhu cầu giao thông.
- Điều khiển tự động: Hệ thống tự động chuyển đổi giữa các giai đoạn đèn theo chu kỳ đã cài đặt.

Ví dụ chu kỳ đèn giao thông: đèn xanh: 30 giây, đèn vàng: 5 giây, đèn đỏ: 35 giây

### 5.6.3 Một số chương trình ví dụ về lập trình điều khiển đèn giao thông

**Ví dụ 0:** Chu kỳ điều khiển 3 đèn giao thông

Chương trình này điều khiển hệ thống đèn giao thông với thời gian cố định cho mỗi giai đoạn:

- Đèn xanh: 10 giây
- Đèn vàng: 3 giây
- Đèn đỏ: 10 giây.

```
.MODEL Small
.STACK 100h

.DATA
    RedLED DB 60h      ; LED đỏ địa chỉ cổng 60H
    YellowLED DB 61h    ; LED vàng địa chỉ cổng 61H
    GreenLED DB 62h    ; LED xanh địa chỉ cổng 62H

.CODE
Start:
    MOV AX, @Data
    MOV DS, AX

    ; Bật LED đỏ
    MOV AL, 1
    OUT RedLED, AL
    CALL Delay        ; Trì hoãn 10 giây
```

```

; Tắt LED đỏ, bật LED vàng
MOV AL, 0
OUT RedLED, AL
MOV AL, 1
OUT YellowLED, AL
CALL Delay      ; Trì hoãn 3 giây

; Tắt LED vàng, bật LED xanh
MOV AL, 0
OUT YellowLED, AL
MOV AL, 1
OUT GreenLED, AL
CALL Delay      ; Trì hoãn 10 giây

; Tắt LED xanh
MOV AL, 0
OUT GreenLED, AL

JMP Start      ; Lặp lại chu kỳ

Delay PROC
    MOV CX, 5000      ; Số vòng lặp tạm
    DelayLoop:
        DEC CX
        JNZ DelayLoop
    RET
Delay ENDP

END Start

```

## 5.7 Lập trình điều khiển robot đơn giản và một số ví dụ

### 5.7.1 Cơ bản về robot

Lập trình điều khiển robot đơn giản thường liên quan đến việc điều khiển các bộ phận chuyển động của robot, chẳng hạn như động cơ, thông qua các lệnh lập trình. Trong giới hạn của môn học này chúng ta tập trung vào việc điều khiển các robot có cấu trúc đơn giản, ví dụ như robot di chuyển trên bánh xe.

Để điều khiển một robot đơn giản, chúng ta cần xác định các yếu tố sau:

- Các bộ phận chuyển động: Xác định các động cơ hoặc cơ cấu chấp hành mà chúng ta có thể điều khiển (ví dụ: động cơ cho bánh xe trái và bánh xe phải).
- Cảm biến (tùy chọn): Các cảm biến có thể cung cấp thông tin về môi trường xung quanh robot (ví dụ: cảm biến khoảng cách, cảm biến va chạm). Trong các ví dụ đơn giản ban đầu, chúng ta có thể bỏ qua cảm biến để tập trung vào điều khiển chuyển động cơ bản.
- Bộ điều khiển: Vị điều khiển (như đã đề cập trong các phần trước) sẽ đóng vai trò là bộ não của robot, nhận lệnh và điều khiển các động cơ.

- Giao diện điều khiển: Cách chúng ta gửi lệnh đến robot. Điều này có thể thông qua bàn phím (như đã đề cập), các nút bấm, hoặc một chương trình điều khiển khác.

Nguyên tắc điều khiển robot di chuyển cơ bản:

- Di thẳng: Điều khiển cả hai động cơ bánh xe quay cùng chiều và cùng tốc độ.
- Rẽ trái: Giảm tốc độ hoặc dừng động cơ bánh xe trái và giữ nguyên hoặc tăng tốc độ động cơ bánh xe phải.
- Rẽ phải: Giảm tốc độ hoặc dừng động cơ bánh xe phải và giữ nguyên hoặc tăng tốc độ động cơ bánh xe trái.
- Di lùi: Điều khiển cả hai động cơ bánh xe quay ngược chiều và cùng tốc độ.
- Dừng lại: Dừng cả hai động cơ.

### 5.7.2 Một số ví dụ về lập trình điều khiển robot đơn giản

**Ví dụ 0:** Điều khiển robot di chuyển bằng phím mũi tên (tiếp nối ví dụ điều khiển bàn phím)

Giả sử robot có hai động cơ, một cho bánh xe trái và một cho bánh xe phải, được điều khiển thông qua các cổng MOTOR\_LEFT và MOTOR\_RIGHT. Chúng ta sẽ sử dụng các phím mũi tên để điều khiển robot:

- Mũi tên lên: Di thẳng
- Mũi tên xuống: Di lùi
- Mũi tên trái: Rẽ trái
- Mũi tên phải: Rẽ phải

```
.MODEL small
.STACK 100h

MOTOR_LEFT EQU 0AH ; Cổng điều khiển động cơ trái
MOTOR_RIGHT EQU 0BH ; Cổng điều khiển động cơ phải

; Giá trị điều khiển động cơ (ví dụ)
FORWARD EQU 01h ; Quay thuận
BACKWARD EQU 02h ; Quay nghịch
STOP EQU 00h ; Dừng

.CODE
Start:
    ; Thiết lập đoạn dữ liệu
    Mov AX, @Data
    Mov DS, AX

MainLoop:
    ; Đọc phím từ bàn phím (sử dụng ngắt 16h)
    Mov AH, 0
    Int 16h
```

```

; Kiểm tra phím và điều khiển động cơ
Cmp AH, 48h ; Mã Scan của phím mũi tên lên
JE ForwardAction

Cmp AH, 50h ; Mã Scan của phím mũi tên xuống
JE BackwardAction

Cmp AH, 4Bh ; Mã Scan của phím mũi tên trái
JE LeftTurnAction

Cmp AH, 4Dh ; Mã Scan của phím mũi tên phải
JE RightTurnAction

Jmp MainLoop ; Nếu không phải phím điều khiển, tiếp tục vòng lặp

ForwardAction:
Mov AL, FORWARD
Out MOTOR_LEFT, AL
Out MOTOR_RIGHT, AL
Jmp MainLoop

BackwardAction:
Mov AL, BACKWARD
Out MOTOR_LEFT, AL
Out MOTOR_RIGHT, AL
Jmp MainLoop

LeftTurnAction:
Mov AL, STOP ; Dừng bánh xe trái
Out MOTOR_LEFT, AL
Mov AL, FORWARD ; Quay bánh xe phải
Out MOTOR_RIGHT, AL
Jmp MainLoop

RightTurnAction:
Mov AL, FORWARD ; Quay bánh xe trái
Out MOTOR_LEFT, AL
Mov AL, STOP ; Dừng bánh xe phải
Out MOTOR_RIGHT, AL
Jmp MainLoop

End Start

```

### Ví dụ 1: Điều khiển robot di chuyển theo một chuỗi hành động lập trình sẵn

Trong ví dụ này, robot sẽ thực hiện một chuỗi các hành động được lập trình trước, ví dụ như đi thẳng một khoảng thời gian, rẽ phải, rồi đi thẳng tiếp.

```

.MODEL small
.STACK 100h

```

```

MOTOR_LEFT EQU 0AH ; Cỗng điều khiển động cơ trái
MOTOR_RIGHT EQU 0BH ; Cỗng điều khiển động cơ phải

; Giá trị điều khiển động cơ (ví dụ)
FORWARD EQU 01h ; Quay thuận
BACKWARD EQU 02h ; Quay nghịch
STOP EQU 00h ; Dừng

.CODE
Start:
    ; Thiết lập đoạn dữ liệu
    Mov AX, @Data
    Mov DS, AX

    ; Hành động 1: Đi thẳng trong 2 giây
    Mov AL, FORWARD
    Out MOTOR_LEFT, AL
    Out MOTOR_RIGHT, AL
    Call Delay2s

    ; Hành động 2: Rẽ phải trong 1 giây
    Mov AL, FORWARD
    Out MOTOR_LEFT, AL
    Mov AL, STOP
    Out MOTOR_RIGHT, AL
    Call Delay1s

    ; Hành động 3: Đi thẳng trong 3 giây
    Mov AL, FORWARD
    Out MOTOR_LEFT, AL
    Out MOTOR_RIGHT, AL
    Call Delay3s

    ; Dừng robot
    Mov AL, STOP
    Out MOTOR_LEFT, AL
    Out MOTOR_RIGHT, AL

    ; Kết thúc chương trình
    Mov AH, 4Ch
    Int 21h

Delay1s:
    ; Hàm tạo độ trễ 1 giây (tương tự như các ví dụ trước)
    Mov CX, 1000 ; Giả sử 1000 vòng lặp tương ứng 1 giây
DelayLoop1s:
    Loop DelayLoop1s
    Ret

```

```

Delay2s:
    ; Hàm tạo độ trễ 2 giây
    Push CX
    Mov CX, 2
DelayLoop2sOuter:
    Call Delay1s
    Loop DelayLoop2sOuter
    Pop CX
    Ret

Delay3s:
    ; Hàm tạo độ trễ 3 giây
    Push CX
    Mov CX, 3
DelayLoop3sOuter:
    Call Delay1s
    Loop DelayLoop3sOuter
    Pop CX
    Ret

End Start

```

## 5.8 Các phương pháp vào ra dữ liệu

### 5.8.1 Giới thiệu

Chương Kỹ thuật trao đổi dữ liệu giữa máy vi tính và các thiết bị ngoại vi được gọi là vào/ra hay I/O (Input/Output). Thiết bị liên lạc với máy vi tính qua các giao tiếp vào/ra. Người dùng có thể nhập chương trình và dữ liệu bằng cách dùng bàn phím và chạy các chương trình để lấy kết quả. Như vậy, các thiết bị vào/ra kết nối với máy vi tính cung cấp cách thức liên lạc tiện lợi với thế giới bên ngoài. Các thiết bị vào/ra phổ biến gồm có bàn phím, màn hình, máy in và ổ đĩa cứng.

Đặc tính của các thiết bị vào/ra thường khác với đặc tính của máy vi tính. Chẳng hạn như tốc độ của các thiết bị thường chậm hơn máy vi tính, độ dài từ (word) và định dạng dữ liệu cũng khác nhau giữa thiết bị và máy tính. Để hai bên có thể liên lạc được với nhau cần có các mạch giao tiếp giữa thiết bị vào/ra và máy tính. Giao tiếp cung cấp trao đổi dữ liệu vào/ra qua buýt vào/ra. Buýt này thông thường chuyên tải 3 loại tín hiệu: địa chỉ thiết bị, dữ liệu và lệnh.

Có ba phương pháp trao đổi dữ liệu giữa máy vi tính và các thiết bị vào/ra: vào/ra lập trình (programmed I/O) hay thăm dò, vào/ra bằng ngắt và truy nhập trực tiếp bộ nhớ (Direct Memory Access DMA). Dùng vào/ra thăm dò, vi xử lý chạy một chương trình thực hiện toàn bộ các trao đổi dữ liệu giữa vi xử lý và các thiết bị bên ngoài. Đặc tính chủ yếu của phương pháp này là thiết bị thực hiện các chức năng được chỉ định bởi chương trình bên trong bộ nhớ của vi xử lý. Nói cách khác, vi xử lý điều khiển hoàn toàn các trao đổi dữ liệu.

Với vào/ra bằng ngắn, thiết bị có thể bắt vi xử lý dừng việc thực hiện chương trình hiện thời để thiết bị có thể chạy chương trình khác gọi là chương trình phục vụ ngắn. Chương trình này đáp ứng yêu cầu của thiết bị. Sau khi kết thúc chương trình này, câu lệnh trả về từ ngắn để trả lại quyền điều khiển cho chương trình bị ngắn.

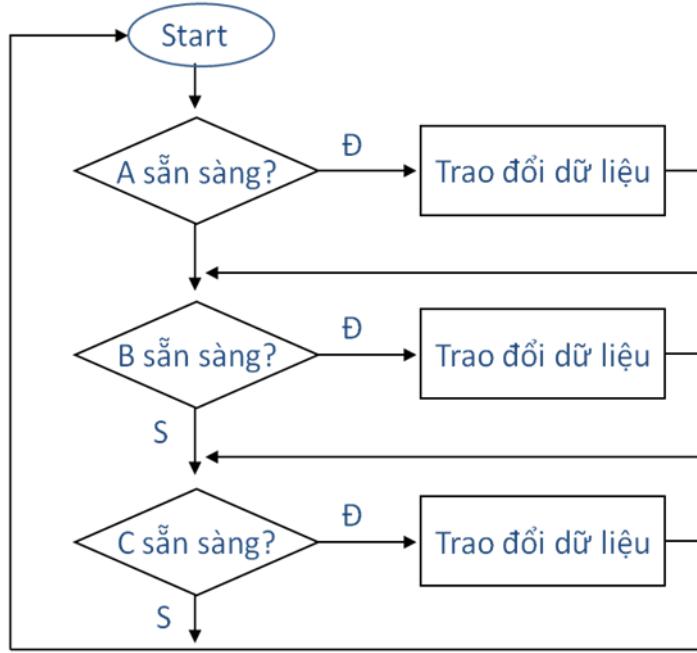
Truy nhập bộ nhớ trực tiếp là kỹ thuật vào/ra mà trong đó dữ liệu có thể được trao đổi giữa bộ nhớ của máy tính với thiết bị như ổ cứng mà không cần sự can thiệp của vi xử lý. Thông thường, phương pháp này cần sử dụng vi mạch đặc biệt gọi là vi mạch DMA.

Trong máy tính sử dụng hệ điều hành, người dùng thường làm việc với thiết bị vào/ra ảo. Người dùng không phải quan tâm tới các đặc tính của thiết bị. Thay vào đó, người dùng thực hiện trao đổi dữ liệu thông qua các dịch vụ vào/ra do hệ điều hành cung cấp. Về căn bản, hệ điều hành đóng vai trò giao tiếp giữa chương trình người dùng và phần cứng thiết bị. Hệ điều hành hỗ trợ tạo nhiều các thiết bị lô-gíc hay thiết bị vào/ra ảo và cho phép người dùng liên lạc trực tiếp với các thiết bị này. Chương trình người dùng hoàn toàn không biết được việc ánh xạ giữa thiết bị ảo và thiết bị vật lý. Như vậy, khi thiết bị ảo gán cho thiết bị vật lý khác thì không phải thay đổi chương trình người dùng.

### **5.8.2 Vào/ra bằng phương pháp thăm dò**

Vấn đề điều khiển vào/ra dữ liệu sẽ trở nên đơn giản nếu thiết bị ngoại nút nào cũng sẵn sàng để làm việc với CPU. Ví dụ, bộ phận đo 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 hiển thị thông tin đó. Như vậy khi CPU muốn có thông tin về nhiệt độ của hệ thống thì nó chỉ việc đọc cổng phôi 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 sẵn 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 chia sẻ thời gian hoạt động cho việc trao đổi dữ liệu và việc kiểm tra trạng thái sẵn sàng của thiết bị ngoại vi thông qua các tín hiệu móc nối (handshake signal).



Hình 5. 17. Vào/ra lập trình với nhiều thiết bị

Các mạch kết nối trong **ERROR! REFERENCE SOURCE NOT FOUND.** và **REF \_REF268425050 \H \\* MERGEFORMAT ERROR! REFERENCE SOURCE NOT FOUND.** là các ví dụ tiêu biểu cho phương pháp vào/ra lập trình. Với bàn phím, CPU liên tục kiểm tra trạng thái các phím và nếu có phím được bấm CPU sẽ đọc thông tin trên cổng vào để xác định phím nào được bấm. Với bộ hiển thị LED, CPU liên tục đưa dữ liệu ra các cổng ra để thiết bị có thể hiển thị các thông tin.

Khi số lượng các thiết bị vào/ra tăng lên thì thời gian dành cho việc xác định trạng thái của thiết bị vào/ra cũng tăng lên nhanh chóng như trong **ERROR! REFERENCE SOURCE NOT FOUND..** CPU kiểm tra lần lượt các thiết bị để phát hiện trạng thái sẵn sàng trao đổi dữ liệu của từng thiết bị và thực hiện các lệnh trao đổi dữ liệu. Các thiết bị được kiểm tra thăm dò theo trật tự ngẫu nhiên hoặc theo mức độ ưu tiên của các thiết bị. Cách thức này dù đơn giản song có nhược điểm là thời gian quét trạng thái của các thiết bị chiếm tỷ trọng rất đáng kể trong suốt quá trình vào/ra nhất là khi các thiết bị chưa có dữ liệu để trao đổi.

### 5.8.3 Vào/ra bằng ngắn

#### 5.8.3.1 Giới thiệu

Nhược điểm của vào/ra thăm dò là máy tính cần kiểm tra bit trạng thái bằng cách chờ đáp ứng của thiết bị vào/ra. Với các thiết bị chậm, việc chờ làm giảm khả năng xử lý dữ liệu khác của máy tính. Kỹ thuật ngắn cho phép giải quyết vấn đề này.

Với vào/ra bằng ngắn, thiết bị khởi xướng việc trao đổi vào/ra. Thiết bị được nối với chân tín hiệu ngắn (INT) trên vi mạch của vi xử lý. Khi thiết bị cần trao đổi dữ liệu, thiết bị sinh ra tín hiệu ngắn. Máy tính sẽ hoàn thành câu lệnh hiện thời và lưu nội dung của bộ đếm chương trình và các thanh ghi trạng thái. Sau đó, máy tính tự động nạp địa

chỉ của chương trình phục vụ ngắn vào thanh ghi đếm chương trình. Chương trình này thường do người dùng viết và máy tính thực hiện chương trình này để trao đổi dữ liệu với thiết bị. Câu lệnh cuối của chương trình này khôi phục thanh ghi đếm chương trình bị dừng và thanh ghi trạng thái của vi xử lý.

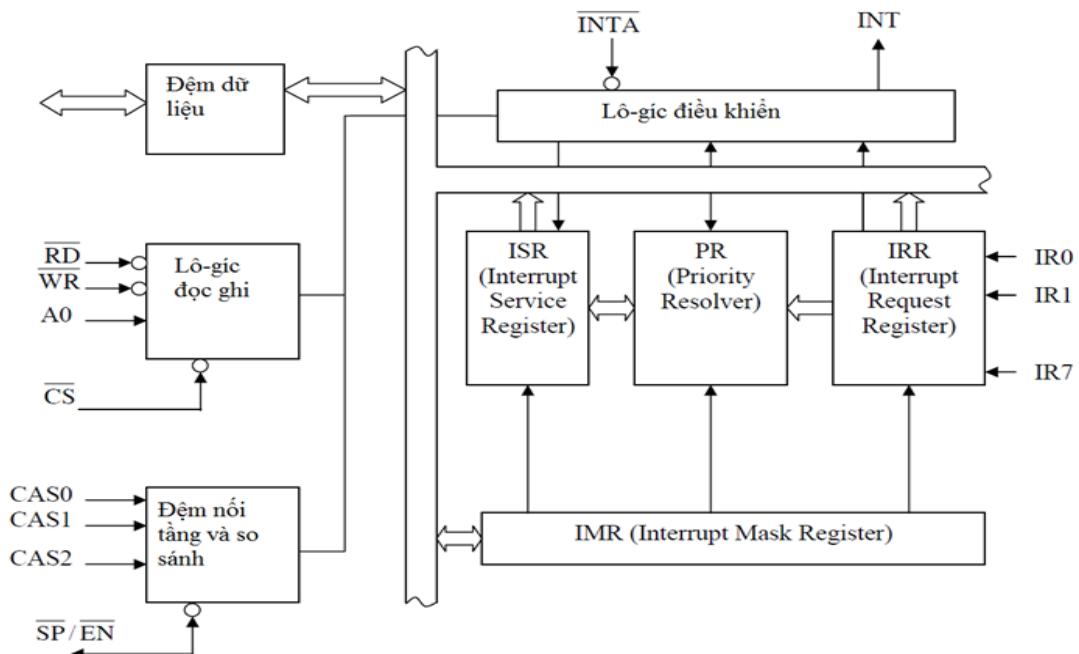
Vi xử lý thường cung cấp một hay nhiều tín hiệu ngắn trên vi mạch. Như vậy, để xử lý các yêu cầu ngắn từ nhiều thiết bị cần có cơ chế đặc biệt. Thường có các cách sau: thăm dò và quay vòng. Thăm dò sử dụng phần mềm chung cho tất cả các thiết bị vì vậy làm giảm tốc độ đáp ứng ngắn. Khi có tín hiệu ngắn phần mềm thăm dò kiểm tra trạng thái của các thiết bị theo thứ tự ưu tiên bắt đầu với thiết bị được ưu tiên cao nhất. Khi xác định được thiết bị yêu cầu trao đổi dữ liệu, phần mềm thăm dò chuyển quyền điều khiển cho phần mềm phục vụ ngắn.

