

MỤC LỤC

Trang

LỜI NÓI ĐẦU	7
-------------------	---

Chương 1 TỔNG QUAN VỀ KIẾN TRÚC MÁY TÍNH

1.1. ĐẠI CƯƠNG	9
1.2. KIẾN TRÚC VÀ TỔ CHỨC CỦA MÁY TÍNH	10
1.2.1. Kiến trúc máy tính	10
1.2.2. Tổ chức máy tính/cấu trúc máy tính	11
1.3. CẤU TRÚC CHUNG CỦA HỆ THỐNG MÁY TÍNH IBM-PC	12
1.3.1. Đơn vị xử lý trung tâm	12
1.3.2. Bộ nhớ trong	13
1.3.3. Hệ thống vào/ra	14
1.3.4. Bus liên kết hệ thống	14
1.4. TẠI SAO PHẢI NGHIÊN CỨU KIẾN TRÚC MÁY TÍNH VÀ TỔ CHỨC MÁY TÍNH	17
1.5. TÓM TẮT LỊCH SỬ PHÁT TRIỂN CỦA KIẾN TRÚC MÁY TÍNH	18
1.5.1. Các thế hệ máy tính	18
1.5.2. Sự phát triển của các tập lệnh	20
1.6. PHÂN LOẠI KIẾN TRÚC MÁY TÍNH	21
1.6.1. Phân loại kiến trúc máy tính theo lịch sử phát triển bộ nhớ trong	21
1.6.2. Phân loại kiến trúc máy tính theo lịch sử phát triển tập lệnh	21
CÂU HỎI VÀ BÀI TẬP CHƯƠNG 1	21

Chương 2 KIẾN TRÚC TẬP LỆNH

2.1. BIỂU DIỄN DỮ LIỆU TRONG MÁY TÍNH	23
2.1.1. Thông tin – biểu diễn và xử lý thông tin	23

2.1.2. Đơn vị thông tin	25
2.1.3. Số nguyên và số thực	26
2.1.4. Biểu diễn ký tự	41
2.1.5. Cấu trúc biểu diễn dữ liệu	44
2.2. KIỂU DỮ LIỆU VÀ ĐỘ CHÍNH XÁC DỮ LIỆU	45
2.2.1. Kiểu dữ liệu	45
2.2.2. Độ chính xác kiểu dữ liệu cơ bản	45
2.2.3. Dự phòng cho dữ liệu có độ chính xác thay đổi	47
2.3. TẬP CÁC THANH GHI	48
2.3.1. Khái niệm thanh ghi	48
2.3.2. Các loại thanh ghi	48
2.4. CÁC KIỂU LỆNH	51
2.4.1. Nhóm lệnh số học và logic	51
2.4.2. Nhóm lệnh truy cập dữ liệu bộ nhớ	52
2.4.3. Nhóm lệnh điều khiển	52
2.4.4. Lệnh đặc quyền	53
2.4.5. Lệnh vectơ	53
2.5. CÁC PHƯƠNG THỨC ĐỊNH ĐỊA CHỈ	54
2.5.1. Định vị tức thời	56
2.5.2. Định vị thanh ghi	56
2.5.3. Định vị bộ nhớ	56
2.6. CÁC VẤN ĐỀ VỀ THIẾT KẾ ĐỊA CHỈ	59
CÂU HỎI VÀ BÀI TẬP CHƯƠNG 2	61

Chương 3

CPU, ĐƯỜNG TRUYỀN VÀ HỆ THỐNG VÀO/RA

3.1. KIẾN TRÚC CƠ BẢN CỦA MỘT MÁY TÍNH ĐIỆN TỬ SỐ VÀ ĐƠN VI XỬ LÝ TRUNG TÂM	65
3.1.1. Kiến trúc cơ bản của máy tính điện tử số	65
3.1.2. Các thành phần của CPU	71
3.1.3. Thiết bị 3 trạng thái	73
3.1.4. Các mạch cảng đơn giản	75

3.1.5. Ví dụ một số mạch chức năng trong ALU	76
3.1.6. Lệnh xử lý vào/ra dữ liệu	78
3.2. ĐƯỜNG TRUYỀN (BUS)	79
3.2.1. Các kiểu đường truyền	79
3.2.2. Đường truyền và tín hiệu điều khiển	80
3.3. HỆ THỐNG VÀO/RA	80
3.3.1. Cấu trúc phần cứng của hệ thống xử lý vào/ra dữ liệu trong máy tính	80
3.3.2. Môđun vào/ra	81
3.3.3. Lập trình điều khiển vào/ra	82
3.3.4. Các phương pháp vào/ra dữ liệu	82
CÂU HỎI VÀ BÀI TẬP CHƯƠNG 3	88

Chương 4

KIẾN TRÚC HỆ THỐNG NHỚ

4.1. CÔNG NGHỆ HỆ THỐNG NHỚ	92
4.1.1. Tổ chức bộ nhớ theo phân cấp	92
4.1.2. Một số khái niệm cơ bản liên quan đến việc vào/ra dữ liệu với bộ nhớ	94
4.1.3. Tổ chức bộ nhớ cache và phương pháp truy nhập	94
4.1.4. Các kiểu bộ nhớ	110
4.2. HỆ THỐNG NHỚ CHÍNH	122
4.2.1. Tổ chức phối ghép giữa CPU và bộ nhớ	122
4.2.2. Kỹ thuật bộ nhớ ảo	124
4.2.3. Tái định vị và bảo vệ chương trình	126
4.2.4. Mở rộng bộ nhớ và tổ chức bank nhớ	136
4.3. CÁC VẤN ĐỀ VỀ THIẾT KẾ BỘ NHỚ	153
4.3.1. Tốc độ bộ nhớ so với tốc độ CPU	153
4.3.2. Vùng địa chỉ bộ nhớ	153
4.3.3. Tốc độ và giá thành	153
CÂU HỎI VÀ BÀI TẬP CHƯƠNG 4	154

Chương 5

KỸ THUẬT ĐƯỜNG ỐNG VÀ RISC

5.1.	KỸ THUẬT ĐƯỜNG ỐNG	158
5.1.1.	Kỹ thuật đường ống đơn vị số học	160
5.1.2.	Kỹ thuật đường ống đơn vị lệnh	161
5.1.3.	Đơn vị chức năng định thời biểu	165
5.2.	MẠCH XỬ LÝ VECTƠ ỐNG	165
5.3.	MÁY TÍNH VỚI TẬP LỆNH ĐƠN GIẢN HÓA	166
5.3.1.	Đại cương	166
5.3.2.	Cạnh tranh giữa RISC và CISC	166
5.3.3.	Tổng quan về kỹ thuật cài đặt RISC	174
	CÂU HỎI VÀ BÀI TẬP CHƯƠNG 5	180
	TÀI LIỆU THAM KHẢO	181

Chương 1

TỔNG QUAN VỀ KIẾN TRÚC MÁY TÍNH

TÓM TẮT: Lý thuyết: 3 tiết.

Mục tiêu cần đạt được	Các mục chính	Bài tập bắt buộc	Bài tập làm thêm
<ul style="list-style-type: none">– Cung cấp khái niệm tổng quát về kiến trúc máy tính và cấu trúc máy tính; giới thiệu một tổ chức máy tính điển hình để có cái nhìn tổng quan máy tính hoạt động như thế nào và lịch sử phát triển của máy tính.– Kết thúc chương, sinh viên cần nắm được ý nghĩa, vị trí của môn học; kiến trúc – cấu trúc chung và nguyên lý hoạt động cơ bản của hệ thống máy tính.	<ul style="list-style-type: none">1.1. Đại cương1.2. Kiến trúc và tổ chức của máy tính.1.3. Cấu trúc chung của hệ thống máy tính IBM-PC.1.4. Tóm tắt lịch sử phát triển của máy tính.1.5. Phân loại kiến trúc máy tính.	Câu hỏi và bài tập cuối chương: 1; 2; 4 ý 1, 2.	Câu hỏi và bài tập cuối chương còn lại.

1.1. ĐẠI CƯƠNG

Môn học **Kiến trúc máy tính** là môn học nghiên cứu, khảo sát, thiết kế và cấu trúc hoạt động căn bản của một hệ thống **máy tính**. Kiến trúc máy tính là một bản thiết kế (blueprint) mô tả có tính chất chức năng về các yêu cầu (đặc biệt là tốc độ và các kết nối tương hỗ) và việc thiết kế đảm bảo vận hành cho những bộ phận khác nhau của một máy tính – tập trung chủ yếu vào việc CPU (Central Processing Unit) hoạt động nội tại như thế nào và truy cập các *địa chỉ trong bộ nhớ* bằng cách nào. Môn học này giúp học viên hiểu rõ bản chất và những đặc trưng của các hệ thống máy tính hiện đại. Để hiểu rõ được bản chất và các đặc trưng của hệ thống máy tính, làm chủ được hệ thống máy tính, đòi hỏi phải đầu tư nhiều công sức do:

- Tính đa dạng của máy tính thể hiện trong giá cả, kích thước, khả năng vận hành và ứng dụng. Máy tính ngày nay gồm rất nhiều chủng loại – từ các máy đơn chip giá chỉ vài đôla cho đến các siêu máy tính giá hàng triệu đôla – với kích thước, hiệu suất và ứng dụng thay đổi trên một phạm vi rộng lớn.

- Sự thay đổi nhanh chóng về công nghệ máy tính, từ kỹ thuật mạch tích hợp dùng để xây dựng nên các thành phần máy tính cho đến việc gia tăng sử dụng những khái niệm về tổ chức song song trong việc kết hợp các thành phần đó.

Mặc dù có sự hiện diện của tính đa dạng và tốc độ thay đổi công nghệ trong lĩnh vực máy tính, nhiều khái niệm cơ bản vẫn được áp dụng rộng khắp. Trong giáo trình này, các yếu tố cơ bản về kiến trúc và tổ chức máy tính, mối quan hệ giữa chúng cũng như nhiều bài toán gặp phải trong thiết kế máy tính hiện nay sẽ được thảo luận chi tiết.

Chương này sẽ trình bày cách tiếp cận mô tả trong việc khảo sát hệ thống máy tính, qua đó cung cấp cho người học một bức tranh chung về tổ chức của giáo trình được sử dụng cho môn học kiến trúc máy tính.

1.2. KIẾN TRÚC VÀ TỔ CHỨC CỦA MÁY TÍNH

Hai thuật ngữ *tổ chức máy tính* và *kiến trúc máy tính* là hai thuật ngữ cần được phân biệt khi mô tả một hệ thống máy tính.

1.2.1. Kiến trúc máy tính (Computer Architecture)

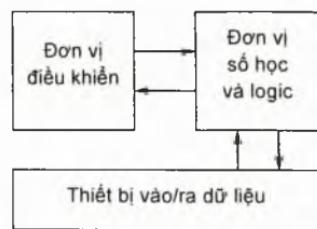
Kiến trúc máy tính đề cập đến những thuộc tính hệ thống mà lập trình viên có thể quan sát được. Nói cách khác, đó là các thuộc tính có ảnh hưởng trực tiếp đến việc thực thi một chương trình; ví dụ như tập chi thị của máy tính, số bit được sử dụng để biểu diễn dữ liệu, cơ chế nhập/xuất, kỹ thuật định địa chỉ bộ nhớ,...

Kiến trúc của hệ thống máy tính có thể chia làm hai loại theo quá trình phát triển: trước năm 1946 và từ năm 1946 đến nay.

➤ *Kiến trúc phi Von Neumann*

(trước năm 1946), máy tính không có bộ nhớ lưu trữ chương trình bên trong, gồm các thành phần sau (hình 1.1):

- Một đơn vị số học và logic (ALU – Arithmetic and Logic Unit): Có khả năng thao tác trên dữ liệu nhị phân.
- Một đơn vị điều khiển (CU – Control Unit): Có nhiệm vụ thông dịch các chỉ thị trong bộ nhớ và làm cho chúng được thực thi.



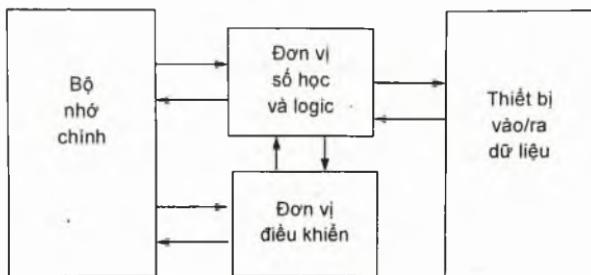
Hình 1.1. Sơ đồ khái niệm kiến trúc phi Von Neumann của hệ thống máy tính

- Thiết bị nhập/xuất (Input/Output device): Được vận hành bởi đơn vị điều khiển.

- Hệ thống dây dẫn vận chuyển tín hiệu điện (BUS).

➤ **Kiến trúc Von Neumann:** Vào cuối năm 1945, Von Neumann đưa ra ý tưởng thiết kế máy tính có chương trình được lưu trữ bên trong, là *kiến trúc chung của hệ thống máy tính* ngày nay, bao gồm các thành phần sau (hình 1.2):

- Một đơn vị số học và logic: Có khả năng thao tác trên dữ liệu nhị phân.
- Một đơn vị điều khiển: Có nhiệm vụ giải mã các chỉ thị trong bộ nhớ và làm cho chúng được thực thi.
- Một bộ nhớ chính: Để lưu trữ dữ liệu và chương trình đang thực thi.
- Thiết bị nhập/xuất: Được vận hành bởi đơn vị điều khiển.
- Hệ thống dây dẫn vận chuyển tín hiệu điện (BUS).



Hình 1.2. Sơ đồ khái niệm kiến trúc Von Neumann

1.2.2. Tổ chức máy tính/cấu trúc máy tính (Computer Organization)

Tổ chức máy tính quan tâm đến các đơn vị vận hành và sự kết nối giữa chúng, nhằm hiện thực hóa những đặc tả về kiến trúc, chẳng hạn như về tín hiệu điều khiển, giao diện giữa máy tính với các thiết bị ngoại vi, kỹ thuật bộ nhớ được sử dụng,...

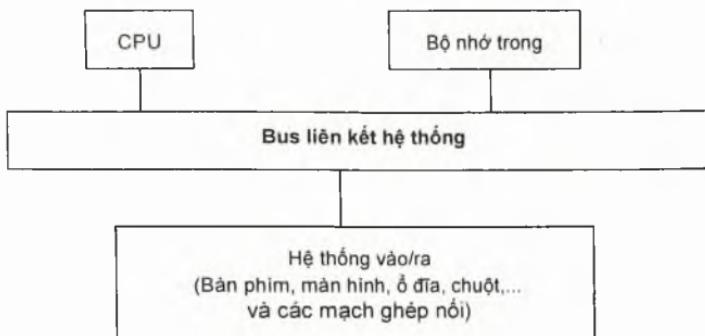
Để minh họa rõ hơn về hai khái niệm này, ta xét phép toán nhân. Việc máy tính có trang bị phép toán này hay không là vấn đề thuộc về kiến trúc máy tính. Trong khi đó, việc cài đặt phép toán thông qua một đơn vị nhân đặc biệt hay qua cơ chế sử dụng lặp đi lặp lại đơn vị cộng của hệ thống là vấn đề thuộc về tổ chức máy tính. Ở đây sự chọn lựa sử dụng cơ chế nào phụ thuộc vào các yếu tố như tần số sử dụng phép toán, tốc độ tương đối của cả hai cách tiếp cận, giá cả và kích

thuộc vật lý của một đơn vị nhân đặc biệt. Sau đây xem xét cấu trúc một hệ thống máy tính cụ thể.

1.3. CẤU TRÚC CHUNG CỦA HỆ THỐNG MÁY TÍNH IBM-PC

Cấu trúc chung của hệ thống máy tính IBM-PC ngày nay gồm các thành phần cơ bản (hình 1.3):

- Đơn vị xử lý trung tâm (Central Processing Unit – CPU).
- Bộ nhớ trong (Internal Memory).
- Hệ thống vào/ra (Input/Output System – I/O System).
- Bus liên kết hệ thống (System Bus).



Hình 1.3. Sơ đồ khái niệm về cấu trúc chung của máy tính IBM-PC

1.3.1. Đơn vị xử lý trung tâm (CPU)

➤ *Chức năng:*

- Xử lý dữ liệu.
- Điều khiển hoạt động của hệ thống.

➤ *Hoạt động:*

- CPU hoạt động theo xung nhịp, nạp lần lượt từng lệnh từ bộ nhớ, giải mã lệnh để phát tín hiệu điều khiển thực hiện lệnh. Việc thực hiện chương trình thực chất là sự lặp lại quá trình nạp lệnh, giải mã lệnh và thực thi lệnh được nạp.
- Mỗi lệnh được thực hiện trong một chu kỳ lệnh. Chu kỳ lệnh là khoảng thời gian từ khi bắt đầu nạp lệnh đến khi thực hiện xong lệnh. Các chu kỳ lệnh không

nhất thiết phải bằng nhau, mỗi chu kỳ lệnh xảy ra trên một hoặc nhiều xung nhịp đồng hồ hệ thống.

– Mỗi lệnh được thực hiện theo nhiều giai đoạn, phụ thuộc vào loại lệnh. Một lệnh thường trải qua 5 giai đoạn: nhận lệnh (Instruction Fetch), giải mã lệnh (Instruction Decode), tạo địa chỉ toán hạng (Operand Address), nhập toán hạng (Accept Operand), thực thi lệnh (Instruction Execute).

– Trong quá trình thực hiện chương trình, CPU trao đổi dữ liệu với bộ nhớ và các thiết bị vào/ra.

➤ *Các thành phần cơ bản:*

– *Đơn vị điều khiển* (Control Unit): Điều khiển hoạt động của CPU và các thành phần khác của máy tính.

– *Đơn vị số học và logic* (Arithmetic & Logic Unit – ALU): Thực hiện các chức năng xử lý dữ liệu.

– *Các thanh ghi*: Là các ngăn nhớ đặc biệt nằm trong CPU để chứa các thông tin tạm thời phục vụ cho quá trình thực hiện chương trình.

– *Bus bên trong*: Dùng để kết nối và trao đổi thông tin giữa các thành phần với nhau.

– Ngoài bốn thành phần cơ bản trên, trong CPU còn có bộ nhớ *cache*: Là bộ nhớ đệm có tốc độ truy nhập cao, giúp vào/ra dữ liệu nhanh hơn, tăng hiệu năng làm việc của CPU.

1.3.2. Bộ nhớ trong

Bộ nhớ trong được chế tạo từ các chip nhớ bán dẫn, gồm hai loại:

➤ **Bộ nhớ BIOS ROM** (Basic Input Output System Read Only Memory): Là bộ nhớ ROM, chứa chương trình khởi động đầu tiên của máy tính và chương trình điều khiển vào/ra cơ bản trong hệ thống máy tính đúng nghĩa như tên gọi của nó và nó chứa thông tin cố định trong hệ thống. Đặc trưng cơ bản của bộ nhớ ROM ngày nay là mất nguồn điện nuôi thì không mất thông tin. Do vậy, nó thường dùng để chứa thông tin cố định trong hệ thống hoặc làm bộ nhớ ngoài lưu trữ thông tin ổn định, lâu dài.

➤ **Bộ nhớ chính** (Main Memory): Là bộ nhớ RAM (Read Only Memory), là thành phần nhớ được nối trực tiếp với CPU và được điều khiển bởi CPU.

Các chương trình đang thực hiện phải nằm trong bộ nhớ chính. Bộ nhớ chính gồm các ngăn nhớ và mỗi ngăn nhớ có một địa chỉ xác định, các ngăn nhớ được tổ chức theo byte. Bộ nhớ chính có tốc độ cao, dung lượng nhỏ và chứa chương trình đang thực thi, dữ liệu đang cần xử lý trong máy tính. Đặc trưng cơ bản của bộ nhớ RAM là truy nhập ngẫu nhiên, tốc độ truy nhập cao và mất nguồn điện nuôi thi mất thông tin, được dùng làm bộ nhớ chính trong máy tính IBM-PC.

1.3.3. Hệ thống vào/ra

➤ **Chức năng:** Dùng để trao đổi thông tin giữa máy tính với thế giới bên ngoài.

➤ **Các thành phần:**

– Các thiết bị ngoại vi (các thiết bị vào/ra): Làm nhiệm vụ chuyên đổi thông tin ở dạng vật lý nào đó về dạng dữ liệu phù hợp với máy tính và ngược lại.

– Các mạch ghép nối vào/ra: Các thiết bị ngoại vi không được nối ghép trực tiếp với CPU mà phải thông qua các mạch ghép nối vào/ra. Trong mạch ghép nối vào/ra có thanh ghi dữ liệu được đánh địa chỉ xác định, và nối trực tiếp với cổng vào/ra. Các thiết bị vào/ra được ghép nối thông qua cổng, địa chỉ của cổng chính là địa chỉ của thanh ghi dữ liệu.

Các loại thiết bị vào/ra điển hình:

- Các thiết bị vào: Bàn phím, chuột, máy quét (scanner),...
- Các thiết bị ra: Màn hình, máy in, máy vẽ,...
- Thiết bị vừa vào vừa ra: Các loại ổ đĩa lưu trữ dữ liệu (bộ nhớ ngoài).

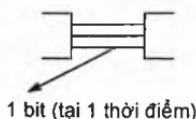
1.3.4. Bus liên kết hệ thống

➤ **Chức năng:** Liên kết các thành phần khác nhau trong hệ thống, do vậy còn gọi là Bus liên kết hệ thống.

➤ **Định nghĩa Bus:**

– Là tập hợp các đường dây dẫn điện để vận chuyển thông tin – tín hiệu điện (các bit) từ phần mạch này đến các phần mạch khác trong phạm vi máy tính.

– Bit là từ viết tắt của "Binary digit".



Bản chất vật lý: + Không có điện áp → truyền 0;
+ Có điện áp → truyền 1.

- Tập các đường dây vận chuyển thông tin đồng thời tại một thời điểm được gọi là *độ rộng của Bus*.

Ví dụ: 8 đường dây vận chuyển thông tin đồng thời thì độ rộng là 8 bit.

➤ *Phân loại Bus:*

Bus chia thành ba loại là: Bus địa chỉ; Bus dữ liệu; Bus điều khiển.

- *Bus địa chỉ:*

- CPU muốn trao đổi dữ liệu với ngăn nhớ nào, với cổng vào/ra nào thì cần xác định được địa chỉ của đối tượng cần trao đổi dữ liệu.

- Bus địa chỉ vận chuyển tín hiệu địa chỉ từ CPU đến bộ nhớ hay cổng vào/ra để xác định ngăn nhớ nào hay cổng vào/ra nào cần trao đổi thông tin.

- Bus địa chỉ, nói tóm quát gồm n đường dây $A_0 \div A_{n-1}$ thì gọi độ rộng Bus là n bit và n bit này được dùng để đánh địa chỉ. Do đó có khả năng quản lý tối đa 2^n giá trị địa chỉ ngăn nhớ hay 2^n byte nhớ (vì bộ nhớ chính quản lý theo byte).

Ví dụ:

$$8088/8086: \text{Bộ vi xử lý có } n = 20 \rightarrow \text{quản lý tối đa } 2^{20} \text{ byte} = 1\text{MB.}$$

$$80286: \quad n = 24 \rightarrow \text{quản lý tối đa } 2^{24} \text{ byte} = 2^4 \times 2^{20} = 16\text{MB.}$$

$$80386: \quad n = 32 \rightarrow \text{quản lý tối đa } 2^{32} \text{ byte} = 2^2 \times 2^{30} = 4\text{GB.}$$

$$\text{Pentum II:} \quad n = 36 \rightarrow \text{quản lý tối đa } 2^{36} \text{ byte} = 2^6 \times 2^{30} = 64\text{GB.}$$

Tóm lại, Bus địa chỉ có độ rộng và chỉ vận chuyển tín hiệu điện theo một chiều, đó là từ CPU ra ngoài.

- *Bus dữ liệu:*

- Vận chuyển lệnh từ bộ nhớ đến CPU.

- Vận chuyển dữ liệu giữa các thành phần khác nhau trong phạm vi máy tính như CPU, bộ nhớ, hệ thống vào/ra.

- Bus dữ liệu ký hiệu $D_0 \div D_{m-1}$ thì độ rộng Bus là m bit.

- m thường có giá trị là: 8, 16, 32, 64.

Ví dụ:

$$8088: \quad m = 8, \text{ tức là vận chuyển một lúc 1 byte.}$$

$$8086/80286: \quad m = 16, \text{ tức là vận chuyển một lúc 2 byte.}$$

$$80386/486: \quad m = 32, \text{ tức là vận chuyển một lúc 4 byte.}$$

$$\text{Pentum II:} \quad m = 64, \text{ tức là vận chuyển một lúc 8 byte.}$$

Tóm lại, Bus dữ liệu có độ rộng và vận chuyển tín hiệu điện theo cả hai chiều.

- Bus điều khiển: Là tập hợp các tín hiệu điều khiển riêng lẻ, hoặc là phát ra từ CPU để điều khiển đọc/ghi bộ nhớ hay hệ thống vào/ra, hoặc là phát ra từ bộ nhớ hay hệ thống vào/ra đến yêu cầu CPU.

Một số tín hiệu điều khiển điển hình:

- Các tín hiệu phát ra từ CPU điều khiển ghi, đọc bộ nhớ hay công vào/ra

Có bốn tín hiệu điều khiển cơ bản:

+ Memory Read (MEMR): Là tín hiệu điều khiển đọc dữ liệu từ một ngãnh nhớ có địa chỉ xác định đưa lên Bus dữ liệu. Khi Bus địa chỉ xác định ngãnh nhớ cần đọc thì MEMR sẽ điều khiển mở ngãnh nhớ để đưa dữ liệu lên Bus dữ liệu.

+ Memory Write (MEMW): Điều khiển ghi dữ liệu có sẵn trên Bus dữ liệu đến ngãnh nhớ có địa chỉ xác định. Khi Bus địa chỉ xác định ngãnh nhớ cần ghi và dữ liệu trên Bus dữ liệu đã ổn định thì MEMW sẽ điều khiển mở ngãnh nhớ để đưa dữ liệu từ Bus dữ liệu vào ngãnh nhớ.

+ Input Output Read (IOR): Là tín hiệu điều khiển đọc dữ liệu từ cổng.

+ Input Output Write (IOW): Là tín hiệu điều khiển ghi dữ liệu ra cổng.

- Các tín hiệu điều khiển ngắn: Là các tín hiệu yêu cầu CPU dừng công việc hiện tại để chuyển sang thực hiện công việc khác, còn gọi là yêu cầu CPU ngắn. Có các dạng sau:

+ Non Maskable Interrupt (NMI) – tín hiệu ngắn không che được: Nó là tín hiệu từ mạch bên ngoài gửi đến chân NMI của CPU và CPU phải ngắn ngay sau khi hoàn thành lệnh đang thực hiện.

+ Interrupt Request (INTR) – tín hiệu ngắn che được: Là tín hiệu được gửi từ mạch điều khiển ưu tiên ngắn gửi đến chân INTR của CPU yêu cầu CPU ngắn. CPU có thể ngắn, có thể không.

+ Interrupt Acknowledge (INTA): Là tín hiệu phát ra từ chân INTA của CPU, báo cho mạch bên ngoài biết CPU có chấp nhận ngắn hay không.

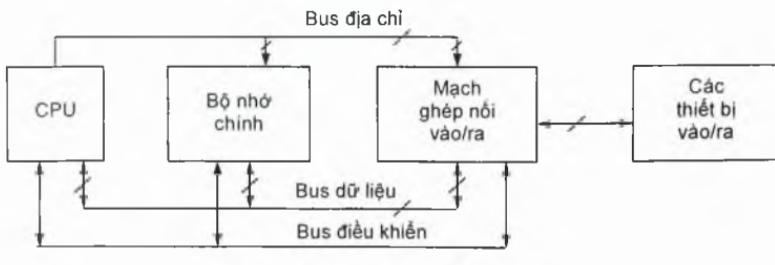
Trên đây là nhóm tín hiệu liên quan đến việc CPU chuyển từ trạng thái này sang trạng thái khác.

- Các tín hiệu điều khiển chuyển nhượng Bus (thực chất là chuyển nhượng quyền điều khiển hệ thống): Bình thường CPU toàn quyền điều khiển hệ thống, sử dụng các Bus địa chỉ, Bus dữ liệu, Bus điều khiển. Khi có một tín hiệu thực hiện xin quyền điều khiển hệ thống thì CPU có thể chuyển nhượng quyền sử dụng Bus. Có các dạng sau:

+ Bus Request (BRQ) – HOLD: Là tín hiệu điều khiển từ mạch bên ngoài gửi đến chân HOLD của CPU, yêu cầu CPU chuyển nhượng quyền sử dụng Bus.

+ Bus Grant (BGT) – HLDA (Hold Acknowledge): Tín hiệu này phát ra từ CPU trả lời có chấp nhận chuyển nhượng Bus hay không.

Sau đây là sơ đồ khái phác ghép Bus (hình 1.4).



Hình 1.4. Sơ đồ khái phác ghép Bus

1.4. TẠI SAO PHẢI NGHIÊN CỨU KIẾN TRÚC MÁY TÍNH VÀ TỔ CHỨC MÁY TÍNH

Máy tính giúp cho việc tính toán, xử lý dữ liệu trong mọi lĩnh vực của đời sống. Ngày nay, các chuyên gia trong bất kỳ lĩnh vực nào cần tính toán, họ thường không quan tâm đến máy tính làm việc như thế nào, mà chỉ coi nó như một hộp đen thực thi chương trình theo một thủ thuật nào đó. Ngược lại, tất cả các sinh viên của ngành khoa học máy tính hay tin học thì cần phải đạt được sự hiểu biết nhất định, sự nhận thức và đánh giá về các thành phần chức năng của một hệ thống máy tính, các nét đặc trưng, hiệu suất, và sự tương tác giữa chúng. Môn học *Kiến trúc máy tính* đề cập nhiều vấn đề liên quan thiết thực đến trình độ chuyên môn, giúp giải quyết tốt công việc sau này. Người học, lập trình viên cần hiểu kiến trúc máy tính để tổ chức chương trình chạy trên một máy tính thực được hiệu quả hơn. Mặt khác, trong việc lựa chọn một hệ thống máy tính để sử dụng, người sử dụng cần phải thực sự hiểu sự kết hợp giữa các thành phần khác nhau để tạo nên một máy tính có tối ưu hay không, như là tốc độ xung nhịp đồng hồ CPU, tốc độ Bus, dung lượng bộ nhớ,...

Ý nghĩa của sự phân biệt tổ chức và kiến trúc máy tính:

Kể từ khi ngành công nghiệp máy tính ra đời cho đến nay, sự phân biệt giữa *kiến trúc* và *tổ chức* máy tính là một yếu tố quan trọng. Nhiều hãng sản xuất máy

tính cho ra đời cả một họ máy chỉ khác nhau về tổ chức, còn kiến trúc hoàn toàn giống nhau. Kết quả là các kiểu máy trong cùng một họ có giá cả và hiệu suất vận hành khác nhau. Hơn thế nữa, một kiến trúc máy có thể tồn tại qua nhiều năm liền, trong khi tổ chức máy dựa trên đó sẽ thay đổi theo bước tiến của công nghệ. Ở đây chúng ta có thể lấy ví dụ trường hợp kiến trúc máy IBM System/370. Kiến trúc này được giới thiệu lần đầu vào năm 1970 với một số kiểu máy khác nhau. Khách hàng với nhu cầu khiêm tốn có thể mua kiểu máy rẻ hơn, chậm hơn vì nâng cấp lên kiểu máy đắt tiền hơn, nhanh hơn khi nhu cầu sử dụng tăng lên. Điểm quan trọng trong việc nâng cấp này là khách hàng không phải bỏ đi những phần mềm đang được sử dụng trên máy cũ của họ. Trải qua năm tháng, hãng IBM đã cho ra đời thêm nhiều kiểu máy mới dựa trên công nghệ cải tiến nhằm thay thế những kiểu máy lỗi thời, mang lại cho khách hàng lợi ích về tốc độ và chi phí thấp, trong khi vẫn bảo toàn sự đầu tư của họ về phần mềm nhờ có sự dùng chung một kiến trúc máy cho tất cả các kiểu máy trong cùng một họ. Bằng cách tổ chức định hướng người sử dụng như vậy, kiến trúc IBM System/370, qua một vài tinh chỉnh không đáng kể, vẫn tồn tại cho đến ngày nay với vai trò ngọn cờ đầu trong dòng sản phẩm IBM.

Trong lớp các máy tính, có lớp được gọi là máy vi tính, hai yếu tố tổ chức và kiến trúc có mối quan hệ rất gần gũi. Những thay đổi về công nghệ không chỉ ảnh hưởng đến tổ chức mà còn dẫn đến sự ra đời của những kiến trúc mạnh hơn, phong phú hơn. Nói chung, đối với những chiếc máy này, thì sự đòi hỏi về tính tương thích từ thế hệ này sang thế hệ khác ít gay gắt hơn. Do vậy, các quyết định thiết kế có tính kiến trúc và thiết kế có tính tổ chức thường có sự tương tác lẫn nhau.

1.5. TÓM TẮT LỊCH SỬ PHÁT TRIỂN CỦA KIẾN TRÚC MÁY TÍNH

1.5.1. Các thế hệ máy tính

➤ *Thế hệ thứ nhất (1940 – 1950): Đèn chân không (Vacuum Tube)*

- *ENIAC – 1945:* Được thiết kế bởi Mauchly và Eckert, chế tạo bởi quân đội Mỹ, nhằm tính toán đường đi của đạn phục vụ cho ngành khoa học đạn đạo trong khoảng thời gian Chiến tranh thế giới thứ hai, nó bao gồm 18000 bóng đèn chân không và 1500 relay (relays), nặng hơn 30 tấn, tiêu thụ một lượng điện năng vào khoảng 140kW và chiếm một diện tích xấp xỉ 1393m², với khả năng thực hiện

5000 phép cộng trong một giây. Việc lập trình trên ENIAC là một công việc vất và vi phải thực hiện nối dây bằng tay qua việc đóng/mở các công tắc cũng như cắm vào hoặc rút ra các dây cáp điện. Với sự ra đời và thành công của máy ENIAC, năm 1946 được xem như năm mở đầu cho kỷ nguyên máy tính điện tử, kết thúc sự nỗ lực nghiên cứu của các nhà khoa học đã kéo dài trong nhiều năm liền trước đó.

- UNIVAC – 1950: Là máy tính thương mại đầu tiên.
- Kiến trúc Von Neumann: Goldstine và Von Neumann đưa ra đề xuất cái tiền ENIAC với ý tưởng là chương trình được lưu trữ trong bộ nhớ. Kiến trúc máy tính với ý tưởng này được biết đến như là kiến trúc "Von Neumann" và là cơ sở cho việc thiết kế mọi máy áo từ đó đến nay.

- Các đặc trưng:

- Sử dụng thiết bị phóng electron.
- Dữ liệu và chương trình được lưu trữ trong bộ nhớ đọc/ghi đơn giản.
- Truy cập bộ nhớ theo địa chỉ.
- Ngôn ngữ máy hoặc ngôn ngữ Assemble.
- Thực thi lệnh tuần tự.

➤ **Thế hệ thứ hai (1950 – 1964): Transistor**

- William Shockley, John Bardeen, and Walter Brattain phát minh ra transistor làm giảm kích thước máy tính và cải thiện độ tin cậy của máy tính.
- Hệ điều hành đầu ra đời: Nạp và thi hành lần lượt từng chương trình.
- Thực hiện chuyển mạch bật tắt bằng điều khiển điện.
- Sử dụng ngôn ngữ lập trình bậc cao.
- Thực hiện phép toán với dấu chấm động.

➤ **Thế hệ thứ ba (1964 – 1974): Mạch tích hợp (Integrated Circuits – IC)**

- Các chip vi xử lý tổ hợp bởi hàng ngàn transistor, toàn bộ mạch nằm trên một bản mạch máy tính.
- Sử dụng bộ nhớ bán dẫn.
- Nhiều mô hình máy tính với các đặc trưng thực thi khác nhau.
- Máy tính nhỏ hơn và không cần các phòng chứa đặc biệt.

➤ *Thế hệ thứ tư (1974 đến nay)*

Độ tích hợp rất cao (Very Large Scale Integration – VLSI)/Độ tích hợp cao (Ultra Large Scale Integration – ULSI).

- Kết hợp hàng triệu transistor.
- Bộ xử lý chip đơn và máy tính bảng mạch đơn đã phổ biến.
- Các máy tính cá nhân (Personal Computer – PC) ra đời.
- Sử dụng tín hiệu phô rộng cho việc truyền dữ liệu.
- Áp dụng trí tuệ nhân tạo (Artificial intelligence – AI).
- Máy tính song song lớn.