#### 5.8.3.2 Bộ xử lý ngắn ưu tiên 8259

Trong trường hợp có nhiều yêu cầu ngắn che được từ bên ngoài phải phục vụ máy tính thường dùng vi mạch có sẵn 8259A để giải quyết vấn đề ưu tiên. Vi mạch 8259A được gọi là mạch điều khiển ngắn lập trình được (Programmable Interrupt Controller, PIC). Đó là một vi mạch cỡ lớn có thể xử lý trước được 8 yêu cầu ngắn với các mức ưu tiên khác nhau để tạo ra một yêu cầu đưa đến đầu vào INTR (yêu cầu ngắn che được của CPU 8086/8088. Nếu nối tầng 1 mạch 8259A chủ với 8 mạch 8259A thợ ta có thể nâng tổng số các yêu cầu ngắn với các mức ưu tiên khác nhau lên thành 64.

#### Các khối chức năng chính của 8259A

Sơ đồ khối của 8259A được trình bày trong **Error! Reference source not found.** dưới đây



Hình 5. 18. Sơ đồ khối 8259

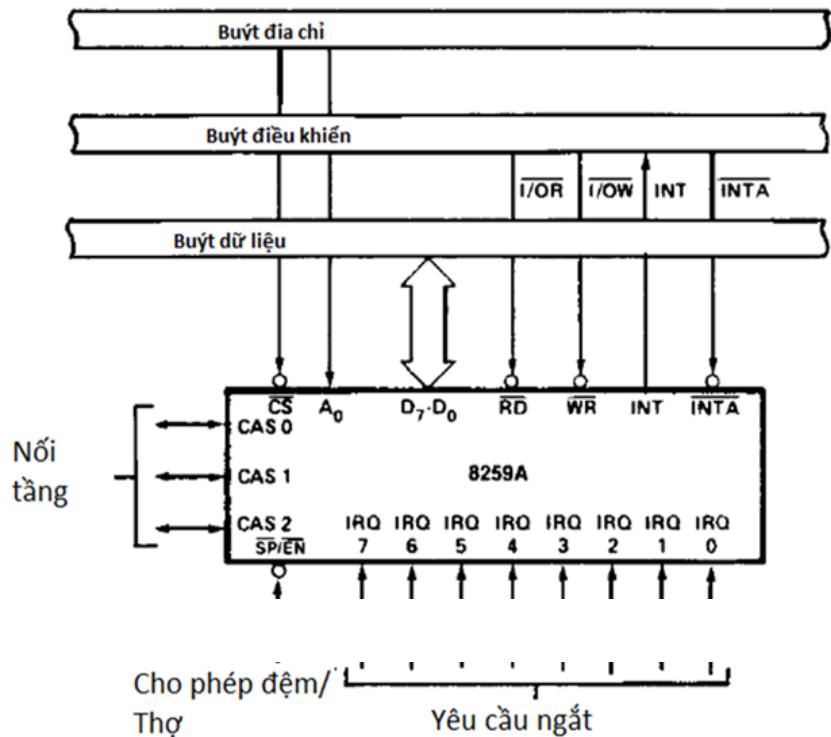
- Thanh ghi IRR: ghi nhớ các yêu cầu ngắt có tại đầu vào IRI.
- Thanh ghi ISR: ghi nhớ các yêu cầu ngắt đang được phục vụ trong số các yêu cầu ngắt IRI.
- Thanh ghi IMR: ghi nhớ mặt nạ ngắt đối với các yêu cầu ngắt IRI.
- Logic điều khiển: khói này có nhiệm vụ gửi yêu cầu ngắt tới INTR của 8086/8088 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 buýt dữ liệu.
- Đệm buýt dữ liệu: dùng để phối ghép 8259A với buýt 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 8086/8080. Ta có thể thấy rõ và hiểu được ý nghĩa của chúng ngay trên **Error! Reference source not found..** Ngoài các tín hiệu này ra, còn có một số tín hiệu đặc biệt khác của 8259A cần phải giới thiệu thêm gồm:

- CAS<sub>0</sub>-CAS<sub>2</sub> [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 buýt 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 buýt dữ liệu thì chân này là tín hiệu ra  $\overline{EN}$  dùng mở đệm buýt dữ liệu để 8086/8088 và 8259A thông vào buýt 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 8086/8088.
- $\overline{INTA}$  [I]: nối với tín hiệu báo chấp nhận ngắt  $\overline{INTA}$  của CPU.

**Error! Reference source not found.**9 dưới đây thể hiện ghép nối 8259A với hệ thống buýt của 8086/8088. Nếu hệ vi xử lý 8086/8088 làm việc ở chế độ MAX thường ta phải dùng mạch điều khiển buýt 8288 và các đệm buýt để cung cấp các tín hiệu thích hợp cho buýt hệ thống. Mạch 8259A phải làm việc ở chế độ có đệm để nối được với buýt hệ thống này.



Hình 5. 19. Ghép nối 8259 với buýt 8086/8088

### Lập trình cho PIC 8259A

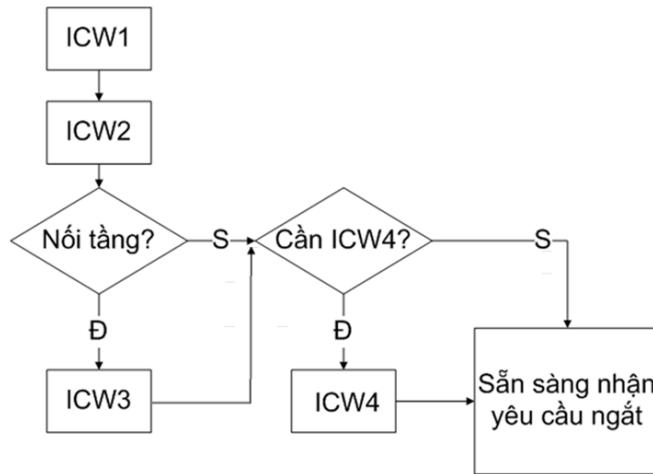
Để mạch PIC 8259A có thể hoạt động được theo yêu cầu, sau khi bật nguồn cấp điện PIC cần phải được lập trình bằng cách ghi vào các thanh ghi (tương đương với các cổng) bên trong *các từ điều khiển khởi đầu (ICW)* và *tiếp sau đó là các từ điều khiển hoạt động (OCW)*.

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 dưới đây.

#### 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. **Error! Reference source not found.** dưới đây thể hiện thứ tự ghi và điều kiện để ghi các điều khiển ICW vào 8259A.

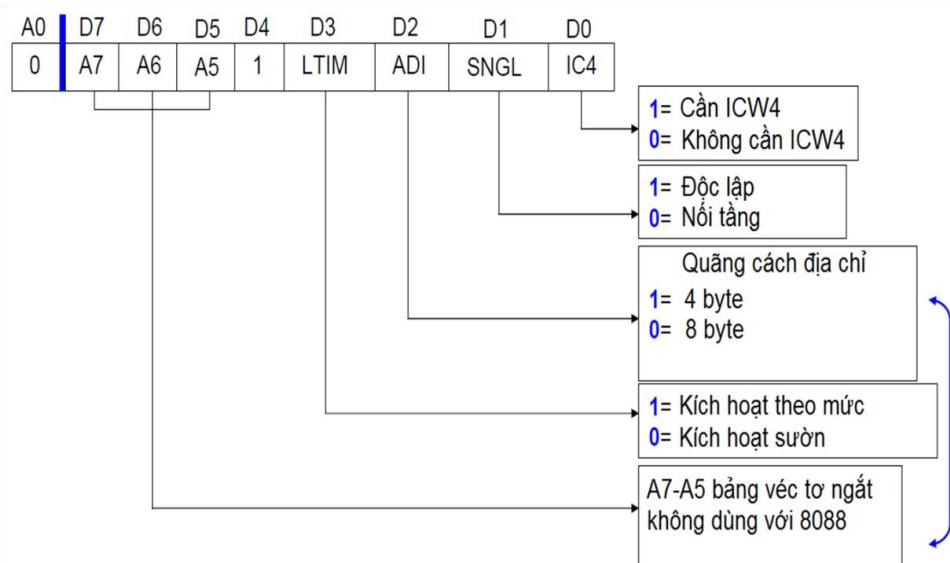


Hình 5. 20. Trình tự sử dụng các thanh ghi khởi đầu

Để xác định các thanh ghi bên trong ta cần sử dụng tín hiệu địa chỉ  $A_0$  và thứ tự ghi để ghi dữ liệu cho các từ điều khiển. Ví dụ  $A_0 = 0$  là dấu hiệu để nhận biết rằng ICW1 được đưa vào thanh ghi có địa chỉ chẵn trong PIC, còn khi  $A_0 = 1$  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**

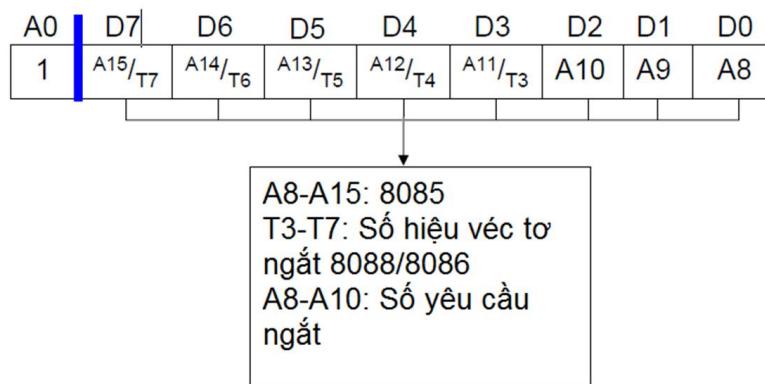
Bit  $D_0$  của ICW1 minh họa trong **Error! Reference source not found.** quyết định 8259A sẽ được nối với họ vi xử lý nào. Để làm việc với hệ 16-32bit (8088 hoặc họ x86) thì ICW nhất thiết phải có  $ICW4 = 0$  (và như vậy các bit của ICW4 sẽ bị xóa về 0). Các bit còn lại của ICW1 định nghĩa cách thức 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.



Hình 5. 21. Dạng thức của ICW1

- o ICW2

**Error! Reference source not found.** mô tả dạng thức của ICW2, được sử dụng để chọn kiểu ngắt (số hiệu ngắt) cho các đầu vào yêu cầu ngắt của mạch 8259A. 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 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.



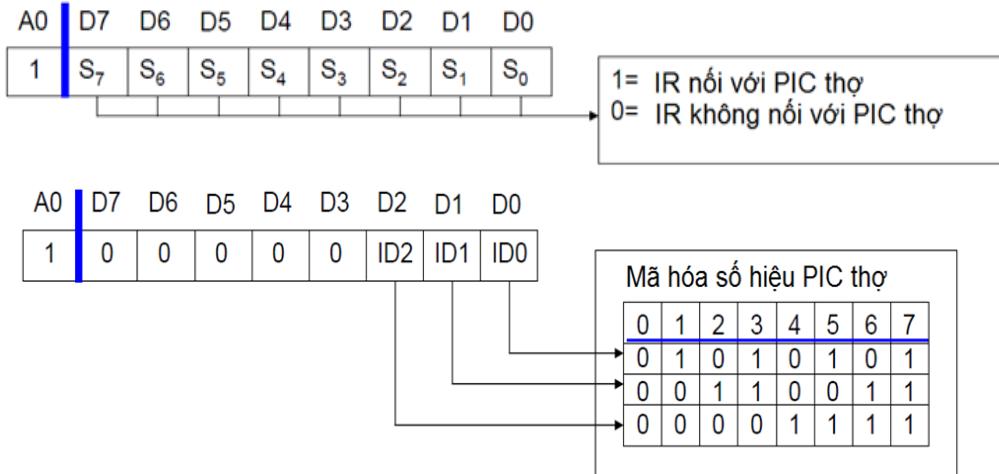
Hình 5. 22. Dạng thức của ICW2

- o ICW3

**Error! Reference source not found.** mô tả dạng thức của ICW3 dành cho mạch 8259A chủ và thợ, được sử dụng khi bit SNGL trong ICW1 có giá trị 0, tức hệ thống bao gồm nhiều mạch 8259A hoạt động ở chế độ nối tầng. 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 đầu 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 CAS<sub>2</sub>-CAS<sub>0</sub>. Nếu bằng nhau thì số hiệu ngắt sẽ được đưa ra buýt khi có INTA.



Hình 5. 23. Dạng thức của ICW3 cho mạch chủ và thợ

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 với nhau ta sẽ phải gán các từ điều khiển khởi đầu như sau: ICW3 = 03H cho mạch chủ. ICW3 = 00H cho mạch thợ thứ nhất và ICW3 = 01H cho mạch thợ thứ hai.

#### o ICW4

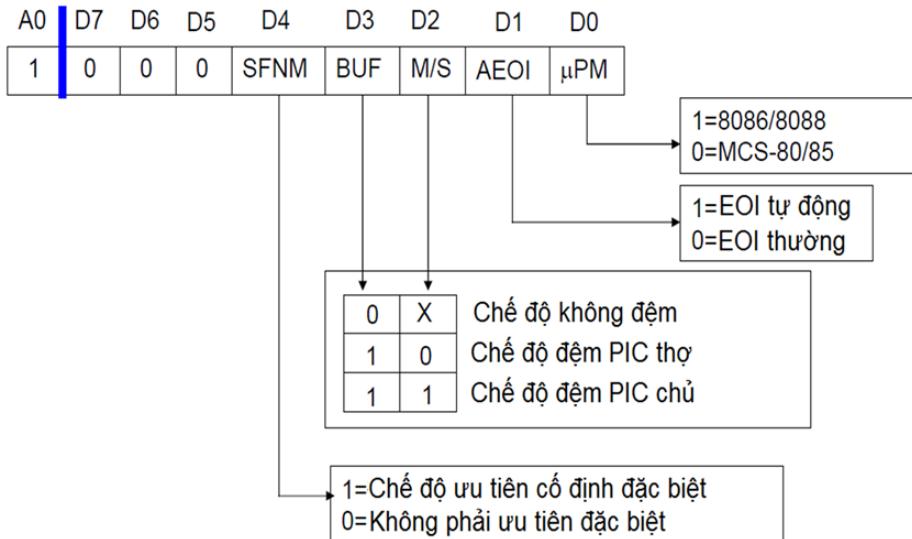
**Error! Reference source not found.** mô tả dạng thức của ICW4, được sử dụng khi trong từ điều khiển ICW1 có chỉ định thêm ICW4 (IC4 = 1). Từ điều khiển khởi đầu này chỉ dùng đến khi trong từ điều khiển ICW1 có IC4 = 1 (cần thêm ICW4)

Bit μPM cho ta khả năng chọn loại vi xử lý để làm việc với 8259A. Bit μPM = 1 cho phép các bộ vi xử lý từ 8086/88 hoặc cao hơn làm việc với 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 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 với 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 tất cả các yêu cầu khác vượt mức ưu tiên cao hơn có thể ngắt yêu cầu khác với mức ưu tiên thấp hơn.

.



Hình 5. 24. Dạng thức của ICW4

Bit BUF cho phép định nghĩa mạch 8259A để làm việc với CPU trong trường hợp có đệm hoặc không có đệm nối với buýt 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 qua buýt hệ thống.

Bit AEOI = 1 cho phép chọn cách kết thúc yêu cầu ngắt tự động. Khi AEOI = 1 thì 8259A tự động xóa ISR<sub>i</sub> = 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.

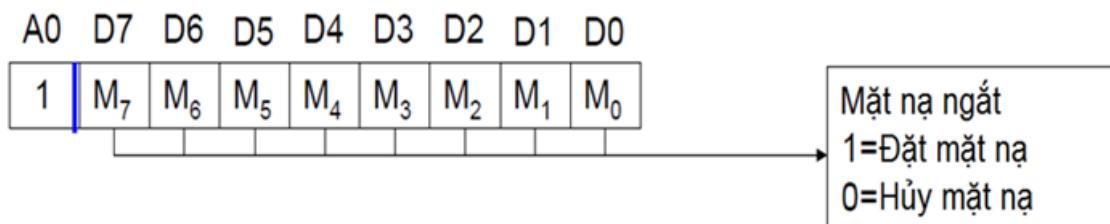
### Các từ điều khiển hoạt động OCW

Các từ điều 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 này sẽ được ghi vào các thanh ghi trong PIC khi A0 = 0, trừ OCW1 được ghi khi A0 = 1.

#### o OCW1

**Error! Reference source not found.** minh họa cấu trúc của OCW1, dùng để ghi giá trị của các bit mặt nạ ngắt vào thanh ghi IMR (Interrupt Mask Register). 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)



Hình 5. 25. OCW1 Trạng thái yêu cầu ngắn

- OCW2

**Error! Reference source not found.** minh họa trạng thái ngắn và chế độ quay mức ưu tiên trong OCW2 của mạch 8259A. Các bit R., SL, và EOI phối hợp với nhau cho phép chọn ra các cách thức kết thúc yêu cầu ngắn khác nhau. Một vài cách thức yêu cầu ngắn còn tác động tới các yêu cầu ngắn được chỉ định danh với mức ưu tiên được giải mã hóa của 3 bit L<sub>2</sub>, L<sub>1</sub>, L<sub>0</sub>.

Trước khi nói về các cách kết thúc yêu cầu ngắn cho các chế độ ta cần mở đầu ngoặc ở đây để giới thiệu các chế độ làm việc của 8259A.

- 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 tại một thời điểm chỉ có một yêu cầu ngắn i được phục vụ (bit ISRI = 1) lúc này tất cả các yêu cầu khác với mức ưu tiên thấp hơn đều bị cấm, tất cả các yêu cầu khác với mức ưu tiên thấp hơn đều có thể ngắn yêu cầu khác với mức ưu tiên thấp hơn.

- Chế độ quay mức ưu tiên (ưu tiên luân phiên) tự động:

Ở chế độ này sau khi một yêu cầu ngắn 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 khác có cơ hội được phục vụ.

- Chế độ quay (đổi) mức ưu tiên chỉ định danh:

Ở chế độ này ta cần chỉ rõ (đích danh) đầu vào IRi nào, với i=L<sub>2</sub>L<sub>1</sub>L<sub>0</sub>, được gán mức ưu tiên thấp nhất, đầu vào IR<sub>i+1</sub> sẽ được tự động gán mức ưu tiên cao nhất.

Trước khi quay

Giả sử IR4 có mức ưu tiên cao

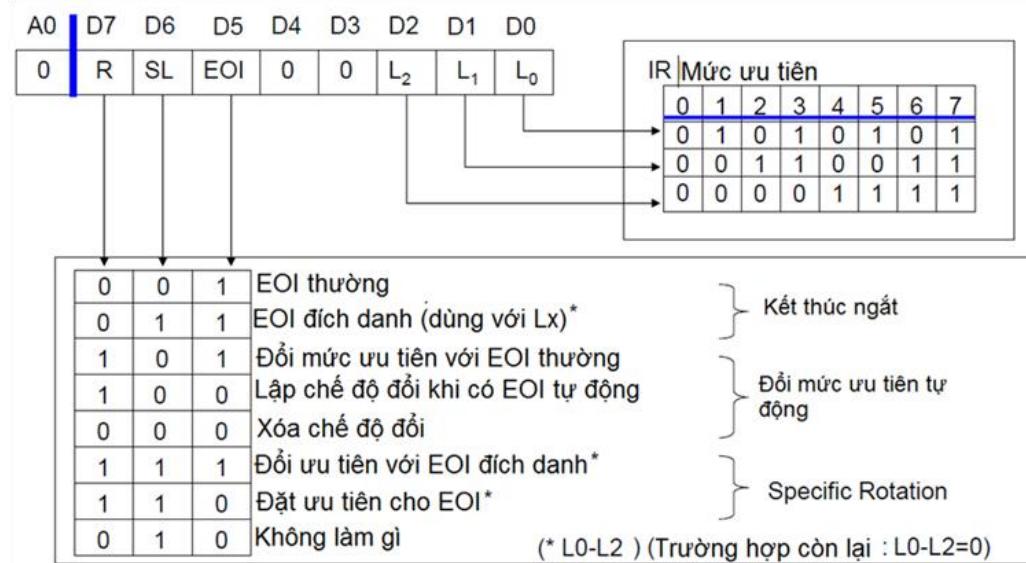
Trạng thái ngắt <small>(ISR)</small>	IS7	IS6	IS5	IS4	IS3	IS2	IS1	IS0
Mức ưu tiên	7	6	5	4	3	2	1	0
Sau khi quay	Trạng thái ngắt	0	1	0	0	0	0	0
Mức ưu tiên	2	1	0	7	6	5	4	3
Sau khi quay	Trạng thái ngắt	0	1	0	0	0	0	0

Hình 5. 26. Trạng thái ngắt và chế độ quay mức ưu tiên

Trở lại các vấn đề liên quan đến OCW, ta sẽ nói rõ việc các bit R, SL và EOI phối hợp với nhau như thế nào để tạo ra các lệnh quy định các cách thức kết thúc yêu cầu ngắt cho các chế độ làm việc khác nhau đã nói đến ở phần trên.

1. Kết thúc yêu cầu ngắt thường: chương trình còn phục vụ ngắt phải có lệnh EOI đặt trước lệnh trở về IRET cho 8259A. Vì mạch này 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 con phục vụ ngắt phải có lệnh EOI chỉ đích danh đặt trước lệnh trở về IRET cho 8259A. 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. 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 được gán mức ưu tiên cao nhất.

Có thể theo dõi cách thức 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 **Error! Reference source not found..**



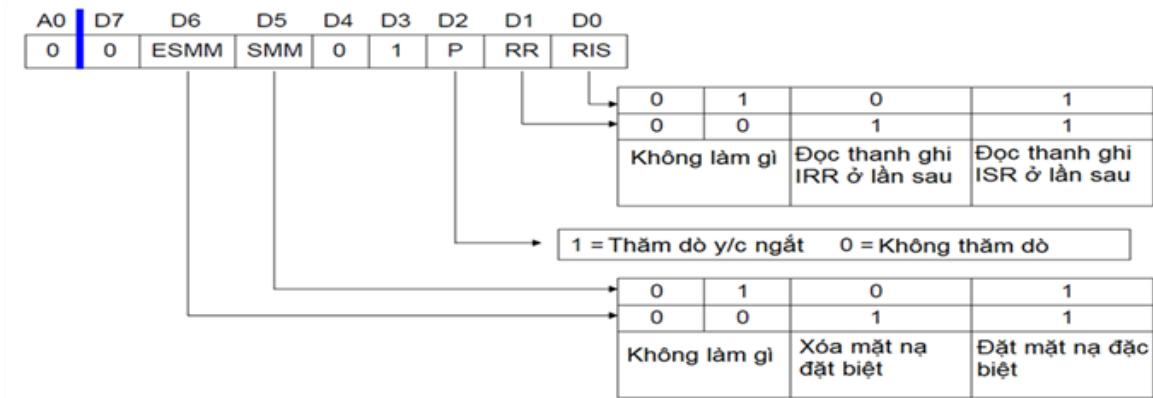
Hình 5. 27. OCW2 xác định xử lý các yêu cầu ngắt

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 chỉ đích danh cho 8259A đặt trước lệnh trả về IRET. Mạch 8259A sẽ xóa bit ISR<sub>i</sub> của yêu cầu ngắt tương ứng và gán luôn mức ưu tiên thấp nhất cho đầu vào IR<sub>i</sub>, với  $i = L_2 L_1 L_0$ .
6. Lập mức ưu tiên: chế độ này cho phép thay đổi mức ưu tiên cố định hoặc mức ưu tiên gán trước đó bằng cách gán mức ưu tiên thấp nhất cho yêu cầu ngắt IR<sub>i</sub> chỉ đích danh với tổ hợp  $i = L_2 L_1 L_0$ . Yêu cầu ngắt IR<sub>i+1</sub> sẽ được gán mức ưu tiên cao nhất.

### o OCW3

**Error! Reference source not found.** minh họa dạng thức của OCW3, được sử dụng để điều khiển các hoạt động đặc biệt trong mạch 8259A. 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 nhất cùng mã của đầu vào đó và.
- Thao tác với mặt nạ đặc biệt.



Hình 5. 28. OCW3

**Error! Reference source not found.** minh họa cấu trúc của các thanh ghi IRR (Interrupt Request Register) và ISR (In-Service Register) trong mạch 8259A, cho phép quản lý và giám sát các yêu cầu ngắn. Các thanh ghi IRR và ISR có thể đọc được sau khi nạp vào 8259A từ điều khiển OCW3 với bit RR = 1: bit RIS = 0 sẽ cho phép đọc IRR. Bit RIS = 1 sẽ cho phép đọc ISR. Dạng thức của các thanh ghi này biểu diễn trên hình dưới đây.

IR7	IR6	IR5	IR4	IR3	IR2	IR1	IR0
D7	D6	D5	D4	D3	D2	D1	D0

- ▶ 0 = Có yêu cầu ngắn
- ▶ 1 = Không có yêu cầu ngắn

IS7	IS6	IS5	IS4	IS3	IS2	IS1	IS0
D7	D6	D5	D4	D3	D2	D1	D0

- ❖ 0 = Yêu cầu ngắn IRI không được phục vụ
- ❖ 1 = Yêu cầu ngắn IRI đang được phục vụ

Hình 5. 29. Thanh ghi IRR và ISR

**Error! Reference source not found.** minh họa dạng thức dữ liệu được trả về từ quá trình thăm dò trạng thái của mạch 8259A. Quá trình này được thực hiện khi OCW3 có bit P = 1, cho phép kiểm tra thông tin về yêu cầu ngắn với mức ưu tiên cao nhất đang hoạt động.

D7	D6	D5	D4	D3	D2	D1	D0
1: có ngắn	X	x	X	x	Số hiệu yêu cầu ngắn		

Hình 5. 30. Dạng thức từ thăm dò trạng thái

Có thể gọi đây là chế độ thăm dò yêu cầu ngắt và chế độ này thường được 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 ngắt hiện tại ta cần làm các thao tác lần lượt như sau:

- Cảm các yêu cầu ngắt bằng lệnh CLI
- Ghi từ lệnh OCW3 với bit P = 1
- Đọc từ thăm dò trạng thái yêu cầu ngắt trên buýt 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 yêu cầu ngắt cảm (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 cảm 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 vúi bit SMM = 0. Mặt nạ đặc biệt không ảnh hưởng tới các yêu cầu ngắt với mức ưu tiên cao hơn)

### **Hoạt động của 8086/8088 với 8259A**

Cuối cùng để có cái nhìn một cách có hệ thống về hoạt động của hệ vi xử lý với CPU 8086/8088 và PIC 8259A khi có 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 8086/8088.
2. 8086/8088 đưa ra xung INTA đầu đến 8259A
3. 8259A dùng xung 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. 8086/8088 đưa ra xung INTA thứ hai đến 8259A
5. Xung INTA thứ hai khiến 8259A đưa ra buýt dữ liệu 1-byte chứa thông tin về số hiệu ngắt của yêu cầu ngắt vừa được nhận biết.
6. 8086/8088 dùng số hiệu ngắt để tính ra địa chỉ ngắt của vectơ ngắt tương ứng.
7. 8086/8088 cất FR, xóa các cờ IF và TF và cất địa chỉ trả về CS: IP vào ngăn xếp.
8. 8086/8088 lấy địa chỉ CS: IP của chương trình phục vụ ngắt từ bảng vectơ ngắt và thực hiện chương trình đó.

### **5.8.4 Vào/rã bằng truy nhập trực tiếp bộ nhớ (Direct memory Access)**

#### **5.8.4.1 Khái niệm về phương pháp truy nhập trực tiếp vào bộ nhớ**

Trong các cách điều khiển việc trao đổi dữ liệu giữa thiết bị ngoại vi và hệ vi xử lý bằng cách thăm dò trạng thái sẵn sàng của thiết bị ngoại vi hay bằng cách ngắt bộ vi xử lý đã trình bày ở các chương trước, dữ liệu thường được chuyển từ bộ nhớ qua bộ vi xử lý để rồi từ đó ghi vào thiết bị ngoại vi hoặc ngược lại, từ thiết bị ngoại vi nó

được đọc vào bộ vi xử lý để rồi từ đó được chuyển đến bộ nhớ. Vì thế tốc độ trao đổi dữ liệu phụ thuộc rất nhiều vào tốc độ thực hiện của các lệnh MOV, IN và OUT của bộ vi xử lý và do đó việc trao đổi dữ liệu không thể tiến hành nhanh được.

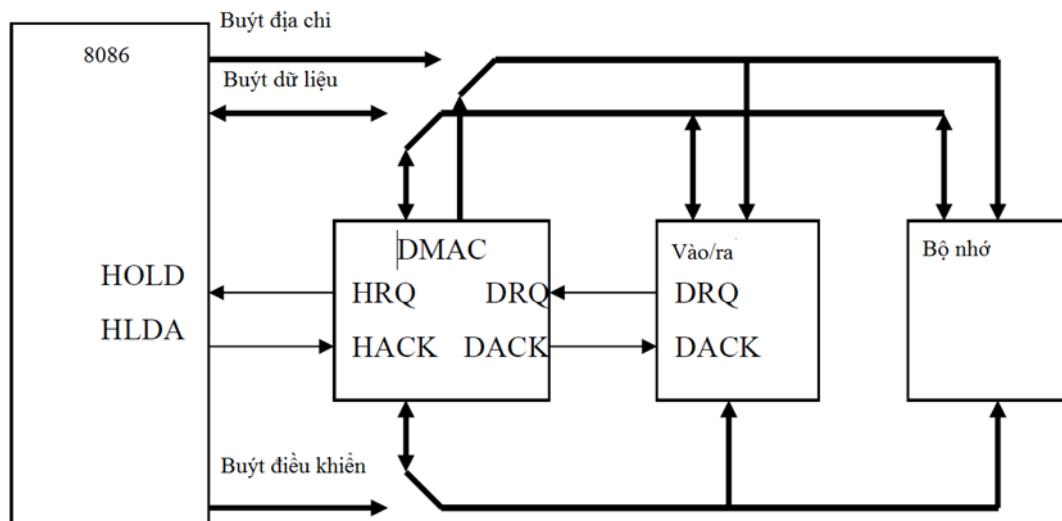
Trong thực tế có những khi ta cần trao đổi dữ liệu thật nhanh với thiết bị ngoại vi: như khi cần đưa dữ liệu hiện thị ra màn hình hoặc trao đổi dữ liệu với bộ điều khiển đĩa. Trong các trường hợp đó ta cần có khả năng ghi /đọc dữ liệu trực tiếp với bộ nhớ thì mới đáp ứng được yêu cầu về tốc độ trao đổi dữ liệu. Để làm được điều này các hệ vi xử lý nói chung đều phải dùng thêm mạch chuyên dụng để điều khiển việc truy nhập trực tiếp vào bộ nhớ DMAC (Direct Memory Access Controller)

Ví dụ dưới đây minh họa điều này. Trong khi một mạch DMAC như 8237A của Inter có thể điều khiển việc chuyển một byte trong một mảng dữ liệu từ bộ nhớ ra thiết bị ngoại vi chỉ hết 4 chu kỳ đồng hồ thì bộ vi xử lý 8086/8088 phải làm hết cỡ 4 chu kỳ:

LAP:	MOV AL, (SI ) ; 10	; số chu kỳ đồng hồ
	OUT PORT, AL ; 10	
	INC SI ; 2	
	LOOP LAP ; 17	
		; CỘNG: 39 chu kỳ

Để hỗ trợ cho việc trao đổi dữ liệu với thiết bị ngoại vi bằng cách truy nhập trực tiếp vào bộ nhớ. CPU thường có tín hiệu yêu cầu treo HOLD để mỗi khi thiết bị cần dùng buýt cho việc trao đổi dữ liệu với bộ nhớ thì thông qua chân này mà báo cho CPU biết. Đến lượt CPU, khi nhận được yêu cầu treo thì nó tự treo lên (tự tách ra khỏi hệ thống bằng cách đưa các bit vào trạng thái trở kháng cao) và đưa xung HLDA ra ngoài để thông báo CPU cho phép sử dụng buýt.

Sơ đồ khối của một hệ vi xử lý có khả năng trao đổi dữ liệu theo kiểu DMA được thể hiện trên **Error! Reference source not found.** dưới đây.



Hình 5. 31. Hệ vi xử lý với DMAC

Ta nhận thấy trong hệ thống này, khi CPU tự tách ra khỏi hệ thống bằng cách tự treo (ứng với vị trí hiện thời của các công tắc chuyển mạch), DMAC phải chịu trách nhiệm điều khiển toàn bộ hoạt động trao đổi dữ liệu của hệ thống. Như vậy, DMAC phải có khả năng tạo ra được các tín hiệu điều khiển cần thiết giống như các tín hiệu của CPU và bản thân nó phải là một thiết bị lập trình được. Quá trình hoạt động của hệ thống trên có thể được tóm tắt như sau:

Khi thiết bị ngoại vi có yêu cầu trao đổi dữ liệu kiểu DMA với bộ nhớ, nó đưa yêu cầu DREQ=1 đến DMAC, DMAC sẽ đưa yêu cầu treo HRQ=1 đến chân HOLD của CPU. Nhận được yêu cầu treo, CPU sẽ treo các buýt của mình và trả lời chấp nhận treo qua tín hiệu HLDA=1 đến chân HACK của DMAC, DMAC sẽ thông báo cho thiết bị ngoại vi thông qua tín hiệu DACK=1 là nó cho phép thiết bị ngoại vi trao đổi dữ liệu kiểu DMA. Khi quá trình DMA kết thúc thì DMAC đưa ra tín hiệu HRQ=0.

### 5.8.5 Các phương pháp trao đổi dữ liệu

Trong thực tế tồn tại 3 kiểu trao đổi dữ liệu bằng cách truy nhập trực tiếp vào bộ nhớ như sau:

- Treo CPU một khoảng thời gian để trao đổi cả mảng dữ liệu.
- Treo CPU để trao đổi từng byte.
- Tận dụng thời gian không dùng buýt để trao đổi dữ liệu.

#### 5.8.5.1 Trao đổi cả mảng dữ liệu

Trong chế độ này CPU bị treo trong suốt quá trình trao đổi mảng dữ liệu. Chế độ này được dùng khi ta có nhu cầu trao đổi dữ liệu với ổ đĩa hoặc đưa dữ liệu ra hiển thị. Các bước để chuyển một mảng dữ liệu từ bộ nhớ ra thiết bị ngoại vi:

1. CPU phải ghi từ điều khiển và từ chế độ làm việc vào DMAC để quy định cách thức làm việc, địa chỉ đầu của mảng nhớ, độ dài của mảng nhớ, . . .
2. Khi thiết bị ngoại vi có yêu cầu trao đổi dữ liệu, nó đưa DREQ =1 đến DMAC.
3. DMAC đưa ra tín hiệu HRQ đến chân HOLD của CPU để yêu cầu treo CPU. Tín hiệu HOLD phải ở mức cao cho đến hết quá trình trao đổi dữ liệu.
4. Nhận được yêu cầu treo, CPU kết thúc chu kỳ buýt hiện tại, sau đó nó treo các buýt của mình và đưa ra tín hiệu HLDA báo cho DMAC được toàn quyền sử dụng buýt.
5. DMAC đưa ra xung DACK để báo cho thiết bị ngoại vi biết là có thể bắt đầu trao đổi dữ liệu.
6. DMAC bắt đầu chuyển dữ liệu từ bộ nhớ ra thiết bị ngoại vi bằng cách đưa địa chỉ của byte đầu ra buýt địa chỉ và đưa ra tín hiệu MEMR=0 để đọc một byte từ bộ nhớ ra buýt dữ liệu. Tiếp đó DMAC đưa ra tín hiệu IOW =0 để ghi đưa dữ liệu ra thiết bị ngoại vi. DMAC sau đó giảm bộ đếm số byte còn phải chuyển, cập nhật địa chỉ của byte cần đọc tiếp, và lặp lại các động tác trên cho tới khi hết số đếm (TC).
7. Quá trình DMA kết thúc, DMAC cho ra tín hiệu HRQ=0 để báo cho CPU biết để CPU dành lại quyền điều khiển hệ thống.

### *5.8.5.2 Treo CPU để trao đổi từng byte.*

Trong cách trao đổi dữ liệu này CPU không bị treo lâu dài trong một lần nhưng thỉnh thoảng lại bị treo trong khoảng thời gian rất ngắn đủ để trao đổi 1-byte dữ liệu (CPU bị lấy mất một số chu kỳ đồng hồ). Do bị lấy đi một số chu kỳ đồng hồ như vậy lên tốc độ thực hiện một công việc nào đó của CPU chỉ bị suy giảm chứ không dừng lại. Cách hoạt động cũng tương tự như phần trước, chỉ có điều mỗi lần DMA yêu cầu treo CPU thì chỉ có một byte được trao đổi.

### *5.8.5.3 Tận dụng thời gian CPU không dùng buýt để trao đổi dữ liệu.*

Trong cách trao đổi dữ liệu này, ta phải có các logic phụ bên ngoài cần thiết để phát hiện ra các chu kỳ xử lý nội bộ của CPU (không dùng đến buýt ngoài) và tận dụng các chu kỳ đó vào việc trao đổi dữ liệu giữa thiết bị ngoại vi với bộ nhớ. Trong cách làm này thì DMA và CPU luân phiên nhau sử dụng buýt và việc truy nhập trực tiếp bộ nhớ kiểu này không ảnh hưởng gì tới hoạt động bình thường của CPU.

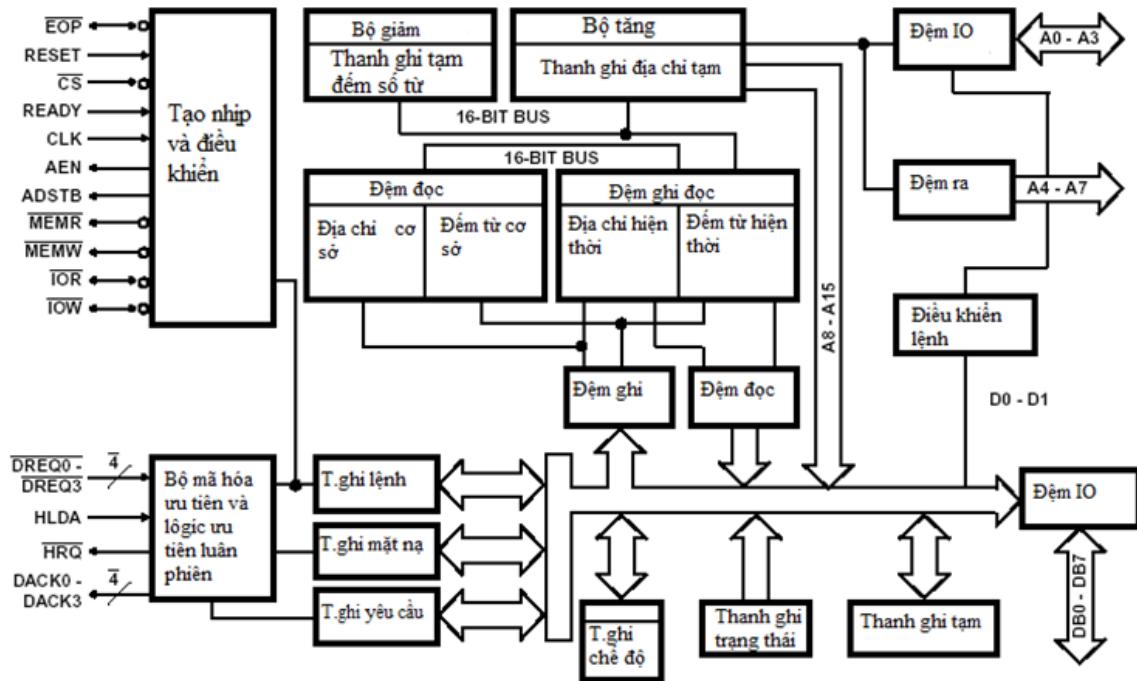
## **5.8.6 Bộ điều khiển truy nhập trực tiếp vào bộ nhớ Intel 8237A**

### *5.8.6.1 Giới thiệu*

DMA 8237A có thể thực hiện truyền dữ liệu theo 3 kiểu: kiểu đọc (từ bộ nhớ ra thiết bị ngoại vi), kiểu ghi (từ thiết bị ngoại vi đến bộ nhớ) và kiểu kiểm tra.

Trong chế độ truyền kiểu đọc thì dữ liệu được đọc từ bộ nhớ rồi đưa ra thiết bị ngoại vi. Trong chế độ truyền kiểu ghi thì dữ liệu được đọc từ thiết bị ngoại vi rồi đưa vào bộ nhớ. Khi 8237A làm việc ở chế độ kiểm tra thì tuy địa chỉ được đưa đến bộ nhớ nhưng DMA không tạo ra các xung điều khiển để tiến hành các thao tác ghi/đọc bộ nhớ hay thiết bị ngoại vi.

Ngoài ra mạch 8237A còn hỗ trợ việc trao đổi dữ liệu giữa các vùng khác nhau của bộ nhớ và cũng chỉ riêng trong chế độ làm việc này, dữ liệu cần trao đổi mới phải di qua DMA nhưng với tốc độ cao hơn khi đi qua CPU nhưng với tốc độ cao hơn khi đi qua CPU (trong trường hợp này ta có thể đọc được dữ liệu đó trong thanh ghi tạm). Sơ đồ khái trúc bên trong của mạch 8237A -5 được thể hiện trên **Error! Reference source not found.** dưới đây.