1.5.2. Sự phát triển của các tập lệnh

Sự tiến bộ chủ yếu của kiến trúc máy tính liên quan chặt chẽ với sự phát triển trong thiết kế tập lệnh.

- Kiến trúc tập lệnh (Instruction Set Architecture – ISA): Sự liên kết trung tượng giữa phần cứng và phần mềm mức thấp (Lowest – level Software):
 - + 1950: Một thanh ghi tích luỹ đơn (Single Accumulator).
 - + 1953: Các thanh ghi chỉ số cộng tích luỹ (Accumulator plus Index Registers): Manchester Mark I, IBM 700 series.
 - Cho phép xây dựng chương trình theo môđun tùy theo việc thực thi:
 - + 1963: Ứng dụng ngôn ngữ bậc cao: B5000.
 - + 1964: Đưa ra khái niệm họ máy tính: IMB 360.
 - Các máy có các loại thanh ghi với chức năng riêng:
 - + 1977 – 1980: Máy tính với tập lệnh phức hợp (Complex Instruction Sets computer – CISC) – kiến trúc CISC: Vax, Intel 432.
 - + 1963 – 1976: Kiến trúc Load – Store: CDC 6600, Cray 1.
 - + 1987: Máy tính với tập lệnh đơn giản hóa (Reduced Instruction Set Computer – RISC) – kiến trúc RISC: Mips, Sparc, HP-PA, IBM RS6000.

Sự tiến bộ chủ yếu của kiến trúc máy tính liên quan chặt chẽ với sự phát triển trong thiết kế tập lệnh.

1.6. PHÂN LOẠI KIẾN TRÚC MÁY TÍNH

Ngày nay có nhiều loại máy tính với các thuộc tính hệ thống khác nhau, và có nhiều cách phân loại khác nhau tùy theo góc nhìn.

1.6.1. Phân loại kiến trúc máy tính theo lịch sử phát triển bộ nhớ trong

– Kiến trúc Phi Von Neumann: Kiến trúc máy tính không có thành phần bộ nhớ chính với chức năng lưu trữ chương trình và dữ liệu, là kiến trúc máy tính trước năm 1946.

– Kiến trúc Von Neumann: Kiến trúc máy tính có thành phần bộ nhớ chính với chức năng lưu trữ chương trình và dữ liệu, là kiến trúc máy tính từ năm 1946.

1.6.2. Phân loại kiến trúc máy tính theo lịch sử phát triển tập lệnh

– Kiến trúc Load – Store (1963 – 1976).

– Kiến trúc CISC (1977 – 1980): Máy tính với tập lệnh phức hợp (Complex Instruction Sets computer – CISC).

– Kiến trúc RISC (1987): Máy tính với tập lệnh đơn giản hóa (Reduced Instruction Set Computer – RISC).

CÂU HỎI VÀ BÀI TẬP CHƯƠNG 1

1. Trình bày sơ đồ khối kiến trúc chung của hệ thống máy tính.
2. Trình bày sơ đồ khối cấu trúc chung của máy tính IBM-PC.
3. Phân biệt máy tính Von Neumann và Phi Von Neumann.
4. Trong các câu hỏi sau, hãy chọn câu trả lời thích hợp nhất:
 - 1) Trong khái niệm "chương trình được lưu trữ" do Von Neumann đưa ra, máy tính lấy chỉ thị từ:
 - a) Bộ nhớ;
 - b) Việc đóng/mở các công tắc điện do người vận hành máy thực hiện;
 - c) Đĩa cứng;
 - d) Chương trình.

- 2) Khái niệm "chương trình được lưu trữ" cho phép
- a) Nhiều thuật toán có thể được thực hiện bên trong máy mà không cần phải nối dây lại;
 - b) Chương trình có thể được thay đổi trực tiếp thông qua các giá trị lưu trong bộ nhớ;
 - c) Giảm thời gian thực thi chương trình;
 - d) Cả (a), (b) và (c) đều đúng.
- 3) Bộ vi xử lý Intel 80486 là
- a) Bộ vi xử lý 32 bit;
 - b) Bộ vi xử lý 16 bit có bộ đồng xử lý toán học được cài đặt bên trong CPU;
 - c) Bộ vi xử lý có bộ đồng xử lý toán học được cài đặt sẵn trong CPU;
 - d) Cả (a) và (c) đều đúng.

Chương 2

KIẾN TRÚC TẬP LỆNH

TÓM TẮT: – Lý thuyết: 6 tiết;
– Bài tập lớn: 10 tiết.

Mục tiêu cần đạt được	Các mục chính	Bài tập bắt buộc	Bài tập làm thêm
<ul style="list-style-type: none">Cung cấp các khái niệm cơ bản về thông tin, dữ liệu, quá trình xử lý dữ liệu và phương pháp biểu diễn dữ liệu ở dạng số hệ cơ số 2 (hệ 2) trong máy tính điện tử số. Cung cấp kiến thức tổng quan về tập thanh ghi, các dạng lệnh trong máy tính cũng như các phương thức định địa chỉ toán hạng.Kết thúc chương, người học cần nắm các khái niệm về thông tin, dữ liệu, đơn vị thông tin, phương pháp biểu diễn dữ liệu trong máy tính qua số hệ 2. Có khả năng thực hiện các phép toán trên số hệ 2, chuyển đổi được một số qua các hệ cơ số khác nhau. Nắm chắc chức năng của các loại thanh ghi cơ bản, các dạng lệnh và phương thức định địa chỉ toán hạng trong câu lệnh.	<ul style="list-style-type: none">Biểu diễn dữ liệu trong máy tính.Kiểu dữ liệu và độ chính xác dữ liệu.Tập các thanh ghi.Các kiểu lệnh.Các phương thức định địa chỉ toán hạng bộ nhớ.Các vấn đề về thiết kế địa chỉ.	<p>Câu hỏi và bài tập cuối chương: từ 1 đến 8; từ 16 đến 21 và 23, 27, 32.</p>	<p>Câu hỏi và bài tập cuối chương còn lại.</p>

2.1. BIỂU DIỄN DỮ LIỆU TRONG MÁY TÍNH

2.1.1. Thông tin – biểu diễn và xử lý thông tin

➤ *Thông tin*

Thông tin là sự hiểu biết, nhận thức của con người về thế giới khách quan; hay nói cách khác, thông tin là một đại lượng phi vật chất, mà con người chỉ có thể cảm nhận thông qua thế giới hiện thực khách quan. Muốn lưu trữ và truyền tải thông tin thì chúng ta phải vật chất hóa thông tin thông qua một đại lượng vật lý nào đó.

Tuy là đại lượng phi vật chất, nhưng bản thân thông tin lại có giá trị nhất định nào đó. Việc lượng giá thông tin phụ thuộc vào xác suất xuất hiện của thông tin. Nếu xác suất xuất hiện càng cao thì thông tin có giá trị càng thấp; ngược lại, xác suất xuất hiện thông tin càng thấp thì giá trị của thông tin càng cao.

➤ **Dữ liệu**

Dữ liệu chính là thông tin đã được vật chất hoá thông qua một đại lượng vật lý nào đó, hay nói cách khác, dữ liệu là một đại lượng mang tin. Muốn có thông tin thì chúng ta phải tập hợp và xử lý dữ liệu.

Trong thực tế, dữ liệu có thể là các loại tín hiệu vật lý (sóng điện từ, sóng âm thanh, tín hiệu điện, ký tín ám hiệu, tín hiệu ánh sáng....), có thể là các số liệu, hình ảnh, chữ viết,...

Ví dụ: Khi người bạn của bạn đọc dòng thông báo về kết quả điểm thi các môn của bạn trên màn hình, bạn đó sẽ cảm nhận được bạn là người học giỏi hay khá, có tư chất phù hợp với công việc gì sau này,... thông qua xử lý dòng thông báo trên màn hình trong não bộ của người đó.

Một ví dụ khác, bảng tính lương được lưu trữ trên máy tính, muốn lấy thông tin về những người có mức lương thấp hơn lương tối thiểu để có chính sách trợ cấp khó khăn thì phải qua một quá trình xử lý trên máy tính. Quá trình xử lý thông tin có thể là tìm kiếm lấy ra hay lọc theo một điều kiện nhất định.... hoặc có thể là giải mã dữ liệu đã được mã hoá,...

Cùng một thông tin, con người có thể biểu diễn thông tin này qua các đại lượng mang tin khác nhau, thể hiện qua các dạng dữ liệu khác nhau, ví dụ như văn bản (chữ viết), âm thanh, hình ảnh, đồ họa,... Hay nói cách khác là, thông tin nằm trong dữ liệu. Xử lý thông tin bao gồm nhiều quá trình xử lý dữ liệu (truyền tin, lọc nhiễu, loại bỏ thông tin dư thừa, lưu trữ, tìm kiếm, lấy ra, sao chép, mã hoá giải mã,...) để lấy ra thông tin hữu ích nhằm phục vụ đời sống con người.

➤ **Biểu diễn thông tin và xử lý dữ liệu trong máy tính**

Khi nói biểu diễn thông tin trong máy tính, chính là biểu diễn dữ liệu trong máy tính, tuy nhiên dùng thuật ngữ biểu diễn dữ liệu thì chính xác hơn.

Trong máy tính, thông tin được vật chất hoá thông qua tín hiệu điện, hay nói cách khác, dữ liệu trong máy tính được biểu diễn, lưu trữ và truyền tải thông qua tín hiệu điện.

Trong máy tính có rất nhiều bóng đèn, mỗi bóng đèn ở một trong hai trạng thái là sáng hay tắt:

- Trạng thái tắt (không có điện hay mức điện áp thấp) → cho ta tín hiệu 0.
- Trạng thái sáng (có điện hay mức điện áp cao) → cho ta tín hiệu 1.

Tập các tín hiệu 0/1 cho ta một số hệ cơ số 2 (hệ nhị phân – binary) biểu diễn một giá trị dữ liệu nào đó. Ví dụ, có 8 bóng đèn đánh số từ 0 đến 7, mỗi bóng ở một trạng thái tương ứng như sau: tắt, sáng, tắt, tắt, tắt, tắt, tắt, sáng; tức là, ta có tập các tín hiệu 0/1 tương ứng là 0, 1, 0, 0, 0, 0, 0, 1 và biểu diễn một số nhị phân là 01000001_b. Nếu dữ liệu lưu trữ là số thì tập tín hiệu này biểu diễn số 65, nếu dữ liệu lưu trữ là ký tự thì tập tín hiệu này biểu diễn chữ "A".

Quá trình xử lý dữ liệu trong máy tính có thể tóm tắt như sau:

- Nhận dữ liệu đầu vào đã được số hoá.
- Xử lý dữ liệu thông qua dãy các lệnh.
- Đưa ra kết quả xử lý và lưu trữ dữ liệu đầu ra.

2.1.2. Đơn vị thông tin

Như chúng ta đã biết, thông tin có thể tạo ra, phát sinh, truyền đi, lưu trữ và chọn lọc. Thông tin cũng có thể bị méo mó, sai lệch do nhiều tác động hay do tác động của con người,...

Người ta có thể *định lượng* tin tức bằng cách do độ bất định của hành vi, trạng thái. Xác suất xuất hiện một tin *càng thấp* thì lượng tin *càng cao* vì độ bất ngờ của nó càng lớn. Ta hãy tưởng tượng, nếu bàn thân được thông báo cho biết điểm thi trong khi ta đã biết chính xác nó rồi, lúc đó ta nói lượng tin bằng 0 (giá trị tin tức đem lại bằng không). Còn nếu tin tức đó được đưa đến trong tâm trạng ta đang chờ đợi, nhất là khi đang nghĩ mình có thể bị điểm kém, mà lại nhận được tin đúng đắn là được điểm cao thì thông tin này có lượng tin càng cao, càng giá trị.

Để định lượng tin tức, năm 1948 Shannon đã đưa ra công thức tính *lượng tin*, được gọi là *Entropi*:

$$H = - \sum_{i=1}^n p_i \log_a(p_i),$$

trong đó p_i là xác suất xuất hiện sự kiện i của hệ và hệ có khả năng khác nhau.

Để chỉ ra được đơn vị đo, cần chỉ rõ ra được cơ số hoá hàm logarit:

- Nếu $a = 2$ thì đơn vị đo được gọi là *đơn vị nhị phân* (Binary uniT – bit).
- Nếu $a = e$ thì đơn vị đo được gọi là *đơn vị tự nhiên* (NAtural uniT – Nat).
- Nếu $a = 10$ thì đơn vị đo được gọi là *đơn vị thập phân* (Hartley) – ít dùng.

Trong trường hợp hệ có hai khả năng ($a = 2$), ví dụ như lượng tin của việc xuất hiện mặt sấp (hoặc ngửa) khi tung đồng tiền thì lượng tin của việc xuất hiện mặt sấp (hoặc ngửa) là:

$$H = -\sum_{i=1}^2 \frac{1}{2} \log_2 \left(\frac{1}{2} \right) = -\log_2 \left(\frac{1}{2} \right) = \log_2 2 = 1 \text{ (bit)}.$$

Như vậy, trong trường hợp hệ có hai trạng thái ($a = 2$), hệ cơ số 2 cho ta đơn vị đo thông tin 1 bit (bit viết tắt bởi Binary unit).

Trong máy tính để biểu diễn một giá trị số, chúng ta dùng *hệ cơ số 2* hoặc nói ngắn gọn hơn là *hệ 2* (Binary number system, viết tắt là hệ B). Trong đó chỉ tồn tại hai chữ số 0 và 1 để biểu diễn các giá trị số (ứng với hai trạng thái: không có điện và có điện). 0 và 1 cũng là các giá trị có thể có của một chữ số hệ 2.

Mỗi một bit cho ta biết được trạng thái của một tín hiệu điện trên một đường dây tại một thời điểm: điện áp ở mức cao (có điện) là 1, điện áp ở mức thấp (không có điện) là 0.

Bit là đơn vị cơ sở để xác định dung lượng của bộ nhớ, bộ nhớ được tổ chức theo byte.

2.1.3. Số nguyên và số thực

a) Hệ đếm

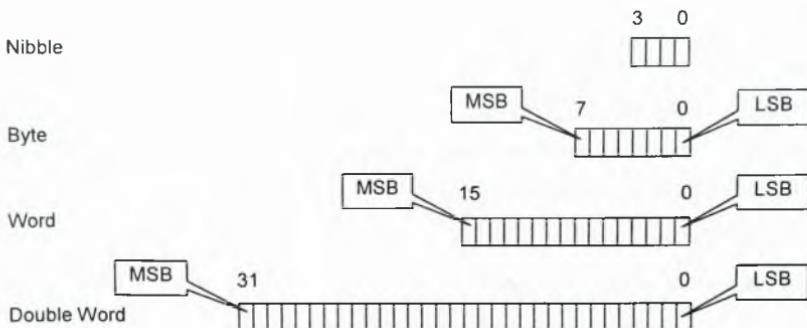
➤ *Hệ đếm thập phân (hệ cơ số 10)*

Trong cuộc sống hàng ngày, ta dùng *hệ đếm thập phân* hay còn gọi là *hệ cơ số 10*, hoặc nói ngắn gọn là *hệ 10* (Decimal number system, viết tắt là hệ D) để biểu diễn các giá trị số. Điều này là rất tự nhiên, vì từ xa xưa một con người bình thường đã biết dùng 10 ngón tay để đếm. Trong hệ thống này, ta dùng tổ hợp của các chữ số 0, 1, ..., 9 để biểu diễn các giá trị số, đi kèm theo tập hợp có thể dùng thêm chữ D hoặc d ở cuối để chỉ ra rằng đó là hệ 10 (một chữ số không có ký hiệu chữ đi kèm ở sau thì ta ngầm hiểu đó là số hệ 10).

➤ *Hệ đếm nhị phân (hệ cơ số 2)*

Hệ đếm nhị phân hay còn gọi là *hệ cơ số 2*, hoặc nói ngắn gọn là *hệ 2* là hệ đếm dùng trong máy tính. Một số hệ 2 gồm các bit được đánh dấu bằng chữ B hoặc b đi kèm ở cuối để phân biệt với các hệ khác khi làm việc cùng một lúc với nhiều hệ đếm khác nhau. Một cụm 4 bit sẽ tạo thành 1 *nibble*, một cụm 8 bit tạo

thành 1 byte, một cụm 16 bit tạo thành 1 word (một từ), một cụm 32 bit tạo thành 1 double word (từ kép). Chữ số đầu tiên bên trái trong dãy các số hệ 2 loại 8 bit, 16 bit, 32 bit... gọi là *bit có ý nghĩa lớn nhất* (Most Significant Bit – MSB); còn bit cuối cùng bên phải (bit 0) trong dãy gọi là *bit có ý nghĩa bé nhất* (Least Significant Bit – LSB). Ứng với thứ tự đếm 1, 2, 3,... ở hệ 10 thì ở hệ 2 ta có 1, 10b, 11b,...



Hình 2.1. Các đơn vị đo độ dài của số hệ 2 dẫn xuất từ bit

➤ Hệ đếm thập lục phân

Nếu ta dùng số hệ 2 để biểu diễn các số có giá trị lớn sẽ gặp điều bất tiện là số hệ 2 thu được quá dài. Ví dụ, để biểu diễn số 255 ta cần 8 bit như sau:

$$255 = 11111111_b.$$

Trong thực tế, để viết kết quả cho gọn lại, người ta tìm cách nhóm 4 số hệ 2 (1 nibble) thành một số *hệ đếm thập lục phân* hay còn gọi là *hệ cơ số 16*, hoặc cho gọn là *hệ 16*. Hệ 16 dùng tất cả 4 bit để biểu diễn các giá trị cho một chữ số, một chữ số có giá trị từ 0 đến 15. Để làm được điều này người ta sử dụng các chữ số có sẵn ở hệ 10 (0, 1, 2, ..., 9) để biểu diễn các giá trị số tương ứng từ 0 đến 9; còn các số từ 10 đến 15, ta dùng thêm các chữ cái A, B, C, D, E, F hoặc a, b, c, d, e, f để biểu diễn các giá trị còn lại tương ứng là 10, 11, 12, 13, 14, 15. Để phân biệt với các số hệ khác, ta kèm theo chữ H hoặc h ở cuối để phân biệt với các hệ đếm khác. Ví dụ số 255 được biểu diễn qua hệ 16 là 255 = FFh.

b) Chuyển đổi giữa các hệ đếm

Ta gọi a là giá trị cơ số của hệ đếm ($a = 2, 8, 10$ hoặc 16), n là số chữ số biểu diễn giá trị cho một số P , trong đó có k ($k \leq n$) chữ số phần nguyên, chỉ số vị trí

của các chữ số phần nguyên trong P là từ 0 đến $k - 1$ tính từ dấu phẩy thập phân sang trái; chỉ số vị trí các chữ số phần thập phân trong P là từ -1 đến $-n + k$, m_i là giá trị của chữ số có chỉ số vị trí i trong P ($0 < m_i < a$, $i = -n + k, \dots, k - 1$).

Ta có công thức chuyển đổi số P hệ cơ số a về số P hệ cơ số 10 như sau:

$$P = a^{k-1} \times m_{k-1} + a^{k-2} \times m_{k-2} + \dots + a^1 \times m_1 + m_0 \\ + a^{-1} \times m_{-1} + \dots + a^{-n+k} \times m_{-n+k}. \quad (1.1)$$

➤ **Đổi số hệ 2 sang hệ 10**

Muốn đổi một số từ hệ 2 sang số hệ 10 tương ứng, ta áp dụng công thức (1.1) ở trên với cơ số $a = 2$; có nghĩa là chỉ cần tính các giá trị 2^i tương ứng với các chỉ số khác 0 thứ i của số hệ 2 rồi cộng chúng lại.

Ví dụ: $10111,11_b = 2^4 + 2^2 + 2^1 + 1 + 2^{-1} + 2^{-2} = 25,75.$

➤ **Đổi số hệ 16 sang hệ 10**

Tương tự như đổi số hệ 2 sang hệ 10. Ta xét ví dụ sau:

$$1AF7h = 1 \times 16^3 + 10 \times 16^2 + 15 \times 16^1 + 7 = 6903.$$

➤ **Đổi số hệ 10 sang hệ 2**

- **Đổi phần nguyên sang hệ 2:**

– *Cách 1:* Lấy số hệ 10 cần đổi trừ đi 2^x (với x là giá trị lớn nhất của số m_i chọn sao cho 2^x nhỏ hơn hoặc bằng số hệ 10 cần đổi), ghi lại giá trị 1 cho chữ số hệ 2 ứng với 2^x . Tiếp tục làm như vậy đối với hiệu số vừa tạo ra và các chữ số 2^i bậc thấp hơn cho tới khi đạt tới 2^0 và ghi lại các giá trị 0 hoặc 1 cho chữ số hệ 2 thứ i tuỳ theo quan hệ giữa số dư và luỹ thừa tương ứng:

$$= 1 \text{ khi số dư } \geq 2^i;$$

$$= 0 \text{ khi số dư } \leq 2^i.$$

Ví dụ: Đổi 34 sang số hệ 2:

Các giá trị 2^i cần tính đến ($2^5 = 32$ là giá trị 2^x sát dưới nhất so với 34):

$$2^5 \quad 2^4 \quad 2^3 \quad 2^2 \quad 2^1 \quad 2^0$$

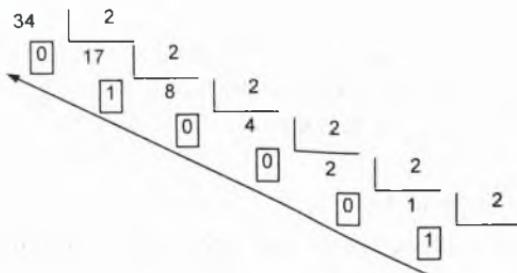
Các chữ số hệ 2 tính được tương ứng:

$$1 \quad 0 \quad 0 \quad 0 \quad 1 \quad 0$$

Như vậy $34 = 100010_b$.

– Cách 2: Lấy số cần đổi chia cho 2 và ghi nhớ phần dư, tiếp theo lấy thương của phép chia vừa nhận được chia cho 2 và ghi nhớ phần dư. Cứ như vậy cho đến khi thương bằng 0. Đảo ngược thứ tự dãy các số dư, ta sẽ được các chữ số của số hệ 2 cần tìm.

Ví dụ: Đổi số 34 sang số hệ 2 ta thực hiện như sau:



Các số dư trong khung sẽ được sắp xếp theo chiều mũi tên (lấy ngược lại với thứ tự thực hiện phép chia), ta được kết quả là 100010b.

Trong trường hợp số hệ 10 có thêm phần lẻ sau dấu thập phân cần đổi sang số hệ 2 thì ta phải thực hiện đổi riêng rẽ từng phần rồi sau đó ghép kết quả lại.

• *Riêng với phần lẻ sau dấu phẩy thập phân ta làm như sau:*

Lấy số cần đổi nhân với 2, tích nhận được sẽ gồm phần nguyên và phần lẻ thập phân, ghi nhớ phần nguyên; lấy phần lẻ thập phân của tích vừa thu được nhân với 2. Cứ tiếp tục như vậy cho đến khi tích được chẵn bằng 1 (không còn phần lẻ thập phân). Lấy các phần nguyên đã lưu theo thứ tự thực hiện phép nhân, sắp xếp lại ta được các chữ số sau dấu phẩy nhị phân (phần lẻ nhị phân) cần tìm.

Ví dụ: Đổi 0,125 sang số hệ 2:

$$\begin{aligned}0,125 \times 2 &= 0,250 \\0,250 \times 2 &= 0,500 \\0,500 \times 2 &= 1,000\end{aligned}$$

Kết quả ta thu được số hệ 2 là 0,001b (như thứ tự phần được đóng khung).

Kết hợp hai ví dụ trên với số 34,125 đổi sang số hệ 2 ta được kết quả là

$$34,125 = 100010,001b.$$

➤ Đổi số hệ 10 sang hệ 16

Thực hiện tương tự như đổi số hệ 10 về số hệ 2, ta thực hiện chia hết quanh phần thương liên tiếp cho 16, lấy giá trị phần dư rồi sắp xếp theo thứ tự ngược lại.

Tóm lại, để đổi một số ra cách biểu diễn ở các hệ cơ số khác nhau nhanh chóng, tránh thực hiện quá nhiều phép chia, ta thực hiện theo thứ tự:

Số hệ 10 \longleftrightarrow Số hệ 16 \longleftrightarrow Số hệ 2 \longleftrightarrow Số hệ 8 (hệ bát phân)

Vì việc thực hiện đổi số hệ 16 sang số hệ 2 rất đơn giản, chỉ việc đổi từng chữ số hệ 16 ra hệ 2 rồi ghép chúng lại theo thứ tự. Từ số hệ 2 về số hệ 8 ta lấy từng cụm 3 bit một bắt đầu từ LSB để đổi ra các số từ 0 đến 7, sau đó ghép đúng theo thứ tự.

Ví dụ: Đổi số 125 ra hệ 16.

Ta lấy 125 chia 16 được 7 dư 13 = C, nhớ C. Lấy 7 chia 16 được 0, dư 7. Kết quả ta được 7Ch.

Ví dụ: Đổi số 125 ra hệ 2.

Trước hết ta đổi 125 về hệ 16, ta được 7Ch. Tiếp tục đổi số hệ 16 là 7Ch về số hệ 2 như sau:

$$7h = 0111b;$$

$$Ch = 1101b.$$

Ghép lại ta được số nhị phân: 01111101b.

Nếu tiếp tục đổi về hệ cơ số 8 (nhóm đủ 3 bit từ phải sang trái) ta được:

$$\underline{01111101}b = 175(8).$$

c) Các phép toán số học với số hệ 2

➤ Phép cộng

Phép cộng các số hệ 2 giống như khi ta thực hiện với số hệ 10. Quy tắc phép cộng số hệ 2 được chỉ ra trong bảng 2.1.

Bảng 2.1. $y = a + b$

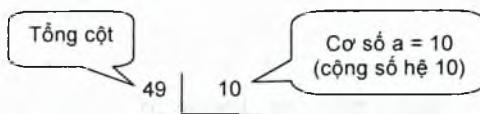
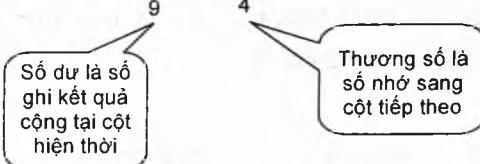
a	b	y	Nhớ
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Bảng 2.2. Mở rộng bảng 2.1, $y = a + b + c + d + e + f + g$

a	b	c	d	e	f	g	y	Nhớ
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	1	0
0	0	0	0	0	1	0	1	0
0	0	0	0	0	1	1	0	1
0	0	0	1	1	1	0	2	
0	0	1	1	1	1	1	1	2
1	0	1	1	1	1	1	0	3
1	1	1	1	1	1	1	1	3

Nhận xét: Máy tính không thực hiện phép cộng theo như bảng 2.2 mà chỉ thực hiện cộng hai toán hạng. Bảng 2.2 là cách giúp ta cộng tay nhanh với nhiều toán hạng, ta thấy kết quả tổng cột chia cho 2 (cơ số hệ đếm nhị phân) là giá trị cần nhớ sang cột tiếp theo, còn phần dư chính là y (phần ghi kết quả cộng cột). Như vậy cách thực hiện không khác gì với số hệ 10.

Ví dụ, cộng nhiều số thập phân, đầu tiên ta cộng cột hàng đơn vị như sau:

Giá trị nhớ: 4 1979 1187 + 677 129 5679 <hr/> 458 <hr/> 9		
--	--	---

Ta được giá trị là 49, lấy 49 chia cho 10 (cơ số hệ đếm thập phân), được 4 dư 9, viết 9 và nhớ 4. Giá trị nhớ sẽ được dùng để cộng với cột ở vị trí tiếp theo ngay sau đó.

Ví dụ tương tự ở hệ 2, cộng nhiều số nhị phân 8 bit. Đầu tiên cộng các số nhị phân ở cột có trọng số là 0, được 1, nhớ 2. Tương tự ta cộng các cột tiếp theo như sau:

Giá trị nhớ: 134555432

$$\begin{array}{r}
 01111011 \\
 11001111 \\
 + 10111111 \\
 00111101 \\
 01111111 \\
 \hline
 01111110 \\
 \hline
 1101000011
 \end{array}$$

Tổng cột

5

Cơ số $a = 2$
(cộng số hệ 2)

1

2

Số dư là số ghi
kết quả cộng tại
cột hiện thời

Thương số là
số nhớ sang
cột tiếp theo

Bảng 2.1 chính là cách mà các bộ cộng trong các khối tính toán số học của máy tính thực hiện.

➤ Phép trừ

Phép trừ các số hệ 2 giống như làm với số hệ 10. Bảng 1.3 là quy tắc thực hiện phép trừ số nhị phân bằng tay.

Bảng 2.3. Quy tắc trừ số nhị phân $y = a - b$

A	b	y	Mượn
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Trong máy tính, không có bộ trừ, chỉ có bộ cộng để thực hiện phép cộng. Do vậy, với phép toán $C = A - B$, máy tính sẽ thực hiện là $C = A + (-B)$, trong đó $(-B)$ là số bù 2 của B .

- Số bù 2:

Trong máy tính để tận dụng các bộ cộng đã có sẵn, với phép trừ được chuyển đổi thành phép cộng số bị trừ với số đảo dấu của số trừ, số đảo dấu của số trừ chính là một số âm. Khi thực hiện phép trừ trong máy tính tất yếu cũng sẽ có kết quả là một số âm. Vấn đề đặt ra đối với số hệ 2 là phải biểu diễn số âm nhị phân như thế nào cho thích hợp để đáp ứng được việc sử dụng các bộ cộng trong máy tính. Việc sử dụng số bù 2 chính là cách biểu diễn số có dấu trong máy tính ngày nay.

Vậy, số bù 2 của số A là số đảo dấu của chính số A đó.

Ví dụ, số bù 2 của 5 là -5 và ngược lại, số bù 2 của -5 là 5.

- Phương pháp xây dựng số bù 2

– Trước hết xây dựng số bù 1 của A. Số bù 1 của A chính là số nhận được khi ta đảo tất cả các bit của số A.

Ví dụ: $5 = 00000101_b$, số bù 1 của 5 là 11111010_b .

– Số bù 2 của A bằng số bù 1 của A cộng 1.

Ví dụ: Số bù 2 của 5 = số bù 1 của 5 + 1 = $11111010_b + 1 = 11111011_b = -5$.

Bảng 2.4. Biểu diễn các số theo số hệ 2, số hệ 2 có dấu và số bù 2

Số hệ 2 (8 bit)	Số hệ 10 tương đương (số không dấu = số nguyên dương)	Số hệ 10 tính theo số bù 2
0000 0000	0	0
0000 0001	1	+1
0000 0010	2	+2
...
0111 1101	125	+125
0111 1110	126	+126
0111 1111	127	+127
1000 0000	128	-128
1000 0001	129	-127
1000 0010	130	-126
...
1111 1101	253	-3
1111 1110	254	-2
1111 1111	255	-1

➤ Phép nhân

Phép nhân số hệ 2 thực hiện giống như với số hệ 10. Chỉ cần chú ý khi cộng kết quả tuân thủ theo bảng 1.2 (quy tắc cộng mở rộng).

Quy tắc thực hiện phép nhân hệ 2 được chỉ ra trong bảng 2.5.

Bảng 2.5. Quy tắc thực hiện phép nhân $y = a \times b$

a	b	y
0	0	0
0	1	0
1	0	0
1	1	1

Trên cơ sở quy tắc vừa nêu và để đơn giản ta thực hiện ví dụ một phép nhân hai số hệ 2 với độ dài 4 bit để làm sáng tỏ thuật toán nhân thực hiện trong máy tính:

$$\begin{array}{r}
 \begin{array}{l} 1001b \\ \times \quad 0110b \\ \hline \end{array} & \begin{array}{l} \text{Số bị nhân} = 9 \\ \text{Số nhân} = 6 \end{array} \\
 \begin{array}{r} 0000 \\ 1001 \\ + 1001 \\ \hline 0000 \end{array} & \begin{array}{l} \text{Thành phần 1 của tổng tích luỹ} \\ \text{Thành phần 2 của tổng tích luỹ} \\ \text{Thành phần 3 của tổng tích luỹ} \\ \text{Thành phần 4 của tổng tích luỹ} \end{array} \\
 0110110b & \text{Tổng tích luỹ bằng } 54
 \end{array}$$

Độ dài cực đại của kết quả trong trường hợp này là 8 bit. Nếu nhân các số 8 bit (hoặc 16 bit) thì độ dài cực đại của kết quả là 16 bit (hoặc 32 bit). Mỗi lần nhân một bit khác 0 của số nhân với số bị nhân, ta thu được chính số bị nhân đã dịch trái nó một số lần (một số bit) tương ứng với vị trí các bit khác 0, tạo ra 1 thành phần của tổng tích luỹ. Tổng của các thành phần như trên là kết quả của phép nhân.

Phân tích quá trình trên ta thấy, phép nhân có thể thực hiện theo thuật toán cộng và dịch như sau:

- Thành phần đầu tiên của tổng tích luỹ thu được là tích của LSB trong số nhân với số bị nhân. Nếu $LSB = 0$ thì thành phần này bằng 0; nếu $LSB = 1$ thì thành phần này bằng chính số bị nhân.
- Mỗi thành phần thứ i tiếp theo của tổng tích luỹ sẽ tính được bằng cách tương tự, nhưng phải dịch trái đi i bit (có thể bỏ qua các thành phần = 0).
- Tổng của các thành phần là tích cần tìm.

Để minh họa thuật toán trên, ta dùng luôn nó để rút gọn ví dụ đã làm ở trên:

\times	1001	Số bị nhân = 9
	<u>0110</u>	Số nhân = 6
	1001	Số bị nhân dịch trái 1 lần
+	<u>1001</u>	Số bị nhân dịch trái 2 lần
	0110110	Tổng tích luỹ bằng 54

Trong máy tính, phép nhân được thực hiện bởi bộ cộng và bộ dịch trái theo thuật toán cộng và dịch vừa trình bày.

➤ Phép chia

Phép chia là phép tính ngược của phép nhân. Từ đó suy ra phép chia có thể thực hiện bằng các phép trừ và phép dịch phải liên tiếp cho tới khi không thể trừ được nữa (do không còn gì để trừ hoặc số bị trừ nhỏ hơn số chia). Sau đây là thuật toán thực hiện phép chia thông qua ví dụ cụ thể:

Ví dụ: $35 / 5 = 7$. Ta quan sát các bước phải thực hiện khi thực hiện phép chia tay:

$$\begin{array}{r}
 100011_2 \quad | \quad 101 \\
 -\underline{000} \quad \quad \quad 0111_2 \\
 1000 \\
 -\underline{101} \\
 0111 \\
 -\underline{0101} \\
 \quad \quad \quad 01 \\
 -\underline{101} \\
 \quad \quad \quad 000
 \end{array}$$

Trong các phép tính ở trên, ta liên tục cần phải dự đoán và kiểm tra để tìm ra kết quả đúng. Công việc này rất khó khăn với các mạch điện tử của khối tính toán số học (vốn là các phần tử để thực hiện phép cộng và dịch trong máy tính).

Sau đây là một thuật toán khắc phục nhược điểm nêu trên:

Trong máy tính:

- Nếu số chia là 1 byte thì số bị chia phải là 2 byte; do vậy, khi số bị chia là 1 byte, ta phải mở rộng bit dấu cho số bị chia lên 8 bit cao.
 - Nếu số chia là 2 byte thì số bị chia phải là 4 byte; do vậy, số bị chia là 1 word, ta phải mở rộng bit dấu cho số bị chia lên 16 bit cao.
1. Kiểm tra số bị chia, mở rộng bit dấu nếu cần.

2. Đổi số chia ra số bù 2 của nó (để bước tiếp theo làm phép cộng với số chia thay cho phép trừ): $K =$ số bù 2 của số chia dịch trái đi 8 bit (nếu số chia là 8 bit) hoặc 16 bit (nếu số chia là số 16 bit). $Q =$ số bị chia.

$$3. H = Q + K.$$

– Nếu H có MSB = 0 (có nghĩa là phần này của số bị chia chia được cho chia) thì bit tương ứng của thương bằng 1: $Q = H$.

– Nếu kết quả này có MSB = 1 (có nghĩa là phần này của số bị chia không chia được cho số chia) thì bit tương ứng của thương bằng 0.