Hình 5. 32. Sơ đồ khối 8237A

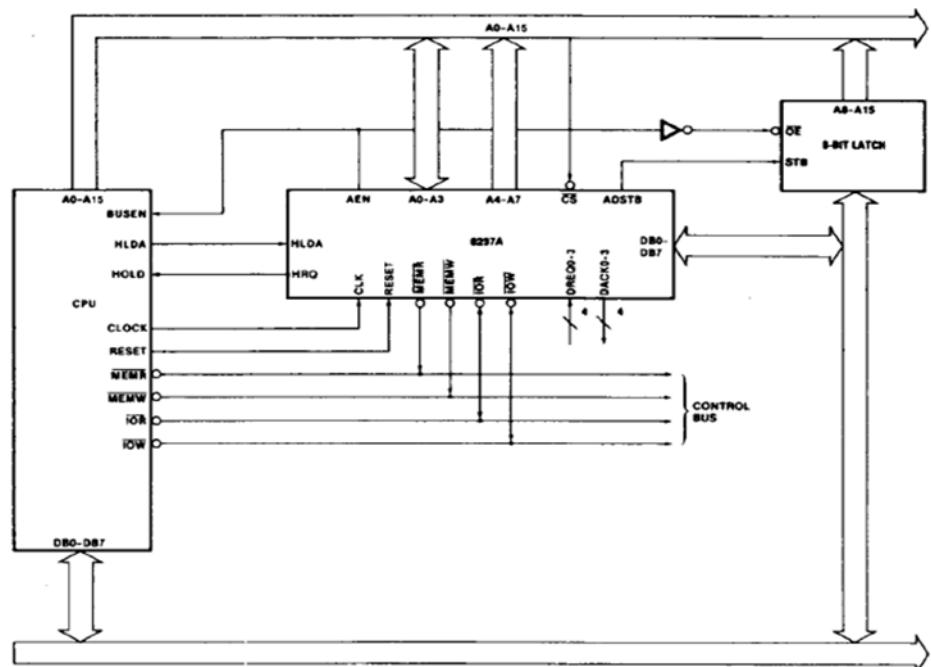
Mạch DMAC 8237A chứa 4 kênh trao đổi dữ liệu DMA với mức ưu tiên lập trình được. DMAC 8237A có tốc độ truyền 1 MB/s cho mỗi kênh, một kênh có thể truyền một mảng có độ dài 64KB.

#### 5.8.6.2 Các tín hiệu của 8237A -5

- CLK[I]: tín hiệu đồng hồ của mạch. Để mạch có thể làm việc tốt với hệ 8086/8088 thì tín hiệu CLK của hệ thống thường được đảo trước khi đưa vào CLK của 8237A.
- CS [I]: tín hiệu chọn vỏ 8237A chân này thường được nối với đầu ra của bộ giải mã địa chỉ. bộ giải mã địa chỉ này không cần dùng đến đầu vào IO/M vì bản thân DMAC đã được cung cấp các xung điều khiển mới của hệ thống.
- RESET[I]: tín hiệu nối với tín hiệu khởi động của hệ thống. Khi mạch 8237A được khởi động riêng thanh ghi mặt nạ được lập còn các bộ phận sau bị xóa:
  - Thanh ghi lệnh
  - Thanh ghi trạng thái
  - Thanh ghi yêu cầu DMA
  - Thanh ghi tạm thời
  - Mạch lật byte đầu /byte cuối (First/Last)
- READY[I]: tín hiệu sẵn sàng, nối với READY của hệ thống để gây ra các chu kỳ đợi đón với các thiết bị ngoại vi và các bộ nhớ chậm.
- HLDA [I]: tín hiệu báo chấp nhận yêu cầu treo từ CPU
- DREQ<sub>0</sub>-DREQ<sub>3</sub>[I]: các tín hiệu yêu cầu treo từ thiết bị ngoại vi. Cực tính của các tín hiệu này có thể lập trình được. Sau khi khởi động các tín hiệu này được định nghĩa là các tín hiệu kích hoạt mức cao.

- DB<sub>0</sub>-DB<sub>7</sub>[I, O]: tín hiệu hai chiều nối buýt địa chỉ và buýt dữ liệu của hệ thống các tín hiệu này được dùng khi lập trình cho DMAC và khi DMAC hoạt động các chân này chứa 8-bit địa chỉ cao A<sub>8</sub>-A<sub>15</sub> của mảng nhớ dữ liệu cần chuyển. Trong chế độ chuyển dữ liệu giữa các vùng của bộ nhớ tại các chân này có các dữ liệu được chuyển.
- IOR [I, O] VÀ IOW [I, O]: là các chân tín hiệu hai chiều dùng trong khi lập trình cho DMAC và trong các chu kỳ đọc và ghi.
- EOP [I, O]: là tín hiệu hai chiều dùng để yêu cầu DMAC kết thúc quá trình DMA. Khi là đầu ra nó được dùng để báo cho bên ngoài biết một kênh nào đó đã chuyển xong số byte theo yêu cầu, lúc này nó thường dùng như một yêu cầu ngắt để CPU xử lý việc kết thúc quá trình DMA.
- A<sub>0</sub>-A<sub>3</sub>[I, O]: là các tín hiệu hai chiều dùng để chọn các thanh ghi trong 8237A khi lập trình và khi đọc (đầu vào), hoặc để chuyển 4-bit địa chỉ thấp nhất của địa chỉ mảng nhớ cần chuyển (đầu ra).
- A<sub>4</sub>-A<sub>7</sub>[O]: các chân để chứa 4-bit địa chỉ phần cao trong byte địa chỉ thấp của địa chỉ mảng nhớ cần chuyển.
- HRQ[O]: tín hiệu yêu cầu treo đến CPU. Tín hiệu này thường được đồng bộ với tín hiệu CLK của hệ thống rồi được đưa đến chân HOLD của 8086/8088.
- DACK<sub>0</sub>-DACK<sub>3</sub>[0]: là các tín hiệu trả lời các yêu cầu DMA cho các kênh. Các tín hiệu này có thể được lập trình để hoạt động theo mức thấp hoặc mức cao. Sau khi khởi động, các tín hiệu này được định nghĩa là các xung tích cực thấp.
- AEN [0]: tín hiệu cho phép mạch nối vào DB<sub>0</sub>-DB<sub>7</sub> chốt lấy địa chỉ của vùng nhớ cần trao đổi theo kiểu DMA. Tín hiệu này cũng cho phép cấm các mạch đệm buýt địa chỉ và dữ liệu hoặc mạch tạo tín hiệu điều khiển của CPU nối vào các buýt tương ứng khi DMAC hoạt động.
- ADSTB [0]: xung cho phép chốt các bit địa chỉ phần cao A<sub>8</sub>-A<sub>15</sub> có mặt trên DB0-DB7.
- MEMR [0] và MEMW [0]: là các chân tín hiệu do DMAC tạo ra và dùng khi đọc/ghi bộ nhớ trong khi hoạt động.

**Error! Reference source not found.** dưới đây minh họa cách ghép nối các tín hiệu của 8237A với buýt hệ thống.



Hình 5. 33. Ghép nối 8237 với buýt hệ vi xử lý

#### 5.8.6.3 Các thanh ghi bên trong của DMAC 8237A

Các thanh ghi bên trong DMAC 8237A được CPU 8086/8088 chọn để làm việc nhờ các bit địa chỉ thấp A0-A3. **Error! Reference source not found.** dưới đây chỉ ra cách thức chọn ra các thanh ghi đó.

Bảng 5. 6. Địa chỉ các thanh ghi 8237A

Bit địa chỉ				Địa chỉ	Chọn chức năng	R/W?
A3	A2	A1	A0			
0	0	0	0	X0	Thanh ghi địa chỉ bộ nhớ kênh 0	R/W
0	0	0	1	X1	Thanh ghi đếm từ kênh 0	R/W
0	0	1	0	X2	Thanh ghi địa chỉ bộ nhớ kênh 1	R/W
0	0	1	1	X3	Thanh ghi đếm từ kênh 1	R/W
0	1	0	0	X4	Thanh ghi địa chỉ bộ nhớ kênh 2	R/W
0	1	0	1	X5	Thanh ghi đếm từ kênh 2	R/W
0	1	1	0	X6	Thanh ghi địa chỉ bộ nhớ kênh 3	R/W
0	1	1	1	X7	Thanh ghi đếm từ kênh 3	R/W
1	0	0	0	X8	Thanh ghi trạng thái / lệnh	R/W
1	0	0	1	X9	Thanh ghi yêu cầu	W
1	0	1	0	XA	Thanh ghi mặt nạ cho một kênh	W
1	0	1	1	XB	Thanh ghi chế độ	W
1	1	0	0	XC	Xóa flip-flop đầu/cuối	W
1	1	0	1	XD	Xóa toàn bộ các thanh ghi / đọc thanh ghi tạm	W/R
1	1	1	0	XE	Xóa thanh ghi mặt nạ	W
1	1	1	1	XF	Thanh ghi mặt nạ	W

Các **Error! Reference source not found.** dưới đây cho biết các thanh ghi trên t heo các quan điểm ứng dụng khác nhau để dễ tra cứu địa chỉ cho chúng khi lập trình với DMAC 8237A.

Bảng 5. 7. Địa chỉ các thanh ghi trong dùng cho các kênh

Kênh	$\overline{IOR}$	$\overline{IOW}$	A3	A2	A1	A0	Thanh ghi	R/W?
0	1	0	0	0	0	0	Địa chỉ cơ sở và địa chỉ hiện hành	W
	0	1	0	0	0	0	Địa chỉ hiện hành	R
	1	0	0	0	0	1	Bộ đếm cơ sở và bộ đếm hiện hành	W
	0	1	0	0	0	1	Bộ đếm hiện hành	R
1	1	0	0	0	1	0	Địa chỉ cơ sở và địa chỉ hiện hành	W
	0	1	0	0	1	0	Địa chỉ hiện hành	R
	1	0	0	0	1	1	Bộ đếm cơ sở và bộ đếm hiện hành	W
	0	1	0	0	1	1	Bộ đếm hiện hành	R
2	1	0	0	1	0	0	Địa chỉ cơ sở và địa chỉ hiện hành	W
	0	1	0	1	0	0	Địa chỉ hiện hành	R
	1	0	0	1	0	1	Bộ đếm cơ sở và bộ đếm hiện hành	W
	0	1	0	1	0	1	Bộ đếm hiện hành	R
3	1	0	0	1	1	0	Địa chỉ cơ sở và địa chỉ hiện hành	W
	0	1	0	1	1	0	Địa chỉ hiện hành	R
	1	0	0	1	1	1	Bộ đếm cơ sở và bộ đếm hiện hành	W
	0	1	0	1	1	1	Bộ đếm hiện hành	R

Bảng 5. 8. Các thanh ghi điều khiển và trạng thái

$\overline{IOR}$	$\overline{IOW}$	A3	A2	A1	A0	Thanh ghi
1	0	1	0	0	0	Ghi thanh ghi lệnh
0	1	1	0	0	0	Đọc thanh ghi trạng thái
1	0	1	0	0	1	Ghi thanh ghi yêu cầu
1	0	1	0	1	0	Ghi thanh ghi mặt nạ
1	0	1	0	1	1	Ghi thanh ghi chế độ
1	0	1	1	0	0	Xóa flip-flop đầu/cuối
1	0	1	1	0	1	Xóa tất cả các thanh ghi nội
0	1	1	1	0	1	
1	0	1	1	1	0	Địa chỉ cơ sở và địa chỉ hiện hành
	1	1	1	1	0	Địa chỉ hiện hành
	0	1	1	1	1	Bộ đếm cơ sở và bộ đếm hiện hành
	1	1	1	1	1	Bộ đếm hiện hành

#### Thanh ghi địa chỉ hiện thời:

Đây là thanh ghi 16-bit dùng để chứa địa chỉ của vùng nhớ phải chuyển. Mỗi kênh có riêng thanh ghi này để chứa địa chỉ. Khi 1-byte được truyền đi. Các thanh ghi này tự động tăng hay giảm tùy theo trước nó được lập trình như thế nào.

#### Thanh ghi số đếm hiện thời:

Thanh ghi 16-bit này dùng để chứa số byte mà kênh phải truyền (nhiều nhất là 16KB). Mỗi kênh có thanh ghi số byte của mình. Các thanh ghi này được ghi bằng giá trị nhỏ hơn 1 so với số byte thực chuyển.

#### Thanh ghi địa chỉ cơ sở và thanh ghi số đếm cơ sở:

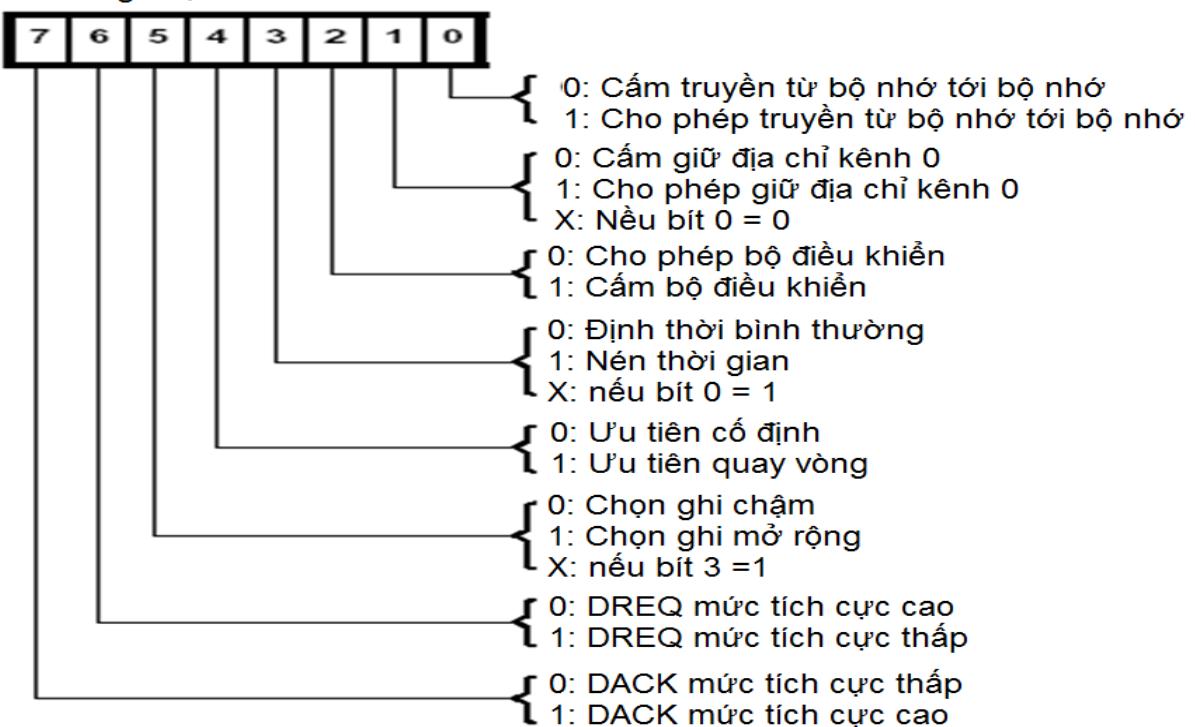
Các thanh ghi này được dùng để chứa địa chỉ và số đếm cho mỗi kênh khi chế độ tự động khởi đầu được sử dụng.

Trong chế độ này một quá trình DMA kết thúc thì các thanh ghi địa chỉ hiện thời và số đếm hiện thời được nạp lại giá trị cũ lấy từ thanh ghi địa chỉ cơ sở và thanh ghi số đếm cơ sở. Khi các thanh ghi địa chỉ hiện thời và số đếm hiện thời được lập trình thì các thanh ghi địa chỉ cơ sở và thanh ghi số đếm cơ sở cũng được lập trình bắt kể chế độ tự khởi đầu có được sử dụng hay không.

### Thanh ghi lệnh:

**Error! Reference source not found.** minh họa dạng thức và chức năng chi tiết của thanh ghi lệnh trong bộ điều khiển DMA (DMAC), được sử dụng để lập trình và quản lý hoạt động của DMA. Thanh ghi này sẽ bị xóa khi khởi động hoặc khi thực hiện lệnh xóa toàn bộ thanh ghi.

Thanh ghi lệnh



Hình 5. 34. Cấu trúc thanh ghi lệnh của DMAC

Các bit của thanh ghi này quyết định các phương thức làm việc khác nhau của 8237A. Ta sẽ giải thích sau đây ý nghĩa của các bit.

Bit D0 cho phép DMAC dùng kênh 0 và kênh 1 để chuyển dữ liệu giữa 2 vùng nhớ. Địa chỉ của byte dữ liệu ở vùng đích được chứa trong thanh ghi địa chỉ của kênh 1. Số byte chuyển được đặt trong thanh ghi đếm của kênh 1. Byte cần chuyển lúc đầu được đọc từ vùng gốc vào thanh ghi tạm để rồi từ đó nó được gửi đến vùng đích trong bước tiếp theo (hoạt động như lệnh MOVSB nhưng với tốc độ cao).

Bit D1=1 dùng để cho phép kênh 0 giữ nguyên địa chỉ trong chế độ truyền giữ liệu giữa 2 vùng nhớ. Điều này khiến cho toàn bộ các ô nhớ vùng đích được nạp cùng một byte dữ liệu.

Bit D2 cho phép DMAC hoạt động hay không.

Bit D3 quyết định byte cần chuyển được truyền với 4 hay 2 chu kỳ đồng hồ.

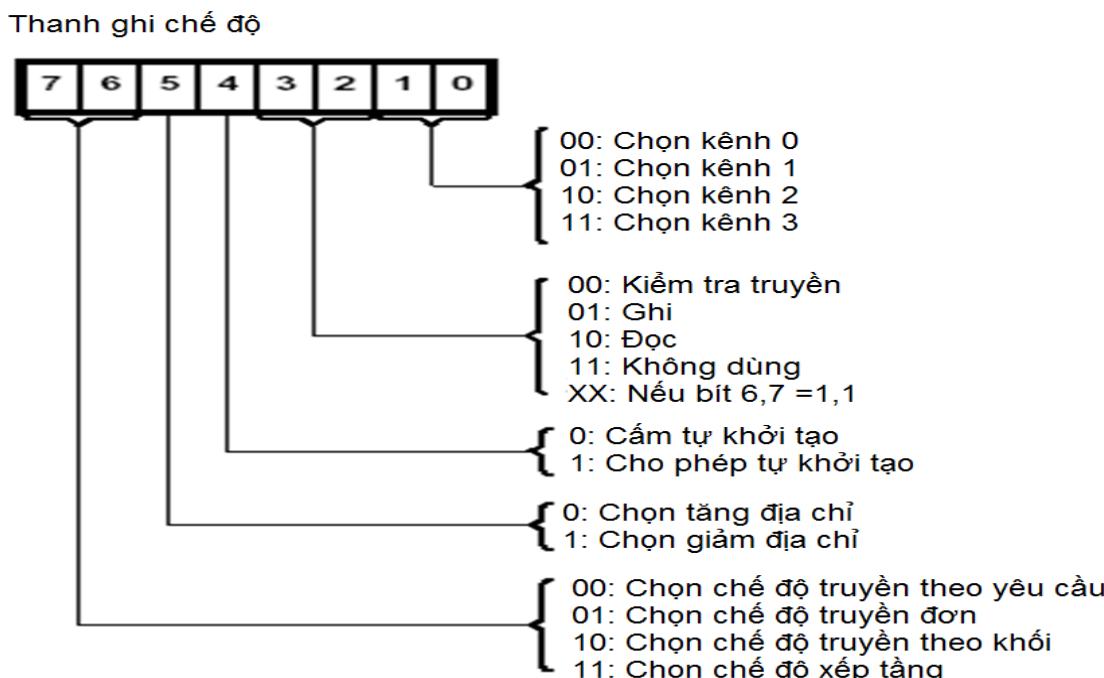
Bit D4 cho phép chọn chế độ ưu tiên cố định (kênh 0 có mức ưu tiên cao nhất. Kênh 3 có mức ưu tiên thấp nhất) hoặc chế độ ưu tiên luân phiên (kênh 0 lúc đầu có mức ưu tiên cao nhất. Sau khi kênh này được chọn để chuyển dữ liệu thì nó được nhận mức ưu tiên thấp nhất. Kênh 1 lại trở thành kênh có mức ưu tiên cao nhất)

Bit D5 cho phép chọn thời gian ghi bình thường hay kéo dài cho tiết bị ngoại vi chậm.

Các bit D6 và D7 cho phép chọn cực tính tích cực của các xung DRQ0-DRQ4 và DACK0- DACK4.