4. Số $K =$ số K dịch phải đi 1 bit và làm lại bước 3 cho đến khi nhận được $H = 0$ (chia hết) hoặc $H <$ số chia (chia còn dư).

5. Kết quả của thương số là các bit 0, 1 ghép theo trình tự các bit tương ứng của thương.

Ví dụ: $103 / 6 = 17$ dư 1. Thực hiện phép chia này ở hệ 2 theo thuật toán ta có: Số chia = 00000110b;

Số bù 2 của số chia = 11111010b;

Số bị chia = 01100111b = 0000000001100111b (mở rộng bit dấu).

Thuật toán chia số nhị phân số 103 / 6 xem bảng 2.6.

Bảng 2.6. Mô phỏng thuật toán chia số 103 / 6 ở hệ cơ số 2 trong máy tính

Tên toán hạng, thao tác	Giá trị	Thương
$Q =$ Số bị chia	0000000001100111	
$K =$ Số bù 2 của số chia $<< 8$ bit	1111101000000000	
$H = Q + K$	1111101001100111	0
Q	0000000001100111	
$K = K >> 1$ bit (giữ nguyên bit dấu)	1111110100000000	
$H = Q + K$	111111011100111	0
Q	0000000001100111	
$K = K >> 1$ bit (giữ nguyên bit dấu)	1111111010000000	
$H = Q + K$	111111101100111	0
Q	0000000001100111	
$K = K >> 1$ bit (giữ nguyên bit dấu)	1111111101000000	

Tên toán hạng, thao tác	Giá trị	
$H = Q + K$	1111111110100111	0
Q	0000000001100111	
$K = K >> 1$ bit (giữ nguyên bit dấu)	1111111110100000	
$H = Q + K > 6$	0000000000000111	1
$Q = H$	00000000000000111	
$K = K >> 1$ bit (giữ nguyên bit dấu)	1111111111010000	
$H = Q + K$	1111111111010111	0
Q	00000000000000111	
$K = K >> 1$ bit (giữ nguyên bit dấu)	1111111111101000	
$H = Q + K$	1111111111101011	0
Q	00000000000000111	
$K = K >> 1$ bit (giữ nguyên bit dấu)	1111111111110100	
$H = Q + K$	1111111111111011	0
Q	00000000000000111	
$K = K >> 1$ bit (giữ nguyên bit dấu)	11111111111111010	
$H = Q + K < 6$	0000000000000001	1

Vậy ta có kết quả = $10001_2 = 17$, số dư $H = 1$.

d) Biểu diễn số nguyên

Phần cứng của máy tính cần giới hạn độ lớn của một số để có thể lưu nó trong các thanh ghi hay các ô nhớ. Do vậy, các số nguyên được chia thành các số 8 bit, 16 bit, 32 bit, 64 bit,... tương ứng với chúng có MSB là bit 7, 15, 31, 63 và LSB đều là bit 0.

➤ Số nguyên không dấu

Số nguyên không dấu (unsigned integer) biểu diễn các số nguyên dương. Số nguyên không dấu rất thích hợp để biểu diễn các đại lượng luôn dương. Chẳng hạn, địa chỉ của ô nhớ, bộ đếm, hay mã ASCII của ký tự. Bởi vì các số nguyên không dấu được định nghĩa là các số nguyên dương, nên không cần dùng bit nào để biểu diễn bit dấu. Do đó 8 bit của 1 byte hay 16 bit của 1 word đều được dùng để biểu diễn giá trị.

– Số nguyên không dấu lớn nhất có thể chứa trong 1 byte là

$$11111111b = FFh = 255.$$

– Số nguyên không dấu lớn nhất có thể chứa trong một word là

$$1111111111111111b = FFFFh = 65535.$$

Chú ý rằng, LSB = 1 thì số nguyên là số lẻ và ngược lại nó là số chẵn.

➤ **Số nguyên có dấu**

Số nguyên có dấu (signed integer) có thể là số dương hoặc âm. MSB được dùng để biểu diễn dấu của số, MSB = 1 cho biết đó là số âm, MSB = 0 cho biết đó là số dương. Các số nguyên có dấu được lưu trữ trong máy tính dưới dạng số bù 2.

Một ưu điểm của việc biểu diễn số âm trong máy tính bằng số bù 2 là phép trừ có thể thực hiện bằng phương pháp lấy bù và phép cộng. Các vi mạch thực hiện phép cộng và bù các bit được thiết kế tương đối dễ dàng.

e) Biểu diễn số thực

Trong máy tính, số thực được biểu diễn và được viết dưới dạng dấu phẩy động (dạng số mũ). Còn trong thực tế con người biểu diễn số thực ở hai dạng:

– Dạng bình thường, chẳng hạn như: 3.14; 3.0; -24.1234567; -0.0002. Việc dùng dấu chấm thay vì dấu phẩy để tách phần nguyên và phần thực của số thực là dùng cách viết của các nước Anh, Mỹ.

– Dạng viết với dạng số mũ gọi là *số thực dấu chấm động*.

Để hiển thị ra màn hình một số thực dạng bình thường, hay dạng viết dấu chấm động phải thông qua giả lập bằng phần mềm.

Số thực được biểu diễn trong bộ nhớ máy tính như thế nào được đề cập ở mục 2.2 – Kiểu dữ liệu và độ chính xác dữ liệu.

Số chấm động: Một số thực R được biểu diễn dưới dạng số có dấu chấm động chuẩn hoá có dạng như sau:

$$R = (-1)^S \times M \times a^E,$$

trong đó: M là phần định trị được chuẩn hoá;

E là số mũ;

a là cơ số cho biết số ở hệ cơ số nào;

S đặc trưng cho dấu của số, S = 0 thì R là số dương, ngược lại S = 1 thì R là số âm.

Ví dụ:

– Số thực là 84537.8765275 được chuẩn hoá ở dạng dấu chấm động là

$$84537.8765275 = (-1)^0 \times 8.45378765275 \times 10^4.$$

– Số thực là 0.0045378765275 được chuẩn hoá ở dạng dấu chấm động là

$$0.0045378765275 = (-1)^0 \times 4.5378765275 \times 10^{-3}.$$

– Số thực là 1011.001b được chuẩn hoá ở dạng dấu chấm động là

$$1011.001b = (-1)^0 \times 1.011001b \times 2^3.$$

– Số thực là -537.8765275 được chuẩn hoá ở dạng dấu chấm động là

$$-537.8765275 = (-1)^1 \times 5.378765275 \times 10^2.$$

– Số thực là -0.00045378765275 được chuẩn hoá ở dạng dấu chấm động là

$$-0.00045378765275 = (-1)^1 \times 4.5378765275 \times 10^{-4}.$$

– Số thực là 0.00011011011001b được chuẩn hoá ở dạng dấu chấm động là

$$0.00011011011001b = (-1)^0 \times 1.1011011001b \times 2^{-4}.$$

Hầu hết các máy PC ngày nay đều sử dụng chuẩn IEEE 754 (Institute of Electrical & Electronics Engineers) cho số học dấu chấm động nhị phân (binary floating – point Arithmetic) theo nguyên lý che số 1 (hidden 1 principle) trong phần hệ số.

Dạng tổng quát của số dấu chấm động hệ 2 theo nguyên lý che số 1 như sau:

$$R = (-1)^S \times 1.M \times 2^{E - Bias} \quad (2.1)$$

Số thực dấu chấm (phẩy) động được biểu diễn bằng 2 phần chính và một bit dấu, theo khuôn dạng:

S	E	M
---	---	---

Trong đó:

– Phần định trị M (Mantissa) chỉ lưu phần lẻ nhị phân của phần định trị đã được chuẩn hoá (chứa các bit của số sau dấu chấm nhị phân). Phần nguyên luôn bằng 1 không lưu.

– Phần mũ E (Exponent) dùng để điều chỉnh lại vị trí của dấu chấm (phẩy) nhị phân về đúng vị trí thật trong số đó. Vì giá trị E lưu trữ phải luôn không âm,

mà trong thực tế số mũ thật sự lại có thể âm, nên sử dụng thủ thuật là dùng một Bias lấy lại giá trị thực của số mũ. Tuỳ vào từng loại số thực khác nhau trong máy tính và độ dài bit của trường E mà Bias được xác định giá trị cụ thể khác nhau và Bias luôn là số dương. Ví dụ, nếu ta chọn 8 bit cho phần mũ thì số $2^8 / 2 - 1 = 2^7 - 1 = 127$ sẽ được chọn làm Bias.

– Bit dấu: bit cao nhất, nếu bit dấu = 1 thì số thực là số âm, ngược lại số thực là số dương.

Ví dụ 1: Số thập phân -4.5 biểu diễn dạng nhị phân là $-100.1b$ và được chuẩn hoá như sau:

$$-100.1b = (-1)^1 \times 1.001b \times 2^2.$$

Như vậy, ta thấy giá trị lưu trữ M, E, S sẽ là:

– Phần biểu diễn dấu chấm động của nó có phần định trị là $1.001b$. Như vậy giá trị M = 001...b (không được bỏ hai chữ số 0 phía trước, mặc dù viết độc lập nó không có nghĩa).

– Phần mũ là 2, tức $E - Bias = 2 \Rightarrow E = Bias + 2$.

– Bit dấu S = 1.

Ví dụ 2: Số thập phân 0.0625 biểu diễn ở dạng nhị phân là $0.0001b$ và được chuẩn hoá như sau:

$$0.0001b = 1.00...b \times 2^{-4}.$$

Như vậy, ta thấy giá trị lưu trữ M, E, S sẽ là:

– Phần biểu diễn dấu chấm động của nó có phần định trị là $1.000b$. Như vậy giá trị M = 000...b.

– Phần mũ là -4 , tức $E - Bias = -4 \Rightarrow E = Bias - 4$.

– Bit dấu S = 0.

Và như vậy, nếu dùng 8 bit để lưu trữ E thì Bias = 127 và E = 123; tức là đảm bảo E luôn không âm.

Tóm lại: Với công thức biểu diễn số chấm động ở dạng nhị phân trên máy tính như công thức (2.1) ở trên thì phần nguyên luôn bằng 1, phần lẻ sau dấu chấm (phẩy) động là M; phần mũ là E-Bias, trong đó Bias là giá trị khử dấu cho phần mũ. Việc lưu trữ số dấu chấm động trên máy tính chỉ lưu các giá trị S, M, E. Việc xem xét cụ thể một số dạng dấu chấm động được trình bày ở mục 2.2.

2.1.4. Biểu diễn ký tự

Không phải mọi số liệu mà máy tính xử lý đều là các con số. Các thiết bị ngoại vi như màn hình hay máy in đều có xu hướng làm việc với các ký tự. Ngoài ra, các chương trình như chương trình xử lý văn bản chuyên làm việc với dữ liệu kiểu ký tự. Cũng như tất cả các dữ liệu khác, các ký tự cũng cần phải được mã hóa thành dạng nhị phân để máy tính có thể xử lý chúng. Một kiểu mã hóa thông dụng nhất cho các ký tự đó là mã ASCII (American Standard Code For Information Interchange – Mã chuẩn của Mỹ dùng để trao đổi thông tin). Trước đây mã ASCII được sử dụng trong thông tin với các thiết bị teletype, ngày nay mã ASCII được sử dụng trên tất cả các máy tính cá nhân.

a) Mã 1 byte

Hệ thống mã ASCII chuẩn sử dụng 7 bit để mã hóa cho một ký tự, do đó có tổng cộng $2^7 = 128$ mã ASCII. Bảng 2.7 trình bày các mã ASCII và các ký tự tương ứng với chúng.

Chú ý rằng, chỉ có 95 ký tự ứng với 95 mã ASCII từ 32 đến 126 là có khả năng hiển thị (in được); các mã từ 0 đến 31 và mã 127 được dùng đến với các mục đích điều khiển quá trình truyền thông, do đó không có khả năng in được. Hầu hết các máy vi tính chỉ sử dụng các ký tự in được và một số ký tự điều khiển như CR (về đầu dòng), LF (xuống dòng), HT (Tab), BS (xoá lùi), BEL (đưa tiếng bip ra loa).

Vì mỗi ký tự ASCII được mã hóa bằng 7 bit, nên mã của một ký tự chứa vừa vặn trong 1 byte với MSB = 0. Các ký tự in được có thể hiển thị ra màn hình hoặc ra máy in, trong khi đó các ký tự điều khiển lại dùng để điều khiển các thiết bị này. Ví dụ, để hiển thị ký tự A trên màn hình, chương trình sẽ gửi mã ASCII 41h đến màn hình; còn để lùi con trỏ trở lại đầu dòng, chương trình sẽ gửi mã ASCII có giá trị 13 (0Dh – mã ASCII của ký tự điều khiển CR) đến màn hình.

Ngoài ra, máy tính cũng hiển thị một số ký tự đặc biệt ứng với một số mã ASCII không in được. Như ta thấy bộ điều khiển màn hình của IBM-PC có thể hiển thị một bộ ký tự mở rộng 256 ký tự (bảng 2.7).

Ở Việt Nam cũng đã đưa ra và sử dụng một số bộ mã 8 bit (1 byte) để mã hóa các ký tự có dấu giúp cho việc soạn thảo tiếng Việt như bộ mã TCVN3-ABC, VNI Win,...

Bảng 2.7. Bảng mã ASCII với 128 ký tự đầu (Bảng mã ASCII chuẩn)

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

Bảng 2.8. Bảng mã ASCII với 128 ký tự nữa sau (Bảng mã ASCII mở rộng)

Hexa-decimal	8	9	A	B	C	D	E	F
0	à	ã	á				α	≡
	128	144	160	176	192	208	224	112
1	ÿ	ô	ó				õ	±
	129	145	161	177	193	209	225	241
2	É	ă	ú				γ	≥
	130	146	162	178	194	210	226	242
3	Õ	Ô	ú				π	≤
	131	147	163	179	195	211	227	243
4	ø	ù	ë				Σ	∫
	132	148	164	180	196	212	228	244
5	À	Ù	ë				σ	J
	133	149	165	181	197	213	229	245
6	ò	Û	ê				μ	÷
	134	150	166	182	198	214	230	246
7	ő	߱	ܺ				τ	≈
	135	151	167	183	199	215	231	247
8	ò	ܺ	ܻ				Φ	•
	136	152	168	184	200	216	232	248
9	ò	ê					θ	.
	137	153	169	185	201	217	233	249
A	ó	ܵ	ܶ				Ω	-
	138	154	170	186	202	218	234	250
B	Ù	ܴ	ܵ				δ	√
	139	155	171	187	203	219	235	251
C	ø	ܶ	ܷ				∞	^
	140	156	172	188	204	220	236	252
D	õ	ܶ	ܸ				ϕ	²
	141	157	173	189	205	221	237	253
E	Ä		ܶ				€	■
	142	158	174	190	206	222	238	254
F	Ã	f	ܷ				∞	
	143	159	175	191	207	223	239	255

b) Mã 2 byte

Unicode là viết tắt của từ Universal Code, bộ mã tiêu chuẩn quốc tế đa ngôn ngữ (tiêu chuẩn mã hoá ký tự 16 bit do Unicode Consortium phát triển trong thời gian từ năm 1988 đến năm 1991). Do dùng 2 byte để hiện từng ký tự, Unicode cho phép thể hiện hầu hết mọi ngôn ngữ viết trên thế giới (trong đó có tiếng Việt) bằng một tập ký tự đơn nhất.

Song song với tổ chức Unicode Consortium còn có tổ chức ISO (Tổ chức chuẩn quốc tế chính thống) cũng nghiên cứu một bộ mã đa ngôn ngữ dùng trong công nghệ thông tin là ISO/IEC 10646. Unicode và ISO từ năm 1993 đã thống nhất cùng nhau phát triển và đồng nhất hai bộ mã ở miền 16 bit. Unicode là bộ mã 16 bit (có 65.536 ô mã).

2.1.5. Cấu trúc biểu diễn dữ liệu

Trong thực tế có nhiều kiểu dữ liệu cần biểu diễn trong máy tính, mỗi kiểu dữ liệu được lưu trữ theo một phương pháp biểu diễn khác nhau với cấu trúc khác nhau. Thường có hai loại dữ liệu là dữ liệu cơ bản và dữ liệu do người sử dụng định nghĩa. Như vậy, cấu trúc biểu diễn dữ liệu rất đa dạng và hệ thống kiến trúc máy tính phải đáp ứng được điều này. Chính vì vậy, ngày nay các hệ thống máy tính có thể biểu diễn, xử lý mọi bài toán trong cuộc sống: giải quyết tính toán toán học, xử lý đồ họa tĩnh/dộng, xử lý âm thanh, quản lý, mô phỏng....

Ví dụ:

– Biểu diễn số thực trong máy tính phải biểu diễn thông qua số thực có dấu chấm động và máy tính lưu trữ giá trị của R theo cấu trúc bao gồm: giá trị M (chi lấy phần lẻ nhị phân), giá trị của S và số mũ E, còn cơ số 2 là mặc định (trong máy tính dữ liệu luôn luôn được biểu diễn ở dạng cơ số 2).

– Trong một đơn vị cần quản lý một người thì phải lưu giữ được các thông tin về con người đó. Thông tin về bản thân mỗi con người cần được lưu trữ trong một vùng nhớ và vùng nhớ chứa nội dung của một dữ liệu mang đặc thù riêng thường được gọi là *cấu trúc dữ liệu kiểu bản ghi*. Tức là, dữ liệu kiểu bản ghi này có cấu trúc và bao gồm nhiều kiểu dữ liệu khác nhau, mỗi kiểu dữ liệu con trong nó không nhất thiết có độ dài bằng nhau và mỗi kiểu dữ liệu con trong nó có thể có cấu trúc lưu trữ khác nhau.

2.2. KIỂU DỮ LIỆU VÀ ĐỘ CHÍNH XÁC DỮ LIỆU

2.2.1. Kiểu dữ liệu

Trong máy tính, dữ liệu được biểu diễn thông qua số nhị phân (mức điện áp cao/thấp). CPU không thể tự phân biệt được một giá trị nhị phân là kiểu dữ liệu gì (ký tự, hay số nguyên có dấu, hay số nguyên không dấu hay số thực,...) để thực hiện áp dụng phép tính phù hợp. Khi thực hiện chương trình, để chương trình dịch chỉ ra đúng kiểu dữ liệu và lệnh áp dụng phép tính phù hợp thì trong chương trình người lập trình phải khai báo kiểu dữ liệu.

Ví dụ: Giả sử trong máy tính, một biến X = CCh, Y = 05h, thi X có thể là một số nguyên không dấu 8 bit (quy đổi ra hệ 10 bằng +204). Khi thực hiện phép nhân CPU sẽ được yêu cầu thực hiện lệnh Mul (với CPU 8086); và X có thể là một số nguyên có dấu 8 bit (quy đổi ra hệ 10 bằng -52), khi thực hiện phép nhân CPU sẽ được yêu cầu thực hiện lệnh IMul (với CPU 8086). Vậy, để chương trình dịch ánh xạ được giá trị đích thực của dữ liệu thì ta phải khai báo kiểu dữ liệu cho biến X.

2.2.2. Độ chính xác kiểu dữ liệu cơ bản

Phần này tài liệu đề cập đến độ chính xác của kiểu dữ liệu số thực. Trong máy tính số thực cũng được lưu trữ ở dạng một số nhị phân theo một cấu trúc cho số thực dấu chấm động đã được chuẩn hoá theo công thức (2.1).

$$R = (-1)^S \times 1.M \times 2^{E - \text{Bias}}$$

Có hai kiểu dữ liệu số thực cơ bản thường dùng là kiểu *Short real* và kiểu *Long real*.

➤ *Kiểu Short real*

Kiểu này dùng 4 byte để lưu trữ, gồm 1 bit dấu, 8 bit phần mũ, 23 bit phần định trị (không chứa phần nguyên của phần định trị), Bias = 127.

Kiểu Short real được lưu trữ trong máy tính theo cấu trúc sau (hình 2.2):

31	30	23 22		0
S		E	M	

Hình 2.2. Biểu diễn số dấu chấm (phẩy) động 32 bit trong máy tính

Hình 2.2 mô tả sự sắp xếp của một số dấu chấm động 32 bit, trong đó:

– S là một bit dấu.

– E là Exponent (phần mũ) luôn lớn hơn hoặc bằng 0, chiếm 8 bit.

– M là Mantissa (phần định trị) – chi lưu phần lẻ sau dấu chấm nhị phân chiếm 23 bit.

Độ chính xác dữ liệu là 2^{-127} vì $E \geq 0$, nên $E - Bias \geq -127$, do vậy giá trị dữ liệu biểu diễn nhỏ nhất với kiểu short real là 2^{-127} .

Ví dụ:

+ Biểu diễn số 4.9 ở dạng dấu chấm động 32 bit (short real):

Áp dụng cách biểu diễn số dấu chấm động ở hình 2.2, ta có

$$4.9 = 100.1110011001100\dots b.$$

Sau khi chuẩn hóa ta có

$$100.1110011001100\dots b = (-1)^0 \times 1.00111001100110011001100b \times 2^2.$$

Cộng thêm Bias vào phần mũ E, ta nhận được

$$E = 127 + 2 = 129 = 10000001b.$$

Do phần nguyên trong phần định trị không được lưu, nên ta được phần định trị M đầy đủ 23 bit sau dấu chấm nhị phân là

$$M = 001\underline{1100110011001100}1100b$$

(không được bỏ các chữ số 0 không có nghĩa khi tách riêng).

Vậy số biểu diễn trên máy cho giá trị 4.9 là

$$0\underline{10000001}\underline{00111001100110011001100}b$$

hay $0100000010011\underline{1001100110011001100}b = 409CCCCCh.$

+ Biểu diễn số -0.75 ở dạng dấu chấm động 32 bit (short real):

Ta có biểu diễn nhị phân của $0.75 = 0.11b$. Do đó $-0.75 = -0.11b$ và
 $-0.11b = (-1)^1 \times 1.100000000000000000000000b \times 2^{-1}$.

Phần định trị M được lưu sẽ là $100000000000000000000000b$ (đủ 23 bit).

Phần mũ: $E - Bias = -1$, ta được phần mũ E = $126 = 01111110b$.

Phần nguyên không được lưu, nên ta có dạng short real của -0.75 là

$$1\underline{01111110}\underline{100000000000000000000000}b$$

hay $1011\underline{111101000000000000000000}b = BF400000h.$

Như vậy ta thấy độ chính xác dữ liệu của kiểu dữ liệu Short real là 2^{-127} .

➤ Kiểu Long real

Kiểu này dùng 8 byte để lưu trữ, gồm 1 bit dấu, 11 bit phần mũ, 52 bit phần định trị (không chứa phần nguyên của phần định trị), Bias = 1023.

Kiểu Long real được lưu trữ trong máy tính theo cấu trúc sau (hình 2.3):

63	62	52	51	0
S	E		M	

Hình 2.3. Biểu diễn số dấu chấm (phẩy) động 64 bit trong máy tính

Hình 2.3 mô tả sự sắp xếp của một số dấu chấm động 64 bit, trong đó:

- S là một bit dấu.
- E là Exponent (phần mũ) đã cộng thêm số Bias = 1023, chiếm 11 bit.
- M là Mantissa (phần định trị) – chỉ lưu phần lẻ sau dấu chấm nhị phân, chiếm 52 bit.

Độ chính xác dữ liệu là 2^{-1023} .

2.2.3. Dự phòng cho dữ liệu có độ chính xác thay đổi

Để dự phòng cho biểu diễn dữ liệu có độ chính xác cao hơn, người ta đưa ra số thực *Temporary real* được lưu trữ trong cấu trúc 80 bit.

Temporary real dùng 10 byte để lưu trữ, 1 bit dấu, 15 bit phần mũ, 64 bit phần định trị (không chứa phần nguyên của phần định trị), Bias = 16383.

Kiểu Temporary real được lưu trữ trong máy tính theo cấu trúc sau (hình 2.4):

79	78	64	63	0
S	E		M	

Hình 2.4. Biểu diễn số dấu chấm (phẩy) động 80 bit trong máy tính

Hình 2.4 mô tả sự sắp xếp của một dấu chấm động 80 bit, trong đó:

- S là một bit dấu.
- E là Exponent (phần mũ) đã cộng thêm số Bias = 16383, chiếm 15 bit.
- M là Mantissa (phần định trị) – chỉ lưu phần lẻ sau dấu chấm nhị phân, chiếm 64 bit.

Độ chính xác dữ liệu là 2^{-16383} .

2.3. TẬP CÁC THANH GHI

2.3.1. Khái niệm thanh ghi

Trong kiến trúc máy tính, CPU chứa các thanh ghi, mỗi thanh ghi là một nơi lưu trữ dữ liệu với dung lượng nhỏ, có chức năng riêng, mà nội dung của nó được truy nhập nhanh hơn so với các nơi lưu trữ khác có sẵn trong máy tính. Mỗi thanh ghi được thiết kế cho mục đích riêng này không là một phần của vùng nhớ thông thường trong máy tính. Đa số, nhưng không phải tất cả, các máy tính hiện đại thuộc kiến trúc load – store. Thông thường, dữ liệu được nạp từ vùng nhớ dung lượng lớn hơn (nó có thể là cache hoặc RAM) vào trong các thanh ghi, dữ liệu được xử lý hoặc kiểm tra theo một cách nào đó (sử dụng các lệnh của máy: các lệnh số học, logic hoặc so sánh) và sau đó được lưu trữ trở lại vào bộ nhớ (có thể vào một vị trí khác trong bộ nhớ). Một thuộc tính chung của các chương trình máy tính là định vị tham chiếu: Các giá trị như nhau thường được truy nhập lặp đi lặp lại và việc lưu giữ các giá trị được sử dụng thường xuyên này trong các thanh ghi làm tăng hiệu năng của chương trình.

Các thanh ghi trong CPU là loại bộ nhớ tốt nhất trong hệ thống nhớ và là cách thức giúp CPU truy nhập dữ liệu nhanh nhất. Toán hạng được sử dụng thường nằm trong một thanh ghi và được chỉ ra trong một phần của câu lệnh đã được giải mã. Câu lệnh đã được giải mã thuộc tập lệnh của CPU. Nội dung của các thanh ghi thường được đưa vào từ bộ nhớ và có thể nói việc sử dụng các thanh ghi làm cho hiệu năng thực thi chương trình là cao nhất. Việc làm này được thực hiện bởi một chương trình dịch trong giai đoạn biên dịch chương trình (giai đoạn sinh mã lệnh).

Tóm lại: Thanh ghi thực chất là bộ nhớ bán dẫn có tốc độ truy nhập cực cao (mức CPU) và có dung lượng nhỏ. Tập các thanh ghi nằm trong CPU. Mỗi thanh ghi là một đơn vị lưu trữ dữ liệu trung gian/tạm thời, có chức năng riêng biệt không thể thiếu trong CPU.

2.3.2. Các loại thanh ghi

Dung lượng của các thanh ghi thường được xác định bởi số bit mà chúng có, ví dụ thanh ghi 8 bit hay thanh ghi 32 bit. Ngày nay, các thanh ghi trong CPU được coi như là tập các thanh ghi, chúng được xây dựng từ các mạch *flip-flops* với các phương pháp khác nhau tùy theo các loại máy tính khác nhau.

Một CPU thường chứa vài loại thanh ghi, chúng có thể được phân loại tùy theo chức năng của chúng hoặc theo lệnh thao tác lên chúng.

Trong phần này ta đề cập các loại thanh ghi theo chức năng của chúng. Ngày nay, trong kiến trúc máy tính thường có các loại thanh ghi sau:

– IR (Instruction Register) – thanh ghi lệnh: Dùng để chứa lệnh CPU đang thực hiện.

– PC (Program Counter) – bộ đếm chương trình: PC còn được gọi là con trỏ lệnh (IP – Instruction Pointer) dùng để chứa địa chỉ của lệnh tiếp theo sẽ được thực hiện ngay sau lệnh đang thực hiện. Nó trỏ đến ô nhớ chứa lệnh tiếp theo mà CPU cần truy nhập. Giá trị của PC tự động tăng tuần tự khi nhận lệnh, tạo ra địa chỉ của ô nhớ chứa lệnh tiếp theo cần nạp. Khi CPU thực hiện các lệnh rẽ nhánh (lệnh nhảy) thì giá trị của PC thay đổi đột biến (có thể tăng, có thể giảm). Trong trường hợp của máy tính đơn giản thì PC là bộ đếm chương trình loại 4 bit, có khả năng quản lý và truy nhập bộ nhớ có 16 ngăn nhớ.

– MAR (Memory Address Register) – thanh ghi địa chỉ bộ nhớ: Dùng để chứa địa chỉ ngăn nhớ tiếp theo cho CPU đọc/ghi.

– MBR (Memory Buffer Register) – thanh ghi đệm bộ nhớ: Dùng để chứa dữ liệu chuẩn bị ghi tới bộ nhớ hoặc nhận dữ liệu đọc từ bộ nhớ.

– IOAR (Input Output Address Register) – thanh ghi địa chỉ vào/ra: Dùng để xác định một thiết bị vào/ra cần trao đổi dữ liệu.

– IOBR (Input Output Buffer Register) – thanh ghi đệm vào/ra: Dùng để trao đổi dữ liệu giữa một môđun vào/ra và CPU.

– Acc (Accumulator) – thanh ghi chung: Là thanh ghi dùng để chứa nội dung một toán hạng của thao tác hoặc kết quả của một thao tác.

– TMP (Temporary) – thanh ghi tạm: Thường dùng để chứa nội dung toán hạng thứ hai trong các thao tác.

– FR (Flag Register) – thanh ghi cờ: Dùng để chứa các cờ, mỗi cờ là một bit và phản ánh thông tin về trạng thái làm việc của CPU hoặc trạng thái của kết quả sau khi thực hiện lệnh. Thanh ghi cờ tạo ra mối quan hệ logic giữa các lệnh được thực hiện trong chương trình.

Ví dụ: Trong CPU 8086 có các cờ sau phản ánh kết quả sau khi thực hiện lệnh:

+ Cờ ZF (Zero Flag) – cờ zero: Nếu kết quả phép tính bằng 0 thì ZF = 1.

- + Cờ SF (Sign Flag) – cờ dấu: Nếu kết quả phép tính là một số âm thì SF = 1.
 - + Cờ CF (Carry Flag) – cờ nhớ: Nếu sau khi thực hiện phép tính có nhớ CF = 1.
 - + Cờ OF (Overflow Flag) – cờ tràn: Nếu kết quả là số bù 2 vượt quá giới hạn biểu diễn dành cho nó thì OF = 1.
 - + Cờ AF (Auxiliary Flag) – cờ nhớ phụ: Nếu có nhớ từ 4 bit thấp sang 4 bit cao thì AF = 1.
 - + Cờ PF (Parity Flag) – cờ chẵn lẻ: Nếu số bit trong kết quả là chẵn thì PF = 1.
- Ngoài ra ở CPU 8086 còn có 3 bit cờ phản ánh trạng thái làm việc của nó là:
- + Cờ TF (Trap Flag) – cờ bẫy: Nếu T = 1 thì CPU làm việc ở chế độ chờ từng lệnh (chế độ này dùng khi cần tìm lỗi của một chương trình).
 - + Cờ IF (Interrupt) – cờ cho phép ngắt: Nếu I = 1 thì CPU cho phép các yêu cầu ngắt (ngắt che được) được tác động.
 - + Cờ DF (Direction) – cờ hướng: Nếu D = 1 thì CPU làm việc với một chuỗi ký tự từ phải sang trái (DF còn gọi là cờ lùi).

Ý nghĩa của các cờ khá rõ ràng; riêng cờ tràn, ta cần làm rõ cách thức mà máy tính xác định giá trị của nó để phát hiện lỗi.

Giả thiết ta làm việc với số bù 2 dài 8 bit, kết quả đẻ ở AL. Gọi C_{6,7} là cờ nhớ từ bit 6 lên bit 7 (b7), trong đó b7 là MSB và cũng chính là bit dấu (SF) của AL:



Quan hệ giữa cờ OF với cờ C_{6,7} tuân theo phương trình sau:

$$OF = CF \oplus C_{6,7} \quad (\oplus \text{ là phép XOR}).$$

Nghĩa là, khi thực hiện các phép toán với số bù 2 (số có dấu), hiện tượng tràn sẽ xảy ra và thể hiện qua cờ OF = 1 nếu có nhớ từ MSB (tức là SF) sang CF và không có nhớ từ C_{6,7} vào chính nó (SF) hoặc ngược lại.

Ví dụ:

$$\begin{array}{r} + 01111111 = 127 \\ \underline{00000001} = \underline{1} \\ 10000000 = -128 \text{ (sai)} \end{array}$$

Tràn số → kết quả sai, cụ thể:

$C_{6,7} = 1$ (có nhớ từ bit 6 lên bit 7);

$CF = 0$ (không có nhớ từ MSB sang).

Do vậy: $OF = CF \oplus C_{6,7} = 1$.

$$\begin{array}{r} + 10000000 = -128 \\ \underline{10000001} = \underline{-127} \\ 10000001 = 001 \text{ (sai)} \end{array}$$

Tràn số → kết quả sai, cụ thể:

$C_{6,7} = 0$ (không có nhớ từ bit 6 lên bit 7);

$CF = 1$ (có nhớ từ MSB sang).

Do vậy: $OF = CF \oplus C_{6,7} = 1$.

2.4. CÁC KIỀU LỆNH

Một tập lệnh (instruction set), hoặc kiến trúc tập lệnh (instruction set architecture – ISA) là một phần của kiến trúc máy tính liên quan tới việc thực thi chương trình, nó bao gồm các kiểu dữ liệu; các lệnh; các thanh ghi; phương pháp định địa chỉ; kiến trúc bộ nhớ; ngắt; bẫy lỗi và việc vào/ra với bên ngoài. Một kiến trúc tập lệnh bao gồm cả việc đặc tả một tập các mã thi hành (opcode) hay còn gọi là ngôn ngữ máy và được thi hành bởi một loại vi xử lý riêng biệt. Tuỳ theo yêu cầu chức năng của từng loại máy tính cần xây dựng mà các nhà thiết kế đưa ra các tập lệnh khác nhau. Và chung nhất, một tập lệnh của máy tính có thể được chia làm nhiều nhóm lệnh với các chức năng thực hiện các thao tác khác nhau: lệnh thao tác, lệnh truy cập bộ nhớ, lệnh điều khiển, lệnh đặc quyền, lệnh vectơ.

2.4.1. Nhóm lệnh số học và logic

Nhóm lệnh số học và logic bao gồm: cộng, trừ, nhân, chia, AND, OR, XOR, NOT, dịch trái, dịch phải, quay trái, quay phải,...

Ví dụ: Với máy tính họ vi xử lý 80x86 có các lệnh cụ thể sau (toán hạng có thể là một trong các thanh ghi hoặc biến):

ADD đích, nguồn ; đích = đích + nguồn: nguồn không đổi sau câu lệnh.

MUL số_nhân ; $DXAX = AX \times số_nhân$ hoặc $AX = AL \times số_nhân$.

DIV số_chia ;

SUB đích, nguồn ; đích = đích - nguồn: nguồn không đổi sau câu lệnh.

AND đích, nguồn ; đích = đích AND nguồn (AND từng bit).

OR đích, nguồn ; đích = đích OR nguồn (OR từng bit).

XOR đích, nguồn ; đích = đích XOR nguồn (XOR từng bit).

NOT đích	; đích = đích đã đảo tất cả các bit: 0 → 1, 1 → 0.
SHL đích, 1	; dịch trái tất cả các bit của đích đi 1 bit, 0 → LSB.
SHR đích, 1	; dịch phải tất cả các bit của đích đi 1 bit, 0 → MSB.
...	

2.4.2. Nhóm lệnh truy cập dữ liệu bộ nhớ

Nhóm lệnh này bao gồm các lệnh thực hiện thao tác vào/ra dữ liệu giữa CPU và bộ nhớ.

Ví dụ với máy tính 8086 có các lệnh sau:

MOV đích, nguồn	; đích ← nguồn, copy nội dung nguồn sang đích. Đích ; có thể là một thanh ghi hay ô nhớ nào đó và nguồn thì ; ngược lại.
PUSH nguồn	; cất nội dung toán hạng nguồn vào ngăn xếp (stack) ; trong bộ nhớ, toán hạng nguồn là một thanh ghi.
POP đích	; lấy nội dung từ nhớ trong ngăn xếp đưa vào toán hạng ; đích, toán hạng đích là một thanh ghi.

2.4.3. Nhóm lệnh điều khiển

Là tập các lệnh phục vụ cho việc điều khiển rẽ nhánh chương trình.

Ví dụ, trong máy tính 8086 có các lệnh sau:

CMP đích, nguồn	; so sánh nội dung đích và nguồn, không lưu kết quả ; chỉ ảnh hưởng đến các bit cờ làm điều kiện nhảy cho ; các lệnh nhảy có điều kiện.
JMP địa_chi_đích	; lệnh nhảy không điều kiện, chuyển điều khiển đến ; thực hiện lệnh có địa chỉ là địa_chi_đích.
Lệnh_nhảy_có_điều_kiện địa_chi_đích	; lệnh nhảy có điều kiện bao gồm ; nhiều lệnh nhảy khác nhau, có chức năng chuyển điều ; khiển đến thực hiện lệnh có địa chỉ là địa_chi_đích khi ; điều kiện nhảy thỏa mãn.

Lệnh gọi chương trình con.

Chương trình con là một nhóm các lệnh thực hiện một nhiệm vụ nào đó và nhiệm vụ đó có thể được thực hiện nhiều lần trong chương trình. Do vậy, chương trình con có thể được gọi ra thực hiện từ một vài nơi trong chương trình. Khi lệnh

gọi chương trình con được nạp thì thanh ghi IP (thanh ghi con trả lệnh) sẽ chứa địa chỉ của lệnh đứng ngay sau lệnh gọi chương trình con (tạm gọi là lệnh A), nhưng để thực hiện chương trình con thì IP phải được nạp địa chỉ của lệnh đứng đầu chương trình con, do vậy giá trị địa chỉ của lệnh A phải được cất vào ngăn xếp (stack) trong bộ nhớ. Khi kết thúc một lần thực hiện chương trình con, lệnh A phải được nạp, có nghĩa là phải có việc chuyển điều khiển trở về tiếp tục thực hiện chương trình gọi chương trình con, chính vì vậy địa chỉ của lệnh A này phải được nạp trở lại IP khi kết thúc thực hiện chương trình con (lấy lại địa chỉ lệnh A từ Stask đưa vào IP).

2.4.4. Lệnh đặc quyền

Mức đặc quyền (Privilege Level – PL) gán cho một chương trình cho biết chương trình có thẩm quyền làm những gì khi nó thực hiện một nhiệm vụ. Có 4 mức đặc quyền (từ 0 đến 3). Mức đặc quyền 0 (PL = 0) bao gồm các chương trình sơ cấp quản lý các tài nguyên của CPU và bộ nhớ. Các chương trình này phải nhỏ gọn, có khả năng vận hành tốt, không bị hỏng do phần mềm khác. Ví dụ các chương trình trong BIOS ROM có mức đặc quyền 0.

Lệnh đặc quyền là lệnh chỉ có thể thực hiện ở mức đặc quyền 0 (PL = 0).

Ví dụ sau đây là các lệnh đặc quyền trong CPU 80286:

- + **LGDT:** Nạp thanh ghi GDTR (Global Descriptor Table Register – Thanh ghi bảng bộ mô tả toàn cục).
- + **LLDT:** Nạp thanh ghi LDTR (Local Descriptor Table Register – Thanh ghi bảng bộ mô tả cục bộ).
- + **LIDT:** Nạp thanh ghi IDTR (Interrupt Descriptor Table Register – Thanh ghi bảng bộ mô tả ngắn).
- + **LTR:** Nạp phần chọn của TR (Task Register – Thanh ghi nhiệm vụ).
- + **LMSW:** Nạp từ trạng thái máy MSW (Machine Status Word).
- + **CLTS:** Xoá bit TS (Task Switch) trong thanh ghi MSW.
- + **HALT:** Dừng hoạt động của CPU.

2.4.5. Lệnh vectơ

Mỗi giá trị có thứ tự cần được xử lý trong bộ nhớ gọi là xử lý vectơ. Tổng số các giá trị cần được xử lý gọi là chiều dài vectơ. Các giá trị đó được xử lý bằng một mã lệnh duy nhất. Thường là các lệnh xử lý xâu ký tự (hoặc mảng), hướng xử

ý tự (từ trái sang phải hay từ phải sang trái) phụ thuộc vào cờ hướng DF (Direction Flag).

Ví dụ: Trong 8086, khi cờ hướng DF = 0, CPU xử lý xâu ký tự từ trái sang (từ địa chỉ thấp đến địa chỉ cao). Sau mỗi lần xử lý xâu, nội dung thanh ghi số tăng lên để trả đến ký tự tiếp theo (tăng 1 nếu ký tự là mã 1 byte, tăng lên 2 nếu ký tự là mã 2 byte). Ví dụ sau minh họa quá trình xử lý vectơ trong 8086:

; S1 = 'Ha Noi' (kiểu byte) và S2 = 'Ha Nam' (kiểu byte).

CLD ; DF = 0, xử lý dữ liệu từ địa chỉ thấp → cao.

Mov CX, 6 ; số lần lặp trong CX.

Lea SI, S1 ; S1 là chuỗi nguồn, được trả bởi thanh ghi SI.

Lea DI, S2 ; S2 là chuỗi đích, được trả bởi thanh ghi DI.

lap: MOVSB ; copy 1 phần tử trong S1 sang S2, sau mỗi lần lặp SI và
; DI đều tự động tăng 1 để trả đến phần tử tiếp theo.

Loop lap

ra: ; nội dung xâu S2 = 'Ha Noi'.

2.5. CÁC PHƯƠNG THỨC ĐỊNH ĐỊA CHỈ

Trong việc xác định các định dạng của lệnh, những yếu tố sau cần phải được cân nhắc:

- Số lệnh đã được xác định trước cho tập lệnh.
- Khả năng địa chỉ và phương thức xác định địa chỉ.
- Sự thuận lợi cho giải mã lệnh.
- Kiểu của trường bit trong lệnh (cố định hay thay đổi).
- Giá phải trả cho yêu cầu phân cứng để giải mã lệnh và thi hành lệnh.

Một lệnh khi đã được giải mã ở dạng mã máy thường có định dạng sau:

OP-CODE	ADDRESS(ES)
---------	-------------

Trong đó:

- OP-CODE là phần mã thi hành (ngôn ngữ máy);
- Phần ADDRESS(ES) có thể có 0, 1, 2, hoặc 3 trường địa chỉ để xác định địa chỉ của các toán hạng có trong lệnh.

Như vậy, một tập lệnh trong máy tính gồm các lệnh có thể có định dạng khác nhau, song phần Op-code là không thể thiếu, còn phần Address có thể có hoặc không tùy theo số toán hạng xuất hiện trong câu lệnh. Và số toán hạng xuất hiện trong câu lệnh có thể là 0, 1, 2, 3 tùy theo câu lệnh cũng như tùy theo kiến trúc lập lệnh.

Ví dụ trong máy tính 8086, mã gọi nhô cho tập lệnh có 3 dạng định dạng lệnh như sau:

1. Mã_lệnh ; câu lệnh không có toán hạng.
lệnh **NOP** ; CPU nghỉ 1 chu kỳ máy, trong định ; dạng lệnh ở mã máy không có trường ; address.
 2. Mã_lệnh toán_hạng_đích ; câu lệnh có 1 toán hạng.
lệnh **INC AX** ; tăng nội dung thanh chứa lên 1, trong ; định dạng lệnh ở mã máy có 1 trường ; address.
 3. Mã_lệnh toán_hạng_đích, toán_hạng_nguồn
lệnh **ADD AX, 5** ; $AX = AX + 5$, ; trong định dạng lệnh ở mã máy có 2 ; trường address.

Và trong máy tính Intel RISC 860, có lệnh với định dạng:

Mã lệnh toán hàng đích, toán hàng nguồn, kết quả.

OR 25, R0, R8 ; OR 25 với R0 và lưu kết quả vào R8,
; trong định dạng lệnh ở mã máy có 3 trường address.

Đối với các lệnh ở định dạng lệnh có trường address (có toán_hạng_dịch hoặc toán_hạng_nguồn) thì địa chỉ toán hạng phải được chỉ ra trong câu lệnh theo một phương thức nào đó, thường gọi là chế độ địa chỉ. Ta sẽ xem xét đến vấn đề chế độ địa chỉ ở phần sau.

Các chế độ địa chỉ cho phép xác định địa chỉ của toán hạng trong thao tác. Các phương pháp xác định địa chỉ toán hạng (các phương pháp định vị toán hạng) cho phép xác định nơi chứa toán hạng. Nơi chứa toán hạng có thể là ngay trên lệnh, hoặc trên thanh ghi, hoặc trong bộ nhớ. Khi truy cập toán hạng bộ nhớ, mặc

Định được hiểu là truy cập vào một ô nhớ nào đó trong bộ nhớ, để truy cập được nhớ thì địa chỉ ô nhớ phải được chỉ ra.

Có 3 nhóm định vị toán hạng:

- Định vị tức thời;
- Định vị thanh ghi;
- Định vị bộ nhớ.

Các mục nhớ tiếp theo sau trình bày các phương thức định địa chỉ sẽ có sử dụng các ví dụ là mã lệnh gọi nhớ cho tập lệnh của CPU 8086.

2.5.1. Định vị tức thời

Trong trường hợp này, toán hạng nằm ngay trong lệnh, không có yêu cầu trả nhập bộ nhớ. Nói cách khác, nội dung toán hạng nằm ngay sau phần Op-code.

Ví dụ:

MOV AX, 70 ; đưa giá trị 70 vào thanh ghi AX,
; 70 là giá trị toán hạng nằm ngay trong câu lệnh.

2.5.2. Định vị thanh ghi

Mỗi một thanh ghi trong CPU ngầm định được xác định địa chỉ qua tên thanh ghi. Trong câu lệnh các thanh ghi là nơi chứa dữ liệu. Trong ví dụ ở mục 2.5.1 toán hạng đích AX là thuộc nhóm định vị theo địa chỉ thanh ghi.

Cũng có thanh ghi được xác định ngầm định bao hàm bởi lệnh – không có mặt trong câu lệnh (địa chỉ của toán hạng thứ nhất được ngầm định là thanh chứa).

Ví dụ:

Lệnh **Mul BL** ; lấy BL nhân với AL, AL là thanh ghi được,
; bao hàm bởi lệnh MUL (AL không xuất hiện trong lệnh).

2.5.3. Định vị bộ nhớ

Phương pháp định vị bộ nhớ cho phép xác định địa chỉ của ô nhớ chứa nội dung toán hạng. Có các phương pháp định vị bộ nhớ như sau:

a) Định vị trực tiếp

Địa chỉ ô nhớ chứa dữ liệu nằm ngay trong lệnh.

Địa chỉ = [giá trị cụ thể].

Ví dụ:

MOV AX, [100h] ; copy nội dung của từ nhớ có địa chỉ DS:100h vào AX.

MOV BX, w ; w là tên biến đã được khai báo – tên biến là địa chỉ.

b) Định vị gian tiếp

Phương pháp định địa chỉ này sử dụng một thanh ghi hay một vùng nhớ trong bộ nhớ chính để chứa địa chỉ của toán hạng trong lệnh. Phương pháp này có sử dụng nhiều hơn một mức trung gian để định vị toán hạng.

Ví dụ trong CPU 8086, các thanh ghi thường dùng làm nơi chứa địa chỉ toán hạng, đó là: BX, BP, SI, DI, SP. Ví dụ lệnh sau:

MOV BX, 100h

MOV AX, [BX] ; copy nội dung của từ nhớ có địa chỉ 100h vào AX,
; trong trường hợp này BX chứa địa chỉ của toán hạng,
; nó đóng vai trò là con trỏ trỏ đến từ nhớ cần truy nhập.

c) Định vị cơ sở

Địa chỉ của toán hạng bằng tổng của nội dung một trong các thanh ghi cơ sở và giá trị gọi là độ dịch. Khuôn dạng toán hạng thường biểu diễn như sau:

Độ_dịch[tên_thanh_ghi_cơ_sở]

Trong đó độ dịch có thể là:

- Địa chỉ của một biến;
- Một hằng số (âm hoặc dương);
- Địa chỉ của một biến \pm một hằng số.

Chế độ địa chỉ này hữu dụng cho xử lý dữ liệu với mảng một chiều hoặc cấu trúc dữ liệu tương đương.

Ví dụ:

LEA BX, a ; BX – Base: thanh ghi cơ sở, chứa địa chỉ đầu của mảng a.

MOV AX, 10[BX] ; copy nội dung phần tử thứ 6 trong mảng a kiểu word
; vào AX (mỗi phần tử cách nhau 2 byte).

d) Định vị chỉ số

Địa chỉ của toán hạng bằng tổng của nội dung một trong các thanh ghi chỉ số (ví dụ, SI hoặc DI) và giá trị gọi là độ dịch. Khuôn dạng toán hạng thường biểu diễn như sau:

Độ_dịch[tên_thanh_ghi_chi_số]

Trong đó độ dịch có thể là:

- Địa chỉ của một biến;
- Một hằng số (âm hoặc dương);
- Địa chỉ của một biến \pm một hằng số.

Chế độ địa chỉ này cũng hữu dụng cho xử lý dữ liệu với mảng một chiều hoặc cấu trúc dữ liệu tương đương.

Ví dụ:

MOV SI, 10 : SI – Source Index: thanh ghi chỉ số nguồn.

MOV AX, a[SI] : copy nội dung phần tử thứ 6 trong mảng a kiểu word vào AX.

e) **Định vị cơ sở chỉ số (hoặc chỉ số cơ sở)**

Trong chế độ này, địa chỉ của một toán hạng là tổng của:

1. Nội dung của thanh ghi cơ sở;
2. Nội dung của thanh ghi chỉ số;
3. (Có thể) Địa chỉ của một biến kiểu byte;
4. (Có thể) Một hằng số (âm hoặc dương).

Khuôn dạng của toán hạng: Toán hạng có thể được viết ở nhiều dạng khác nhau, sau đây là một số dạng thường dùng:

1. *Biến_nhỏ [thanh_ghi_cơ_sở] [thanh_ghi_chi_số]*
2. *[thanh_ghi_cơ_sở + thanh_ghi_chi_số + biến_nhỏ + hằng_số]*
3. *Biến_nhỏ [thanh_ghi_cơ_sở + thanh_ghi_chi_số + hằng_số]*
4. *Hằng_số [thanh_ghi_cơ_sở + thanh_ghi_chi_số + biến_nhỏ]*

Chú ý: Thứ tự các phần tử trong dấu ngoặc là tùy ý.

Ví dụ: W là một biến kiểu byte, BX chứa giá trị 2, SI chứa giá trị 5.

Khi đó lệnh: **MOV AL, W[BX][SI]** sẽ chuyển nội dung của ô nhớ có địa chỉ DS: (offset(W) + 2 + 5) hay DS: (offset(W) + 7) vào AL.

Lệnh trên có thể được viết dưới dạng sau:

MOV AX, [W + BX + SI] hoặc **MOV AX, W[BX + SI]**

Cách độ địa chỉ này hữu ích khi thao tác với mảng dữ liệu hai chiều hoặc cấu trúc dữ liệu phức tạp tương đương.

2.6. CÁC VẤN ĐỀ VỀ THIẾT KẾ ĐỊA CHỈ

Nguyên tắc chung trong thiết kế địa chỉ là phải tạo ra một không gian địa chỉ với các giá trị địa chỉ tăng tuyến tính. Điều này có nghĩa là, giá trị địa chỉ phải tăng đều, không ngắt quãng, giá trị địa chỉ của mỗi đối tượng phải là duy nhất và không được phép trùng nhau (ví dụ mỗi ô nhớ chỉ có một giá trị địa chỉ duy nhất và không được phép có từ 2 ô nhớ có cùng giá trị địa chỉ). Để đảm bảo yêu cầu này thì trong máy tính phải có bộ giải mã địa chỉ.

Nguyên tắc thứ hai là phải đáp ứng được khả năng công nghệ đảm bảo cho chế tạo vi mạch, bảng mạch: phải đảm bảo được số đường dây đưa vào trên bảng mạch hoặc vi mạch là không quá lớn, nếu quá lớn thì sẽ gặp khó khăn về mặt công nghệ, tăng giá thành đầu tư thiết bị trong sản xuất dẫn đến giá thành các module trong máy tính cao hoặc không có khả năng sản xuất. Để đáp ứng được yêu cầu này, chúng ta xem xét một số vấn đề sau.

Khi thiết kế địa chỉ bộ nhớ hay địa chỉ cổng vào/ra, độ lớn của không gian địa chỉ phụ thuộc vào độ lớn của chương trình và dữ liệu thực thi trên máy tính và số lượng thiết bị vào/ra tối đa cần ghép nối với máy tính, tức là phụ thuộc vào nhiệm vụ mà máy tính cần giải quyết, hay nói cách khác là phụ thuộc vào ý định thiết kế máy dụng máy tính nhằm giải quyết các công việc gì và bài toán với độ lớn như thế nào.

Trong máy tính có hai đối tượng mà CPU cần xác định địa chỉ để trao đổi dữ liệu, đó là bộ nhớ và thiết bị ngoại vi, vấn đề đặt ra là làm thế nào để xác định đúng đối tượng cần trao đổi dữ liệu. Thường có hai phương pháp đánh địa chỉ cho đối tượng:

Phương pháp thứ nhất: Thiết bị ngoại vi được đánh chung không gian địa chỉ với bộ nhớ. Trong phương pháp này với n bit địa chỉ thì không gian địa chỉ mà CPU có thể xác định là 2^n giá trị địa chỉ. Nếu dùng k giá trị địa chỉ dành cho thiết bị ngoại vi thì không gian địa chỉ dành cho bộ nhớ chỉ còn $2^n - k$ giá trị. Như vậy, mỗi cổng vào/ra có thể được coi như là một ô nhớ (với cổng 1 byte) hay một từ nhớ (với cổng 2 byte). Trong trường hợp này sử dụng chung thiết bị giải mã địa chỉ cho bộ nhớ và thiết bị ngoại vi. Không gian địa chỉ dành cho bộ nhớ không còn là 2^n giá trị mà đã bị thu hẹp lại.

Phương pháp thứ hai: Thiết bị ngoại vi được đánh không gian địa chỉ tách biệt với bộ nhớ. Giả sử không gian địa chỉ cần cho bộ nhớ là 2^n , tức cần n bit địa chỉ (n đường dây địa chỉ); không gian địa chỉ cần cho các công vào/ra là 2^k (cho thiết bị ngoại vi), tức cần k bit địa chỉ (k đường dây địa chỉ). Như vậy cần $n + k$ đường dây địa chỉ, dẫn đến chi phí công nghệ và giá thành sản xuất cao hơn. Để giảm bớt độ phức tạp công nghệ, giảm giá thành sản xuất, người ta sử dụng phương pháp xác định đối tượng làm việc (tại một thời điểm CPU chỉ làm việc với bộ nhớ hoặc thiết bị ngoại vi, không làm việc cả hai), với phương pháp này, chỉ cần tín hiệu $\overline{IO/M}$ xác định khi nào thì CPU làm việc với bộ nhớ (giả sử $\overline{IO/M} = 1$), khi nào thì CPU làm việc với thiết bị ngoại vi (giả sử $\overline{IO/M} = 0$) và chỉ cần tổng số đường dây địa chỉ là n đường dây để đánh địa chỉ cho cả không gian địa chỉ bộ nhớ và không gian địa chỉ các công vào/ra. Như vậy, tổng số đường dây tín hiệu liên quan đến địa chỉ chỉ còn $n + 1$ đường, do đó đã giảm bớt được $k - 1$ đường dây, giảm bớt được độ phức tạp công nghệ do phải nén tích hợp nhiều đường dây trong vi mạch nhỏ.

Một vấn đề nữa được đặt ra là: Nếu sử dụng m bit dữ liệu và n bit địa chỉ riêng biệt, tức là thiết kế số đường dây địa chỉ tách biệt với đường dây dữ liệu thì tổng số đường dây địa chỉ và dữ liệu trong vi mạch là $m + n$ đường. Điều này tạo ra độ phức tạp về mặt công nghệ lên nhiều so với trường hợp chỉ cần tổng số là đường dây khi $n > m$ (hoặc m đường dây khi $m > n$). Trong thực tế, các hệ thống đã thiết kế theo phương pháp sử dụng chung đường dây vận chuyển tín hiệu địa chỉ và dữ liệu. Trong trường hợp này, phương pháp chốt địa chỉ đã được sử dụng với sự hỗ trợ của mạch chốt địa chỉ (bao gồm các mạch Flip – Flop D) và một tín hiệu chốt địa chỉ ALE (Address Latch Enable), khi $ALE = 1$ thì tín hiệu trên n (hoặc m) đường dây là tín hiệu địa chỉ; khi $ALE = 0$ thì tín hiệu trên n (hoặc m) đường dây là tín hiệu dữ liệu. Như vậy, tổng số đường dây tín hiệu khi áp dụng kỹ thuật chốt địa chỉ đã giảm được $(m - 1)$ đường dây trong trường hợp $n > m$, hoặc giảm được $(n - 1)$ đường dây trong trường hợp $m > n$.

Tóm lại, kiến trúc tập lệnh là một phần của kiến trúc máy tính liên quan tới việc thực thi chương trình, bao gồm các kiểu dữ liệu, các lệnh, các thanh ghi, phương pháp định địa,... Tuỳ theo từng loại kiến trúc và tuỳ theo chức năng máy tính, phương pháp tổ chức xây dựng các thành phần trong máy tính khác nhau mà chúng ta có các hệ thống máy tính với nền tảng phần cứng khác nhau.

CÂU HỎI VÀ BÀI TẬP CHƯƠNG 2

1. Đổi các số nhị phân và số hex sau ra số thập phân:
 - a) 1110b
 - b) 10010011110b
 - c) 46Ah
 - d) FEA7Dh.
2. Đổi các số thập phân sau ra số nhị phân và hex:

97, 629, 973, 1026.
3. Đổi các số nhị phân sau ra các số hệ 10 và hex:
 - a) 111001010101
 - b) 11100101010111001010101
 - c) 101000101101
 - d) 1011001101001101.
4. Thực hiện phép cộng sau:
 - a) 10010101b + 01110110b
 - b) 11100101010111001010101b + 10010101b
 - c) B23CDh + 9F7Eah
 - d) FEEFCh + ABCDEh.
5. Thực hiện các phép trừ sau:
 - a) 10010101b - 01110110b
 - b) 11100101010111001010101b - 10010101b
 - c) B23CDh - 9F7Eah
 - d) FEEFCh - ABCDEh.
6. Đổi các số nguyên thập phân sau ra số hex 16 bit:

234, -16, -32216, 31634, 7899, 65356.
7. Thực hiện phép trừ các số nhị phân và số hex dưới đây bằng cách cộng với số bù 2 của số trừ:
 - a) 10010101b - 01110110b
 - b) 11100101010111001010101b - 10010101b
 - c) B23CDh - 9F7Eah
 - d) FEEFCh - ABCDEh.
8. Đổi các số hệ hex sau ra số thập phân có dấu và không dấu:
 - a) 7FFEh
 - b) 8543h
 - c) FEh
 - d) 7Fh.

22. Sử dụng thuật toán chia số nhị phân để thực hiện phép chia sau:

- a) $31576 / 243$ b) $70576 / 23041$
c) $164 / 81$.

23. Biểu diễn số dấu chấm động trong máy tính theo chuẩn IEEE 32 bit các số sau:

1032.0625, -0.03125, +129.9, -129.8

24. Đổi các số thập phân sau đây thành số nhị phân và số hex:

- a) 24
 - b) 15.25
 - c) 512
 - d) 127
 - e) 2048
 - f) 65535.

25. Tìm giá trị thập phân tương đương của các số hex sau:

26. Tìm giá trị thập phân tương đương của các số nhị phân sau:

27. Biểu diễn các số thập phân sau trong hệ bù 2 (dùng 8 bit và 16 bit)

- a) ± 255 b) ± 5
 c) ± 12 d) $\pm 1.$

28. Hãy liệt kê các số liên tiếp trong hệ 8 bắt đầu từ 668 đến 2008.

29. Hãy liệt kê 20 số liên tiếp trong hệ hex từ 9h.

30. Một đường phố có 1000 ngôi nhà. Người ta muốn đánh địa chỉ cho chúng bằng số nhị phân (bắt đầu từ 0). Hỏi phải sử dụng số nhị phân bao nhiêu bit để có thể đánh địa chỉ cho 1000 ngôi nhà này? Viết ra vùng địa chỉ của 1000 ngôi nhà đó dưới dạng nhị phân và hex.

31. Cho đoạn mã sau trong bộ nhớ máy tính:

1000 0011 1000 0111

Đoạn mã có thể là:

Chi lập trình viên, người đã gõ đoạn mã này vào máy tính mới biết được m
biểu diễn cho cái gì. Hãy tìm giá trị thập phân tương đương của đoạn m
trong từng trường hợp.

32. Sau đây là tên và nội dung (dạng hex) của các thanh ghi 16 bit trong vi xử lý:

$$AX = 1234, BX = 000A, CX = FFFA, DX = 7FFF$$

Hãy tìm giá trị và dấu của chúng trong hệ 10.

33. Thực hiện các phép toán sau trong hệ nhị phân nguyên thuỷ:

a) $12 + 39$	b) $9 + 4$
c) $18 + 37$	d) $2 + 7$.

34. Thực hiện các phép toán sau trong hệ hex:

a) $12 + 34$	b) $FF + F0$
c) $DFA - FFF$	d) $123 - CD$.

35. Biểu diễn số dấu chấm động trong máy tính theo chuẩn IEEE 32 bit các số sau:

a) $+ 27.25$	b) $- 27.25$
c) $+ 12$	d) $- 12$.

Chương 3

CPU, ĐƯỜNG TRUYỀN VÀ HỆ THỐNG VÀO/RA

TÓM TẮT: – Lý thuyết: 8 tiết;
– Bài tập lớn: 6 tiết.

Mục tiêu cần đạt được	Các mục chính	Bài tập bắt buộc	Bài tập làm thêm
<p>– Cung cấp cho sinh viên kiến thức về kiến trúc cơ bản của máy tính điện tử số ngày nay và nguyên lý thực hiện chương trình, điều khiển vào/ra dữ liệu. Các mạch điện tử số cơ bản và nguyên lý thực hiện thao tác tính toán cơ bản của mạch cộng, dịch, cũng như nguyên lý chức năng của mạch điều khiển kết nối Bus với các mạch thành phần trong máy tính.</p> <p>– Kết thúc chương, sinh viên cần nắm được kiến trúc cơ bản của máy tính điện tử số, nguyên lý thực hiện chương trình, điều khiển vào/ra dữ liệu. Có khả năng phân tích, chạy một số mạch chức năng cơ bản làm cơ sở nghiên cứu thực hiện thiết kế ghép nối mạch sau này nếu cần.</p>	<p>3.1. Kiến trúc cơ bản của một máy tính điện tử số và đơn vị xử lý trung tâm (CPU – Central Processing Unit).</p> <p>3.2. Đường truyền.</p> <p>3.3. Hệ thống vào/ra</p>	<p>Câu hỏi và bài tập cuối chương: 1; 2; 11; 12; 13; 15.</p>	<p>Câu hỏi và bài tập cuối chương còn lại.</p>

3.1. KIẾN TRÚC CƠ BẢN CỦA MỘT MÁY TÍNH ĐIỆN TỬ SỐ VÀ ĐƠN VỊ XỬ LÝ TRUNG TÂM

3.1.1. Kiến trúc cơ bản của máy tính điện tử số

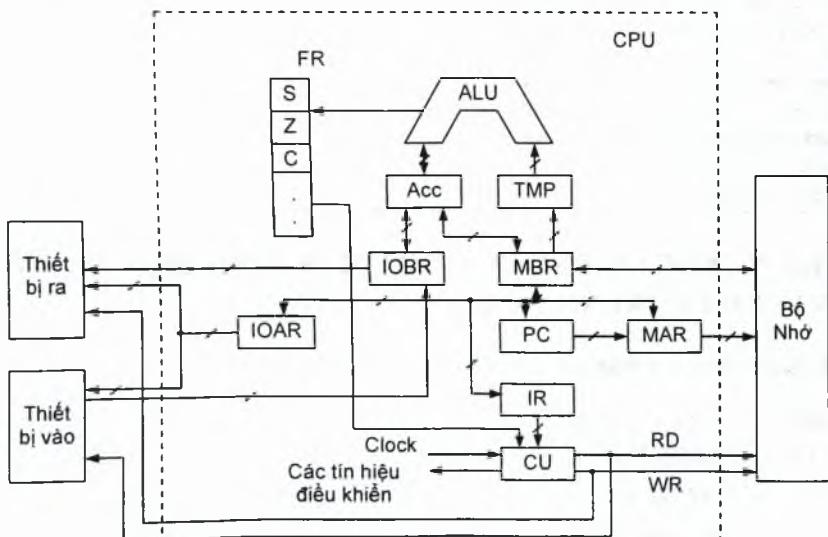
Kiến trúc cơ bản của một máy tính điện tử số được trình bày tổng quát như hình 3.1. Các đường Bus có nét gạch chéo ngang là các Bus dữ liệu và địa chỉ (thể hiện Bus có độ rộng), còn lại là các đường Bus điều khiển, bao gồm:

– Một CPU hoặc bộ xử lý (Processor) thường được gọi là bộ não của máy tính. Nó xử lý tất cả các dữ liệu đầu vào từ các thành phần khác như bộ nhớ, các

thiết bị vào/ra, cũng như đưa dữ liệu đã xử lý ra các thành phần khác trong máy tính hoặc kết nối với máy tính. Để CPU trao đổi dữ liệu với bộ nhớ, trong nó có hai thanh ghi: thanh ghi địa chỉ bộ nhớ MAR (Memory Address Register) được kết nối với các đường địa chỉ của hệ thống Bus, dùng để xác định vị trí nhớ cho việc đọc/ghi tiếp theo; và một thanh ghi đệm bộ nhớ MBR (Memory Buffer Register) được kết nối với các đường dữ liệu của hệ thống Bus, nó dùng để chứa dữ liệu chuẩn bị ghi tới bộ nhớ hoặc nhận dữ liệu đọc từ bộ nhớ. Tương tự, trong CPU có một thanh ghi địa chỉ vào/ra IOAR (Input Output Address Register) dùng để xác định một thiết bị vào/ra cần trao đổi dữ liệu, một thanh ghi đệm vào/ra IOBR (Input Output Buffer Register) dùng để trao đổi dữ liệu giữa một module vào/ra và CPU. Ngoài ra, trong CPU còn có các thanh ghi khác đã được đề cập trong mục 2.3.2.

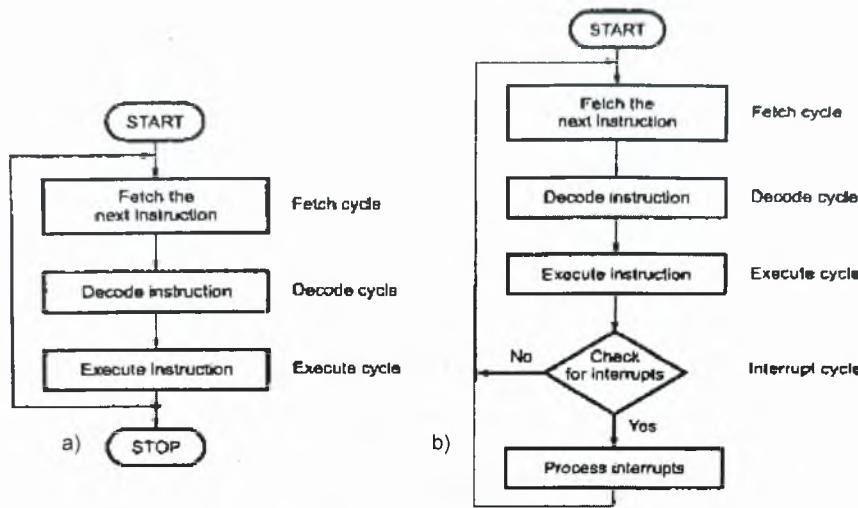
– Một module nhớ bao gồm tập các vị trí nhớ được xác định bởi các giá trị địa chỉ đã được đánh số liên tiếp. Mỗi vị trí chứa một số nhị phân mà có thể hiểu là một lệnh hoặc một giá trị dữ liệu.

– Một module vào/ra nào đó có nhiệm vụ trao đổi dữ liệu giữa một trong các thiết bị bên ngoài tới CPU và bộ nhớ, trong nó có bộ đệm để chứa tạm các dữ liệu trong quá trình trao đổi.



Hình 3.1. Kiến trúc chung của một máy tính điện tử số

Chức năng cơ bản của máy tính là thực hiện chương trình. CPU thực hiện chương trình bằng cách tuần tự thực thi các lệnh trong chương trình. CPU hoạt động theo xung nhịp đồng hồ. Với một lệnh, CPU thực hiện theo ba giai đoạn: nạp lệnh tiếp theo từ bộ nhớ (Fetch the Next Instruction); giải mã lệnh (Decode Instruction) và thực thi lệnh (Execute Instruction).



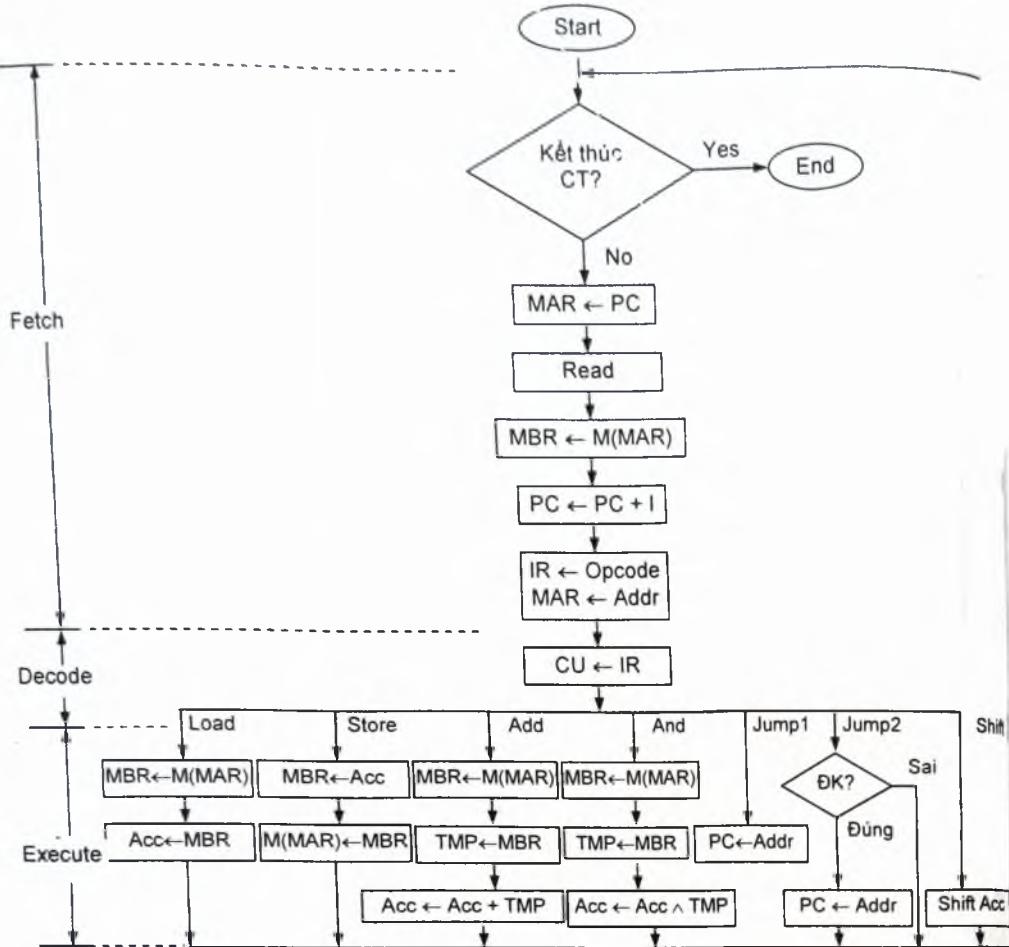
Hình 3.2

a) Chu kỳ lệnh cơ bản; b) Chu kỳ lệnh cơ bản với ngắt.