### Thanh ghi chế độ:

**Error! Reference source not found.** minh họa dạng thức và chức năng của thanh ghi chế độ trong bộ điều khiển DMA (DMAC), được sử dụng để thiết lập chế độ hoạt động cho từng kênh của DMAC. Mỗi kênh có một thanh ghi chế độ riêng, cho phép tùy chỉnh độc lập



Hình 5. 35. Cấu trúc thanh ghi chế độ của DMAC

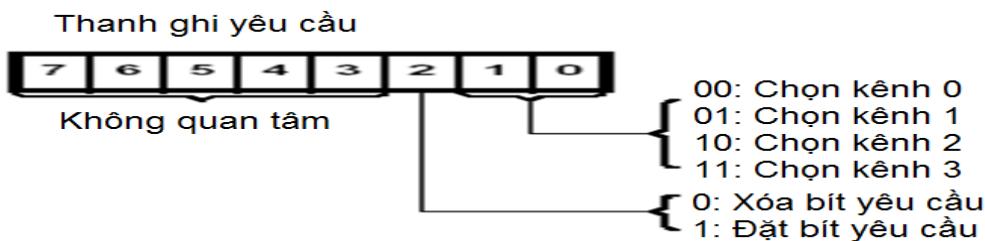
Trong chế độ DMA theo yêu cầu. DMAC tiến hành chuyển dữ liệu cho đến khi có tín hiệu EOP từ bên ngoài hoặc cho đến khi không còn yêu cầu DMA nữa (DREQ trở nên không tích cực)

Trong chế độ DMA chuyển từng byte, chừng nào vẫn còn yêu cầu DMA (DREQ vẫn là tích cực) thì DMAC đưa ra HRQ=0 trong thời gian 1 chu kỳ buýt sau mỗi lần chuyển sang 1 byte. Sau đó nó lại đưa ra HRQ=1. Cứ như vậy DMAC và CPU luân phiên nhau sử dụng buýt cho đến khi đếm hết (TC).

Trong chế độ DMA chuyển cả mảng, cả một mảng gồm một số byte bằng nội dung bộ đếm được chuyển liền một lúc. Chân yêu cầu chuyển dữ liệu DREQ không cần phải giữ được ở mức tích cực suốt trong quá trình chuyển. Chế độ nối tầng được dùng khi có nhiều bộ DMAC được dùng trong hệ thống để mở rộng số kênh có thể yêu cầu DMA.

#### **Thanh ghi yêu cầu:**

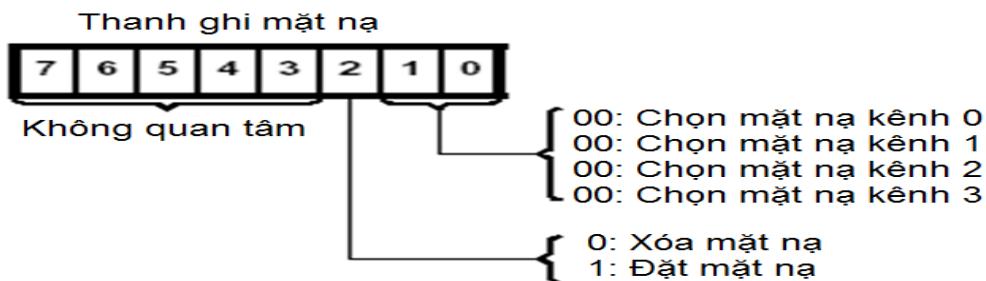
Thanh ghi này dùng để yêu cầu DMA có thể được thiết lập/ xoá theo ý muốn bằng chương trình. Điều này rất có lợi khi ta muốn chuyển dữ liệu giữa các vùng khác nhau của bộ nhớ lúc này các kênh liên quan phải được lập trình ở chế độ chuyển cả mảng. Dạng thức của thanh ghi yêu cầu như sau:



Hình 5. 36. Thanh ghi yêu cầu cho từng kênh của DMAC

#### **Thanh ghi mặt nạ riêng cho từng kênh:**

Bằng thanh ghi này ta có thể lập trình để cấm (cho Bit mặt nạ tương ứng = 1) thay cho phép hoạt động (cho Bit mặt nạ tương ứng = 0) đối với từng kênh một.



Hình 5. 37. Thanh ghi yêu cầu và thanh ghi mặt nạ riêng cho từng kênh của DMAC

#### **Thanh ghi mặt nạ tổng hợp:**

Với thanh ghi này ta có thể lập trình để cấm (Bit mặt nạ tương ứng = 1) thay cho phép hoạt động (Bit mặt nạ tương ứng = 0) đối với từng kênh chỉ bằng một lệnh.

#### **Thanh ghi trạng thái:**

Thanh ghi này cho phép xác định trạng thái của các kênh trong DMAC. Kênh nào đã truyền xong (đạt số đếm TC), kênh nào đang có yêu cầu DMA để trao đổi dữ liệu. Khi một kênh nào đó đạt TC. Kênh đó sẽ tự động bị cấm. Cấu trúc thanh ghi trạng thái như sau:

D7	D6	D5	D4	D3	D2	D1	D0
----	----	----	----	----	----	----	----

D7=1: Kênh 0 có yêu cầu

D0=1: Kênh 0 đạt số đếm

D6=1: Kênh 1 có yêu cầu

D1=1: Kênh 1 đạt số đếm

D5=1: Kênh 2 có yêu cầu

D2=1: Kênh 2 đạt số đếm

D4=1: Kênh 3 có yêu cầu

D3=1: Kênh 3 đạt số đếm

#### 5.8.6.4 Các lệnh đặc biệt cho DMAC 8237A

Có 3 lệnh đặc biệt để điều khiển hoạt động của DMAC 8237A. Các lệnh này chỉ thực hiện bằng các lệnh OUT với các địa chỉ cổng xác định thì theo thanh ghi mà không cần đến giá trị cụ thể của thanh ghi AL.

1. Lệnh xóa mạch lật byte đầu/byte cuối (First/Lát, F/L): F/L là một mạch lật bên trong DMAC có bit để chỉ ra byte nào trong các thanh ghi 16bit (thanh ghi địa chỉ hoặc thanh ghi số đếm) được chọn làm việc. Nếu F/L=1 thì số đó là MSB, còn nếu F/L=0 thì số đó là LSB. Mạch lật F/L tự động thay đổi trạng thái khi ta ghi /đọc các thanh ghi đó. Khi khởi động xong thì F/L=0
2. Lệnh xoá toàn bộ các thanh ghi: lệnh này có tác động như thao tác khởi động. Tất cả các thanh ghi đều bị xoá riêng thanh ghi mặt nạ tổng hợp thì được lập để cấm các yêu cầu trao đổi dữ liệu.
3. Lệnh xoá thanh ghi mặt nạ tổng hợp: Lệnh này cho phép các kênh của DMAC bắt đầu yêu cầu trao đổi dữ liệu.

#### 5.8.6.5 Lập trình cho các thanh ghi địa chỉ và thanh ghi số đếm:

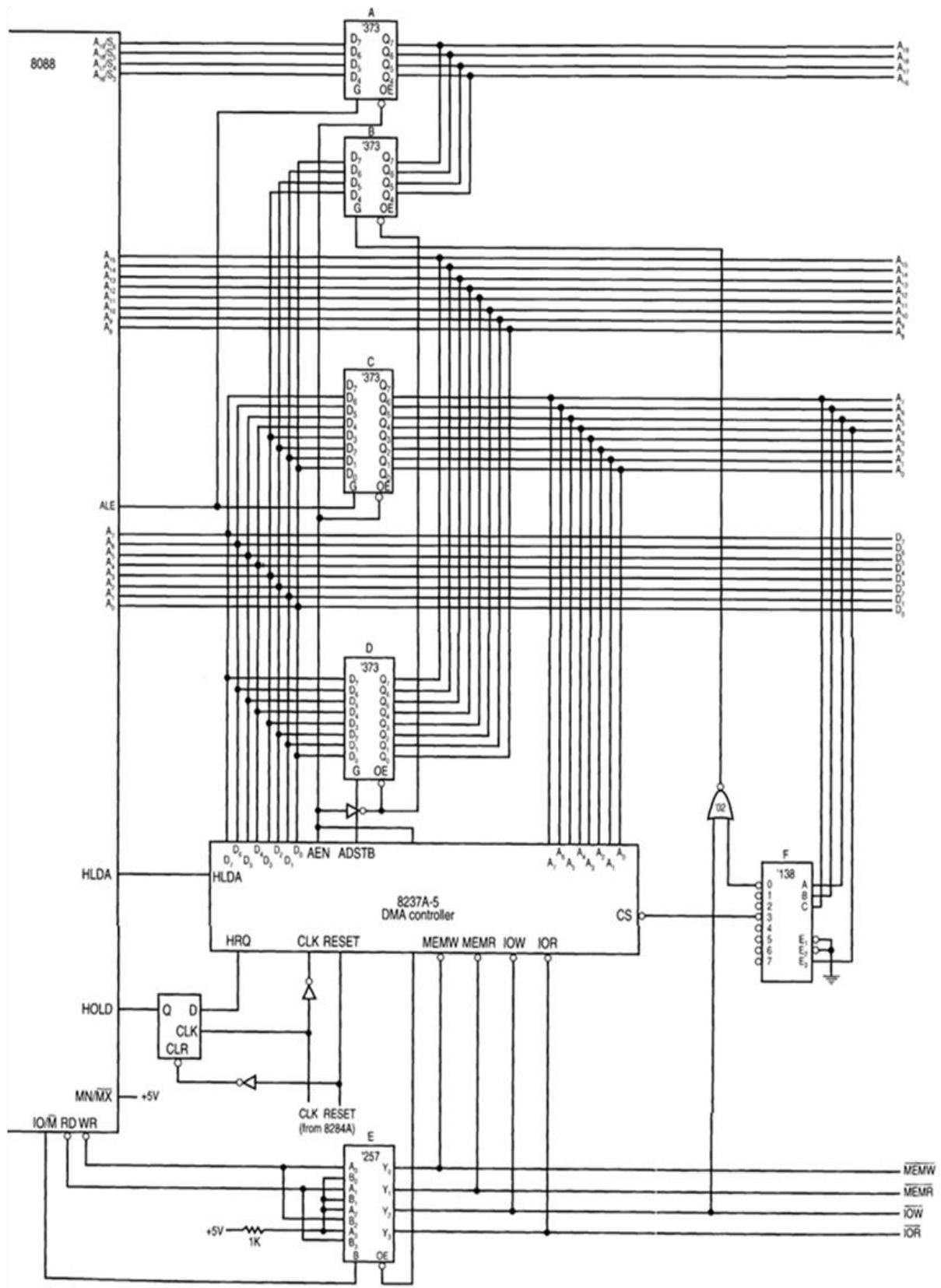
Việc lập trình cho các thanh ghi địa chỉ và thanh ghi số đếm được thực hiện riêng cho mỗi kênh. Cần phải định trước giá trị logic của F/L để thao tác chính xác được với LSB và MSB của các thanh ghi trên. Ngoài ra còn phải cấm các yêu cầu DMA của các kênh trong khi lập trình cho chúng. Có thể tuân theo các bước sau đây để lập trình cho DMAC 8237A:

- + xoá mặt lật F/L
- +cấm các yêu cầu của các kênh
- +ghi LSB rồi MSB của thanh ghi địa chỉ
- +ghi LSB rồi MSB của thanh ghi số đếm

Dưới đây là một đoạn mã cho 8237A có địa chỉ cơ sở 70H và được ghép với vi xử lý 8088 như trong **Error! Reference source not found..**

	ChotB	EQU 010H ; Địa chỉ mạch chốt B
FL	EQU 07CH	; Địa chỉ mạch lật
C0	EQU 070H	; Địa chỉ kênh 0
C1	EQU 072H	; Địa chỉ kênh 1
Dem_C1	EQU 073H	; Địa chỉ kênh 0
CheDo	EQU 07BH	; Địa chỉ thanh ghi chế độ
Lenh	EQU 078H	; Địa chỉ thanh ghi lệnh
MatNa	EQU 07FH	; Địa chỉ thanh ghi mặt nạ
YeuCau	EQU 079H	; Địa chỉ thanh ghi yêu cầu
TThai	EQU 078H	; Địa chỉ thanh ghi trạng thái
SoByte	DW 0100H	; Số byte cần chuyển
A16_19	DB 01H	; 4 bit địa chỉ cao

	Nguon	DW 00000H	; Địa chỉ nguồn
	Dich	DW 04000H	; Địa chỉ đích
	;		
	MOV	AL,A16_19	
	OUT	ChotB, AL	; Gửi địa chỉ cao ra mạch chốt
	OUT	FL, AL	; Xóa mạch lật
	MOV	AX, Nguon	; Địa chỉ nguồn ra kênh 0
	OUT	C0,AL	
	MOV	AL, AH	
	OUT	C0, AL	
	MOV	AX, Dich	; Địa chỉ đích ra kênh 1
	OUT	C1, AL	
	MOV	AL, AH	
	OUT	C1, AL	
	DEC	SoByte	
	MOV	AX, SoByte	
kênh 1	OUT	Dem_C1, AL	; số byte cần chuyển vào bộ đếm
	MOV	AL, AH	
	OUT	Dem_C1, AL	
	MOV	AL, 088H	; Chế độ kênh 0
	OUT	CheDo, AL	
	MOV	AL, 085H	; Chế độ kênh 1
	OUT	CheDo, AL	
	MOV	AL,1	; Chuyển mảng
	OUT	Lenh, AL	
	MOV	AL, 0CH	; Bỏ mặt nạ kênh 0,1
	OUT	MatNa, AL	
	MOV	AL, 4	; Kênh 0 yêu cầu DMA
	OUT	YeuCau, AL	
LAP:	IN	AL,TThai	
	TEST	AL,2	; Kiểm tra bộ đếm kênh 1 xong?
	JZ	LAP	



Hình 5. 38. Ghép nối 8237A với 8088 ở chế độ MIN

## 5.9 Kết luận chương

Chương 5 đã trình bày các nguyên lý và phương pháp phối ghép CPU với bộ nhớ và thiết bị ngoại vi, từ đó xây dựng một hệ thống vi xử lý hoàn chỉnh. Các khái niệm về không gian địa chỉ, tín hiệu điều khiển, và các vi mạch hỗ trợ đã được giải thích chi tiết, minh họa vai trò của chúng trong việc đảm bảo giao tiếp dữ liệu hiệu quả. Đồng thời, chương này cũng giới thiệu các kỹ thuật xử lý ngắn và truy cập trực tiếp bộ nhớ (DMA), giúp tối ưu hóa hiệu suất hoạt động của hệ thống. Những kiến thức này không chỉ cung cấp nền tảng cho việc thiết kế các hệ thống phần cứng thực tiễn mà còn mở ra khả năng áp dụng trong nhiều lĩnh vực như tự động hóa, điều khiển công nghiệp và nghiên cứu phát triển. Như vậy, với việc nắm vững các nội dung của chương 5, người đọc đã có thể hiểu rõ các thành phần cơ bản và cách thức phối ghép chúng trong một hệ thống vi xử lý, chuẩn bị cho những chương tiếp theo về tối ưu hóa và ứng dụng thực tiễn.

## 5.10 Câu hỏi ôn tập

1. Trình bày các thành phần của hệ thống bus và phân loại bus.
2. Mô tả nguyên lý làm việc của bus PCI và sự khác biệt so với các loại bus khác.
3. Giải thích nguyên lý làm việc của bus PCI Express và những cải tiến so với PCI.
4. Liệt kê các thiết bị vào/ra và phân loại các cổng vào/ra.
5. Trình bày nguyên lý hoạt động của bàn phím và cách quét ma trận phím để phát hiện phím được nhấn.
6. So sánh chế độ MIN/MAX của CPU 8086/8088.
7. (\*) Giải thích vai trò của tín hiệu READY và cách CPU xử lý khi tín hiệu này ở mức thấp.
8. Nêu các chức năng của tín hiệu ALE và các chân ADO – AD15 của CPU 8086.
9. (\*) Trình bày cách phối ghép CPU 8086 với bộ nhớ, bao gồm việc giải mã địa chỉ.
10. Mô tả sự khác nhau giữa RAM tĩnh và RAM động.
11. (\*) Giải thích cách hoạt động của mạch giải mã địa chỉ dùng PROM và so sánh với các mạch giải mã dùng 74LS138.
12. (\*) Lập trình đơn giản để bật/tắt đèn LED bảy đoạn sử dụng 7447 và vi xử lý.
13. Giải thích cách phối ghép CPU với các thiết bị ngoại vi và sự khác biệt giữa hai không gian địa chỉ (tách biệt và chung).
14. (\*) Trình bày nguyên lý lập trình ngắn bàn phím thông qua dịch vụ ngắn 16h và 21h.
15. Nêu các tín hiệu của CPU 8086 liên quan đến truy cập bộ nhớ và thiết bị ngoại vi.

## CHƯƠNG 6 KIẾN TRÚC MÁY TÍNH TIỀN TIẾN

*Sự phát triển của kiến trúc máy tính ngày nay không chỉ tập trung vào tăng tốc độ xử lý mà còn khai thác tối đa khả năng song song để nâng cao hiệu suất. Chương này sẽ giới thiệu các kiến trúc tiên tiến như IA-64 với công nghệ EPIC, bộ xử lý SIMD/MIMD, GPU hiệu suất cao và siêu máy tính, giúp tối ưu hóa xử lý từ cấp độ lệnh đến quy mô hệ thống. Những tiến bộ này đã và đang đóng vai trò quan trọng trong đồ họa, trí tuệ nhân tạo, mô phỏng khoa học và xử lý dữ liệu lớn, mở ra nhiều cơ hội mới trong công nghệ tính toán.*

### 6.1 Kiến trúc Intel IA32/64

Trong phần này, chúng ta sẽ đi sâu vào việc khám phá kiến trúc IA-64 (Intel Itanium), một kiến trúc máy tính tiên tiến mang tính đột phá. Kiến trúc này nổi bật như một ví dụ điển hình về hướng tiếp cận thiết kế tập trung vào khai thác tối đa tính toán song song ở cấp độ lệnh. Chúng ta sẽ phân tích những đặc điểm cốt lõi của IA-64, đặc biệt là công nghệ EPIC (Explicitly Parallel Instruction Computing), và so sánh nó với các kiến trúc superscalar truyền thống để làm nổi bật những ưu điểm và sự khác biệt trong cách tiếp cận xử lý song song.

#### 6.1.1 Động lực và tổng quan

##### 6.1.1.1 Bối cảnh và động lực phát triển IA-64

Vào cuối những năm 1970, sự ra đời của các vi xử lý x86 đánh dấu một kỷ nguyên mới trong điện toán. Thế hệ vi xử lý này, ban đầu được thiết kế theo hướng tối giản, chỉ tích hợp vài chục nghìn transistor và thực thi lệnh một cách tuần tự, nghĩa là mỗi lệnh được thực hiện hoàn toàn trước khi lệnh tiếp theo bắt đầu. Đến giữa những năm 1980, sự tiến bộ trong công nghệ bán dẫn cho phép tăng số lượng transistor lên hàng trăm nghìn. Intel đã tận dụng điều này để triển khai kỹ thuật pipeline (hay còn gọi là "ống dẫn lệnh"). Pipeline là một phương pháp tổ chức việc thực thi lệnh bằng cách chia nhỏ quá trình thực thi một lệnh thành nhiều giai đoạn riêng biệt (ví dụ: fetch, decode, execute, memory access, write back). Các giai đoạn này có thể hoạt động đồng thời trên các lệnh khác nhau, tương tự như dây chuyền lắp ráp. Điều này giúp tăng thông lượng xử lý (instruction throughput) bằng cách xử lý nhiều lệnh hơn trong một đơn vị thời gian, mặc dù thời gian để thực hiện một lệnh riêng lẻ có thể không giảm đáng kể.

Cùng thời điểm đó, kiến trúc RISC (Reduced Instruction Set Computing) nổi lên như một giải pháp để tối ưu hóa hiệu quả của pipeline. RISC đặc trưng bởi tập lệnh đơn giản, số lượng ít, độ dài cố định và các định dạng lệnh thống nhất. Điều này giúp đơn giản hóa quá trình giải mã lệnh và các giai đoạn pipeline khác, giảm độ trễ và tăng tần số xung nhịp tiềm năng. Bước tiến lớn tiếp theo là sự kết hợp giữa triết lý RISC và kiến trúc superscalar. Kiến trúc superscalar cho phép thực thi đồng thời nhiều lệnh độc lập thông qua việc tích hợp nhiều đơn vị thực thi (execution units) song song bên trong CPU. Dòng chip Pentium đầu tiên của Intel đã áp dụng kiến trúc superscalar, có khả

năng xử lý đồng thời hai lệnh CISC (Complex Instruction Set Computing). Tuy nhiên, bản thân lệnh CISC phức tạp có thể cản trở hiệu quả của việc thực thi song song.

Các thế hệ vi xử lý sau này như Pentium Pro và Pentium 4 đã giải quyết vấn đề này bằng cách triển khai một kỹ thuật gọi là micro-operation decomposition (phân tách thành vi lệnh). Các lệnh CISC phức tạp được phân tách thành các vi lệnh (micro-ops hoặc uops) đơn giản, có cấu trúc tương tự như lệnh RISC. Điều này cho phép tận dụng tối đa kiến trúc superscalar và các kỹ thuật tối ưu hóa khác như thực thi ngoài thứ tự (out-of-order execution) và đổi tên thanh ghi (register renaming).