Việc thực hiện chương trình thực chất là sự lặp lại quá trình nạp lệnh, giải mã lệnh và thi hành lệnh. Thời gian tính từ khi nạp lệnh đến khi thi hành lệnh hoàn thành được gọi là *một chu kỳ lệnh*. Sau mỗi một chu kỳ lệnh, CPU kiểm tra xem có yêu cầu ngắt hợp lệ hay không (Check for Interrupt?); nếu có, CPU nhận lệnh từ thủ tục phục vụ ngắt (Process Interrupt); sau khi hoàn thành thủ tục phục vụ ngắt, CPU bắt đầu thực hiện chu kỳ lệnh mới từ nơi nó đã bị ngắt (thực hiện chu kỳ lệnh đứng ngay sau lệnh gọi ngắt). Hình 3.2 thể hiện một chu kỳ lệnh cơ bản và chu kỳ lệnh cơ bản với ngắt. Ta có thể phân làm hai loại lệnh: lệnh thao tác dữ liệu bộ nhớ và lệnh vào/ra dữ liệu trực tiếp với thiết bị vào/ra.

a) Các thao tác cơ bản khi thực hiện lệnh thao tác dữ liệu bộ nhớ

Hình 3.3 cho thấy các thao tác cơ bản được thực hiện trong một chu kỳ lệnh thao tác dữ liệu bộ nhớ. Ta sẽ xem xét các giai đoạn trong chu kỳ lệnh cơ bản và giai đoạn xử lý ngắt trong chu kỳ lệnh cơ bản với ngắt.



Hình 3.3. Các thao tác cơ bản trong một chu kỳ lệnh thao tác dữ liệu bộ nhớ

➤ **Giai đoạn nhận lệnh:** Là giai đoạn chung cho tất cả mọi lệnh thao tác dữ liệu. Nếu chương trình chưa kết thúc thì nội dung của bộ đếm chương trình PC được chuyển vào thanh ghi địa chỉ bộ nhớ MAR ($MAR \leftarrow PC$) và một yêu cầu đọc vị trí nhớ có địa chỉ trong MAR (Read $M(MAR)$) được gửi tới bộ nhớ. Dữ liệu yêu cầu (lệnh cần thực hiện) tại vị trí nhớ $M(MAR)$ được chuyển tới thanh ghi bộ nhớ MBR ($MBR \leftarrow M(MAR)$). Lúc này nội dung của PC được tăng lên một giá trị bằng độ dài lệnh vừa nạp là I ($PC \leftarrow PC + I$) để chuẩn bị cho việc nạp lệnh tiếp theo sẽ được thực hiện. Phần opcode của lệnh vừa nạp trong MBR

được gửi vào thanh ghi lệnh (IR) và phần địa chỉ (Addr) trong lệnh vừa nạp được chuyển vào thanh ghi địa chỉ bộ nhớ (MAR).

➤ **Giai đoạn giải mã lệnh:** Nội dung của thanh ghi lệnh được đưa vào đơn vị điều khiển (CU \leftarrow IR). Tại đây, tuỳ theo lệnh mà CPU biết cần thực hiện thao tác gì (nạp toán hạng – Load, lưu toán hạng ra bộ nhớ – Store, cộng – Add, and logic – And, lệnh nhảy – Jump, lệnh dịch phải – Shift,...).

➤ **Giai đoạn thi hành lệnh:** Trong giai đoạn này, các vi thao tác (micro – operations) tuỳ thuộc vào lệnh đã nạp. Hình 3.3 cho thấy một trong bảy lệnh khác nhau có thể được nạp và được thi hành trong giai đoạn này (Load, Store, Add, And, Jump1, Jump2, Shift). Và ta thấy rằng, mỗi một lệnh được thực thi bởi một chuỗi các vi thao tác.

Ví dụ:

– Lệnh Load yêu cầu hai vi thao tác: nạp toán hạng từ vị trí nhớ có địa chỉ chứa trong thanh ghi địa chỉ bộ nhớ MAR vào MBR ($MBR \leftarrow M(MAR)$) và sau đó nội dung của MBR được đưa vào thanh ghi Acc ($Acc \leftarrow MBR$);

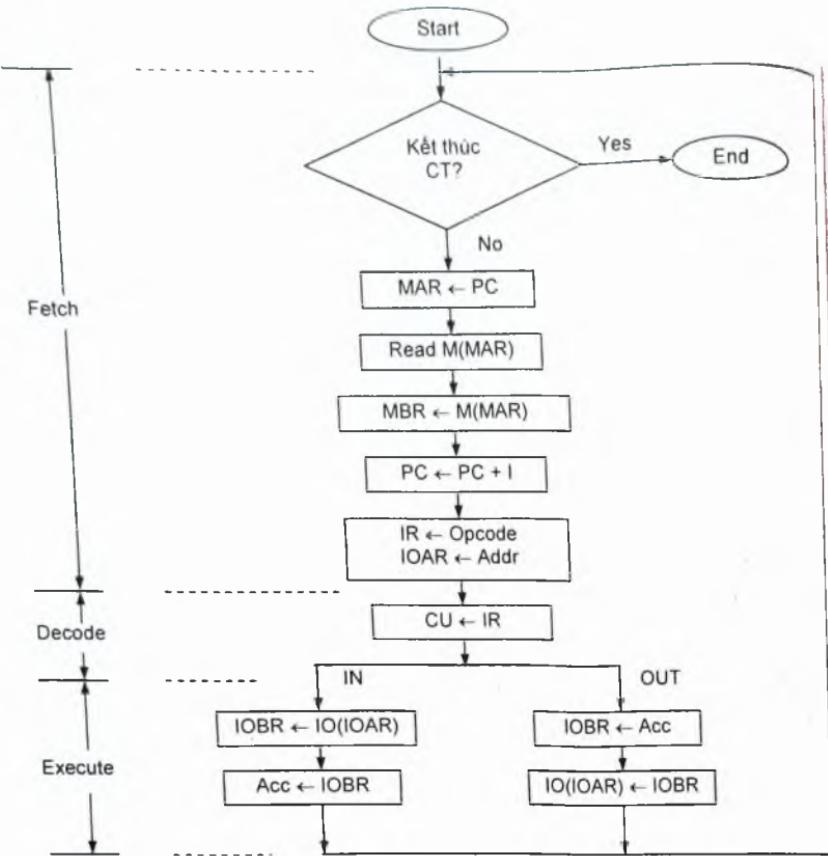
– Lệnh Store yêu cầu hai vi thao tác: chuyển nội dung của Acc vào MBR ($MBR \leftarrow Acc$) và nội dung của MBR được ghi ra vị trí nhớ có địa chỉ chứa trong MAR ($M(MAR) \leftarrow MBR$);

– Lệnh Add có thể yêu cầu ba thao tác: đầu tiên, nạp toán hạng thứ hai từ vị trí nhớ có địa chỉ chứa trong MAR đưa vào thanh ghi dữ liệu MBR ($MBR \leftarrow M(MAR)$), tiếp theo là chuyển nội dung của MBR vào thanh ghi tạm TMP ($TMP \leftarrow MBR$), cuối cùng là thực hiện cộng nội dung của Acc với TMP và kết quả cất trong Acc ($Acc \leftarrow Acc + TMP$),...

b) Các thao tác cơ bản khi thực hiện lệnh vào/ra dữ liệu trực tiếp với thiết bị vào/ra

Hình 3.4 cho thấy các thao tác cơ bản được thực hiện trong một chu kỳ lệnh vào/ra dữ liệu trực tiếp.

➤ **Giai đoạn nhận lệnh:** Là giai đoạn chung cho các lệnh vào/ra dữ liệu, giai đoạn này chỉ khác với giai đoạn nhận lệnh của lệnh xử lý dữ liệu là phần địa chỉ (Addr) của lệnh vừa nạp trong MBR được chuyển vào thanh ghi địa chỉ vào/ra (IOAR).



Hình 3.4. Các thao tác cơ bản trong một chu kỳ lệnh vào/ra dữ liệu trực tiếp

➤ **Giai đoạn giải mã lệnh:** Nội dung của thanh ghi lệnh được đưa vào đơn vị điều khiển (CU ← IR). Tại đây, tùy theo lệnh mà CPU biết cần thực hiện thao tác gì? (nhập dữ liệu – IN, xuất dữ liệu – OUT).

➤ **Giai đoạn thi hành lệnh:** Với lệnh vào/ra dữ liệu trực tiếp, CPU thường dùng lệnh IN/OUT. Lệnh IN đọc nội dung của cổng vào có địa chỉ chứa trong IOAR đưa vào thanh ghi IOBR (IOBR ← IO(IOAR)), sau đó dữ liệu trong IOBR được chuyển vào Acc (Acc ← IOBR). Còn lệnh OUT, đưa nội dung của Acc vào thanh ghi IOBR (IOBR ← Acc), sau đó dữ liệu từ thanh ghi IOBR được chuyển ra cổng ra có địa chỉ chứa trong IOAR (IO(IOAR) ← IOBR).

Trong trường hợp chu kỳ lệnh với ngắt:

Tại thời điểm hoàn thành giai đoạn thi hành lệnh, một thao tác kiểm tra được thực hiện để quyết định yêu cầu ngắt có được chấp nhận hay không. Nếu yêu cầu ngắt được chấp nhận, các thao tác sau được thực hiện:

- t1: MBR \leftarrow PC
- t2: MAR \leftarrow Save_Address
PC \leftarrow Routine_Address
- t3: M(MAR) \leftarrow MBR

Trong bước đầu tiên, nội dung của PC được chuyển vào MBR (MBR \leftarrow PC): lưu địa chỉ của lệnh tiếp theo sau lệnh vừa thi hành xong, để lệnh này có thể được nạp thực hiện sau khi hoàn thành chương trình con phục vụ ngắt; Bước tiếp theo giá trị địa chỉ của vị trí nhớ (Save_Address) mà vị trí nhớ này sẽ lưu nội dung cũ của PC vừa chuyển vào MBR được chuyển vào MAR (MAR \leftarrow Save_address), ví dụ trong máy tính IBM-PC thì Save_address nằm trong một thanh ghi có tên là con trỏ ngăn xếp (Stack Pointer – SP), và PC được nạp địa chỉ của lệnh đầu tiên (Routine_Address) trong chương trình con phục vụ ngắt (PC \leftarrow Routine_Address), hai hành động này là hai vi thao tác riêng biệt và xảy ra đồng thời. Khi bắt kỳ một thao tác nào trong hai thao tác trên hoàn thành thì bước cuối cùng là lưu trữ nội dung của MBR (đang chứa giá trị cũ của PC) ra bộ nhớ. Lúc này vi xử lý sẵn sàng bắt đầu một chu kỳ lệnh mới.

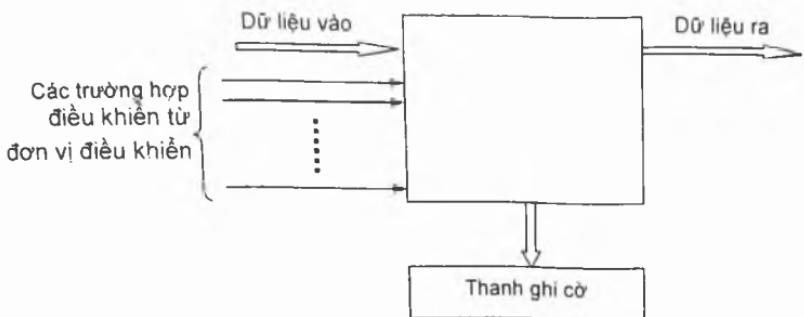
3.1.2. Các thành phần của CPU

CPU gồm đơn vị số học – logic (arithmetic – logic unit – ALU), đơn vị điều khiển (Control Unit – CU) và tập các thanh ghi tốc độ cao (Acc, TMP, MBR, MAR, IOBR, IOAR, PC, IR). Các thanh ghi này (đã đề cập ở mục 2.3.2) được dùng để lưu trữ lệnh, dữ liệu và địa chỉ bộ nhớ hoặc địa chỉ thiết bị vào/ra. Các thành phần trên trao đổi thông tin tín hiệu với nhau thông qua các Bus trong CPU. Phần này ta chỉ đề cập đến ALU và CU.

a) Đơn vị số học và logic (ALU)

➢ **Chức năng:** Thực hiện các phép toán số học và logic.

- Số học: +, -, *, /, đảo dấu, dịch, quay,...
- Logic: AND, OR, XOR, NOT.



Hình 3.5. Sơ đồ khái giao tiếp chung của ALU trong CPU

➤ **Dữ liệu vào**

- Có sẵn trong các thanh ghi đưa vào ALU.
- Từ ngăn nhớ hoặc cổng đưa vào ALU: CPU phát ra tín hiệu tìm dữ liệu và từ ngăn nhớ hoặc cổng đưa qua thanh ghi đưa vào ALU.

➤ **Dữ liệu ra**

Từ ALU đưa vào các thanh ghi: Phải có tín hiệu điều khiển ghi thì dữ liệu mới thực sự được ghi vào thanh ghi. ALU gồm hai khối chính:

- Khối xử lý số học (Arithmetic): Thực hiện các phép toán số học;
- Khối xử lý logic: Thực hiện các phép toán logic.

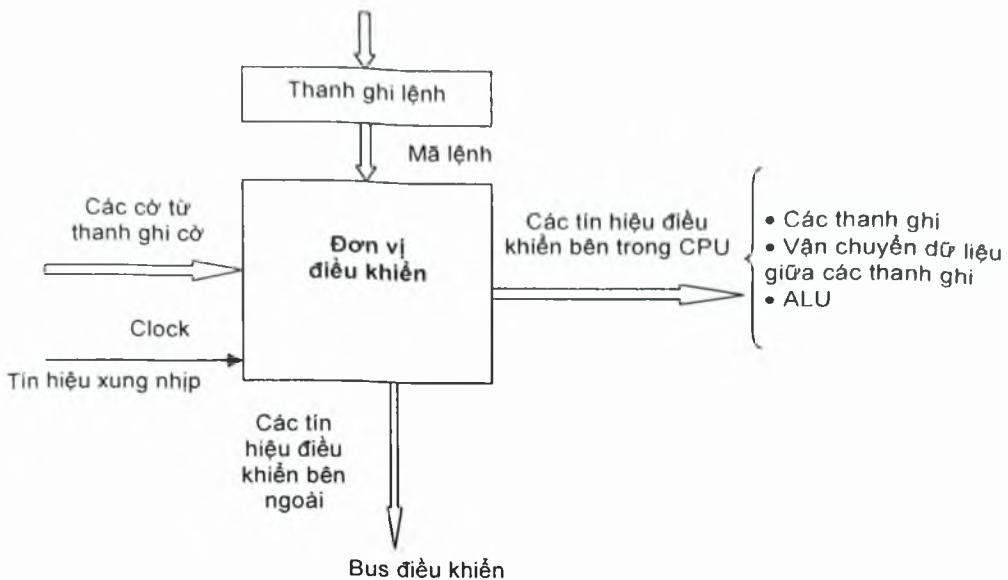
ALU được xây dựng nên từ các mạch dãy tổ hợp. Các mạch dãy tổ hợp được tạo từ các mạch công đơn giản. Ví dụ, mạch cộng đơn giản dùng mạch cổng XOR, mạch cộng đầy đủ, mạch dịch,... (ta sẽ xem xét ở phần sau – mục 3.1.3).

b) **Đơn vị điều khiển (CU)**

Dùng để điều khiển và đồng bộ các hoạt động của hệ thống, cụ thể:

- Điều khiển nhận lệnh từ bộ nhớ và sau đó tăng nội dung PC (bộ đếm chương trình – Program Counter) để chuẩn bị nhận lệnh tiếp theo.
- Giải mã lệnh nằm ở thanh ghi lệnh để xác định yêu cầu của lệnh và phát ra tín hiệu điều khiển thực hiện lệnh đó.
- Nhận các tín hiệu yêu cầu từ bên ngoài, xử lý và đáp ứng các yêu cầu đó.

Đơn vị điều khiển gồm hai khối chính: khối giải mã lệnh nhận lệnh từ thanh ghi lệnh để giải mã lệnh rồi đưa vào khối tạo xung nhịp, và khối tạo xung nhịp nhận tín hiệu lệnh đã giải mã để phát ra chuỗi các xung tương ứng điều khiển thực hiện lệnh.



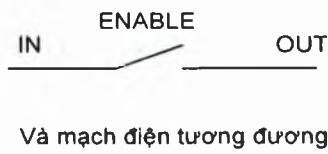
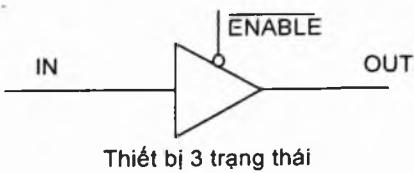
Hình 3.6. Sơ đồ khái niệm giao tiếp chung của CU trong CPU

3.1.3. Thiết bị 3 trạng thái

Thiết bị 3 trạng thái là phương tiện giúp cho việc điều khiển kết nối (về mặt điện) bộ nhớ và các thiết bị lên hệ thống Bus.

Việc chỉ dùng một Bus để kết nối và truyền một loại thông tin giữa các thiết bị khác nhau đòi hỏi tại một thời điểm chỉ một thiết bị được phép phát tín hiệu (mang tải thông tin) lên Bus. Nếu tại một thời điểm lại có nhiều thiết bị được kết nối và cùng phát tín hiệu lên Bus thì sẽ gây ra xung đột trên Bus này.

Để làm chủ được việc kết nối (về mặt điện) các thiết bị lên Bus, người ta sử dụng một thiết bị điện tử có tên là thiết bị 3 trạng thái (tri – state device). Dạng logic và mạch điện tương đương của thiết bị 3 trạng thái *bình thường là thông* như sau:

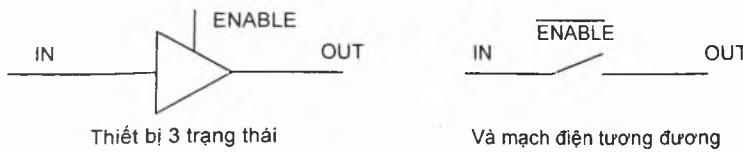


Bảng mô tả chức năng của thiết bị 3 trạng thái *bình thường là thông* như hình 3.7.

ENABLE	IN	OUT (có 3 trạng thái)
0	0	0
0	1	1
1	0	Z (trở kháng cao)
1	1	Z (trở kháng cao)

Hình 3.7. Ký hiệu và bảng chân lý của mạch 3 trạng thái *bình thường là thông*

Dạng logic và mạch điện tương đương của thiết bị 3 trạng thái *bình thường là thông* như sau:



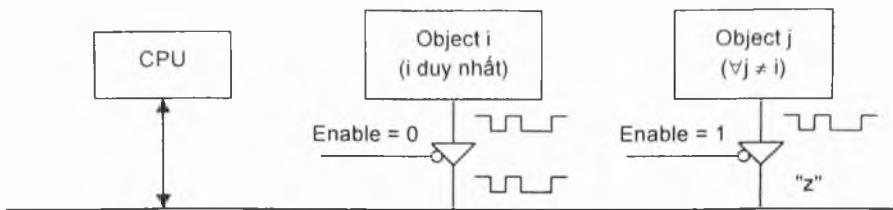
Bảng mô tả chức năng của thiết bị 3 trạng thái *bình thường là ngắt* như hình 3.8.

ENABLE	IN	OUT (có 3 trạng thái)
0	0	Z (trở kháng cao)
0	1	Z (trở kháng cao)
1	0	0
1	1	1

Hình 3.8. Ký hiệu và bảng chân lý của mạch 3 trạng thái *bình thường là ngắt*

Thiết bị 3 trạng thái cho phép CPU hoặc các đơn vị chủ Bus (bus master) khác làm chủ được việc kết nối các đối tượng khác nhau lên Bus. Khả năng làm chủ việc kết nối (về mặt điện) sẽ cho phép tránh sự kiện có hai (hay nhiều) thiết bị cùng được kết nối vào Bus dữ liệu và cùng phát dữ liệu trong cùng một thời điểm (đảm bảo không để xảy ra xung đột). Khi cần chọn đối tượng nào CPU phát ra chỉ của đối tượng đó, và tín hiệu giải mã địa chỉ này kết hợp với tín hiệu điều khiển để tạo ra tín hiệu cho phép ENABLE. Ví dụ, dùng thiết bị 3 trạng thái *bình thường là thông – ở đối tượng có địa chỉ phù hợp, một tín hiệu Enable = "0" để tạo ra, cho phép đối tượng này (thông qua thiết bị 3 trạng thái) kết nối lên Bus*.

Các đối tượng có địa chỉ không phù hợp thì tín hiệu Enable sẽ là "1", thiết bị 3 trạng thái gắn với nó có trạng thái trở kháng cao, nên đối tượng này không kết nối được lên Bus, xem hình 3.9.

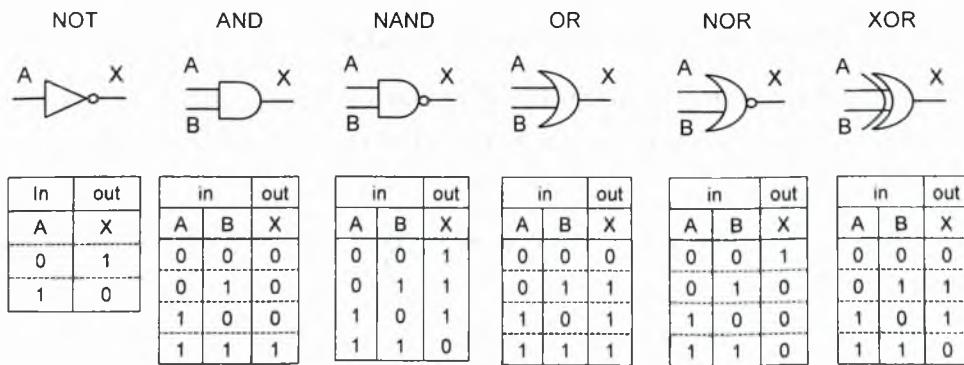


Hình 3.9. Sơ đồ điều khiển kết nối Bus

3.1.4. Các mạch công đồng đơn giản

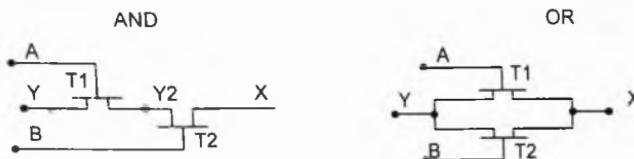
Những thiết bị điện tử rất nhỏ bé có thể thực hiện các chức năng khác nhau trên các giá trị nhị phân (khoảng điện áp từ $0 \div 1V$, gọi là mức thấp hay giá trị 0; khoảng điện áp từ $2 \div 5V$ gọi là mức cao hay giá trị 1) được gọi là *công*, chính các công tạo nên cơ sở phần cứng của tất cả các loại máy tính.

Công có một hoặc nhiều lối vào (input) nhưng chỉ có một lối ra (output). Các giá trị vào hoặc ra chỉ có thể nhận một trong hai giá trị là 0 hoặc 1. Công còn được gọi là *mạch logic* bởi nó thực hiện các phép đại số logic. Chi tiết hoạt động bên trong của công không thuộc khuôn khổ của giáo trình này. Tuy nhiên chúng ta cũng cần nắm một số mạch công đồng đơn giản và chức năng của chúng. Sau đây là 6 mạch công đồng đơn giản: NOT, AND, NAND, OR, NOR, XOR và hình trạng chức năng của từng công tương ứng (hình 3.10).



Hình 3.10. Ký hiệu và hình trạng chức năng của 6 công cơ bản

Các mạch công cơ bản được xây dựng từ các transistor. Hình 3.11 minh họa sơ đồ mạch AND và OR được xây dựng từ hai transistor T1 và T2, hai tín hiệu đầu vào là A và B, tín hiệu ra là X, cuối cùng Y là tín hiệu nguồn hiện thời luôn bằng 1.



Hình 3.11. Sơ đồ mạch của công AND và OR từ các transistor

Với mạch AND:

- Khi $A = 1, B = 1$, T1 và T2 đều thông do vậy $X = Y_2 = Y = 1$.
- Khi $A = 1, B = 0$, T1 thông, $Y_2 = Y = 1$, T2 ngắt, $X = 0$.
- Khi $A = 0, B = 1$, T1 ngắt, $Y_2 = 0$, T2 thông, $X = Y_2 = 0$.
- Khi $A = 0, B = 0$, T1 và T2 cùng ngắt, $X = 0$.

Với mạch OR:

- Khi $A = 1, B = 1$, T1 và T2 đều thông do vậy $X = Y = 1$.
- Khi $A = 1, B = 0$, T1 thông, $X = Y = 1$.
- Khi $A = 0, B = 1$, T2 thông, $X = Y = 1$.
- Khi $A = 0, B = 0$, T1 và T2 cùng ngắt, $X = 0$.

3.1.5. Ví dụ một số mạch chức năng trong ALU

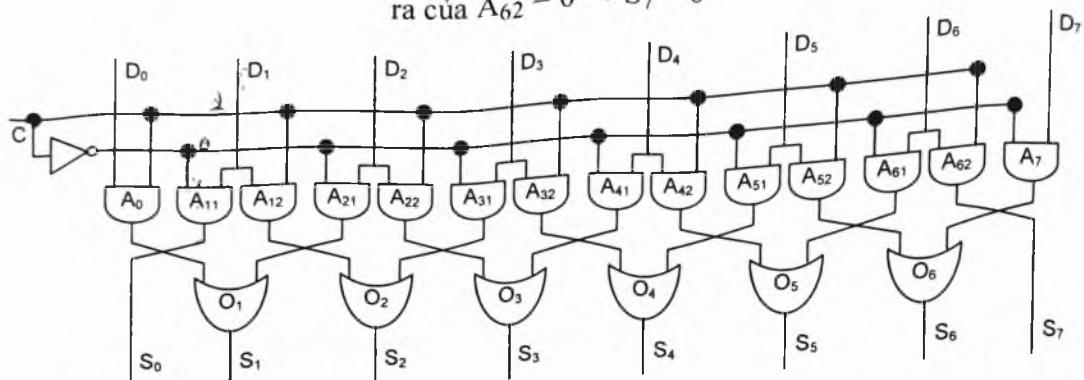
Phản này đưa ra một số mạch chức năng như mạch cộng, mạch dịch để chúng ta có một cái nhìn cơ bản về nguyên lý thao tác thực hiện lệnh từ các mạch công cơ bản.

Hình 3.12 cho thấy cấu tạo và hoạt động của bộ dịch được xây dựng từ các mạch công cơ bản.

Theo hình 3.12, bộ dịch làm việc như sau:

Khi bit C = 0, bộ dịch có chức năng dịch phải tất cả các bit đi 1 bit, cụ thể:

$C = 0 \rightarrow \bar{C} = 1 \rightarrow$ ra của $A_{11} = D_1 \rightarrow S_0 = D_1$
 ra của $A_{21} = D_2$, ra của $A_{00} = 0 \rightarrow S_1 = D_2$
 ra của $A_{31} = D_3$, ra của $A_{12} = 0 \rightarrow S_2 = D_3$
 ra của $A_{41} = D_4$, ra của $A_{22} = 0 \rightarrow S_3 = D_4$
 ra của $A_{51} = D_5$, ra của $A_{32} = 0 \rightarrow S_4 = D_5$
 ra của $A_{61} = D_6$, ra của $A_{42} = 0 \rightarrow S_5 = D_6$
 ra của $A_7 = D_7$, ra của $A_{52} = 0 \rightarrow S_6 = D_7$
 ra của $A_{62} = 0 \rightarrow S_7 = 0$



Trong đó: $D_7 \div D_0$: dữ liệu đầu vào; C : bit điều khiển hướng dịch;
 $S_7 \div S_0$: dữ liệu đầu ra.

Hình 3.12. Sơ đồ ghép nối của bộ dịch từ các mạch cổng đơn giản

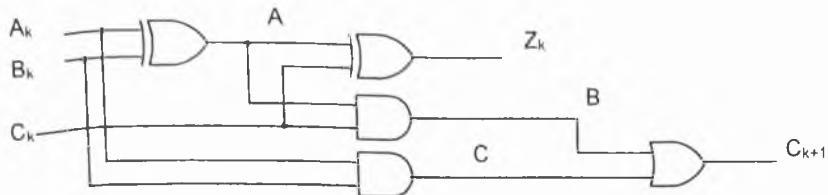
Ví dụ với giá trị cụ thể như sau:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	1	0	1	0	1	1	0
\downarrow							
S_7	S_6	S_5	S_4	S_3	S_2	S_1	S_0
0	1	1	0	1	0	1	1

Ta thấy các giá trị đầu vào là: $11010110_b = D6h = 214$ đã bị dịch phải tất cả đi 1 bit, bit D_0 (S_0) bật ra ngoài và giá trị 0 được đưa vào bit D_7 (S_7). Giá trị đầu ra là $01101011_b = 6Bh = 107$ (phép dịch phải một bit tương ứng với phép chia cho 2 lấy phần nguyên).

Trường hợp khi $C = 1$, người đọc tự chứng minh mạch trên hình 3.12 là bộ dịch trái 1 bit.

Hình 3.13 minh họa mạch cộng đầy đủ xây dựng từ các mạch cổng cơ bản.



Trong đó: A_k, B_k : đầu vào

Z_k : đầu ra

C_k : nhớ từ số thứ $k - 1$ sang

C_{k+1} : nhớ tới số thứ $k + 1$.

Bảng chân lý của bộ cộng đầy đủ

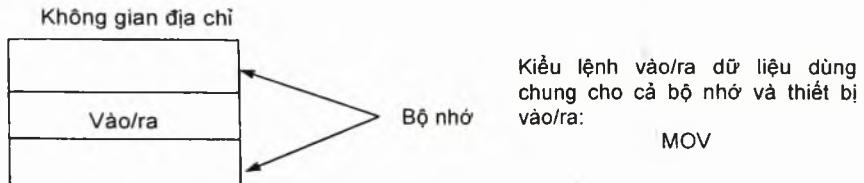
Đầu vào			Đầu ra	
C_k	A_k	B_k	Z_k	C_{k+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Hình 3.13. Mạch cộng đầy đủ và bảng chân lý

3.1.6. Lệnh xử lý vào/ra dữ liệu

Tùy theo phương pháp đánh địa chỉ cho các đối tượng như đã trình bày ở mục 2.6, mà hệ thống máy tính được xác định sử dụng các loại lệnh vào/ra dữ liệu khác nhau.

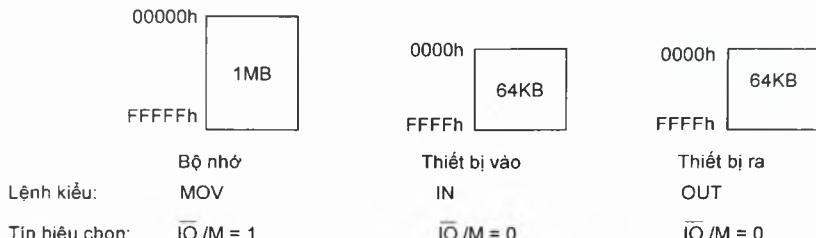
Nếu thiết bị vào/ra được đánh địa chỉ chung với không gian địa chỉ bộ nhớ thì hệ thống máy tính thường sử dụng lệnh trao đổi dữ liệu kiểu MOV cho cả bộ nhớ và thiết bị vào/ra.



Hình 3.14. Bộ nhớ và thiết bị ngoại vi chung không gian địa chỉ

Nếu thiết bị ngoại vi được sử dụng một không gian địa chỉ tách biệt với không gian địa chỉ bộ nhớ thì hệ thống máy tính thường dùng lệnh IN (nhập dữ liệu), OUT (xuất dữ liệu).

Ví dụ: Với máy tính 8086, trong cách phối ghép này bộ nhớ được dùng độc quyền không gian nhớ 1MB mà CPU giành cho nó. Các thiết bị ngoại vi vào/ra dữ liệu với máy tính qua các cổng và được giành riêng một không gian 64KB cho mỗi loại cổng vào/cổng ra. Xong 20 đường địa chỉ của CPU lại dùng chung để xác định địa chỉ cho cả bộ nhớ và thiết bị vào/ra. Tất nhiên, CPU phải dùng tín hiệu \overline{IO}/M để chỉ rõ đối tượng nó cần trao đổi dữ liệu và các lệnh trao đổi dữ liệu thích hợp cho mỗi khoảng không gian đó như hình 3.15.



Hình 3.15. Bộ nhớ và thiết bị ngoại vi có không gian địa chỉ tách biệt

3.2. ĐƯỜNG TRUYỀN (BUS)

3.2.1. Các kiểu đường truyền

Tồn tại hai kiểu đường truyền dữ liệu phục vụ cho việc trao đổi dữ liệu giữa CPU và các phần mạch cũng như thiết bị bên ngoài, đó là kiểu đường truyền nối tiếp và kiểu đường truyền song song.

➢ **Đường truyền nối tiếp:** Tại mỗi thời điểm trên đường truyền chỉ có 1 bit được truyền. Ví dụ, muốn truyền một ký tự (8 bit) thì phải truyền làm 8 lần. Như vậy, với kiểu đường truyền này chỉ cần một đường dây dữ liệu phục vụ cho việc truyền dữ liệu.

➢ **Đường truyền song song:** Tại mỗi thời điểm trên đường truyền có nhiều bit được truyền đồng thời. Ví dụ, muốn truyền một ký tự (8 bit) thì chỉ cần truyền một lần, song phải cần đến một tập 8 đường dây. Như vậy, kiểu đường truyền song song có khả năng truyền dữ liệu nhanh hơn kiểu truyền nối tiếp m lần (với m đường dây dữ liệu), song khả năng truyền tín hiệu đi xa kém hơn kiểu truyền nối tiếp do sóng điện từ trên các đường truyền gây nhiễu lẫn nhau (nhiễu xuyên âm) làm suy giảm tín hiệu trên đường truyền.

3.2.2. Đường truyền và tín hiệu điều khiển

Đường truyền là hệ thống các đường dây dẫn vận chuyển tín hiệu điện từ phần mạch này sang phần mạch khác trong phạm vi máy tính, còn được gọi là Bus.

Trong máy tính, phân loại theo chức năng thì có ba loại Bus sau:

➤ **Bus địa chỉ (Address Bus – Bus A):** Dùng để truyền tín hiệu địa chỉ từ CPU ra ngoài, nhằm xác định đối tượng cần trao đổi dữ liệu với CPU. Bus A bao gồm tập n đường dây dẫn điện.

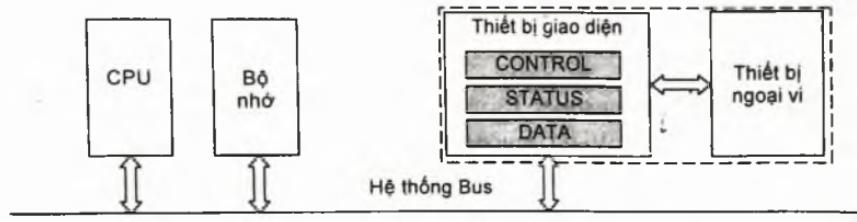
➤ **Bus dữ liệu (Data Bus – Bus D):** Dùng để truyền tín hiệu dữ liệu giữa các phần mạch khác nhau trong máy tính, cụ thể: truyền lệnh từ bộ nhớ trong và CPU; truyền dữ liệu từ CPU ra bộ nhớ và ngược lại; truyền dữ liệu từ các thiết bị vào/ra tới bộ nhớ và ngược lại. Bus D gồm m (thường $m = 8 \times 2^l$) đường dây dẫn điện.

➤ **Bus điều khiển:** Gồm nhiều đường dây dẫn điện, mỗi đường dây có chức năng truyền một tín hiệu điều khiển khác nhau để các phần mạch trong hệ thống máy tính làm việc với nhau được đồng bộ, tránh được sự xung đột khi thực hiện chương trình cũng như khi thực hiện trao đổi dữ liệu (xem mục 1.1.1 phần tử chủ máy tính).

3.3. HỆ THỐNG VÀO/RA

3.3.1. Cấu trúc phần cứng của hệ thống xử lý vào/ra dữ liệu trong máy tính

Đơn vị xử lý trung tâm (CPU) thực hiện trao đổi thông tin với các thiết bị ngoại vi thông qua các thiết bị giao diện (các khối ghép nối) như hình 3.16.



Hình 3.16. Sơ đồ khái niệm kiến trúc hệ thống xử lý vào/ra dữ liệu

Sự có mặt của thiết bị giao diện là do có sự khác biệt rất lớn về dạng thức biểu diễn, truyền tải dữ liệu và tốc độ xử lý dữ liệu giữa đơn vị xử lý trung tâm và

các thiết bị ngoại vi. Bên trong máy tính, dữ liệu được mã hoá thành các số nhị phân và biểu diễn ở dạng tín hiệu điện với hai mức khác nhau (0/1). Dữ liệu trong các thiết bị vào/ra có thể được biểu diễn, truyền tải dưới nhiều dạng khác nhau như: ánh sáng, âm thanh, hình ảnh,... CPU xử lý dữ liệu với tốc độ rất cao, các thiết bị ngoại vi xử lý dữ liệu với tốc độ chậm hơn rất nhiều.

3.3.2. Môđun vào/ra

Trong máy tính, các phần mạch thiết bị được sản xuất theo từng môđun (vi mạch) riêng biệt và lắp ghép với nhau qua các khe cắm, đế cắm, công đã được chuẩn hoá. Đồng thời thiết bị giao diện cũng được chế tạo theo từng môđun, gọi là môđun vào/ra hoặc còn gọi là *control card* (thẻ mạch điều khiển thiết bị).

Thiết bị giao diện là loại thiết bị khả trinh (các mạch điều khiển có thể lập trình). Mỗi loại thiết bị giao diện đều có ba loại thanh ghi, mỗi loại thực hiện một chức năng khác nhau, đó là các thanh ghi điều khiển (control), thanh ghi trạng thái (status) và thanh ghi dữ liệu (data). Mỗi một thanh ghi đều được gắn một địa chỉ xác định, địa chỉ của thanh ghi dữ liệu được gọi là địa chỉ cơ sở của cổng hay gọi tắt là địa chỉ cổng (xem hình 3.16).

- Các thanh ghi điều khiển (Control Register) nhận và chứa các từ điều khiển, xác lập chế độ làm việc của thiết bị.
- Các thanh ghi trạng thái (Status Register) chứa thông tin phản ánh trạng thái làm việc của thiết bị giao diện và thiết bị ngoại vi.
- Các thanh ghi dữ liệu (Data Register) thực hiện chức năng bộ đệm, nơi trung chuyển dữ liệu vào/ra.

Khi CPU đưa một dữ liệu ra ngoài (khi CPU thực hiện lệnh OUT xuất một dữ liệu ra cổng có địa chỉ xác định) thực chất là CPU đưa dữ liệu ra thanh ghi dữ liệu của thiết bị giao diện, thiết bị giao diện sẽ chuyển đổi nó thành dạng thích hợp với thiết bị ngoại vi rồi mới đưa ra ngoài cho thiết bị ngoại vi.

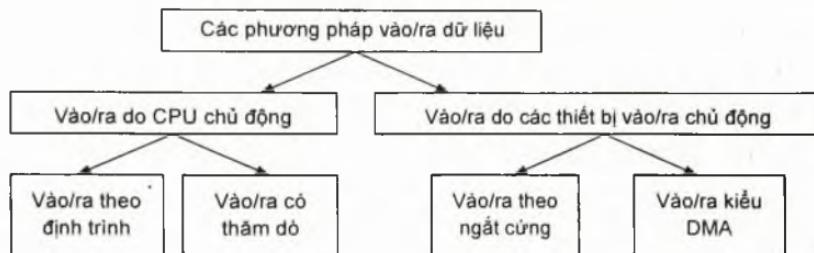
Khi thiết bị ngoại vi gửi một dữ liệu vào máy tính, dữ liệu này được thiết bị giao diện chuyển đổi về dạng phù hợp với máy tính rồi đưa vào thanh ghi dữ liệu trong thiết bị giao diện, CPU nhập dữ liệu từ ngoài vào bằng cách đọc thanh ghi dữ liệu.

3.3.3. Lập trình điều khiển vào/ra

Có nhiều phương pháp vào/ra dữ liệu giữa CPU và thiết bị ngoại vi, song phương pháp nào cũng cần phải có các chương trình điều khiển vào/ra phù hợp. Các chương trình điều khiển vào/ra thường được nhúng sẵn trong các mạch phần cứng như BIOS ROM có các chương trình điều khiển vào/ra cơ bản của hệ thống (các chương trình con phục vụ ngắt của BIOS), các module vào/ra được lập trình điều khiển sẵn và cho phép thiết lập, truyền các thông số làm việc khác nhau để người sử dụng có thể thực hiện thao tác điều khiển vào/ra dữ liệu với thiết bị một cách tiện lợi.

3.3.4. Các phương pháp vào/ra dữ liệu

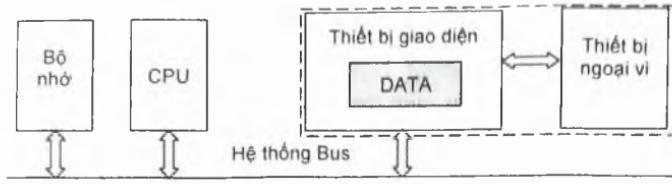
Thiết bị giao diện chỉ giúp CPU kết nối một cách thích hợp về mặt vật lý với các thiết bị ngoại vi, nhưng chưa đảm bảo tính tin cậy của quá trình trao đổi dữ liệu. Điều này xuất phát từ một thực tế khách quan là nhịp làm việc (tốc độ làm việc) của CPU nhanh hơn rất nhiều nhịp làm việc của các thiết bị ngoại vi. Để CPU có thể thực hiện trao đổi dữ liệu với các thiết bị ngoại vi với độ tin cậy cao, cần phải áp dụng các phương pháp vào/ra dữ liệu thích hợp. Có bốn phương pháp vào/ra dữ liệu, nằm trong hai nhóm phương pháp khác nhau (hình 3.17).



a) Phương pháp vào/ra theo định trình

Đây là phương pháp mà trong đó quá trình vào/ra được thực hiện tức thời nhờ các lệnh vào/ra (IN hoặc OUT) và CPU không cần quan tâm đến trạng thái của thiết bị vào/ra (bao gồm thiết bị giao diện và thiết bị ngoại vi) (hình 3.18).

Phương pháp này có nhược điểm là độ tin cậy trong truyền, nhận dữ liệu không cao, dữ liệu truyền nhận dễ bị chồng lên nhau gây mất dữ liệu nếu bên nhận xử lý dữ liệu không kịp.

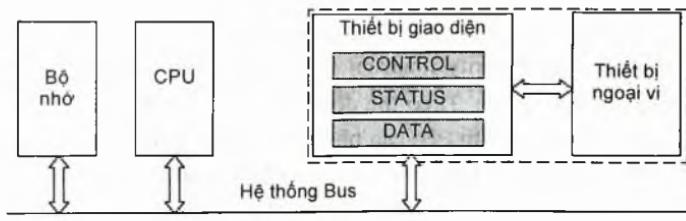


Hình 3.18. Mô hình hệ thống xử lý vào/ra dữ liệu theo phương pháp định trình

Phương pháp vào/ra theo định trình thích hợp với những quá trình vào/ra có chu kỳ cố định và có thể xác định trước.

b) Phương pháp vào/ra có thăm dò

Trong mỗi thiết bị giao diện thường có ít nhất một thanh ghi trạng thái chứa thông tin phản ánh trạng thái làm việc của thiết bị này và của thiết bị ngoại vi. Khi thực hiện vào/ra dữ liệu bằng phương pháp thăm dò, CPU (chương trình vào/ra dữ liệu) luôn thực hiện kiểm tra trạng thái sẵn sàng làm việc của thiết bị trước khi việc vào/ra dữ liệu được thực hiện (hình 3.19). Khi thiết bị (mà CPU cần trao đổi dữ liệu) chưa sẵn sàng làm việc thì CPU lại phải tiếp tục thăm dò, việc thăm dò được lặp đi lặp lại cho đến khi thiết bị sẵn sàng trao đổi dữ liệu với CPU thì quá trình truyền nhận dữ liệu được tiến hành.



Hình 3.19. Mô hình hệ thống xử lý vào/ra dữ liệu theo phương pháp thăm dò

Ưu điểm của phương pháp vào/ra dữ liệu có thăm dò là quá trình trao đổi dữ liệu có độ tin cậy rất cao vì việc truyền nhận dữ liệu chỉ xảy ra khi hai bên truyền và nhận đều sẵn sàng.

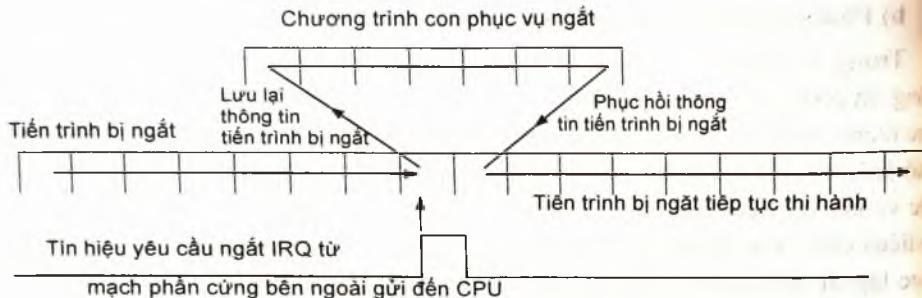
Nhược điểm của phương pháp này là chiếm dụng nhiều thời gian của CPU cho việc thăm dò, dẫn đến hiệu quả hoạt động chung của toàn hệ thống không cao. Trong trường hợp CPU đồng thời phải thực hiện nhiều loại công việc thì thời gian làm việc của CPU bị chia sẻ và bị chiếm dụng làm cho hiệu suất giải quyết các công việc khác của CPU bị hạn chế và có thể ảnh hưởng đến độ tin cậy.

c) Phương pháp vào/ra bằng ngắt cứng

Một số khái niệm về ngắt cứng: Ngắt cứng là sự kiện CPU phải tạm dừng tiến trình đang thực hiện để chuyển sang thực hiện tiến trình phục vụ ngắt khi có yêu cầu ngắt từ phần mạch bên ngoài gửi đến CPU.

Phương pháp vào/ra dữ liệu theo ngắt cứng là phương pháp mà trong đó thiết bị vào/ra chủ động khởi động quá trình vào/ra dữ liệu nhờ hệ thống ngắt cứng.

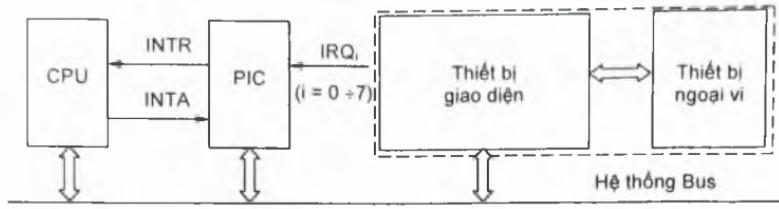
Hình 3.20 mô tả quá trình làm việc của CPU khi có yêu cầu ngắt từ mạch phần cứng bên ngoài gửi đến CPU và CPU chấp nhận ngắt.



Hình 3.20. Mô tả khái niệm ngắt cứng và quá trình ngắt cứng

Thông thường quá trình vào/ra theo ngắt cứng được sự trợ giúp bởi thiết bị điều khiển ngắt PIC (Priority Interrupt Controller). PIC có chức năng nhận các yêu cầu ngắt IRQ_i ($i = 0 \div 7$) có thể đồng thời gửi đến PIC, xử lý ưu tiên ngắt (chọn ra yêu cầu ngắt độ ưu tiên cao nhất) và cung cấp số hiệu ngắt ưu tiên cao nhất cho CPU qua Bus dữ liệu. Căn cứ vào số hiệu ngắt, CPU sẽ lấy được địa chỉ của chương trình con phục vụ ngắt tương ứng với số hiệu ngắt và thực hiện chương trình con phục vụ ngắt này. Trước khi lấy địa chỉ của chương trình con phục vụ ngắt, CPU phải lưu lại các thông tin trạng thái của tiến trình đang thực thi, còn gọi là tiến trình bị ngắt (như địa chỉ của lệnh tiếp theo sẽ được thực hiện, trạng thái đang làm việc của CPU, trạng thái kết quả thực hiện lệnh vừa làm xong,...). Thực hiện xong chương trình con phục vụ ngắt, CPU phải lấy lại các thông tin trạng thái trước đó đã cất để có thể tiếp tục thực thi tiến trình vừa bị ngắt. Các thông tin trạng thái cần lưu trữ trước khi thực hiện chương trình con phục vụ ngắt thường được cất vào stack.

Hình 3.21 mô tả mô hình của hệ thống vào/ra dữ liệu theo phương pháp ngắt cứng.



Hình 3.21. Mô hình hệ thống xử lý vào/ra dữ liệu theo phương pháp ngắt cứng

Trong thực tế, mỗi mạch PIC có thể nhận đồng thời một lúc 8 tín hiệu yêu cầu ngắt từ 8 module vào/ra. Tức là với một mạch PIC thì CPU có thể làm việc vào/ra dữ liệu với 8 loại thiết bị vào/ra. Muốn CPU có thể làm việc vào/ra dữ liệu theo phương pháp này với nhiều thiết bị hơn nữa thì các mạch PIC được ghép nối và lập trình làm việc theo phương pháp nối tầng.

Quá trình vào/ra dữ liệu theo phương pháp ngắt cứng xảy ra cụ thể như sau:

- CPU đang thực hiện tiến trình A.
- Các thiết bị vào/ra có yêu cầu phục vụ, phát ra tín hiệu IRQ_i ($i = 0 \div 7$) tới PIC. Mỗi thiết bị vào/ra đã được ấn định sẵn một số hiệu ngắt nhất định.
- Thiết bị PIC lựa chọn số hiệu ngắt có mức ưu tiên cao nhất, phát tín hiệu INTR (Interrupt Request) tới CPU yêu cầu CPU phục vụ.
- Nếu CPU chấp nhận ngắt, CPU hoàn thành nốt lệnh đang thực hiện, tiến hành lưu trạng thái của tiến trình đang thực hiện và trạng thái hiện thời của CPU.
- CPU phát ra tín hiệu INTA (Interrupt Acknowledge) trả lời PIC, báo sẵn sàng phục vụ yêu cầu ngắt.
- PIC phát ra số hiệu ngắt được chọn tới CPU qua Bus dữ liệu.
- Dựa vào số hiệu ngắt này, CPU xác định được địa chỉ của chương trình con phục vụ ngắt, kích hoạt và thực hiện chương trình con phục vụ ngắt để thực hiện vào/ra dữ liệu với thiết bị được chọn.
- Khi chương trình con phục vụ ngắt kết thúc, CPU lấy lại trạng thái cũ của CPU và của tiến trình A vừa bị ngắt để tiếp tục thi hành.

Ưu điểm của phương pháp này là:

- CPU thực hiện việc vào/ra dữ liệu ngay khi có yêu cầu từ thiết bị vào/ra. Do vậy điều này làm cho độ tin cậy trao đổi dữ liệu cao hơn phương pháp thăm dò. Tuy nhiên so với phương pháp thăm dò, thì vào/ra dữ liệu theo phương pháp này

chưa chắc đã thành công, bởi vì trong thực tế, khi thiết bị có yêu cầu thì chưa chắc CPU đã đáp ứng phục vụ (CPU có thể từ chối yêu cầu ngắt).

– CPU không phải thăm dò trạng thái sẵn sàng của thiết bị vào/ra nên không lãng phí thời gian CPU, do vậy tăng hiệu quả làm việc của CPU.

Do những ưu điểm này mà phương pháp vào/ra dữ liệu theo ngắt cung được dùng để thực hiện với hầu hết các thiết bị vào/ra với máy tính.

Tuy nhiên với phương pháp này, quá trình chuyển dữ liệu vào bộ nhớ vẫn phải qua CPU, tức là CPU vẫn phải thực hiện điều khiển vào/ra dữ liệu nên chưa dành được nhiều nhất thời gian cho xử lý dữ liệu, do vậy hiệu năng hoạt động của toàn hệ thống máy tính chưa phải là cao nhất.

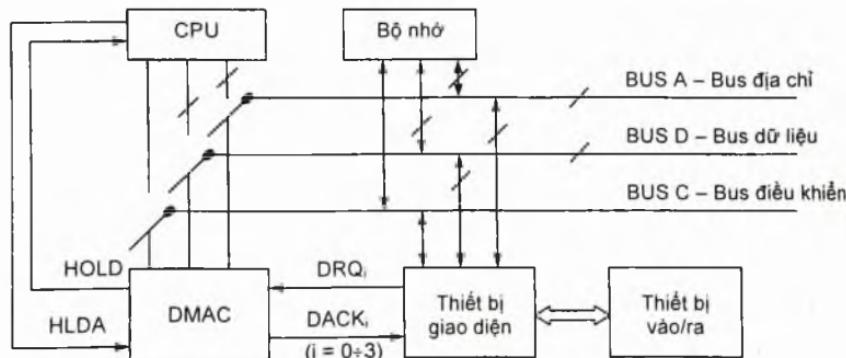
d) Phương pháp vào/ra kiểu truy nhập bộ nhớ trực tiếp (DMA – Direct Memory Access)

Để nâng cao hiệu suất hoạt động của hệ thống, có một phương pháp vào/ra dữ liệu mà CPU cần không phải tham gia điều khiển vào/ra, đó là phương pháp truy cập bộ nhớ trực tiếp (phương pháp DMA).

Quá trình DMA là quá trình vào/ra dữ liệu giữa bộ nhớ và thiết bị ngoại vi mà không thông qua CPU.

Trong quá trình DMA, việc vào/ra dữ liệu không được điều khiển bởi CPU mà được điều khiển bởi một thiết bị phần cứng, đó là bộ điều khiển vào/ra dữ liệu trực tiếp – DMAC (Direct Memory Access Controller).

Để dễ hình dung, hình 3.22 mô tả mô hình hệ thống xử lý vào/ra dữ liệu kiểu DMA bằng cách sử dụng chuyền mạch điện tương đương cho kết nối Bus với CPU và DMAC thay cho tín hiệu điều khiển kết nối bằng thiết bị 3 trạng thái.



Hình 3.22. Mô hình hệ thống xử lý vào/ra dữ liệu kiểu DMA

Một thiết bị DMAC có 4 kênh vào/ra ($i = 0 \div 3$), tức tại một thời điểm DMAC có thể nhận đồng thời 4 yêu cầu vào/ra dữ liệu trực tiếp với bộ nhớ từ 4 thiết bị. Sau đó DMAC phải chọn ra thiết bị có mức ưu tiên cao nhất để cho phép vào/ra dữ liệu kiểu DMA.

Muôn nhiều hơn 4 thiết bị vào/ra có thể vào/ra dữ liệu kiểu DMA thì cần có nhiều thiết bị DMAC, và các thiết bị DMAC phải được ghép nối và lập trình làm việc theo chế độ nối tảng.

Quá trình DMA được thực hiện như sau:

- CPU đang hoạt động bình thường (CPU quản lý hệ thống Bus, bao gồm Bus A, Bus D, Bus C).

- DMAC được xác lập chế độ làm việc, nhận thông tin về địa chỉ đầu khối nhớ chứa dữ liệu và kích thước khối dữ liệu cần truyền.

- Các thiết bị vào/ra phát tín hiệu DRQ_i cho DMAC, DMAC chọn thiết bị có mức ưu tiên cao nhất.

- DMAC phát tín hiệu BRQ (Bus Request)/HOLD = 1 cho CPU, yêu cầu CPU chuyển nhượng Bus.

- Nếu CPU chấp nhận, CPU thực hiện nốt chu kỳ máy, CPU phát ra tín hiệu BGT (Bus Grant)/HLDA (HOLD Acknowledge) trả lời chấp nhận chuyển nhượng Bus.

- CPU tự tách ra khỏi hệ thống Bus, tắt cả các chân tín hiệu A, D và một số chân tín hiệu C ở trạng thái trở kháng cao về mặt logic (hình 3.22 về các dây A, D, C từ CPU ra hệ thống Bus bị ngắt mạch). Quyền điều khiển hệ thống Bus thuộc về DMAC.

- DMAC làm chủ các Bus A, Bus D và một số Bus C. DMAC phát ra tín hiệu DACK_i báo cho thiết bị yêu cầu được chọn vào/ra dữ liệu. DMAC phát địa chỉ ô nhớ đầu tiên của khối dữ liệu cần truyền lên Bus A, phát ra các tín hiệu điều khiển đọc/ghi thiết bị vào/ra và các tín hiệu điều khiển ghi/đọc bộ nhớ, thực hiện điều khiển toàn bộ quá trình vào/ra dữ liệu trực tiếp giữa bộ nhớ và thiết bị ngoại vi. Trong quá trình truyền, DMAC giám bộ đếm và tăng nội dung của con trỏ chứa địa chỉ cho đến khi nội dung bộ đếm bằng 0 thì khối dữ liệu đã chuyển xong.

- Khi một khối dữ liệu đã chuyển xong, DMAC kết thúc quá trình DMA bằng việc phát tín hiệu BRQ/HOLD = 0 cho CPU và trả quyền điều khiển hệ

thông Bus cho CPU. Tất cả các chân tín hiệu của CPU được khai thông với hệ thống Bus.

- CPU tiếp tục làm việc bình thường.

Phương pháp vào/ra dữ liệu kiểu DMA được dùng để thực hiện trao đổi dữ liệu giữa bộ nhớ với các thiết bị ngoại vi có khả năng vào/ra dữ liệu với khối lượng lớn như các loại ổ đĩa cứng, ổ đĩa mềm,...

Trong thực tế tồn tại ba kiểu trao đổi dữ liệu theo phương pháp DMA sau:

- *Treo CPU một khoảng thời gian để 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.

- *Treo CPU để trao đổi từng byte*: Trong cách trao đổi này, CPU không bị treo lâu trong một lần, mà CPU và DMAC luân phiên sử dụng Bus với thời gian đủ để trao đổi 1 byte dữ liệu. Trong kiểu trao đổi này, hoạt động của CPU có ảnh hưởng đến tốc độ thực hiện một công việc nào đó (bị chậm lại), có nghĩa là công việc CPU đang thực hiện không bị dừng lại.

- *Tận dụng thời gian CPU không dùng Bus để trao đổi dữ liệu*: Trong cách trao đổi dữ liệu này, ta phải có các mạch logic phụ bên ngoài cần thiết để phát hiện ra các chu kỳ xử lý nội bộ của CPU (CPU không dùng đến Bus ngoài) và tận dụng các chu kỳ đó cho việc trao đổi dữ liệu. Trong cách này CPU và DMAC cũng luân phiên nhau sử dụng Bus mà không ảnh hưởng gì đến hoạt động bình thường của CPU.

Các chế độ ưu tiên: Để xác định mức ưu tiên cho các kênh, có hai chế độ:

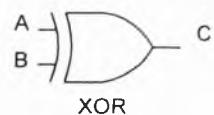
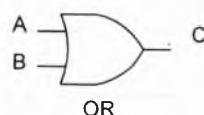
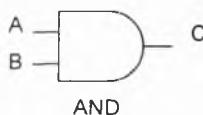
- **Ưu tiên cố định**: Kênh 0 ưu tiên cao nhất, kênh 3 ưu tiên thấp nhất.
- **Ưu tiên vòng**: Kênh nào vừa được phục vụ thì sẽ có mức ưu tiên thấp nhất. Lúc mới lập chế độ thì kênh 0 có mức ưu tiên cao nhất.

CÂU HỎI VÀ BÀI TẬP CHƯƠNG 3

1. Cho biết ý nghĩa khi nói Bus địa chỉ có độ rộng 24 bit.
2. Cho biết ý nghĩa khi nói Bus dữ liệu có độ rộng 32 bit.
3. Trình bày sơ đồ khái niệm của hệ thống vào/ra trong máy tính.
4. Trình bày cấu trúc chung của môđun vào/ra dữ liệu.

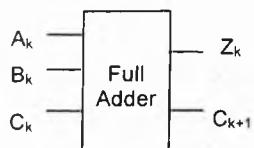
5. Trình bày hai phương pháp vào/ra dữ liệu do CPU chủ động (vào/ra theo định trình và vào/ra kiểu thăm dò).
6. Trình bày phương pháp vào/ra dữ liệu theo ngắt cứng.
7. Trình bày cấu trúc của hệ thống vào/ra theo ngắt cứng.
8. Trình bày quá trình vào/ra dữ liệu theo phương pháp ngắt cứng.
9. Trình bày khái niệm quá trình DMA, cấu trúc của hệ thống vào/ra theo kiểu DMA.
10. Trình bày quá trình vào/ra dữ liệu kiểu DMA (quá trình DMA).
11. Hãy đọc những mô tả dưới đây về các phép toán logic và bộ cộng đầy đủ, sau đó trả lời các ý từ 1 tới 3.

– Các ký hiệu mạch logic cho các phép toán logic chính như sau:



Trong đó: A, B là đầu vào; C là đầu ra

– Sau đây là hình vẽ bộ cộng đầy đủ, thực hiện việc cộng các số nhị phân theo từng chữ số có tính tới việc nhớ. Bảng cho dưới đây là bảng chân lý cho bộ cộng đầy đủ đó.



Trong đó: Ak, Bk: đầu vào;
Zk: đầu ra;
Ck: nhớ từ số thứ k – 1 sang;
Ck+1: nhớ tới số thứ k + 1.

Bộ cộng đầy đủ (full adder)

Bảng chân lý của bộ cộng đầy đủ

Đầu vào			Đầu ra	
Ck	Ak	Bk	Zk	Ck+1
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1		
1	1	0	0	1
1	1	1	1	1

- **Ý 1:** Từ nhóm câu trả lời dưới đây, hãy chọn câu trả lời đúng để điền vào hộp trống trong bảng chân lý của bộ cộng đầy đủ.

Nhóm câu trả lời:

a)

0	0
---	---

c)

1	0
---	---

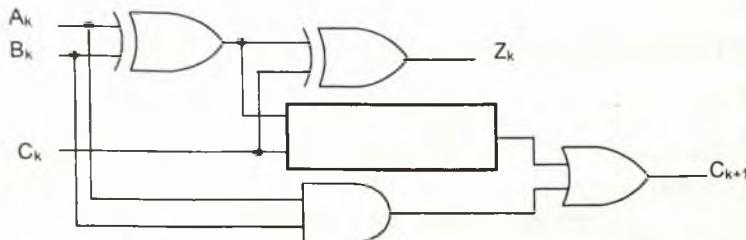
b)

0	1
---	---

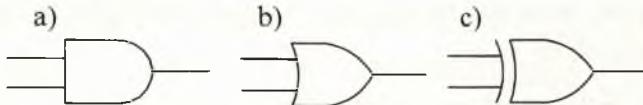
d)

1	1
---	---

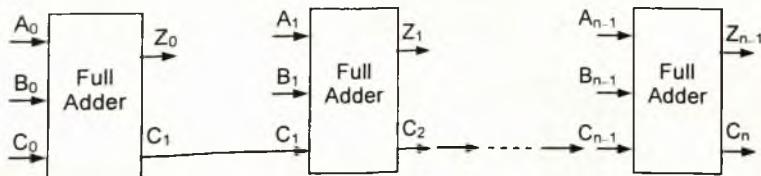
- **Ý 2:** Từ nhóm câu trả lời dưới đây, hãy chọn câu trả lời đúng để điền vào hộp trống trong mạch logic của bộ cộng đầy đủ.



Nhóm câu trả lời:



- **Ý 3:** Khi một mạch logic được cấu tạo bằng các bộ cộng đầy đủ để cộng các số nhị phân n – chữ số được biểu diễn như phần bù hai, việc cộng các chữ số có ý nghĩa nhất (A_n, B_n và C_n) gây ra sự tràn (phần tố đậm của bảng chân lý của bộ cộng đầy đủ). Mạch logic để phát hiện việc này có thể được cấu tạo bằng một mạch XOR. Hãy chọn từ nhóm câu trả lời dưới đây tổ hợp đúng của các đầu vào X và Y cho mạch logic này.



Nhóm câu trả lời:

a) A_{n-1}, B_{n-1}

b) A_{n-1}, Z_{n-1}

c) B_{n-1}, Z_{n-1}

d) C_{n-1}, C_n

e) C_{n-1}, Z_{n-1}

f) C_n, Z_{n-1} .

12. Xét trên khía cạnh logic số, công là

- a) một thiết bị cài đặt một hàm luận lý đơn giản.
- b) một thiết bị có thể lưu trữ dữ liệu.
- c) giao diện của một thiết bị ngoại vi.
- d) một giá trị địa chỉ ô nhớ.

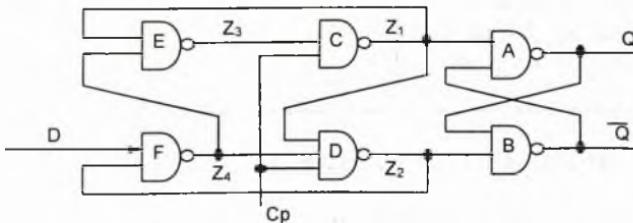
13. Xét trên khía cạnh logic số, ô nhớ là

- a) một thiết bị cài đặt một hàm luận lý đơn giản.
- b) một thiết bị có thể lưu trữ một bit dữ liệu.
- c) một thiết bị có thể lưu trữ dữ liệu.
- d) một thiết bị điều khiển luồng dữ liệu.

14. Quy luật Moore dựa trên cơ sở

- a) sự gia tăng độ trù mật của các transistor được đặt trong một chip máy tính.
- b) sự gia tăng tốc độ của máy tính.
- c) sự phát triển của công nghệ chế tạo chip máy tính.
- d) giá cả phần cứng máy tính.

15. Cho sơ đồ mạch Flip – Flop sau, chứng minh rằng khi $C_p = 0$, trạng thái đầu ra Q không đổi; khi $C_p = 1$ (chuyển từ 0 → 1) thì $Q = D$.



16. Thiết kế mạch logic để phát hiện lỗi trong mã BCD. Lối vào là mã BCD, lối ra ở trạng thái 1 khi có lỗi.

17. Làm thế nào để xây dựng 1 cổng AND có hai lối vào từ những cổng NOR hai lối vào.

18. Hãy xây dựng cổng XOR từ các cổng NOT, AND, OR.

Chương 4

KIẾN TRÚC HỆ THỐNG NHỚ

TÓM TẮT: – Lý thuyết: 7 tiết;
– Bài tập lớn: 6 tiết.

Mục tiêu cần đạt được	Các mục chính	Bài tập bắt buộc	Bài tập làm thêm
<ul style="list-style-type: none">- Cung cấp cho sinh viên kiến thức cơ bản về các loại bộ nhớ bán dẫn trong máy tính, nguyên lý lưu trữ dữ liệu, thao tác đọc/ghi bộ nhớ bán dẫn. Cung cấp phương pháp thiết kế xây dựng bộ nhớ từ các phần tử nhớ 1 bit, phương pháp tổ chức và quản lý bộ nhớ trong máy tính.- Kết thúc chương, sinh viên cần nắm tổng quan về các loại bộ nhớ bán dẫn, nguyên lý đọc/ghi bộ nhớ bán dẫn. Nắm được các phương pháp tổ chức, quản lý, phân phối bộ nhớ trong của máy tính. Có khả năng thiết kế, ghép nối bộ nhớ, mở rộng bộ nhớ trên một máy tính cụ thể.	<ul style="list-style-type: none">4.1. Công nghệ hệ thống nhớ.4.2. Hệ thống nhớ chính.4.3. Các vấn đề về thiết kế bộ nhớ	<p>Câu hỏi và bài tập cuối chương: 1; 2; 3; 10; 11; 12; 13; 14; 18; 19; 21.</p>	<p>Câu hỏi và bài tập cuối chương còn lại.</p>

4.1. CÔNG NGHỆ HỆ THỐNG NHỚ

4.1.1. Tổ chức bộ nhớ theo phân cấp

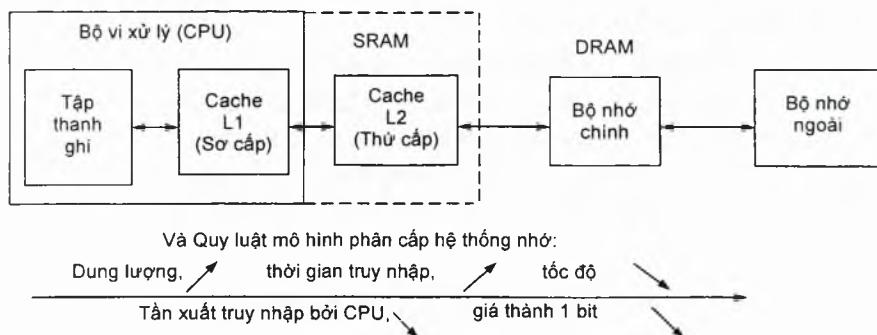
Một trong các chức năng của CPU là tuân tự nhập lệnh từ bộ nhớ và thực hiện lệnh. Tốc độ truy nhập bộ nhớ chính tương đối chậm (với DRAM là khoảng $100\text{ns} = 10^{-7}\text{s}$) so với tốc độ của CPU. Nếu CPU nhập lệnh và dữ liệu trực tiếp từ bộ nhớ chính thì tốc độ xử lý thực của CPU phụ thuộc vào tốc độ truy nhập của bộ nhớ chính. Mặt khác, dung lượng của bộ nhớ chính cũng khá hạn chế, do vậy nếu hệ thống lưu trữ chỉ gồm bộ nhớ chính cũng sẽ hạn chế khả năng của CPU.

Để tăng tốc độ xử lý của CPU, đồng thời đảm bảo khả năng lưu trữ dữ liệu với khối lượng lớn, người ta tổ chức bộ nhớ máy tính theo kiểu hệ thống có phân cấp (hình 4.1).

Ý tưởng chính trong việc sử dụng hệ thống bộ nhớ có phân cấp là tại một thời khoảng thi các lệnh và dữ liệu được sử dụng đều nằm ở một vùng tương đối nhỏ của bộ nhớ chính. Các lệnh và dữ liệu trong vùng này luôn chuyển dịch khi thực hiện chương trình. Cơ sở của việc quản lý hệ thống nhớ có phân cấp, trong đó bao gồm nhiều loại thiết bị nhớ khác nhau là dựa trên *nguyên lý quy chiếu phân vùng*.

Kỹ thuật được sử dụng để giảm thời gian trung bình truy nhập bộ nhớ chính, là thêm một bộ nhớ có tốc độ truy nhập cao (bộ nhớ SRAM) vào hệ thống lưu trữ này. Bộ nhớ loại này được gọi là bộ nhớ *Cache* (bộ nhớ ẩn). Bộ nhớ cache được sử dụng để lưu trữ các lệnh và dữ liệu thường được sử dụng nhiều trong quá trình thực hiện chương trình. Việc quy chiếu truy nhập đến bộ nhớ chính chỉ xảy ra khi không tìm thấy lệnh hoặc dữ liệu cần có trong cache.

Giải pháp cho vấn đề hạn chế kích thước của bộ nhớ chính là sử dụng bộ nhớ ngoài (thiết bị đĩa từ) như là một thành phần của hệ thống nhớ. Bộ nhớ ngoài để lưu trữ các tài nguyên phần mềm của máy tính một cách lâu dài, ổn định. Về mặt tổ chức, bộ nhớ ngoài được tổ chức như một thiết bị vào/ra.



Hình 4.1. Mô hình phân cấp hệ thống nhớ nói chung

Trong hình 4.1:

- Đường ----- thể hiện: Cache L2 có thể đưa một phần vào bộ vi xử lý, hoặc có thể đưa toàn bộ vào bộ vi xử lý.
- Trục mũi tên từ trái sang phải thể hiện vị trí của thành phần nhớ trong hệ thống nhớ, càng xa CPU thì dung lượng càng tăng, thời gian truy nhập tăng tức là tốc độ truy nhập dữ liệu chậm, tần xuất truy nhập bởi CPU giảm, giá thành tính theo bit giảm.

4.1.2. Một số khái niệm cơ bản liên quan đến việc vào/ra dữ liệu và bộ nhớ

Một khái niệm liên quan đến bộ nhớ là đơn vị chuyên giao dữ liệu (unit of transfer). Đối với bộ nhớ trong, đơn vị chuyên giao dữ liệu là số đường dây dữ liệu vào/ra module nhớ. Nó có thể bằng độ dài của từ dữ liệu, nhưng thường là lớn hơn, nó có thể là 64, 128, 256 bit. Để làm sáng tỏ vấn đề này, ta xem xét ba khái niệm liên quan đến bộ nhớ trong.

➤ **Từ dữ liệu (Word):** Là đơn vị "đương nhiên" của tổ chức bộ nhớ. Độ dài của từ dữ liệu thường được tính bằng số bit được sử dụng để biểu diễn một số và bằng số bit tính cho độ dài của lệnh, cách tính này có nhiều ngoại lệ. Ví dụ, máy tính CRAY C90 có độ dài từ dữ liệu là 64 bit, nhưng sử dụng để diễn tả một số nguyên là 46 bit; trong máy tính VAX, các lệnh có độ dài không đều nhau (với sự khác biệt khá lớn) và đơn vị tính độ dài lệnh theo byte và một từ dữ liệu là 32 bit.