- Thực thi ngoài thứ tự: Cho phép CPU thực thi các lệnh không theo đúng thứ tự chương trình nếu không có sự phụ thuộc dữ liệu giữa chúng, tận dụng tối đa các đơn vị thực thi đang rảnh rỗi.
- Đổi tên thanh ghi: Giải quyết các phụ thuộc giả (false dependencies) giữa các lệnh bằng cách gán các thanh ghi vật lý khác nhau cho các thanh ghi logic giống nhau, cho phép thực thi song song nhiều lệnh hơn.

Tuy nhiên, khi số lượng transistor trên chip vượt qua ngưỡng hàng chục triệu, các giới hạn về hiệu năng bắt đầu trở nên rõ ràng. Việc tăng kích thước bộ nhớ đệm (cache), một bộ nhớ tốc độ cao nhỏ nằm gần CPU, chỉ mang lại cải thiện hiệu quả đến một mức độ nhất định. Mặc dù cache giúp giảm độ trễ khi truy cập dữ liệu thường xuyên sử dụng, nó không giải quyết được giới hạn về số lượng lệnh có thể được xử lý đồng thời trên một lõi đơn. Việc bổ sung thêm các đơn vị xử lý song song trên một lõi đơn dẫn đến sự gia tăng đáng kể về độ phức tạp trong thiết kế và quản lý, đặc biệt là trong việc xử lý các tình huống như thực thi ngoài thứ tự và quản lý các pipeline ngày càng dài. Để vượt qua những rào cản này, kiến trúc đa lõi (multi-core) đã được phát triển. Thay vì cố gắng tăng độ phức tạp của một lõi đơn, kiến trúc đa lõi tích hợp nhiều lõi xử lý độc lập trên cùng một chip. Mỗi lõi có khả năng thực thi lệnh một cách độc lập, có bộ nhớ đệm riêng (thường là L1 và đôi khi L2) và chia sẻ bộ nhớ đệm cấp cao hơn (như L3). Điều này mang lại hiệu năng tính toán cao hơn đáng kể bằng cách thực hiện song song các tác vụ khác nhau trên các lõi khác nhau. Tuy nhiên, để khai thác tối đa sức mạnh của kiến trúc đa lõi, cần có sự thay đổi đồng bộ giữa phần cứng và phần mềm. Các nhà phát triển phần mềm cần viết các ứng dụng có khả năng tận dụng khả năng xử lý song song bằng cách sử dụng các kỹ thuật như đa luồng (multithreading) hoặc xử lý song song. Sự chuyển đổi sang kiến trúc đa lõi đánh dấu một bước ngoặt lớn trong thiết kế vi xử lý, chuyển trọng tâm từ việc tăng tốc độ của một lõi đơn sang việc tăng khả năng xử lý song song tổng thể của chip.

#### 6.1.1.2 Tổng quan về kiến trúc IA-64 và công nghệ EPIC

Kiến trúc IA-64 là một bước tiến đột phá trong lĩnh vực xử lý song song, được thiết kế để tối đa hóa khả năng song song của vi xử lý. Thay vì dựa vào bộ xử lý tự động phân tích luồng công việc, IA-64 chuyển nhiệm vụ này sang trình biên dịch tại thời điểm biên dịch. Sự khác biệt này giúp giảm bớt độ phức tạp của phần cứng, mang

lại hiệu quả xử lý vượt trội. Hình 2. 1. Tóm tắt các điểm khác biệt chính giữa IA-64 và cách tiếp cận superscalar truyền thống.

Bảng 6. 1. Kiến trúc Superscalar truyền thống so với IA-64

<b>Đa luồng song song (Superscalar)</b>	<b>IA-64</b>
Các lệnh kiểu RISC, mỗi lệnh độc lập	Các lệnh được nhóm thành cụm ba lệnh song song
Nhiều đơn vị thực thi song song	Nhiều đơn vị thực thi song song
Sắp xếp và tối ưu luồng lệnh trong thời gian chạy	Sắp xếp và tối ưu trong thời gian biên dịch
Dự đoán nhánh với thực thi dự đoán một nhánh	Thực thi dự đoán song song cả hai nhánh
Tải dữ liệu khi cần, ưu tiên tìm trong bộ đệm	Tải dữ liệu dự đoán trước khi cần

Công nghệ EPIC (Explicitly Parallel Instruction Computing) là trụ cột lõi của IA-64. Công nghệ này kết hợp nhiều đặc điểm quan trọng như:

- Song song ở mức lệnh (Instruction-Level Parallelism - ILP): Trình biên dịch phân tích và tối ưu luồng công việc trước khi thi hành, giúp giảm thiểu tình phức tạp của bộ xử lý.
- Từ lệnh dài (Long Instruction Words - LIW): Lệnh được nhóm thành các bó song song, giúp gia tăng hiệu quả thi hành.
- Dự đoán nhánh tiên tiến: IA-64 thi hành song song cả hai nhánh của lệnh rẽ, giảm thời gian chờ.
- Tải trước dự đoán: Dữ liệu được tải trước khi cần, đảm bảo luân chuyển nhanh chóng.

#### 6.1.1.3 Lợi thế của công nghệ EPIC trong IA-64

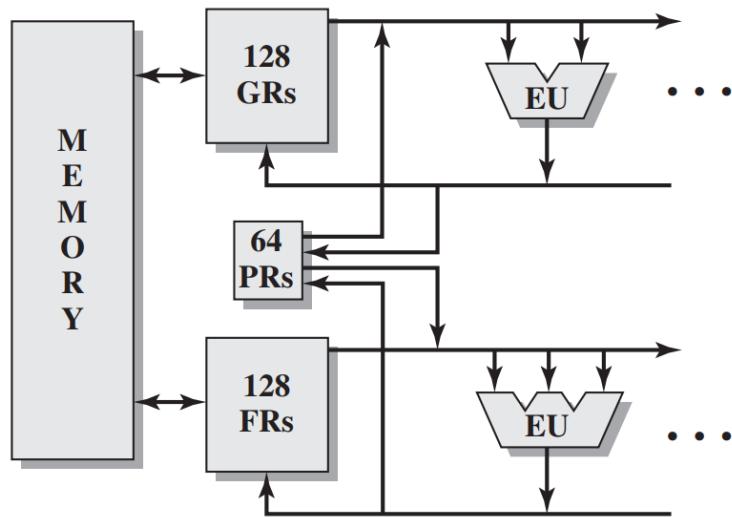
Cốt lõi của IA-64 là ý tưởng "song song rõ ràng". Thay vì để bộ xử lý thực hiện việc lập lịch động, trình biên dịch sẽ đảm nhiệm vai trò lập lịch và xác định các lệnh nào có thể chạy song song. Thông tin này được nhúng vào mã máy, giúp giảm bớt logic phức tạp trong bộ xử lý. Một số lợi ích đáng chú ý của cách tiếp cận này bao gồm:

- Đơn giản hóa bộ xử lý: Bộ xử lý IA-64 không cần triển khai các mạch logic phức tạp như trong kiến trúc superscalar ngoài thứ tự, giúp tiết kiệm tài nguyên và tăng hiệu suất.
- Tối ưu hóa tại thời điểm biên dịch: Trình biên dịch có thời gian phân tích toàn bộ chương trình để xác định cơ hội song song, thay vì chỉ dựa vào vài nanosecond tại thời điểm chạy.

- Thực hiện hiệu quả hơn: Nhờ dự đoán và chuẩn bị trước dữ liệu, IA-64 giảm thiểu thời gian chờ và cải thiện khả năng tận dụng tài nguyên.

Với những cải tiến này, IA-64 đã tạo tiền đề cho các bộ xử lý như Itanium, mở ra một hướng đi mới trong thiết kế vi xử lý hiệu năng cao. Mặc dù việc chuyển từ kiến trúc x86 sang IA-64 là một quyết định lớn, Intel tin rằng đây là bước đi cần thiết để đáp ứng yêu cầu công nghệ trong tương lai.

### 6.1.2 Tổ chức Tổng Quát của IA-64



GR = General-purpose or integer register

FR = Floating-point or graphics register

PR = One-bit predicate register

EU = Execution unit

Hình 6. 1. Tổ chức tổng quan của kiến trúc IA-64

Kiến trúc IA-64, như các kiến trúc bộ xử lý hiện đại, có thể được triển khai theo nhiều cách khác nhau, tùy thuộc vào yêu cầu thiết kế và công nghệ sử dụng. Hình 2. 1. đề xuất các đặc điểm chung về tổ chức của một máy IA-64. IA-64 tận dụng triệt để những tiến bộ trong công nghệ vi xử lý để mang lại hiệu suất vượt trội. Dưới đây là các yếu tố quan trọng trong thiết kế và tổ chức của bộ xử lý IA-64:

#### 6.1.2.1 Hệ Thống Thanh Ghi Lớn

Một trong những đặc điểm nổi bật nhất của IA-64 là sở hữu một hệ thống thanh ghi có quy mô vượt trội so với các kiến trúc truyền thống. Tổng cộng 256 thanh ghi, bao gồm:

- 128 thanh ghi 64-bit: Được sử dụng cho các phép toán số nguyên, logic, và các thao tác chung.
- 128 thanh ghi 82-bit: Chuyên xử lý các phép toán dấu phẩy động và đồ họa.
- 64 thanh ghi điều kiện 1-bit: Dành cho các lệnh thực thi có điều kiện, giúp bộ xử lý đưa ra quyết định nhanh chóng dựa trên các điều kiện cụ thể.

Khác với kiến trúc superscalar truyền thống, nơi một số lượng nhỏ thanh ghi logic được ánh xạ vào các thanh ghi vật lý thông qua các kỹ thuật như đổi tên thanh ghi và phân tích phụ thuộc, IA-64 loại bỏ hoàn toàn cách tiếp cận này. Nhờ tập hợp lớn các thanh ghi sẵn có, IA-64 giảm tải đáng kể khối lượng công việc của bộ xử lý, đồng thời hỗ trợ xử lý song song hiệu quả hơn.

#### 6.1.2.2 *Hỗ Trợ Đa Đơn Vị Thực Thi Song Song*

Kiến trúc IA-64 được thiết kế để khai thác tối đa khả năng xử lý song song, cho phép hiệu suất vượt trội. Một bộ xử lý IA-64 điển hình có thể tích hợp 8 hoặc nhiều hơn các đơn vị thực thi hoạt động đồng thời, cao hơn đáng kể so với các kiến trúc siêu vô hướng phổ biến hiện nay (thường chỉ có 4 đường ống xử lý). Điều này có nghĩa là, nếu một chương trình cần thực thi 8 lệnh số nguyên song song, bộ xử lý có 4 đường ống sẽ phải chia nhỏ các lệnh này thành hai chu kỳ xử lý. Ngược lại, bộ xử lý IA-64 với 8 đường ống có thể xử lý toàn bộ 8 lệnh cùng lúc, từ đó cải thiện hiệu suất rõ rệt. Số lượng đơn vị thực thi mà một bộ xử lý IA-64 thực sự có được phụ thuộc vào thiết kế chip cụ thể và số lượng transistor khả dụng. Nhờ thiết kế này, IA-64 tối ưu hóa khả năng xử lý song song và khai thác hiệu quả tài nguyên phần cứng.

#### 6.1.2.3 *Các Loại Đơn Vị Thực Thi Trong IA-64*

IA-64 định nghĩa bốn loại đơn vị thực thi chính, mỗi loại phụ trách một nhóm lệnh cụ thể:

- I-unit (Integer Unit): Xử lý các phép toán số học số nguyên, dịch chuyển, logic, so sánh, và các lệnh đa phương tiện số nguyên.
- M-unit (Memory Unit): Đảm nhiệm các thao tác đọc/ghi giữa thanh ghi và bộ nhớ, đồng thời thực hiện một số phép toán số nguyên cơ bản.
- B-unit (Branch Unit): Xử lý các lệnh nhánh, điều khiển luồng thực thi của chương trình.
- F-unit (Floating-Point Unit): Xử lý các phép toán dấu phẩy động, hỗ trợ các tính toán khoa học và xử lý đồ họa phức tạp.

#### 6.1.2.4 *Phân Loại Lệnh Trong IA-64*

Lệnh trong IA-64 được chia thành sáu loại chính, mỗi loại được thiết kế để phù hợp với một loại đơn vị thực thi cụ thể. Hình 2. 1. liệt kê các loại lệnh và các loại đơn vị thực thi mà chúng có thể được thực hiện

Bảng 6. 2 Mối quan hệ giữa loại lệnh và loại đơn vị thực thi

<b>Loại lệnh</b>	<b>Mô tả</b>	<b>Đơn vị thực thi</b>
A	Integer ALU	I-unit or M-unit
I	Non-ALU integers	I-unit
M	Memory	M-unit
F	Floating-point	F-unit

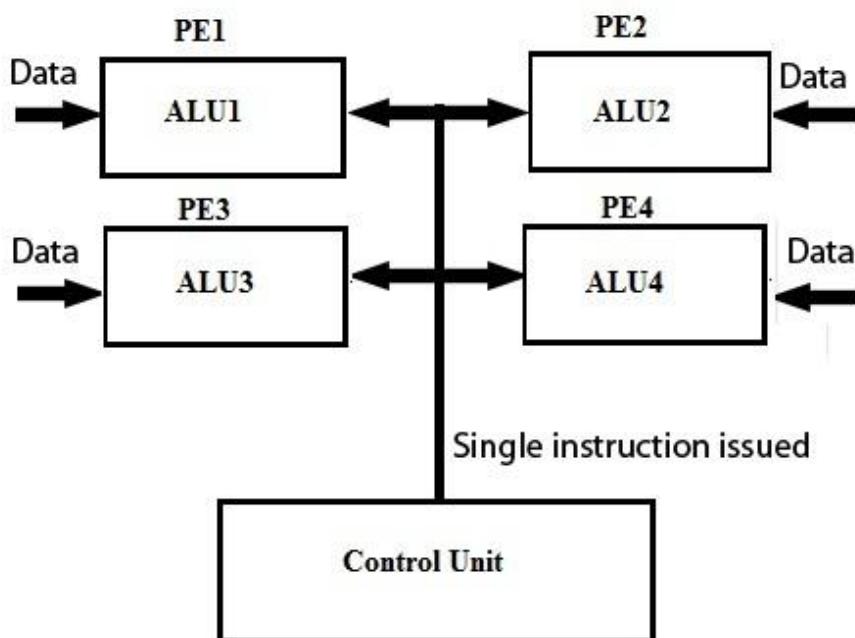
B	Brach	B-unit
X	Extended	I-unit/B-unit

Lệnh mở rộng (X-type) là các lệnh phức tạp đòi hỏi nhiều không gian mã hóa hơn so với lệnh tiêu chuẩn 41-bit. Để giải quyết, các lệnh này sử dụng hai khe trong một bộ lệnh, cho phép chứa đủ thông tin cần thiết. Cách tiếp cận này giúp tăng cường tính linh hoạt và khả năng xử lý hiệu quả các lệnh phức tạp.

## 6.2 Kiến trúc bộ xử lý mảng và song song

### 6.2.1 Tổng Quan về Kiến Trúc Bộ Xử Lý Mảng

Bộ xử lý mảng như hình Hình 2. 1. là một loại kiến trúc xử lý song song, trong đó một tập hợp các bộ xử lý, thường được sắp xếp theo cấu trúc mảng, thực thi cùng một lệnh trên nhiều dữ liệu khác nhau một cách đồng thời. Đây là hiện thực hóa của mô hình xử lý song song dữ liệu SIMD (Single Instruction, Multiple Data), nơi một lệnh duy nhất từ bộ điều khiển trung tâm được áp dụng đồng thời cho nhiều phần tử dữ liệu khác nhau trên các đơn vị xử lý. Kiến trúc này đặc biệt hiệu quả với các bài toán có tính chất lặp đi lặp lại và xử lý trên tập dữ liệu lớn, nơi cùng một chuỗi các thao tác cần được thực hiện trên nhiều phần tử dữ liệu độc lập. Các ví dụ điển hình bao gồm xử lý đồ họa, tính toán ma trận, xử lý tín hiệu số và phân tích dữ liệu lớn.



Hình 6. 2 Kiến trúc bộ xử lý mảng

### 6.2.2 Các Thành Phần Cơ Bản của Bộ Xử Lý Mảng

Một hệ thống bộ xử lý mảng điển hình bao gồm các thành phần chính sau:

- Bộ điều khiển trung tâm (Control Unit): Thành phần này đóng vai trò chỉ huy, đảm nhiệm việc giải mã lệnh và phát ra lệnh điều khiển đồng thời cho toàn bộ mảng các đơn vị xử lý. Bộ điều khiển trung tâm quản lý luồng lệnh

và đảm bảo rằng tất cả các đơn vị xử lý thực thi cùng một lệnh tại một thời điểm nhất định.

- **Mảng các đơn vị xử lý** (Processing Elements - PEs): Đây là tập hợp các bộ xử lý nhỏ, hoạt động đồng bộ dưới sự điều khiển của bộ điều khiển trung tâm. Số lượng các đơn vị xử lý trong mảng có thể rất lớn, tùy thuộc vào thiết kế và mục đích sử dụng. Mỗi đơn vị xử lý có khả năng thực hiện các phép toán số học và logic trên dữ liệu.
- **Bộ nhớ cục bộ** (Local Memory): Mỗi đơn vị xử lý thường được trang bị một bộ nhớ cục bộ riêng để lưu trữ dữ liệu đầu vào, dữ liệu tạm thời và kết quả tính toán. Việc có bộ nhớ cục bộ giúp tối ưu hóa tốc độ truy xuất dữ liệu, giảm thiểu tình trạng nghẽn cỗ chai bộ nhớ và tăng cường hiệu suất xử lý tổng thể.
- **Mạng kết nối** (Interconnection Network): Hệ thống này bao gồm các kênh giao tiếp và cơ chế định tuyến, kết nối các đơn vị xử lý với nhau và với bộ nhớ (nếu có bộ nhớ chia sẻ). Mạng kết nối cho phép các đơn vị xử lý trao đổi dữ liệu với nhau khi cần thiết, tạo điều kiện cho việc phối hợp và thực hiện các phép toán phức tạp.

#### 6.2.3 Nguyên Tắc Hoạt Động của Bộ Xử Lý Mảng

Bộ xử lý mảng hoạt động dựa trên nguyên tắc đồng bộ hóa và phân phối dữ liệu. Quy trình hoạt động cơ bản bao gồm các bước sau:

1. **Phân phối dữ liệu:** Dữ liệu đầu vào của bài toán được chia thành các phần nhỏ hơn và phân phối đến bộ nhớ cục bộ của từng đơn vị xử lý. Việc phân phối dữ liệu thường được thực hiện bởi bộ điều khiển trung tâm hoặc một hệ thống quản lý bộ nhớ.
2. **Phát lệnh:** Bộ điều khiển trung tâm giải mã lệnh tiếp theo trong chương trình và phát ra lệnh điều khiển đồng thời đến tất cả các đơn vị xử lý trong mảng.
3. **Thực thi lệnh song song:** Mỗi đơn vị xử lý thực thi cùng lệnh nhận được từ bộ điều khiển trung tâm, nhưng trên phần dữ liệu riêng được lưu trữ trong bộ nhớ cục bộ của nó. Do tất cả các đơn vị xử lý thực hiện cùng một thao tác đồng thời, kiến trúc này đạt được tính song song dữ liệu cao.
4. **Thu thu thập kết quả (Nếu cần):** Sau khi các đơn vị xử lý hoàn thành tính toán, kết quả có thể được lưu trữ trong bộ nhớ cục bộ hoặc gửi trở lại bộ điều khiển trung tâm hoặc một bộ nhớ chung để tổng hợp.

#### 6.2.4 Ứng Dụng Phổ Biến của Bộ Xử Lý Mảng

Nhờ khả năng xử lý song song dữ liệu hiệu quả, bộ xử lý mảng (Array Processor) được ứng dụng rộng rãi trong nhiều lĩnh vực khác nhau. Dưới đây là một số ứng dụng tiêu biểu cùng với các nghiên cứu liên quan:

- **Bộ xử lý đồ họa (GPU)** hiện đại là một ví dụ điển hình của kiến trúc bộ xử lý mảng, với hàng nghìn đơn vị xử lý song song giúp tăng tốc đáng kể quá trình

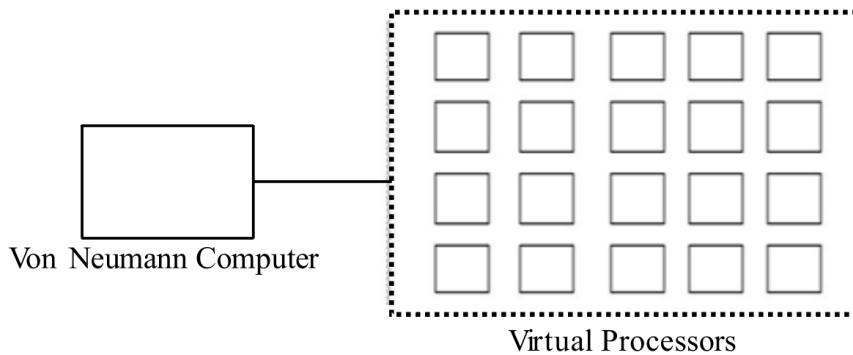
xử lý hình ảnh, video và các hiệu ứng đồ họa phức tạp. NVIDIA đã phát triển kiến trúc CUDA (Compute Unified Device Architecture), cho phép GPU xử lý các tác vụ đồ họa và tính toán song song hiệu quả. Các nghiên cứu về CUDA đã chứng minh hiệu suất vượt trội của GPU trong các ứng dụng đồ họa thời gian thực và mô phỏng 3D (NVIDIA, 2007).