➤ **Các đơn vị được đánh địa chỉ (Addressable unit):** Trong một số hệ thống máy tính, đơn vị được đánh địa chỉ là từ dữ liệu, trường hợp này ta nói máy tính tổ chức và quản lý bộ nhớ theo từ dữ liệu. Còn phần lớn hệ thống máy tính đơn vị được đánh địa chỉ theo byte, ta nói máy tính tổ chức và quản lý bộ nhớ theo byte. Khi đơn vị được đánh địa chỉ theo byte, có sự liên quan giữa số bit địa chỉ (n) và số đơn vị được đánh địa chỉ (A) là $A = 2^n$.

➤ **Đơn vị chuyển giao dữ liệu (unit of transfer):** Đối với bộ nhớ chính, đơn vị chuyển giao dữ liệu là số bit đọc ra hoặc ghi vào bộ nhớ tại một thời điểm. Đơn vị chuyển giao dữ liệu không nhất thiết bằng một từ dữ liệu hoặc một đơn vị định địa chỉ. Đối với bộ nhớ ngoài, dữ liệu thường được chuyển giao theo các đơn vị lớn hơn rất nhiều một từ dữ liệu, đơn vị chuyển giao dữ liệu được gọi là *khối* (block).

4.1.3. Tổ chức bộ nhớ cache và phương pháp truy nhập

a) Tổng quan về tổ chức bộ nhớ cache

Bộ nhớ cache được đặt giữa CPU và bộ nhớ chính, nó đóng vai trò làm bộ nhớ đệm có tốc độ truy nhập cao. Bộ nhớ cache có dung lượng nhỏ hơn bộ nhớ chính rất nhiều lần, song nó có tốc độ truy nhập cao hơn bộ nhớ chính rất nhiều. Việc xây dựng và đưa bộ nhớ cache vào hệ thống nhớ phân cấp dựa theo hai nguyên lý sau:

- Nguyên lý cục bộ hoá về thời gian: Một lệnh hoặc dữ liệu vừa được truy nhập thì thường sẽ được truy nhập ngay sau đó.
- Nguyên lý cục bộ hoá về không gian: Một lệnh hoặc một dữ liệu vừa được truy nhập thì thường những lệnh hoặc dữ liệu lân cận sẽ được truy nhập ngay sau đó.

➤ *Hoạt động của bộ nhớ cache*

– Trao đổi giữa cache và CPU theo 1 byte hoặc một *tùy dãy* dữ liệu, trong tài liệu này, chúng ta xem xét việc trao đổi giữa cache và CPU theo 1 byte dữ liệu.

– Trao đổi giữa cache và bộ nhớ chính theo khối (block) nhớ.

➤ *Tổ chức bộ nhớ cache*

– Bộ nhớ chính được chia làm m *block* bằng nhau, ký hiệu là B_i [$i = 0 \div (m - 1)$].

– Bộ nhớ cache được chia thành k *line*, ký hiệu L_j [$j = 0 \div (k - 1)$]. Mỗi line chứa các thông tin sau: số hiệu thẻ (tag), bit cờ (F) và phần chứa dữ liệu (data). Dung lượng của phần chứa dữ liệu trong 1 line bằng dung lượng của 1 block (có thể nói, khối dữ liệu trong cache là bản sao của khối dữ liệu trong bộ nhớ chính và ngược lại).

– Tag dùng để ghi số hiệu block nhớ được nạp vào line (block nhớ có dữ liệu là bản copy của line). Bit cờ F cho biết nội dung của line đã ghi ra bộ nhớ hay chưa (trong trường hợp ghi bộ nhớ). Nếu F = 0, tức nội dung line đã ghi ra bộ nhớ chính; nếu F = 1, tức nội dung line chưa ghi ra bộ nhớ chính.

Bộ nhớ chính

0	
1	
2	
i	
$m - 1$	

Bộ nhớ cache

Tag	F	Data
0		
1		
j		
$k - 1$		

– Khi CPU truy nhập dữ liệu, dữ liệu đó đã có sẵn trong cache. Trường hợp này được gọi là trúng (hit). Ngược lại, thông tin không tìm thấy trong bộ nhớ cache thì gọi là trượt (miss), khi đó CPU phải đọc thông tin từ bộ nhớ chính và thực hiện nạp cache (nạp từ block vào line).

– Nếu xác suất xuất hiện trường hợp hit cao thì hiệu năng hệ thống cao và ngược lại, xác suất xuất hiện trường hợp miss cao thì hiệu năng hệ thống thấp.

Có hai trường hợp tổ chức cache: mỗi khối chứa 1 byte dữ liệu (hoặc một từ dữ liệu) và mỗi khối chứa nhiều byte dữ liệu (hoặc nhiều từ dữ liệu). Trong tài liệu này, chúng ta chỉ xem xét trường hợp máy tính tổ chức quản lý bộ nhớ theo byte dữ liệu (mỗi khối chứa 1 byte dữ liệu và mỗi khối chứa nhiều byte dữ liệu).

b) Trường hợp mỗi khối chứa 1 byte dữ liệu

➤ *Thao tác đọc bộ nhớ*

Giả sử địa chỉ truy nhập hợp lệ gồm n bit thì n bit này được chia làm hai phần (hình 4.2):

– n_1 bit thấp dùng để đánh số hiệu line trong cache.

– n_2 bit cao còn lại dùng để đánh số hiệu cho các block trong bộ nhớ chính.

Giả sử bộ nhớ cache có k line thì $k = 2^{n_1}$ và bộ nhớ chính có m block nhớ thì $m = 2^{n_2}$, dung lượng bộ nhớ chính cũng là 2^{n_2} .

Khi 1 byte dữ liệu cần được đọc thì CPU cung cấp địa chỉ cho bộ điều khiển bộ nhớ. Bộ điều khiển bộ nhớ tách n bit địa chỉ ra làm hai phần (hình 4.2):

– n_1 bit thấp nhất được đặt vào thanh ghi địa chỉ MAR của cache. Các bit này xác định vị trí của line cần tìm trong cache từ 0 đến ($2^{n_1} - 1$).

– $n_2 = n - n_1$ và có 2^{n_2} block, được đánh số hiệu từ 0 đến ($2^{n_2} - 1$).

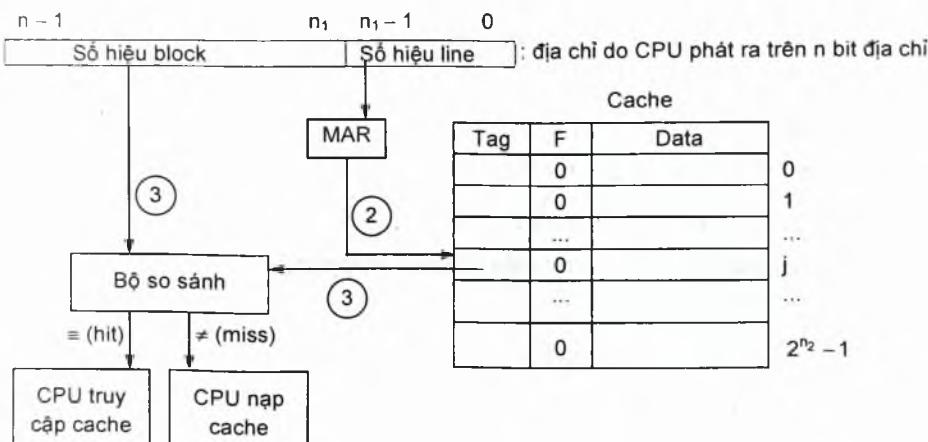
Thao tác đọc dữ liệu từ bộ nhớ được tiến hành như sau (hình 4.2):

– Bước 1: CPU kiểm tra tính hợp lệ của địa chỉ truy nhập, nếu số bit có nghĩa biểu diễn địa chỉ truy nhập lớn hơn số bit hợp lệ (lớn hơn n) thì địa chỉ này không hợp lệ, tiến trình phải dừng. Ngược lại, CPU thực hiện phát ra địa chỉ truy nhập.

– Bước 2: Bộ điều khiển cache tìm đến line dữ liệu trong cache tại vị trí có số thứ tự trùng với phần số thứ tự do n_1 bit thấp xác định.

– Bước 3: Bộ điều khiển cache đọc giá trị trên tag và kiểm tra xem giá trị này có trùng với giá trị do n_2 bit xác định hay không.

- Bước 4:
 - + Nếu trùng giá trị, đây là trường hợp cache *hit* và 1 byte dữ liệu được đọc vào CPU.
 - + Nếu không trùng giá trị, đây là trường hợp cache *miss* thì 1 byte dữ liệu phải được lấy từ bộ nhớ chính. Trong trường hợp này, nếu bit cờ F = 1 (CPU mới ghi dữ liệu ra cache, chưa ghi ra bộ nhớ) thì trước hết cần phai sao lưu dữ liệu đang có trong line tìm được vào bộ nhớ chính theo địa chỉ của nó (tại vị trí khói có số hiệu trùng với giá trị tag trong line), sửa bit cờ F = 0 (ghi nhận nội dung line trùng với nội dung của một block trên bộ nhớ chính), sau đó mới nạp dữ liệu từ block nhớ cần truy cập vào line và cũng ghi luôn số hiệu block vừa nạp vào trong tag của line. Dữ liệu được lấy từ line này cung cấp cho CPU.



Hình 4.2. Thao tác đọc bộ nhớ với mỗi khối (line) chứa 1 byte

➤ Thao tác ghi bộ nhớ

Có một vài kỹ thuật được dùng để ghi dữ liệu vào bộ nhớ chính khi thực hiện các lệnh ghi bộ nhớ: kỹ thuật ghi xuyên (write through) và kỹ thuật sao lưu (copy back).

- Ở loại cache ghi xuyên: Dữ liệu được ghi lên cả cache lẫn bộ nhớ cùng một lúc, trường hợp này không dùng đến bit cờ F và F luôn luôn = 0 (dữ liệu có trong line thì cũng được ghi luôn vào block nhớ). Kỹ thuật này làm cho thời gian ghi bộ nhớ tăng lên, hiệu suất thấp và tốn thêm đường dây dữ liệu nối trực tiếp từ CPU đến bộ nhớ.

– Ở loại cache sao lưu: Dữ liệu chỉ được ghi ra cache và bit F trên line đ~~đ~~
lập ($F = 1$), ghi nhận nội dung của line khác với nội dung của mọi block trên bộ
nhớ chính. Sau đó khi đọc line, nếu line này cần được thay thế nội dung bằng nội
dung của một block khác từ bộ nhớ chính (trong trường hợp đọc cache mà là
cache miss), thì bit cờ F được kiểm tra để xác định xem có cần thực hiện thao
tác sao lưu dữ liệu từ line này sang bộ nhớ chính hay không; nếu $F = 1$ thì cần thi
hiện sao lưu; nếu $F = 0$ thì không cần thực hiện sao lưu ($F = 0$ tức là line đã đ~~đ~~
ghi ở block nhớ). Kỹ thuật này làm tăng tốc độ thao tác với bộ nhớ và được gọi là
kỹ thuật sao lưu có dụng cờ.

Sau khi đã ghi nội dung của line ra block nhớ, thì số hiệu của block nhớ đ~~đ~~
được ghi sẽ được ghi vào trường tag của line.

c) Trường hợp mỗi khối chứa nhiều byte dữ liệu

Nếu có nhiều byte dữ liệu trong một block (hay trong một line) thì có ba
phương pháp tổ chức cache:

- Phương pháp ánh xạ trực tiếp.
- Phương pháp ánh xạ hoàn toàn.
- Phương pháp ánh xạ liên kết tập hợp.

> *Phương pháp ánh xạ trực tiếp (Direct Mapping)*

Phương pháp này quy định:

B_0 chỉ nạp vào L_0

B_1 chỉ nạp vào L_1

...

B_{k-1} chỉ nạp vào L_{k-1}

B_k chỉ nạp vào L_0

B_{k+1} chỉ nạp vào L_1

...

B_j chỉ nạp vào $L_{i=j \bmod k}$.

Xác suất xuất hiện cache hit trong kỹ thuật này là $\frac{k}{m}$.

Như đã đề cập ở trên, m là số block nhớ, k là số line trên cache.

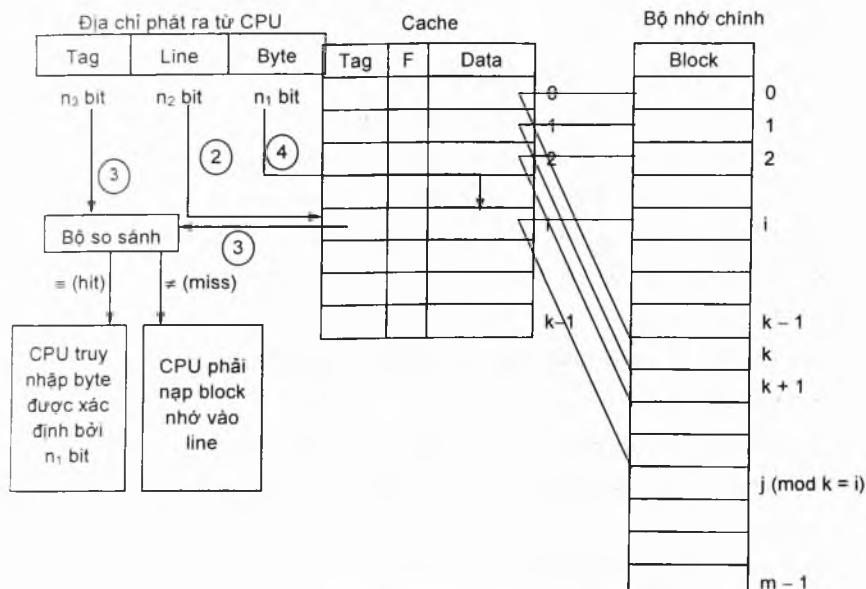
Hoạt động truy nhập: Giả sử địa chỉ truy nhập hợp lệ gồm n bit thì n bit này được chia làm ba thành phần (hình 4.3):

- n_1 bit thấp nhất đánh số hiệu byte trong line (block) cần truy cập.
- n_2 bit cao tiếp theo dùng đánh số hiệu line trong cache.
- n_3 bit cao nhất dùng đánh số hiệu block nhớ cần truy nhập.

Như vậy, dung lượng bộ nhớ chính = $2^{n_1} * 2^{n_2}$.

Dung lượng cache = $2^{n_2} * 2^{n_1}$.

Số địa chỉ bit hợp lệ là $n = n_3 + n_2 + n_1$.



Hình 4.3. Tổ chức và truy nhập cache theo kỹ thuật ánh xạ trực tiếp

• Thao tác đọc cache:

Gọi: LI (Line Index) là số hiệu line cần truy nhập;

BI (Block Index) là số hiệu block nhớ cần truy nhập.

– Bước 1: CPU kiểm tra tính hợp lệ của địa chỉ truy nhập.

+ Trường hợp 1: Nếu số bit có nghĩa biểu diễn địa chỉ truy nhập lớn hơn số bit hợp lệ (lớn hơn n), địa chỉ này không hợp lệ, tiến trình phải dừng.

+ Trường hợp 2: Nếu $LI \neq BI \text{ mod } 2^n$, thì địa chỉ này không hợp lệ, tiến trình phải dừng (số lượng line trong cache là 2^n).

Ngược lại, CPU thực hiện phát ra địa chỉ truy nhập lên Bus A.

- Bước 2: Căn cứ vào n_2 bit dùng để đánh số cho line, bộ điều khiển cache xác định được line cần truy nhập.

- Bước 3: Bộ điều khiển cache đọc giá trị tag trên tag thuộc line vừa tìm được và so sánh với giá trị do n_3 bit xác định.

+ Nếu đây là trường hợp cache *hit*, thực hiện bước 4 của thao tác.

+ Nếu là trường hợp cache *miss* thì CPU phải truy cập vào block nhớ có địa chỉ xác định bởi n_3 bit và nạp nội dung của nó vào line vừa tìm được. Trước khi nạp cache, nếu $F = 1$ thì CPU sao lưu dữ liệu đang có trong line tìm được vào bộ nhớ chính theo địa chỉ của nó (tại vị trí khối có số hiệu trùng với giá trị tag trong line), sửa $F = 0$ (ghi nhận nội dung line trùng với nội dung của một block trên bộ nhớ chính). Sau đó mới nạp dữ liệu từ block nhớ cần truy cập vào line và cũng ghi luôn số hiệu block vừa nạp vào trong tag của line. Và cuối cùng thực hiện bước 4 của thao tác.

- Bước 4: Byte dữ liệu cần đọc có địa chỉ do n_1 bit thấp xác định trong line được đọc vào CPU.

Ưu điểm của phương pháp này là số đường dây dữ liệu nối giữa cache và bộ nhớ chính là ít nhất, do vậy công nghệ chế tạo là đơn giản nhất, chi phí sản xuất và giá thành thấp.

Nhược điểm của kỹ thuật này là xác suất xuất hiện cache hit thấp, do vậy hiệu năng của hệ thống không cao. Đồng thời không dành hết không gian địa chỉ là 2^n cho bộ nhớ.

Ví dụ 1: CPU có 24 bit địa chỉ, Bộ nhớ chính 256KB chia làm 512 block nhớ, bộ nhớ cache có dung lượng 8KB. Khi CPU được lệnh phát ra địa chỉ truy nhập bộ nhớ là 308842h. Hãy trình bày chi tiết phương pháp đọc cache theo kỹ thuật ánh xạ trực tiếp cho trường hợp phát ra địa chỉ trên và cho biết địa chỉ ô nhớ cần truy cập trong bộ nhớ.

Giải:

✓ *Trình bày phương pháp đọc cache của CPU*

- Bước 1: Xác định dung lượng từng block (line) và số bit n_1 :

Dung lượng 1 block = dung lượng bộ nhớ / số block nhớ

$$= 256\text{KB} / 512 = 0,5\text{KB} = 512 \text{ byte} = 2^9 \text{ byte.}$$

Vậy số bit để đánh địa chỉ lêch trong block (line) là $n_1 = 9$.

- Bước 2: Xác định số line trong cache và số bit n_2 :

Số line trong cache = dung lượng cache / dung lượng 1 line (block)

$$= 8\text{KB} / 0,5\text{KB} = 16 \text{ (line)} = 2^4.$$

Vậy số bit để đánh số hiệu line trong cache là $n_2 = 4$.

- Bước 3: Xác định số bit để đánh số hiệu block trong bộ nhớ chính:

Số block nhớ trong bộ nhớ chính như đã cho là 512 block = 2^9 block.

Vậy số bit để đánh số hiệu block nhớ là $n_3 = 9$.

- Bước 4: Số bit địa chỉ hợp lệ là $n = n_3 + n_2 + n_1 = 9 + 4 + 9 = 22$.

- Bước 5: Thực hiện phương pháp đọc cache với giá trị địa chỉ là 308842h.

+ Đổi giá trị địa chỉ ra nhị phân $308842_h = 001100001000100001000010_b$.

+ CPU kiểm tra địa chỉ truy nhập này gồm 22 bit bằng số bit địa chỉ hợp lệ, địa chỉ hợp lệ theo trường hợp 1.

+ CPU xác định số hiệu line cần truy nhập được xác định bởi n_2 bit là 0100b (các bit được gạch chân). Vậy số hiệu line cần truy nhập là LI = 4.

+ CPU xác định số hiệu block nhớ (BI) cần truy nhập được xác định bởi các bit cao còn lại tiếp theo sau các bit n_2 , các bit này có giá trị là

$$00110000100b = 256 + 128 + 4 = 388.$$

+ CPU xác định tính hợp lệ tiếp theo của địa chỉ cần truy nhập phải thỏa mãn:

Địa chỉ hợp lệ phải thỏa mãn đẳng thức

$$LI = BI \bmod 2^4.$$

Trong bài này: $LI = 4, BI = 388, 2^4 = 16$.

Ta thấy $4 = 388 \bmod 16$ nên địa chỉ phát ra là hợp lệ.

- + CPU phát địa chỉ lên Bus A.
- + Bộ điều khiển cache thực hiện truy cập vào line 4, đọc giá trị trên tag của nó và thực hiện so sánh với $B1 = 388$. Có hai trường hợp xảy ra:
 - Nếu giá trị trên tag $\neq 388$, tức là block nhớ cần truy cập chưa được nạp vào line (cache), đây là trường hợp cache miss. CPU phải tiến hành nạp cache: trước khi nạp cache, nếu line 4 có $F = 1$, tức là nội dung nó được ghi từ CPU ra và chưa ghi ra bộ nhớ, CPU phải ghi nội dung của nó ra block nhớ có số hiệu bằng giá trị trên tag của line, sửa $F = 0$ (còn trường hợp $F = 0$ thì thao tác này không cần thiết); tiếp theo CPU ra bộ nhớ đọc block nhớ 388, nạp vào line 4, sửa tag của line 4 thành 388. Cuối cùng CPU tiến hành đọc byte trong line.

- Nếu giá trị trên tag $= 388$ thì CPU tiến hành đọc byte trong line.

- + CPU tiến hành đọc byte trong line có địa chỉ được xác định bởi n_1 bit, đó là giá trị: 001000010b (9 bit thấp nhất trong giá trị địa chỉ cần truy nhập), đây là byte có địa chỉ tương đối trong line là 66.

Xác định địa chỉ ô nhớ cần truy nhập trong bộ nhớ

- Ô nhớ cần truy nhập trong bộ nhớ chính là ô nhớ đã được nạp vào địa chỉ 66 trong line 4 của bộ nhớ cache.

– Địa chỉ vật lý của ô nhớ cần truy cập được xác định bằng phép ghép các bit biểu diễn số hiệu block cần truy cập và các bit biểu diễn địa chỉ tương đối của byte trong line. Trong đó $n_3 = 9$ bit địa chỉ xác định số hiệu block sẽ được đặt là 9 bit cao, còn $n_1 = 9$ bit xác định địa chỉ tương đối của byte trong line sẽ được ghép ở vị trí 9 bit thấp. Ta có giá trị địa chỉ vật lý của byte cần truy cập trong bộ nhớ là:

$$11\underbrace{0000}_{}1\underbrace{0000}_{}1\underbrace{0000}_{}10b = 30842h$$

n_3 n_1

• Thao tác ghi cache

Tương tự như thao tác ghi cache trong trường hợp mỗi line chứa 1 byte.

➤ Phương pháp ánh xạ liên kết hoàn toàn (Full Associative Mapping)

Phương pháp cho phép mỗi một block có thể nạp vào một line bất kỳ, do vậy một block có thể nạp vào k line.

Xác suất xuất hiện cache hit trong kỹ thuật này là:

$$\frac{m * k - m + 1}{m * k} > \frac{k}{m}.$$

Như đã đề cập ở trên, m là số block nhớ, k là số line trên cache.

Hoạt động truy nhập: Giả sử địa chỉ truy nhập hợp lệ gồm n bit thì n bit địa chỉ này được chia làm hai thành phần (hình 4.4):

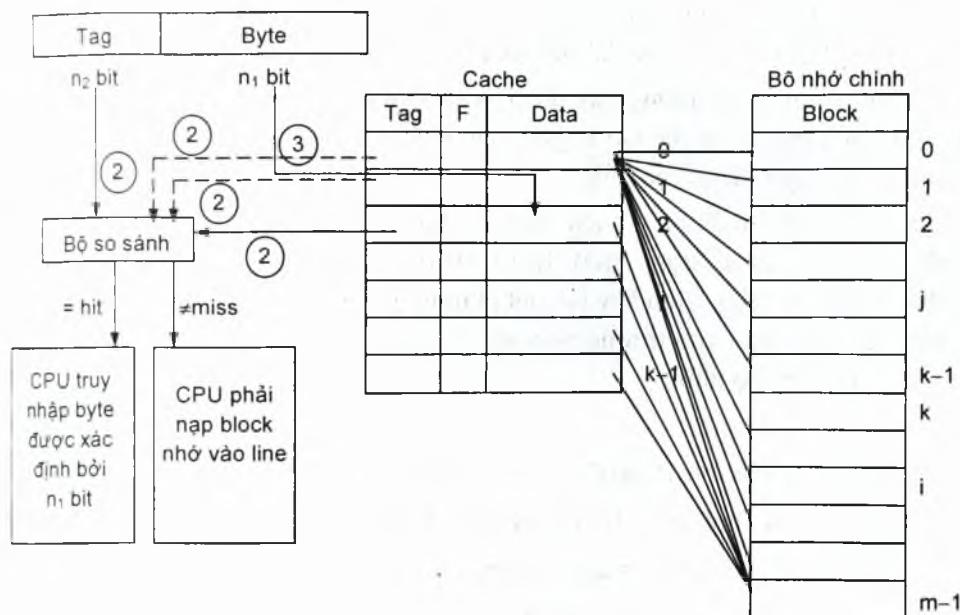
$$n = n_2 + n_1 \text{ (bit).}$$

Trong đó:

- n_1 bit thấp đánh số hiệu byte trong line (block) cần truy cập.
- n_2 bit cao dùng đánh số hiệu block nhớ cần truy nhập.

Như vậy, dung lượng bộ nhớ chính $= 2^{n_2} * 2^{n_1} = 2^n$.

Số bit địa chỉ hợp lệ là $n = n_2 + n_1$.



Hình 4.4. Tổ chức và truy nhập cache theo kỹ thuật ánh xạ liên kết hoàn toàn.

• Thao tác đọc cache

- Bước 1: CPU kiểm tra tính hợp lệ của địa chỉ truy nhập, nếu số bit có nghĩa biểu diễn địa chỉ truy nhập lớn hơn số bit hợp lệ (lớn hơn n), địa chỉ này không hợp lệ, tiến trình phải dừng.

– Bước 2: Bộ điều khiển cache đọc lần lượt các giá trị trên tag thuộc từng line trong cache và so sánh với giá trị do n_2 bit xác định.

+ Nếu đây là trường hợp cache *hit*, thực hiện bước 3 của thao tác.

+ Nếu là trường hợp cache *miss* thì CPU phải chọn một line nào đó để nạp cache, nếu không có line rỗng CPU phải chọn 1 line đang chứa block nhớ để nạp cache (đổi line), nếu bit cờ F của line được chọn bằng 1 thì CPU phải sao lưu nội dung của line ra block nhớ có số hiệu bằng giá trị trên tag của line, sửa F = 0 (ghi nhận nội dung line trùng với nội dung của một block trên bộ nhớ chính). Sau đó truy nhập vào block nhớ có địa chỉ xác định bởi n_2 bit cao nhất trong n bit địa chỉ hợp lệ và nạp nội dung của nó vào line được chọn, sửa tag của line bằng số hiệu block nhớ vừa nạp. Và cuối cùng thực hiện bước 3 của thao tác.

– Bước 3: Byte dữ liệu cần đọc có địa chỉ do n_1 bit thấp xác định trong line được đọc vào CPU.

Ưu điểm của phương pháp này là xác suất xuất hiện cache hit cao do một block có thể nạp vào k line bất kỳ nên hiệu năng của hệ thống cao.

Nhược điểm là số đường dây dữ liệu nối giữa cache và bộ nhớ chính là lớn nhất, do vậy công nghệ chế tạo là phức tạp nhất và gặp nhiều khó khăn nhất, chi phí sản xuất và giá thành cao nhất.

Ví dụ 2: CPU có 24 bit địa chỉ, bộ nhớ chính 256KB chia làm 512 block nhớ, bộ nhớ cache có dung lượng 8KB. Khi CPU được lệnh phát ra địa chỉ truy nhập bộ nhớ là 30815h. Hãy trình bày chi tiết phương pháp đọc cache theo kỹ thuật ánh xạ liên kết hoàn toàn cho trường hợp phát ra địa chỉ trên và chỉ ra địa chỉ ô nhớ cần truy cập trong bộ nhớ.

Giải:

☞ *Trình bày phương pháp đọc cache của CPU*

– Bước 1: Xác định dung lượng từng block (line) và số bit n_1 :

$$\begin{aligned}\text{Dung lượng 1 block} &= \text{dung lượng bộ nhớ} / \text{số block nhớ} \\ &= 256\text{KB} / 512 = 0,5\text{KB} = 512 \text{ byte} = 2^9 \text{ byte.}\end{aligned}$$

Vậy số bit để đánh địa chỉ lêch trong block (line) là $n_1 = 9$.

– Bước 2: Xác định số line trong cache:

$$\begin{aligned}\text{Số line trong cache} &= \text{dung lượng cache} / \text{dung lượng 1 line (block)} \\ &= 8\text{KB} / 0,5\text{KB} = 16 (\text{line}).\end{aligned}$$

- Bước 3: Xác định số bit để đánh số hiệu block trong bộ nhớ chính.
 - + Số block nhớ trong bộ nhớ chính như đã cho là $512 \text{ block} = 2^9 \text{ block}$.
 - + Số bit để đánh số hiệu block nhớ là $n_2 = 9$.
 - Bước 4: Số bit địa chỉ hợp lệ là $n = n_2 + n_1 = 9 + 9 = 18$.
 - Bước 5: Thực hiện phương pháp đọc cache với giá trị địa chỉ là $30815h$:
 - + Đổi giá trị địa chỉ ra nhị phân $30815h = 00110000100000010101b$.
 - + CPU xác định tính hợp lệ của địa chỉ cần truy nhập: Địa chỉ truy nhập này gồm 18 bit, bằng số bit địa chỉ hợp lệ, nên địa chỉ phát ra là hợp lệ. CPU phát địa chỉ lên Bus A.
 - + Bộ điều khiển cache xác định số hiệu block nhớ cần truy nhập được xác định bởi các bit cao tính từ bit 9 (các bit được gạch chân), BI = $110000100b = 388$.
 - + Bộ điều khiển cache thực hiện truy cập vào lần lượt từng line trong 16 line, đọc giá trị trên tag của nó và thực hiện so sánh với BI. Có hai trường hợp xảy ra:
 - Nếu không tìm thấy giá trị trùng nhau, tức là block nhớ cần truy cập chưa được nạp vào line (cache), đây là trường hợp cache miss. CPU phải tiến hành nạp cache: CPU phải chọn một line nào đó để nạp cache, nếu không có line rỗng CPU phải chọn 1 line đang chứa block nhớ để nạp cache (đổi line), nếu bit cờ F của line được chọn bằng 1 thì CPU phải sao lưu nội dung của line ra block nhớ có số hiệu bằng giá trị trên tag của line, sửa F = 0 (ghi nhận nội dung line trùng với nội dung của một block trên bộ nhớ chính). Sau đó truy nhập vào block nhớ có số hiệu 388 và nạp nội dung của nó vào line được chọn, ghi giá trị 388 vào tag của line vừa nạp. Cuối cùng CPU tiến hành đọc byte trong line.
 - Nếu tìm thấy giá trị trùng nhau thì đó chính là line cần truy cập, CPU tiến hành đọc byte trong line.
 - + CPU tiến hành đọc byte trong line có địa chỉ được xác định bởi n_1 bit, đó là giá trị: $000010101b$ (9 bit thấp nhất trong giá trị địa chỉ cần truy nhập), đây là byte có địa chỉ tương đối trong line là 21.
- ☞ Xác định địa chỉ ô nhớ cần truy nhập trong bộ nhớ*
- Trong kỹ thuật ánh xạ liên kết hoàn toàn, địa chỉ vật lý của ô nhớ cần truy cập chính là địa chỉ truy cập được phát ra vì thành phần địa chỉ phát ra chỉ có hai

phần, đó là số hiệu block cần truy nhập (các bit cao) và số hiệu byte cần đọc trong block (line). Vậy địa chỉ vật lý ở nhớ cần truy cập trong bộ nhớ là 30815h.

• Thao tác ghi cache

Tương tự như thao tác ghi cache trong trường hợp mỗi line chứa 1 byte.

➤ Phương pháp ánh xạ liên kết tập hợp (Set Associative Mapping)/ Kỹ thuật tập liên hợp

Phương pháp này tổ chức cache thành q set, mỗi set gồm p line ($p = k/q$) và số lượng line trong các set bằng nhau. Ký hiệu set là S_j ($j = 0 \div q - 1$).

Phương pháp này quy định:

B_0 chỉ nạp vào S_0

B_1 chỉ nạp vào S_1

...

B_{q-1} chỉ nạp vào S_{q-1}

B_q chỉ nạp vào S_0

B_{q+1} chỉ nạp vào S_1

...

B_i chỉ nạp vào $S_{j = i \bmod q}$

Xác suất xuất hiện cache hit trong kỹ thuật này là

$$\frac{m * k - m * q + q * k}{m * k}$$

Như đã đề cập ở trên, m là số block nhớ, k là số line trên cache.

Ta thấy, xác suất xuất hiện cache hit của phương pháp này lớn hơn phương pháp ánh xạ trực tiếp, song nhỏ hơn phương pháp ánh xạ liên kết hoàn toàn.

Hoạt động truy nhập: Địa chỉ truy nhập hợp lệ gồm n bit và được chia làm ba thành phần (hình 4.5):

- n_1 bit thấp nhất đánh số hiệu byte trong line (block) cần truy cập.
- n_2 bit cao tiếp theo dùng đánh số hiệu set trong cache.
- n_3 bit cao nhất dùng đánh số hiệu block nhớ cần truy nhập.

Như vậy, dung lượng bộ nhớ chính = $2^{n_1} * 2^{n_2}$.

Số bit địa chỉ hợp lệ do CPU phát ra là $n = n_3 + n_2 + n_1$.

• Thao tác đọc cache

Gọi: SI (Set Index) là số hiệu set cần truy nhập;

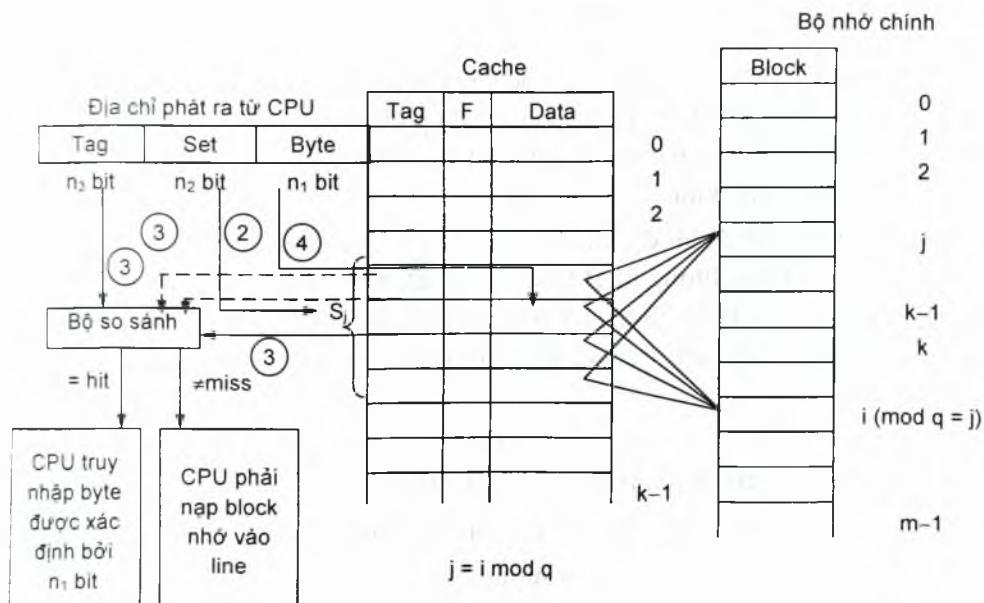
BI (Block Index) là số hiệu block nhớ cần truy nhập.

- Bước 1: CPU kiểm tra tính hợp lệ của địa chỉ truy nhập.

- Trường hợp 1: Nếu số bit có nghĩa biểu diễn địa chỉ truy nhập lớn hơn số bit hợp lệ (lớn hơn n), địa chỉ này không hợp lệ, tiến trình phải dừng.

+ Trường hợp 2: Nếu $SI \neq (BI \bmod 2^{n_2})$ thì địa chỉ này không hợp lệ, tiến trình phải dừng (số lượng set trong cache là 2^{n_2}).

Ngược lại, CPU thực hiện phát ra địa chỉ truy nhập lên Bus A, sang bước 2.



Hình 4.5. Tổ chức và truy nhập cache theo kỹ thuật ánh xạ liên kết tập hợp

- Bước 2: Căn cứ vào n_2 bit dùng để đánh số hiệu cho set, bộ điều khiển cache xác định được set cần truy nhập.

- Bước 3: Bộ điều khiển cache đọc lần lượt từng giá trị trên tag thuộc từng line trong set vừa tìm được và so sánh với giá trị BI do n_3 bit xác định.