- Xử lý tín hiệu số: Bộ xử lý mảng được sử dụng rộng rãi trong xử lý tín hiệu số (DSP) để thực hiện các phép toán phức tạp như biến đổi Fourier nhanh (FFT) và lọc tín hiệu. Các nghiên cứu của Smith (1997) trong cuốn "The Scientist and Engineer's Guide to Digital Signal Processing" đã chỉ ra hiệu quả của bộ xử lý mảng trong việc xử lý âm thanh và hình ảnh. Ví dụ, trong lĩnh vực y tế, GPU được sử dụng để xử lý dữ liệu từ MRI và CT scan, giúp cải thiện độ chính xác và tốc độ chẩn đoán. Trong lĩnh vực viễn thông, bộ xử lý mảng được dùng để mã hóa và giải mã tín hiệu, đặc biệt trong các chuẩn nén video như H.264 và H.265.
- Trí tuệ nhân tạo và học sâu: Bộ xử lý mảng, đặc biệt là GPU, đóng vai trò then chốt trong việc huấn luyện và suy luận các mô hình học sâu. Nghiên cứu của LeCun et al. (2015) trong bài báo "Deep Learning" đã nhấn mạnh tầm quan trọng của GPU trong việc xử lý các phép toán ma trận phức tạp và tăng tốc quá trình huấn luyện mô hình. Các ứng dụng phổ biến bao gồm nhận dạng hình ảnh, xử lý ngôn ngữ tự nhiên (NLP), và xe tự lái. Ví dụ, các mô hình như mạng neural tích chập (CNN) và mạng neural hồi quy (RNN) đều được huấn luyện hiệu quả trên GPU

### 6.3 Kiến trúc SIMD và MIMD

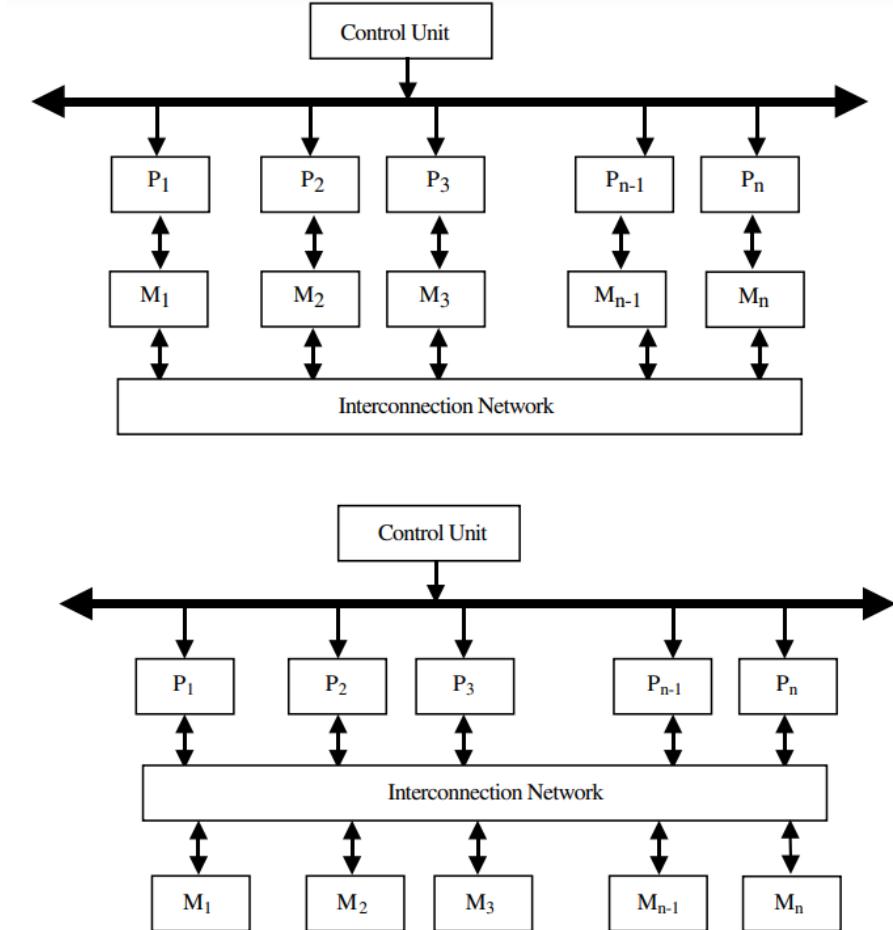
#### 6.3.1 Kiến Trúc SIMD

SIMD (Single Instruction, Multiple Data) là một trong những kiến trúc cốt lõi trong tính toán song song, nơi cùng một lệnh được thực thi đồng thời trên nhiều tập dữ liệu. Mô hình này mang lại khả năng khai thác hiệu quả tính song song của dữ liệu, đặc biệt trong các ứng dụng có khối lượng lớn các phép tính lặp lại trên các bộ dữ liệu lớn, chẳng hạn như đồ họa máy tính, xử lý tín hiệu số, hoặc các bài toán mô phỏng khoa học. Cấu trúc cơ bản của SIMD bao gồm hai thành phần chính: một máy tính phía trước theo kiến trúc von Neumann và một mảng bộ xử lý song song như được minh họa trong Hình 2. 1.. Hai thành phần này hoạt động chặt chẽ với nhau để tối ưu hóa việc thực thi các chương trình song song, đồng thời vẫn giữ được tính đơn giản trong việc lập trình và triển khai.



Hình 6. 3. Kiến trúc mô hình SIMD

Máy tính phía trước đóng vai trò như một bộ điều khiển trung tâm, thực thi các chương trình theo cách tuần tự thông thường. Chương trình được viết bằng các ngôn ngữ lập trình truyền thống, giống như khi lập trình trên một hệ thống thông thường. Tuy nhiên, điểm khác biệt lớn nhất là máy tính phía trước có thể gửi các lệnh song song đến mảng bộ xử lý để thực hiện các thao tác trên các tập dữ liệu lớn. Mảng bộ xử lý trong kiến trúc SIMD là tập hợp các bộ xử lý giống hệt nhau, mỗi bộ xử lý có một lượng nhỏ bộ nhớ cục bộ. Điều này cho phép mỗi bộ xử lý thực thi đồng thời cùng một lệnh trên dữ liệu của mình. Các bộ xử lý hoạt động hoàn toàn đồng bộ, đảm bảo rằng tất cả hoặc thực hiện cùng một thao tác tại một thời điểm, hoặc không làm gì cả. Điều này không chỉ loại bỏ các vấn đề phức tạp liên quan đến đồng bộ hóa mà còn làm cho việc lập trình trên SIMD trở nên dễ dàng hơn. Một đặc điểm quan trọng của kiến trúc này là mảng bộ xử lý được kết nối với bus bộ nhớ của máy tính phía trước, cho phép máy này truy cập ngẫu nhiên vào các bộ nhớ cục bộ của từng bộ xử lý. Điều này mang lại sự linh hoạt khi quản lý dữ liệu và đảm bảo rằng tất cả bộ xử lý có thể hoạt động trên các phần dữ liệu được phân phối.



Hình 6. 4. Hai sơ đồ SIMD.

SIMD hỗ trợ hai sơ đồ kết nối chính giữa các bộ xử lý và bộ nhớ, được minh họa qua các hệ thống nổi tiếng như ILLIAC IV và BSP (xem Hình 2. 1.):

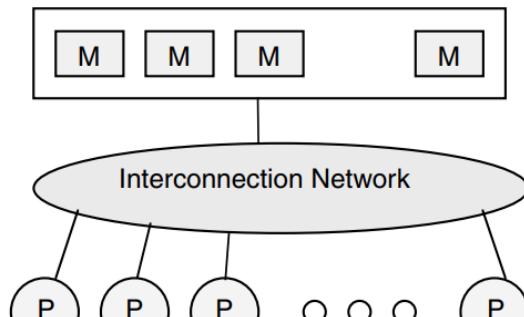
**Sơ đồ thứ nhất:** Trong sơ đồ này, mỗi bộ xử lý được tích hợp với một bộ nhớ cục bộ riêng và có thể giao tiếp với các bộ xử lý khác thông qua một mạng kết nối. Nếu mạng không hỗ trợ giao tiếp trực tiếp giữa hai bộ xử lý cụ thể, chúng có thể truyền dữ liệu qua các bộ xử lý trung gian. Ví dụ, hệ thống ILLIAC IV sử dụng mạng kết nối hình ma trận  $8 \times 8$ , trong đó mỗi bộ xử lý có thể kết nối trực tiếp với bốn bộ xử lý lân cận: bên trái, bên phải, phía trên, và phía dưới. Mô hình này đặc biệt hiệu quả với các ứng dụng có tính chất không gian, nơi dữ liệu được chia thành các khối liền kề để xử lý.

**Sơ đồ thứ hai:** Sơ đồ này sử dụng một mạng kết nối để liên kết các bộ xử lý và mô-đun bộ nhớ, cho phép truyền dữ liệu linh hoạt. Dữ liệu có thể được chuyển từ bộ xử lý này sang bộ xử lý khác thông qua các mô-đun bộ nhớ trung gian hoặc thông qua các bộ xử lý trung gian. Kiến trúc này được ứng dụng trong hệ thống BSP (Bộ xử lý Khoa học của Burroughs), mang lại sự linh hoạt cao trong việc giao tiếp dữ liệu giữa các thành phần, đáp ứng nhu cầu xử lý phức tạp.

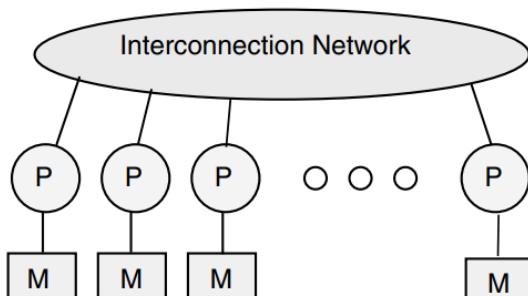
SIMD có một ưu điểm lớn là sự tương đồng giữa lập trình tuần tự và song song, giúp đơn giản hóa việc phát triển ứng dụng. Các chương trình có thể được viết và chạy theo cách truyền thống trên máy tính chính, trong khi các tác vụ xử lý dữ liệu lớn được

thực hiện song song bởi mảng bộ xử lý. Nhờ sự đồng bộ hóa chặt chẽ giữa các bộ xử lý, SIMD loại bỏ nhu cầu sử dụng các cơ chế đồng bộ hóa phức tạp. Điều này làm cho SIMD đặc biệt hiệu quả trong các bài toán yêu cầu xử lý số học trên các tập dữ liệu lớn, như các phép toán ma trận trong khoa học và kỹ thuật, kết xuất hình ảnh trong đồ họa máy tính, hoặc xử lý tensor và dữ liệu lớn trong ứng dụng trí tuệ nhân tạo.

### 6.3.2 Kiến Trúc MIMD



Shared Memory MIMD Architecture



Message Passing MIMD Architecture

Hình 6. 5. Kiến trúc bộ nhớ chung so với kiến trúc truyền thông điệp.

Kiến trúc song song MIMD (Multiple-Instruction Multiple-Data) là nền tảng của nhiều hệ thống xử lý hiện đại, nơi các bộ xử lý và bộ nhớ được liên kết với nhau qua mạng liên kết. Hai mô hình chính của kiến trúc MIMD là hệ thống bộ nhớ chia sẻ (shared memory) và hệ thống truyền thông điệp (message passing). Chúng khác biệt về cách trao đổi dữ liệu giữa các bộ xử lý, mỗi mô hình mang những ưu điểm và thách thức riêng. Hình 6.5 minh họa cấu trúc tổng quát của hai loại này.

#### **Hệ thống bộ nhớ chia sẻ**

Trong kiến trúc bộ nhớ chia sẻ, các bộ xử lý (thường được ký hiệu là 'P' trong hình) cùng sử dụng một không gian bộ nhớ chung toàn cục (ký hiệu là 'M'). Điều này có nghĩa là bất kỳ bộ xử lý nào cũng có thể truy cập trực tiếp đến bất kỳ vị trí nào trong bộ nhớ chung. Để hiện thực hóa điều này một cách hiệu quả về chi phí, thay vì sử dụng các bộ nhớ đa công phức tạp và đắt đỏ, các hệ thống bộ nhớ chia sẻ thường kết nối các bộ xử lý với bộ nhớ chung thông qua một bus hệ thống hoặc một mạng liên kết (Interconnection Network) cùng với các bộ điều khiển bộ nhớ đệm (cache controller). Bộ điều khiển bộ nhớ đệm đóng vai trò quan trọng trong việc giảm thiểu

độ trễ khi truy cập bộ nhớ, bằng cách lưu trữ các bản sao của dữ liệu được sử dụng thường xuyên bởi mỗi bộ xử lý.

Một trong những ưu điểm nổi bật của mô hình bộ nhớ chia sẻ là khả năng truy cập bộ nhớ cân bằng giữa các bộ xử lý. Điều này có nghĩa là về lý thuyết, mỗi bộ xử lý có quyền truy cập bộ nhớ với tốc độ tương đương nhau, bất kể vị trí dữ liệu nằm ở đâu trong bộ nhớ chung. Chính vì đặc điểm này, kiến trúc bộ nhớ chia sẻ thường được gọi là SMP (Symmetric Multiprocessor), nhấn mạnh tính đối xứng trong khả năng truy cập tài nguyên. Từ góc độ lập trình, kiến trúc bộ nhớ chia sẻ mang lại sự thuận tiện và dễ dàng hơn. Lập trình viên không cần phải quá bận tâm đến việc quản lý dữ liệu một cách tường minh giữa các bộ xử lý. Thay vào đó, họ có thể chia sẻ dữ liệu bằng cách đặt chúng vào bộ nhớ chung và cho phép các bộ xử lý truy cập trực tiếp. Tuy nhiên, điều này cũng đặt ra thách thức về việc đảm bảo tính nhất quán của dữ liệu (data coherence) khi nhiều bộ xử lý cùng truy cập và sửa đổi các vùng nhớ chung. Các cơ chế đồng bộ hóa như khóa (locks) hoặc semaphores thường được sử dụng để giải quyết vấn đề này.

Trong thực tế, kiến trúc bộ nhớ chia sẻ đã được ứng dụng rộng rãi trong nhiều hệ thống thương mại. Các máy chủ Balance và Symmetry của Sequent Computer là những ví dụ điển hình cho kiến trúc SMP. Tương tự, các máy chủ đa xử lý của Sun Microsystems và các hệ thống của Silicon Graphics Inc. cũng dựa trên nguyên tắc bộ nhớ chia sẻ để cung cấp khả năng xử lý song song mạnh mẽ. Như vậy, kiến trúc bộ nhớ chia sẻ trong MIMD cung cấp một mô hình lập trình trực quan và khả năng truy cập bộ nhớ cân bằng, làm cho nó trở thành một lựa chọn phổ biến cho nhiều ứng dụng song song. Tuy nhiên, khi số lượng bộ xử lý tăng lên, việc duy trì tính nhất quán của bộ nhớ đệm và giới hạn về băng thông của bus hệ thống có thể trở thành những thách thức đáng kể. Để hiểu rõ hơn về một cách tiếp cận khác trong kiến trúc MIMD, chúng ta sẽ chuyển sang tìm hiểu về hệ thống truyền thông điệp.

### **Hệ thống truyền thông điệp**

Trái ngược với mô hình bộ nhớ chia sẻ, hệ thống truyền thông điệp xây dựng trên nền tảng bộ nhớ phân tán. Trong kiến trúc này, mỗi bộ xử lý không còn chia sẻ một không gian bộ nhớ chung duy nhất. Thay vào đó, mỗi nút trong mạng liên kết bao gồm một bộ xử lý (P) và một bộ nhớ cục bộ (M) riêng biệt, như được thể hiện ở phần dưới của Hình 6.5. Điều này có nghĩa là dữ liệu của mỗi bộ xử lý được lưu trữ và quản lý độc lập trong bộ nhớ cục bộ của nó.

Việc trao đổi dữ liệu giữa các bộ xử lý trong hệ thống truyền thông điệp không diễn ra một cách tự động thông qua việc truy cập bộ nhớ chung. Thay vào đó, các bộ xử lý giao tiếp với nhau một cách tường minh bằng cách gửi và nhận thông điệp (Send/Receive). Khi một bộ xử lý cần dữ liệu nằm trong bộ nhớ cục bộ của một bộ xử lý khác, nó phải gửi một thông điệp yêu cầu dữ liệu. Bộ xử lý nhận được yêu cầu sẽ sau đó gửi lại một thông điệp chứa dữ liệu cần thiết. Phương pháp giao tiếp rõ ràng này đòi hỏi lập trình viên phải trực tiếp quản lý quá trình truyền dữ liệu giữa các bộ xử lý. Họ cần xác định dữ liệu nào cần được chia sẻ, khi nào và làm thế nào để gửi và

nhận các thông điệp. Đồng thời, việc đảm bảo tính nhất quán của dữ liệu trở thành trách nhiệm của lập trình viên, vì không có cơ chế phần cứng tự động duy trì tính nhất quán như trong bộ nhớ chia sẻ. Chính vì những yêu cầu này, việc lập trình trên các hệ thống truyền thông điệp thường được xem là phức tạp hơn so với lập trình trên các hệ thống bộ nhớ chia sẻ.

Tuy nhiên, hệ thống truyền thông điệp sở hữu một ưu điểm vượt trội về khả năng mở rộng quy mô. Do mỗi bộ xử lý có bộ nhớ riêng và giao tiếp thông qua mạng liên kết, việc thêm số lượng lớn bộ xử lý vào hệ thống không bị giới hạn bởi băng thông của một bus chung duy nhất như trong kiến trúc bộ nhớ chia sẻ. Điều này cho phép xây dựng các hệ thống song song với hàng nghìn, thậm chí hàng triệu bộ xử lý. Các hệ thống phổ biến trong những năm 1990 như nCUBE, iPSC/2, và các kiến trúc dựa trên Transputer đã minh chứng cho khả năng mở rộng của mô hình này. Thậm chí, có thể thấy những nguyên tắc của hệ thống truyền thông điệp được phản ánh trong cấu trúc của mạng Internet hiện đại, nơi các máy chủ hoặc máy tính cá nhân đóng vai trò như các nút xử lý độc lập, trao đổi thông tin thông qua các giao thức truyền thông.

### **Kiến Trúc Bộ Nhớ Chia Sẻ Phân Tán (DSM)**

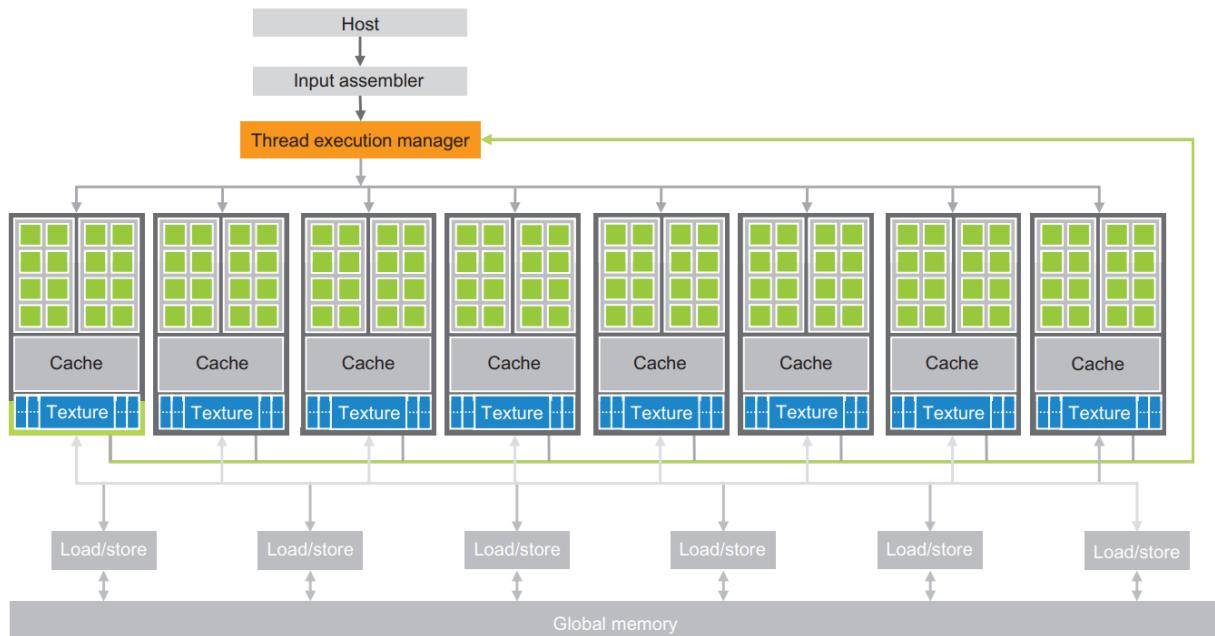
Để khai thác tối đa ưu điểm của cả hai mô hình bộ nhớ chia sẻ và truyền thông điệp, kiến trúc bộ nhớ chia sẻ phân tán (DSM - Distributed Shared Memory) đã ra đời. DSM là một nỗ lực nhằm dung hòa sự thuận tiện trong lập trình của bộ nhớ chia sẻ với khả năng mở rộng của bộ nhớ phân tán. Trong kiến trúc DSM, bộ nhớ được phân tán vật lý giữa các nút của hệ thống, tương tự như trong hệ thống truyền thông điệp. Mỗi nút có bộ xử lý và bộ nhớ cục bộ riêng. Tuy nhiên, điểm khác biệt nằm ở cách phân mềm quản lý bộ nhớ này. Các cơ chế phần mềm phức tạp được sử dụng để tạo ra một không gian địa chỉ bộ nhớ chung duy nhất từ góc nhìn của lập trình viên. Điều này có nghĩa là, mặc dù dữ liệu thực tế được lưu trữ phân tán, lập trình viên có thể truy cập dữ liệu trên các nút khác nhau như thể chúng nằm trong một bộ nhớ chung duy nhất.

Kiến trúc DSM mang lại lợi ích kép. Thứ nhất, nó thừa hưởng khả năng mở rộng của bộ nhớ phân tán, cho phép xây dựng các hệ thống với số lượng lớn bộ xử lý. Thứ hai, nó cung cấp sự tiện lợi trong lập trình tương tự như kiến trúc bộ nhớ chia sẻ, vì lập trình viên không cần phải quản lý trực tiếp việc truyền thông điệp. Các cơ chế DSM sẽ tự động lo việc di chuyển dữ liệu giữa các bộ nhớ cục bộ khi cần thiết, một quá trình thường được gọi là "trang bộ nhớ ảo" (paging). Một ví dụ tiêu biểu cho hệ thống sử dụng kiến trúc DSM là SGI Origin2000. Hệ thống này sử dụng phần cứng dựa trên mô hình truyền thông điệp, với các nút có bộ xử lý và bộ nhớ cục bộ riêng. Tuy nhiên, thông qua các cơ chế phần cứng và phần mềm đặc biệt, Origin2000 hỗ trợ lập trình theo mô hình bộ nhớ chia sẻ, giúp đơn giản hóa quá trình phát triển ứng dụng song song. Điều này giúp giảm bớt đáng kể sự phức tạp trong việc quản lý truyền thông giữa các bộ xử lý.

## **6.4 Bộ xử lý đồ họa (GPU) và siêu máy tính**

### **6.4.1 Kiến Trúc GPU Hiện Đại**

**GPU (Bộ xử lý đồ họa)** là một loại phần cứng rất quan trọng trong công nghệ hiện đại. Ban đầu, GPU được thiết kế để xử lý các hình ảnh và đồ họa. Tuy nhiên, ngày nay GPU còn được sử dụng để giải quyết các bài toán lớn và phức tạp trong nhiều lĩnh vực như khoa học, trí tuệ nhân tạo và mô phỏng. Đây là những công việc đòi hỏi khả năng xử lý rất nhanh và mạnh, gọi chung là **tính toán hiệu năng cao (HPC)**. Trong phần này, chúng ta sẽ tìm hiểu về cấu trúc cơ bản của GPU và cách GPU làm việc cùng CPU để tăng tốc độ xử lý bằng cách chia nhỏ công việc thành nhiều phần và xử lý song song.

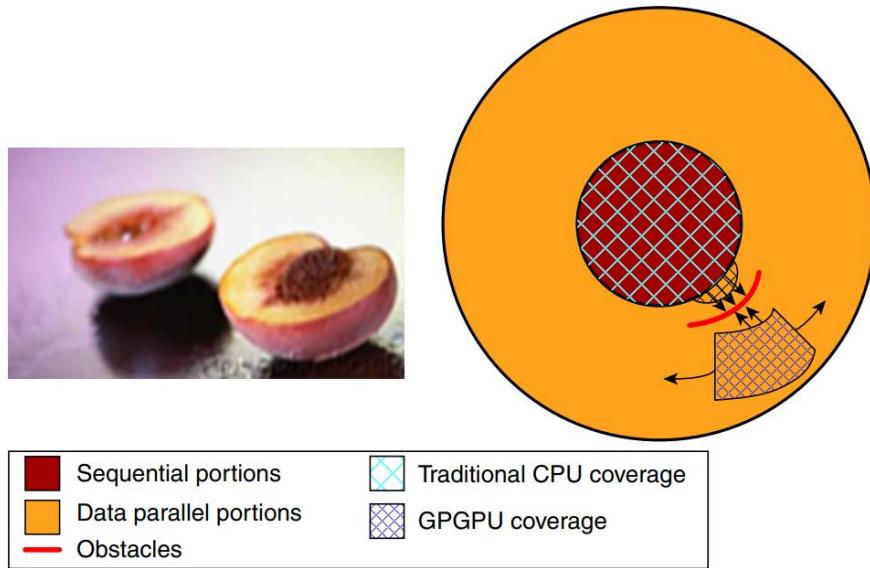


Hình 6. 6. Kiến trúc của GPU hỗ trợ CUDA.

Hình 2. 1. minh họa kiến trúc điển hình của một GPU hỗ trợ CUDA. GPU được tổ chức thành một mạng lưới các đơn vị xử lý gọi là Streaming Multiprocessors (SMs). Mỗi SM là một thành phần chính của GPU, chứa nhiều Streaming Processors (SPs) để xử lý dữ liệu. Ngoài SMs, GPU còn có các thành phần quan trọng khác như bộ nhớ đệm (cache), bộ nhớ kết cấu (texture memory), và bộ nhớ toàn cục (global memory).

- Streaming multiprocessors (SMs) là các đơn vị xử lý chính, mỗi SM chứa nhiều SPs để thực hiện các tác vụ tính toán song song. SM chịu trách nhiệm phân chia công việc cho các SP và thực thi đồng thời hàng ngàn luồng tính toán.
- Cache (bộ nhớ đệm) được dùng để lưu trữ dữ liệu tạm thời, giúp các SP truy xuất dữ liệu nhanh hơn, từ đó cải thiện hiệu suất xử lý.
- Texture memory (bộ nhớ kết cấu) ban đầu được thiết kế để lưu trữ thông tin đồ họa, chẳng hạn như kết cấu và hình ảnh. Tuy nhiên, đối với các ứng dụng tính toán, nó có thể được sử dụng để tăng cường hiệu suất.
- Global memory (bộ nhớ toàn cục) là bộ nhớ chính của GPU, thường là GDDR DRAM. Mặc dù có băng thông cao, bộ nhớ này có độ trễ lớn hơn so với bộ nhớ đệm. Tuy vậy, khả năng xử lý dữ liệu lớn và nhanh của nó rất phù hợp với các ứng dụng tính toán song song.

Các GPU hiện đại còn có khả năng giao tiếp với CPU thông qua giao diện PCI-Express (PCI-E). Điều này giúp GPU nhận dữ liệu từ bộ nhớ hệ thống của CPU và ngược lại một cách nhanh chóng, đảm bảo luồng công việc được xử lý liên tục.



Hình 6. 7. Phạm vi của các phần ứng dụng tuần tự và song song.

Hình 2. 1. minh họa cách GPU và CPU phối hợp với nhau trong một ứng dụng tính toán. Mỗi loại bộ xử lý sẽ đảm nhận một vai trò cụ thể dựa trên tính chất của công việc:

- Phần tuần tự (Sequential portions) là các đoạn mã cần xử lý lần lượt, không thể song song hóa. Các đoạn này được giao cho CPU thực hiện, nhờ khả năng xử lý tuần tự mạnh mẽ.
- Phần song song (Data parallel portions) là những đoạn mã có thể được chia nhỏ và thực thi đồng thời. GPU, với hàng ngàn luồng tính toán song song, sẽ đảm nhiệm phần việc này, đặc biệt là các tác vụ như tính toán ma trận, đồ họa hoặc mô phỏng vật lý.

Với sự phân công này, GPU xử lý phần lớn công việc tính toán song song, trong khi CPU đóng vai trò điều phối và xử lý các tác vụ không thể song song hóa.

Một ví dụ về sức mạnh của GPU là GTX 680, một GPU có thể chạy tới 16.384 luồng đồng thời, đạt hiệu suất trên 1,5 teraflop trong các phép tính dấu phẩy động kép. Hiệu suất này vượt xa CPU, vốn chỉ có thể xử lý từ 2 đến 4 luồng trên mỗi lõi. Tuy nhiên, để tận dụng tối đa sức mạnh của GPU, các ứng dụng cần được thiết kế để tận dụng số lượng lớn luồng tính toán. Thông thường, các ứng dụng hiệu quả sẽ chạy từ 5.000 đến 12.000 luồng đồng thời trên một GPU như GTX 680. Dù GPU rất mạnh trong xử lý song song, CPU vẫn đóng vai trò quan trọng nhờ khả năng xử lý linh hoạt và hiệu quả các đoạn mã không song song.

Dữ liệu giữa GPU và CPU được truyền qua giao diện PCI-Express (PCI-E). Khả năng giao tiếp phụ thuộc vào thế hệ của PCI-E:

- PCI-E Gen2 cho phép tốc độ truyền 4 GB/s theo mỗi hướng, với tổng băng thông đạt 8 GB/s.
- PCI-E Gen3 tăng gấp đôi tốc độ, đạt 8 GB/s theo mỗi hướng, phù hợp với nhu cầu truyền dữ liệu lớn của các ứng dụng hiện đại.

Trong các GPU hiện đại, bộ nhớ toàn cục của GPU có dung lượng lớn và đủ để lưu trữ dữ liệu của ứng dụng. Nhờ vậy, GPU chỉ cần giao tiếp với CPU thông qua PCI-E khi thật sự cần thiết, chẳng hạn như khi gọi một thư viện từ CPU. Điều này giảm sự phụ thuộc vào tốc độ truyền của PCI-E và tối ưu hóa hiệu năng tính toán.

#### **6.4.2 Tại Sao Cần Tăng Tốc Độ Hoặc Song Song Hóa?**

Việc tận dụng sức mạnh của GPU thông qua lập trình song song mang lại những lợi ích to lớn, đặc biệt khi phần cứng liên tục phát triển. Tăng tốc độ tính toán là yếu tố then chốt trong nhiều lĩnh vực:

- Nghiên cứu khoa học: GPU cho phép tăng tốc độ lên đến 100 lần so với CPU trong các ứng dụng song song hóa tốt. Ngay cả với vài giờ điều chỉnh, tốc độ cũng có thể tăng gấp 10 lần. Ví dụ, trong sinh học, mô phỏng các quá trình ở cấp độ phân tử, vốn vượt quá khả năng của kính hiển vi truyền thống, trở nên khả thi hơn nhờ sức mạnh tính toán của GPU. Điều này thúc đẩy các nghiên cứu và ứng dụng trong khoa học và y học.
- Xử lý video và hình ảnh: Các công nghệ như TV HD đòi hỏi khả năng xử lý song song lớn để hiển thị hình ảnh sắc nét. Trong tương lai, các tính năng tiên tiến như tổng hợp hình ảnh và nâng cao độ phân giải video sẽ càng phụ thuộc vào sức mạnh tính toán cao. Tương tự, các ứng dụng xử lý ảnh trên thiết bị cá nhân cũng cần tốc độ nhanh để cải thiện chất lượng hình ảnh.
- Trải nghiệm người dùng: Giao diện cảm ứng mượt mà trên các thiết bị như iPhone đòi hỏi tốc độ xử lý cao để phản hồi nhanh chóng và tự nhiên. Các công nghệ tương lai như màn hình 3D, nhận diện giọng nói và thực tế ảo tăng cường sẽ càng cần đến sức mạnh tính toán vượt trội.
- Trò chơi điện tử: Mô phỏng động chân thực, ví dụ như việc xe bị hư hại trong game, đòi hỏi khả năng tính toán phức tạp trong thời gian thực. GPU đóng vai trò quan trọng trong việc mang lại trải nghiệm chơi game sống động hơn.
- Xử lý dữ liệu lớn (Big Data): Với sự bùng nổ của dữ liệu, việc phân tích hiệu quả đòi hỏi khả năng xử lý song song trên nhiều phần dữ liệu. GPU cho phép xử lý đồng thời, giảm đáng kể thời gian cần thiết để phân tích và khai thác thông tin từ lượng lớn dữ liệu.

#### **6.4.3 Siêu máy tính?**

Siêu máy tính là các hệ thống tính toán hiệu năng cao, được thiết kế để giải quyết những bài toán phức tạp trong khoa học, kỹ thuật và công nghiệp. Chúng thường được sử dụng để dự báo thời tiết, mô phỏng vật lý, nghiên cứu sinh học, hóa học, và xử lý dữ liệu lớn. Với khả năng thực hiện hàng triệu tỷ hoặc thậm chí hàng tỷ phép

tính mỗi giây, siêu máy tính đóng vai trò quan trọng trong việc thúc đẩy tiến bộ công nghệ và khoa học.

Kiến trúc siêu máy tính thường được chia thành hai loại chính: hệ thống song song chật chẽ và hệ thống song song phân tán. Trong hệ thống song song chật chẽ, các bộ xử lý chia sẻ một không gian địa chỉ bộ nhớ chung và giao tiếp qua các bus tốc độ cao hoặc mạng liên kết trực tiếp, phù hợp với các bài toán yêu cầu mức độ giao tiếp lớn giữa các phần tử tính toán. Ngược lại, hệ thống song song phân tán sử dụng bộ nhớ cục bộ và giao tiếp qua mạng bằng cơ chế truyền thông điệp, dễ dàng mở rộng quy mô và thích hợp cho các bài toán lớn như phân tích dữ liệu thiên văn hoặc xử lý dữ liệu cảm biến toàn cầu.

Siêu máy tính có ứng dụng rộng rãi trong nhiều lĩnh vực. Trong nghiên cứu khoa học, chúng được sử dụng để mô phỏng các hiện tượng phức tạp như sự hình thành vũ trụ hay biến đổi khí hậu. Trong y sinh học, siêu máy tính hỗ trợ phân tích trình tự gen, mô phỏng protein và thiết kế thuốc, rút ngắn thời gian phát triển thuốc mới. Chúng cũng là công cụ mạnh mẽ cho trí tuệ nhân tạo, hỗ trợ huấn luyện các mô hình học máy lớn với lượng dữ liệu khổng lồ. Bên cạnh đó, siêu máy tính giúp khám phá vật liệu mới thông qua các mô phỏng cấp độ nguyên tử và xử lý dữ liệu lớn trong nhiều ứng dụng kinh doanh, khoa học và kỹ thuật.

Mặc dù mang lại nhiều lợi ích, siêu máy tính đối mặt với thách thức lớn về tiêu thụ năng lượng và yêu cầu phần mềm phức tạp để tận dụng hiệu quả cấu trúc song song. Các hệ thống như Fugaku của Nhật Bản hay Frontier của Mỹ tiêu thụ hàng chục megawatt năng lượng, đòi hỏi các giải pháp tối ưu hơn trong tương lai.

Sự phát triển của công nghệ lượng tử, kiến trúc tính toán neuromorphic và GPU tiên tiến sẽ mở ra những hướng đi mới cho siêu máy tính. Các hệ thống thế hệ exascale như Aurora, với hiệu suất vượt 1 exaflop, đang được kỳ vọng sẽ giải quyết các bài toán mà hiện nay chưa thể thực hiện được, đánh dấu bước tiến lớn trong lĩnh vực tính toán hiệu năng cao.

## 6.5 Kết luận chương

Chương 6 đã phân tích sâu các khía cạnh của kiến trúc máy tính tiên tiến, từ IA-64 với công nghệ EPIC, bộ xử lý mảng trong kiến trúc SIMD/MIMD, đến vai trò của GPU và siêu máy tính trong tính toán hiệu năng cao. IA-64 mang lại sự đột phá nhờ khai thác song song ở cấp độ lệnh, giảm độ phức tạp phần cứng nhờ tối ưu hóa từ trình biên dịch. Bộ xử lý mảng và SIMD/MIMD cung cấp các mô hình xử lý song song hiệu quả, đặc biệt phù hợp với các bài toán khoa học. GPU hiện đại với khả năng xử lý hàng nghìn luồng song song đã trở thành công cụ quan trọng trong đồ họa, trí tuệ nhân tạo và dữ liệu lớn, trong khi siêu máy tính đầy mạnh giải quyết các bài toán phức tạp trên quy mô toàn cầu. Những tiến bộ này không chỉ cải thiện hiệu suất tính toán mà còn mở ra các khả năng ứng dụng rộng lớn, đặt nền móng cho sự phát triển của các thế hệ công nghệ tiên tiến hơn.

## 6.6 Câu hỏi ôn tập

1. Kiến trúc IA-64 sử dụng công nghệ EPIC có những ưu điểm gì so với kiến trúc superscalar truyền thống?
2. Hệ thống thanh ghi lớn trong IA-64 giúp cải thiện hiệu năng xử lý song song như thế nào?
3. So sánh các đơn vị thực thi (I-unit, M-unit, B-unit, F-unit) trong IA-64 về chức năng và vai trò.
4. SIMD (Single Instruction, Multiple Data) được ứng dụng như thế nào trong xử lý đồ họa và trí tuệ nhân tạo?
5. Mô hình bộ nhớ chia sẻ (Shared Memory) và truyền thông điệp (Message Passing) khác nhau như thế nào trong kiến trúc MIMD?
6. Công nghệ thực thi ngoài thứ tự (Out-of-order Execution) và đổi tên thanh ghi (Register Renaming) cải thiện hiệu suất xử lý như thế nào?
7. Tại sao kiến trúc DSM (Distributed Shared Memory) được xem là sự kết hợp tối ưu giữa mô hình bộ nhớ chia sẻ và bộ nhớ phân tán?
8. (\*) Lợi ích của việc chuyển lập lịch lệnh từ phần cứng sang trình biên dịch trong kiến trúc IA-64 là gì?
9. (\*) Hãy giải thích cách SIMD và MIMD xử lý song song trên các tập dữ liệu lớn, cùng với ví dụ minh họa.
10. (\*) Trong kiến trúc đa lõi (Multi-core), làm thế nào phần mềm cần được tối ưu hóa để tận dụng hiệu quả các lõi xử lý song song?

## TÀI LIỆU THAM KHẢO

1. Stallings W., *Computer Organization and Architecture: Designing for Performance*, 8<sup>th</sup> Edition, Prentice – Hall 2009.
2. Mostafa Abd-El-Barr and Hesham El-Rewini, *Fundamentals of Computer Organization and Architecture*, John Wiley & Sons, Inc, 2005.
3. Hennessy J.L. and Patterson D.A., *Computer Architecture. A Quantitative Approach*, Morgan Kaufmann, 4<sup>th</sup> Edition, 2006.
4. Kirk, David B. and Hwu, Wen-mei W, *Programming Massively Parallel Processors: A Hands-on Approach*, Morgan Kaufmann Publishers Inc, 2010.
5. Patterson, David A. and Hennessy, John L, *Computer Organization and Design RISC-V Edition: The Hardware Software Interface*, Morgan Kaufmann Publishers Inc. 2020.
6. Hồ Khánh Lâm, *Kỹ thuật vi xử lý*, Nhà xuất bản Bưu điện, 2005
7. Trần Quang Vinh, *Cấu trúc máy vi tính*, Nhà xuất bản Giáo dục, 1999.
8. Hoàng Xuân Dậu, *Bài giảng Kiến trúc Máy tính*, Học viện Công nghệ Bưu chính Viễn thông, 2010.
9. Phạm Hoàng Duy, *Bài giảng Kỹ thuật Vi xử lý*, Học viện Công nghệ Bưu chính Viễn thông, 2010.
10. Phạm Văn Cường, *Bài giảng Lập trình Hệ thống và Điều khiển thiết bị*, Học viện Công nghệ Bưu chính Viễn thông, 2007.