- + Nếu đây là trường hợp cache *hit*, thực hiện bước 3 của thao tác.
- + Nếu là trường hợp cache *miss* thì CPU phải tiến hành nạp cache: CPU phải chọn một line nào đó trong set để nạp cache, nếu không có line rỗng CPU phải chọn 1 line đang chứa block nhớ để nạp cache, nếu bit cờ F của line **được chọn** bằng 1 thì CPU phải sao lưu nội dung của line ra block nhớ có số hiệu **bằng giá trị** trên tag của line, sửa F = 0 (ghi nhận nội dung line trùng với nội dung của một block trên bộ nhớ chính). Sau đó truy nhập vào block nhớ có **địa chỉ xác định** bởi n₁ bit cao trong n bit và nạp nội dung của nó vào line được chọn, sửa tag thành số hiệu block vừa nạp. Và cuối cùng thực hiện bước 4 của thao tác.
- Bước 4: Từ dữ liệu hoặc byte dữ liệu cần đọc có địa chỉ do n₁ bit **thấp xác định** trong line được đọc vào CPU.

Ưu điểm và nhược điểm của phương pháp này là trung hoà của hai phương pháp trên. Do một block chỉ có thể nói vào $\frac{k}{q}$ line trong một set nhất định nào đó, nên số đường dây dữ liệu nối giữa cache và bộ nhớ chính giảm đi rất nhiều so với kỹ thuật ánh xạ liên kết hoàn toàn. Do vậy, độ phức tạp về công nghệ có thể chấp nhận được và xác suất xuất hiện cache hit tuy không phải là cao nhất, song là trường hợp chấp nhận được.

Ví dụ 3: CPU có 24 bit địa chỉ, Bộ nhớ chính 256KB chia làm 512 block nhớ, bộ nhớ cache có dung lượng 8KB chia làm 4 set. Khi CPU được lệnh phát ra địa chỉ truy nhập bộ nhớ là 73426h. Hãy trình bày chi tiết phương pháp đọc cache cho trường hợp phát ra địa chỉ trên và địa chỉ vật lý của ô nhớ cần truy nhập trong bộ nhớ.

Giải:

☞ Trình bày phương pháp đọc cache của CPU

- Bước 1: Xác định dung lượng từng block (line) và số bit n₁:

Dung lượng 1 block = dung lượng bộ nhớ / số block nhớ

$$= 256\text{KB} / 512 = 0,5\text{KB} = 512 \text{ byte} = 2^9 \text{ byte.}$$

Vậy số bit để đánh địa chỉ lêch trong block (line) là n₁ = 9.

- Bước 2: Xác định số line trong một set và số bit n₂ biểu diễn số hiệu set:

Số line trong cache = dung lượng cache / dung lượng 1 line (block)
 $= 8\text{KB} / 0,5\text{KB} = 16 (\text{line}) = 2^4.$

$$\begin{aligned} \text{Số line trong 1 set} &= \text{số line trong cache} / \text{số set trong cache} \\ &= 16 / 4 = 4 \text{ line} \end{aligned}$$

Số bit để đánh số hiệu set trong cache là $n_2 = 2$ (số set trong cache = $4 = 2^2$).

- Bước 3: Xác định số bit để đánh số hiệu block trong bộ nhớ chính.

Số block nhớ trong bộ nhớ chính như đã cho là 512 block = 2^9 block.

Số bit để đánh số hiệu block nhớ là $n_3 = 9$.

- Bước 4: Số bit địa chỉ hợp lệ là $n = n_3 + n_2 + n_1 = 9 + 2 + 9 = 20$ (bit).

- Bước 5: Thực hiện phương pháp đọc cache với giá trị địa chỉ là 73426h.

+ Đôi giá trị địa chỉ ra nhị phân 73426 h = 01110011010000100110b.

+ CPU xác định tính hợp lệ của địa chỉ cần truy nhập: Địa chỉ truy nhập này gồm 19 bit, nhỏ hơn số bit địa chỉ hợp lệ, địa chỉ hợp lệ theo trường hợp 1.

+ CPU xác định số hiệu set cần truy nhập được xác định bởi 2 bit là 10b (các bit được gạch chân). Vậy số hiệu set cần truy nhập là SI = 2.

+ CPU xác định số hiệu block nhớ cần truy nhập được xác định bởi các bit cao còn lại tiếp theo sau các bit n_2 , các bit này có giá trị là BI = 11100110b = 230.

+ CPU xác định tính hợp lệ tiếp theo của địa chỉ cần truy nhập, phải thỏa mãn:

Địa chỉ hợp lệ phải thỏa mãn điều kiện

$$SI = BI \bmod 4 \text{ (4 set).}$$

Ta thấy: $2 = 230 \bmod 4$, nên địa chỉ phát ra là hợp lệ. CPU phát địa chỉ lên Bus A.

+ Bộ điều khiển cache thực hiện truy cập vào set 2, thực hiện truy cập vào lần lượt từng line trong 4 line thuộc set 2, đọc giá trị trên tag của nó và thực hiện so sánh với BI, có 2 trường hợp xảy ra:

- Nếu không tìm thấy giá trị trùng nhau, tức là block nhớ cần truy cập chưa được nạp vào line (cache), đây là trường hợp cache miss. CPU phải tiến hành nạp cache: CPU phải chọn một line nào đó trong set 2 để nạp cache, nếu không có line rỗng, CPU phải chọn 1 line đang chứa block nhớ để nạp cache, nếu bit cờ F của line được chọn bằng 1 thì CPU phải sao lưu nội dung của line ra block nhớ có số hiệu bằng giá trị trên tag của line, sửa F = 0 (ghi nhận nội dung line trùng với nội dung của một block trên bộ nhớ chính). Sau đó truy nhập vào block nhớ có số

hiệu 230 và nạp nội dung của nó vào line được chọn, sửa nội dung tag thành 23. Cuối cùng CPU tiến hành đọc byte trong line.

- Nếu tìm thấy giá trị trùng nhau, thì đó chính là line cần truy cập, CPU tiến hành đọc byte trong line.

+ CPU tiến hành đọc byte trong line có địa chỉ được xác định bởi n_1 bit, đó là giá trị: 000100110b (9 bit thấp nhất trong giá trị địa chỉ cần truy nhập), đây là byte có địa chỉ tương đối trong line là 38.

☞ Xác định địa chỉ ô nhớ cần truy nhập trong bộ nhớ

- Ô nhớ cần truy nhập trong bộ nhớ chính là ô nhớ đã được nạp vào địa chỉ 38 trong line có giá trị tag trùng với số hiệu block cần truy nhập của bộ nhớ cache.

- Địa chỉ vật lý của ô nhớ cần truy cập được xác định bằng phép ghép các bit biểu diễn số hiệu block cần truy cập và các bit biểu diễn địa chỉ tương đối của byte trong line, trong đó $n_3 = 9$ bit địa chỉ xác định số hiệu block sẽ được đặt là 9 bit cao, còn $n_1 = 9$ bit xác định địa chỉ tương đối của byte trong line sẽ được ghép ở vị trí 9 bit thấp. Ta có giá trị địa chỉ vật lý của byte cần truy cập trong bộ nhớ là:

$$011100110000100110b = 1CC26h$$

• Thao tác ghi cache

Tương tự như thao tác ghi cache trong trường hợp mỗi line chứa 1 byte.

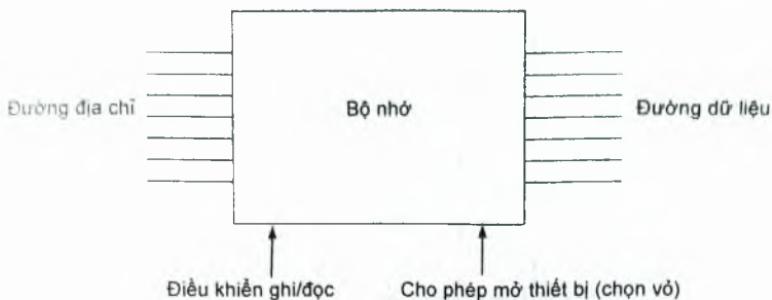
4.1.4. Các kiểu bộ nhớ

Bộ nhớ được chia thành nhiều loại: bộ nhớ bán dẫn, bộ nhớ từ, bộ nhớ quang, bộ nhớ quang từ. Trong phần này chúng ta chỉ xem xét loại bộ nhớ bán dẫn.

a) Tổng quan về bộ nhớ bán dẫn

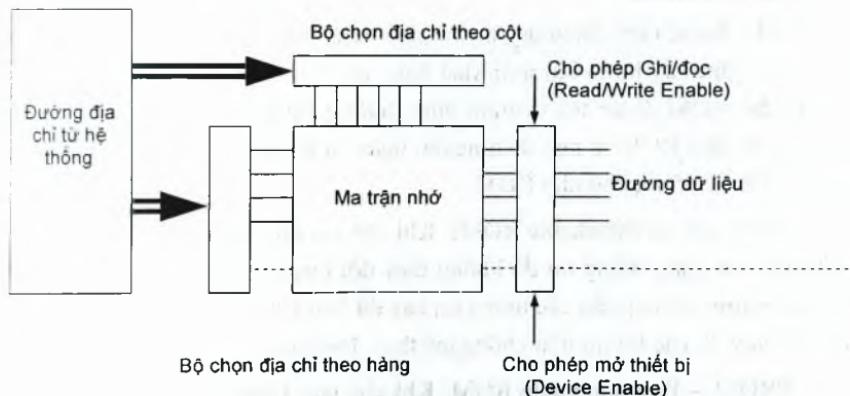
Bộ nhớ bán dẫn thường được sử dụng làm bộ nhớ trong của máy tính. Bộ nhớ trong được nối với các bộ phận khác trong máy tính thông qua Bus dữ liệu và Bus địa chỉ. Điều khiển các mạch nhớ có đường điều khiển cho phép mở (Device enable). Ngoài ra còn có thêm đường điều khiển cho phép đọc/ghi (Read/write enable).

Hình 4.6 giới thiệu sơ đồ khái măch nhớ cơ sở. Nếu số đường dây địa chỉ là n thì số lượng ô nhớ cực đại sẽ là 2^n .



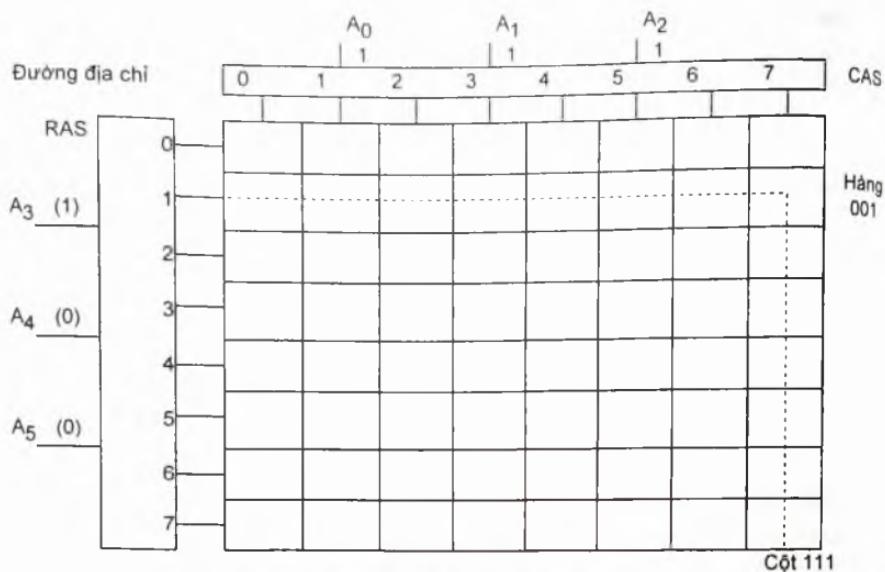
Hình 4.6. Sơ đồ khái măch nhớ cơ sở

Thông thường các ngăn nhớ trong bộ nhớ được bố trí ở dạng ma trận. Để xác định vị trí nhớ hay phần tử nhớ cần có mạch giải mã địa chỉ. Mạch này gồm hai phần: Mạch chọn địa chỉ hàng RAS (Row Address Selector) và mạch chọn địa chỉ cột CAS (Column Address Selector). Đường dây Device Enable cho phép mở các mạch điện lõi ra của bộ nhớ theo 3 trạng thái còn đường dây Read/write enable sẽ xác định dạng thao tác là đọc hay là ghi vào bộ nhớ (hình 4.7).



Hình 4.7. Sơ đồ khái măch nhớ ma trận

Hình 4.8 biểu diễn sơ đồ bộ nhớ có 6 đường địa chỉ được tổ chức dưới dạng ma trận gồm các hàng và cột, nó cho thấy cách xác định ngăn nhớ theo địa chỉ hàng và cột.



Hình 4.8. Sơ đồ khái niệm địa chỉ ngăn nhớ 001111

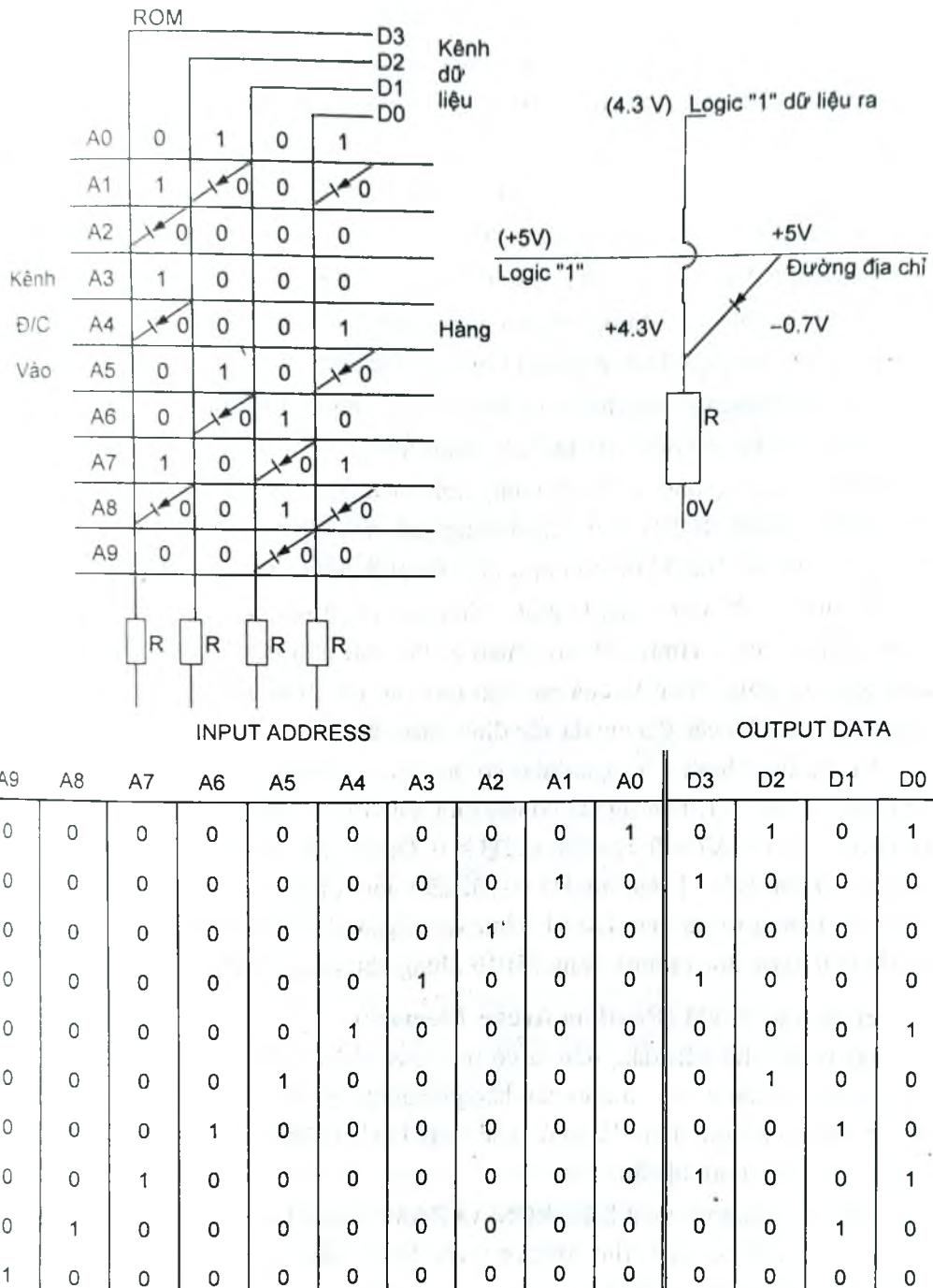
Bộ nhớ bán dẫn được chia thành hai loại là ROM và RAM.

b) Bộ nhớ ROM

ROM – Read Only Memory theo nghĩa tiếng Việt là bộ nhớ chỉ đọc, bộ nhớ này có tên như vậy là do khi mới xuất hiện, nó thực sự là bộ nhớ chỉ đọc. Ngày nay bộ nhớ ROM đã có thể đọc/ghi bình thường bằng điện và đặc trưng cơ bản nhất của bộ nhớ ROM là mất điện nguồn nuôi thì không mất thông tin. Theo lịch sử phát triển, có 5 loại bộ nhớ ROM.

– ROM mặt nạ (Maskable ROM): Khi chế tạo nó được ghi luôn thông tin và vĩnh viễn nội dung thông tin đó không thay đổi được. Là loại ROM do nhà máy khi xuất xưởng đã nạp sẵn các thông tin hay dữ liệu lên. Khi đã được chương trình hoá như vậy thì các bit dữ liệu không thể thay đổi được nữa.

– PROM – Programmable ROM: Khi chế tạo, không có thông tin, song có thể ghi được thông tin vào một lần bằng thiết bị chuyên dùng, nếu ghi sai thì bị loại bỏ. Là loại mạch nhớ ROM mà người sử dụng có thể nạp chương trình vào được nhờ sử dụng một thiết bị đốt đặc biệt gọi là thiết bị đốt PROM. PROM thường được làm bằng transistor lưỡng cực hoặc MOSFET.



Hình 4.9. ROM mặt nạ chế tạo bằng ma trận diot và bảng sự thật của nó

- EPROM – Erasable PROM: Là loại PROM, song có thể xoá được. Việc lưu trữ thông tin được thực hiện bằng các điện tích bãy ở cực gate của MOSFET, còn xoá nội dung nhớ người ta chiêu tia cực tím với cường độ mạnh.

+ Ghi thông tin được nhiều lần bằng thiết bị chuyên dùng.

+ Trước khi ghi phải xoá nội dung cũ bằng tia cực tím.

- EEPROM – Electrically EPROM:

+ Giống RAM nhưng mất nguồn thì không mất thông tin.

+ Cần đọc hay ghi thông tin đều được ngay. Nó ghi thông tin theo từng byte.

Thông tin ghi lần cuối không bị mất khi mất điện nguồn nuôi.

- Flash Memory (bộ nhớ tia chớp): Giống như EEPROM, nó chỉ cho phép đọc cả khối, ghi cả khối, chế tạo với dung lượng lớn – ngày nay ít sử dụng. Có một thời kỳ làm bộ nhớ ngoài cho máy tính xách tay. Ngày nay được dùng làm bộ nhớ ngoài (Flash ROM) như vẫn thường gọi là ổ USB, thẻ nhớ và làm bộ nhớ trong của một số thiết bị điện tử như điện thoại di động, bút ghi âm,...

Bộ nhớ ROM thực chất là một tổ hợp nối ghép sẵn các mạch điện để có các trạng thái cố định. Hình 4.9 giới thiệu ROM mặt nạ được tạo thành do ma trận điol hàng và bảng chân lý của ma trận điol, nó cho thấy nội dung cho trước của từng ngăn nhớ tại các địa chỉ đã xác định khác nhau.

Ví dụ, theo hình 4.9, ngăn nhớ có địa chỉ 1, A9 đến A1 tín hiệu địa chỉ vào đều bằng 0, A0 = 1. Đường D3 có các điol nối với các đường địa chỉ A1, A3, A7, song A1 = A3 = A7 = 0 nên đầu ra D3 = 0. Đường D2 có các điol nối với A0 = 1 và A5 = 0 nên D2 = 1. Đường D1 có các điol nối với A6 = 0 và A8 = 0 nên đầu ra D1 = 0. Tương tự, ta thấy D0 = 1. Như vậy, ngăn nhớ có địa chỉ 0000000001b sẽ có dữ liệu được đọc ra luôn bằng 0101b (đúng như bảng sự thật).

c) Bộ nhớ RAM (Random Access Memory)

Đã là bộ nhớ bán dẫn, đều là bộ nhớ truy nhập ngẫu nhiên, tức là thời gian truy nhập trên tất cả các ô nhớ đều bằng nhau khi có địa chỉ phát ra. Loại bộ nhớ RAM này có nhược điểm là thuộc loại "bay hơi" (volatile), thông tin sẽ bị mất đi khi nguồn điện nuôi bị cắt.

Thời kỳ đầu mới xuất hiện, ROM và RAM có sự khác biệt lớn nhất của RAM so với ROM là Read Write Memory, tức là bộ nhớ ghi/đọc được và mất điện nguồn nuôi thì mất thông tin.

RAM có hai loại chính là SRAM và DRAM.

➤ SRAM (Static RAM – RAM tĩnh)

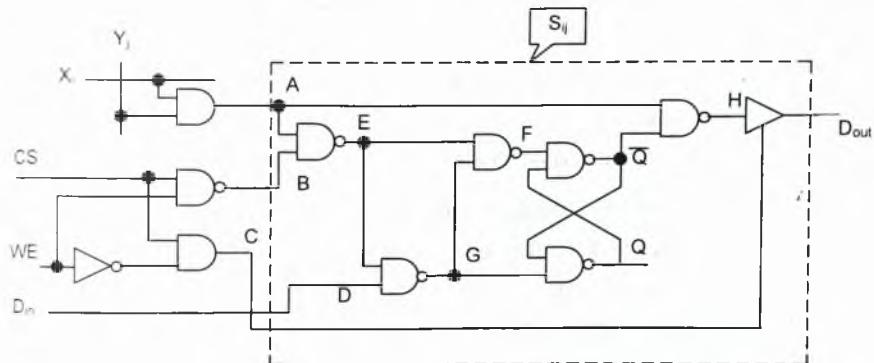
Bộ nhớ được xây dựng từ các phần tử nhớ một bit SRAM. Mỗi bit SRAM được xây dựng trên các mạch điện tử flip-flop:

- Thông tin ổn định;
- Tốc độ nhanh;
- Dung lượng IC nhỏ;
- Giá thành cao.

Trong máy tính nó được dùng làm cache L1 (sơ cấp), L2 (thứ cấp).

Sau đây là ví dụ cấu tạo phần tử nhớ 1 bit của bộ nhớ SRAM. Phần tử nhớ 1 bit SRAM là phần tử nhớ bán dẫn có tính chất là khi ta thiết lập cho phần tử nhớ 1 bit một giá trị 0 hoặc 1 thì nó nhớ mãi giá trị đó cho đến khi ta thiết lập cho nó một giá trị mới.

Hình 4.10 là sơ đồ mạch điện của một phần tử nhớ RAM tĩnh một bit S_{ij} cùng các mạch điện cũng như các tín hiệu để điều khiển sự hoạt động của nó.



Hình 4.10. Mạch điện tử của một phần tử nhớ SRAM 1 bit S_{ij}

Các đường dây truyền tín hiệu trên sơ đồ mạch có ý nghĩa như sau:

- D_{in} : Tín hiệu thông tin (dữ liệu) đầu vào.
- D_{out} : Tín hiệu thông tin (dữ liệu) đầu ra.
- X_i , Y_j : Là các dây địa chỉ, nếu ta tổ chức một mảng các phần tử nhớ hai chiều thì phần tử S_{ij} nằm ở hàng i cột j và X_i sẽ được nối với hàng i của ma trận, còn Y_j sẽ được nối với cột j của ma trận.
- CS (Chip Select): Tín hiệu chọn chip (tín hiệu chọn vỏ), đôi khi được ký hiệu là CE (Chip Enable). Khi có nhiều chip nhớ RAM cùng nối với đường tín

hiệu chung (D_{in} , D_{out}) thì đầu vào CS có nhiệm vụ chọn xem chip RAM nào được truyền thông với Bus dữ liệu.

Với các phần tử nhớ 1 bit có cấu tạo như trên, ta rất dễ kết hợp chúng lại với nhau để tạo nên các bộ nhớ có ngăn nhớ với kích thước mong muốn (8 bit, 1 word). Vẫn đề phôi ghép để xây dựng bộ nhớ từ các phần tử nhớ 1 bit sẽ được đề cập ở phần sau.

+ Điều kiện ghi: $CS = 1$, $X_i = 1$, $Y_j = 1$, $WE = 1$, khi đó Q sẽ **được thiết lập** bằng D_{in} , đồng thời H và D_{out} ở trạng thái trờ kháng cao.

+ Điều kiện đọc: $CS = 1$, $X_i = 1$, $Y_j = 1$, $WE = 0$, khi đó D_{out} sẽ **nhận** giá trị bằng Q và Q không phụ thuộc vào D_{in} .

- WE (Write Enable): Tín hiệu cho phép ghi.

+ Khi $WE = 1$ cho phép ghi thông tin D_{in} vào phần tử nhớ, khi đó:

- $WE = 1$, $\overline{WE} = 0$, $CS = 1 \Rightarrow C = 0$, lúc này đầu ra D_{out} ở trạng thái trờ kháng cao, có thể nói đường dây ra D_{out} ngắt mạch với phần tử nhớ;

- $X_i = 1$, $Y_j = 1 \Rightarrow A = 1$;

- $WE = 1$, $CS = 1 \Rightarrow B = 0 \Rightarrow E = 1$;

- Nếu $D_{in} = 0 \Rightarrow G = 1$:

- * $G = 1$, $E = 1 \Rightarrow F = 0 \Rightarrow \overline{Q} = 1$,

- * $G = 1$, $\overline{Q} = 1 \Rightarrow Q = 0$ hay $Q = D_{in}$;

- Nếu $D_{in} = 1$ với $E = 1 \Rightarrow G = 0$:

- * $G = 0$, $E = 1 \Rightarrow F = 1$,

- * $G = 0 \Rightarrow Q = 1$ hay $Q = D_{in}$,

- * $Q = 1$, $F = 1 \Rightarrow \overline{Q} = 0$.

Vậy ta thấy trong trường hợp này Q luôn = D_{in} .

+ Khi $WE = 0$ cho phép đọc thông tin từ phần tử nhớ, khi đó:

- $WE = 0$, $\overline{WE} = 1$, $CS = 1 \Rightarrow C = 1$, lúc này đầu ra D_{out} luôn bằng H ;

- $X_i = 1$, $Y_j = 1 \Rightarrow A = 1$;

- $WE = 0$, $CS = 1 \Rightarrow B = 1 \Rightarrow E = 0 \Rightarrow G = 1$, $F = 1$.

- Giả sử $Q = 1 \Rightarrow \overline{Q} = F$ NAND $Q = 1$ NAND $1 = Q \Rightarrow Q = 1$. (1*)

- Giả sử $Q = 0 \Rightarrow \overline{Q} = 1 \Rightarrow Q = \overline{Q}$ NAND $G = 1$ NAND $1 = Q$ (2*)

Từ (1*) và (2*) suy ra Q không đổi.

Vậy trong trường hợp này Q không đổi trạng thái (giá trị tín hiệu đã ghi được bảo toàn), tức là không bị phụ thuộc vào D_{in} .

➤ DRAM (Dynamic RAM – RAM động)

Được xây dựng trên cơ sở nhớ là các tụ điện, khi giữa hai cực của tụ điện có điện áp (tích điện) thì bit thông tin là 1, theo thời gian mức điện áp này bị giảm dần do có sự rò điện tích trên tụ. Do vậy DRAM là loại phải được "làm tươi" (refresh) theo chu kỳ, tức là phải nạp lại các dữ liệu đang được lưu giữ theo từng chu kỳ. Làm tươi có thể thực hiện bằng cách nhắc lại thao tác đọc hoặc ghi hoặc cũng có thể bằng những thao tác đặc biệt khác. Mật độ phần tử nhớ rất cao nên giá thành của DRAM tính theo dung lượng trở nên khá rẻ so với SRAM. Nhược điểm là tốc độ hoạt động bị chậm hơn. DRAM là loại thông dụng nhất trong các hệ vi tính hiện nay. Và loại này có các đặc tính sau:

- Thông tin không ổn định, do đó cần có mạch làm tươi để ổn định (mạch bù đắp điện áp bị sụt).
- Tốc độ chậm.
- Dung lượng IC lớn.
- Giá thành thấp.

DRAM được dùng để thiết kế ra bộ nhớ chính, nó có hai loại phổ biến là:

- EDO–DRAM (Extended Data Output DRAM) được cải tiến ở mạch ra dữ liệu để truy nhập nhanh.

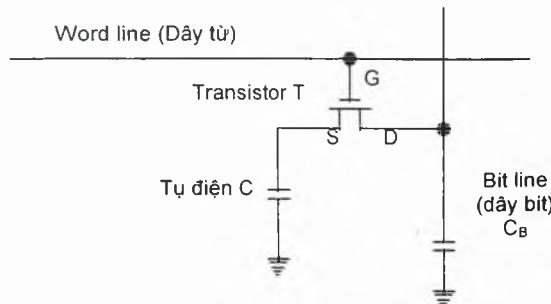
- SDRAM (Synchronous DRAM): Loại này có chân để đưa tín hiệu xung Clock vào nhằm tạo ra sự đồng bộ theo đầu vào của tín hiệu Clock, do đó phân mốc thực hiện đồng thời kiểu Pipeline.

Sau đây là các ví dụ về cấu tạo phần tử nhớ 1 bit của bộ nhớ DRAM. Như ta đã biết, các phần tử SRAM được cấu tạo từ các flip-flop. Một phần tử SRAM 1 bit như hình 4.10 được cấu tạo bởi 6 cổng NAND và một bộ đếm 3 trạng thái. Nếu mỗi cổng hoặc bộ đếm được tạo bởi 3 transistor thì cần đến 19 transistor để nhớ được 1 bit thông tin. Ngày nay người ta sử dụng các phần tử SRAM đơn giản hơn, chứa từ 1 đến 6 transistor.

Để tích hợp được lớn nhất số phần tử trong một vi mạch, mỗi phần tử nhớ phải được chế tạo sao cho đơn giản nhất. Phần tử nhớ DRAM mà chúng ta tìm

hiệu dưới đây chỉ cần 1 transistor cho 1 bit dữ liệu. Chính vì vậy có thể tích hợp vào vi mạch các bit nhớ với mật độ rất cao, giá thành rẻ. Trong phần tử nhớ này người ta thay flip-flop bằng một tụ điện C, giá trị nhớ trong phần tử nhớ này chính là mức điện áp tích nạp trên tụ điện. Ta có thể sử dụng trạng thái tự được nạp, tức trên tụ điện C có một điện áp lớn hơn một giá trị nhất định nào đó biểu diễn giá trị 1 của bit, còn trạng thái không được nạp biểu diễn giá trị 0 của bit. Hình 4.11 minh họa nguyên lý cấu tạo của một phần tử nhớ động 1 bóng (1 transistor); hình 4.12 minh họa nguyên lý cấu tạo và hoạt động của phần tử nhớ động 3 bóng; hình 4.13 minh họa nguyên lý cấu tạo và hoạt động của phần tử nhớ động 4 bóng.

- *Phần tử DRAM 1 transistor*



Hình 4.11. Phần tử nhớ động MOS 1 bóng

Bộ nhớ DRAM được tổ chức thành một ma trận nhớ (thường là ma trận vuông), trong đó dây từ (word line) là một trong các dây hàng của ma trận; còn dây bit (bit line) là một trong các dây cột. Transistor T là một transistor trường (Field Effect Transistor) đóng vai trò một chuyển mạch điện tử. T có 3 cực là cực công G (gate), cực máng D (Drain) và cực nguồn S (Source), trong đó G là cực điều khiển. D sẽ được nối với S khi G có mức điện áp cao ($=1$) so với S, ngược lại thì điện trở giữa D và S sẽ rất lớn.

- *Việc ghi (Write)*

Khi dây từ có mức tích cực ($=1$), T ở trạng thái mở, nối tụ C với dây bit. Nếu thao tác là ghi thì giá trị cần ghi phải đặt trên dây bit. Nếu giá trị đó là 1 thì tụ C sẽ được nạp tới điện áp ứng với giá trị 1 trên dây bit, còn nếu giá trị đó là 0 thì tụ C sẽ bị phóng hết điện, tức có giá trị bằng 0.

- Việc đọc (Read)

Việc đọc phức tạp hơn việc ghi một chút do điện tích trên tụ C ứng với giá trị cần đọc rất nhỏ. Điện áp trên tụ C điều khiển mức điện áp trên C_B , tức đưa ra dữ liệu. Giả sử trên C có mức điện áp V, thì sau khi đọc mức điện áp trên dây bit là

$$V_B = V_X \frac{C}{C + C_B}.$$

Do nhiều phần tử nối vào dây bit nên điện dung C_B tương đối lớn ($C_B \gg C$). Dữ liệu đọc ra có mức điện bé: giá trị mức logic 0 và mức logic 1 trên dây bit không khác xa nhau. Để xác định mức điện áp thay đổi nhỏ trên dây bit cần phải dùng bộ khuếch đại đọc ra có độ nhạy cao.

- Làm tươi (Refresh)

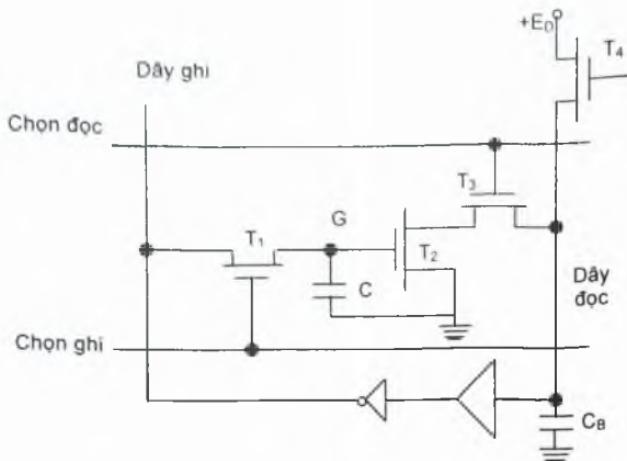
Vì mọi tụ điện đều có điện trở rò và transistor mắc nối tiếp với nó dù ở trạng thái cầm cung có một điện trở rò nhất định, cho nên sau khi được nạp, điện tích trên tụ C sẽ bị phóng điện liên tục, sau một khoảng nào đó sẽ phóng hết điện và không phản ánh chính xác bit dữ liệu nó chứa. Chính vì vậy cần phải nạp điện cho tụ điện (nếu lưu thông tin mức 1) trước khi điện áp trên tụ giảm thấp hơn một ngưỡng nào đó, việc làm này gọi là bù điện áp hay "làm tươi" (refresh). Xong vấn đề là không thể chỉ xét đến các bit nhớ có nội dung là 1 để làm tươi. Để làm tươi các ô nhớ DRAM trong bộ nhớ DRAM, cần phải đọc nội dung của nó rồi viết trở lại. Việc làm tươi này được tiến hành đều đặn theo một chu kỳ nhất định, gọi là chu kỳ làm tươi. Tên gọi RAM động xuất phát từ đặc tính và sự hoạt động này.

Trong các chip DRAM trước đây, mạch điện bổ sung để thực hiện việc làm tươi thường nằm ở ngoài chip nhớ, vì vậy dây nối với mạch ngoài tương đối nhiều. Ngày nay các mạch thực hiện làm tươi thường được chế tạo nằm ngay bên trong chip nhớ, nhờ đó chip nhớ loại này vừa có dung lượng cao, vừa có giao diện đơn giản, chúng được gọi là *quasi – static RAM* (giống như SRAM).

• Phản tử nhớ động 3 transistor

Hoạt động của phản tử nhớ động MOS 3 transistor:

- Trên hình 4.12, dữ liệu lưu ở điện dung C chính là mức điện áp cực G của transistor T_2 . Điện áp trên C điều khiển trạng thái của T_2 . Các dây đọc/ghi bit và dây chọn đọc/ghi đều riêng biệt. Dây chọn đọc điều khiển bóng T_3 , dây chọn ghi điều khiển bóng T_1 . T_4 là bóng bán dẫn mạch nạp trước dùng chung cho cột phản tử nhớ của ma trận.



Hình 4.12. Phản tử nhớ động MOS 3 bóng

– Quá trình đọc: Đầu tiên dây đọc được nạp trước đến mức cao. Tiếp theo, dây chọn đọc nâng lên mức cao làm cho T_3 thông. Giả sử C được nạp điện đến mức lớn hơn điện áp cắt của T_2 (nội dung bit nhớ là 1) làm T_2 thông. C_B phỏng điện qua T_3 . T_2 làm cho dây đọc bit ở mức thấp. Giả sử C không tích điện (nội dung bit nhớ là 0), thì T_2 ngắt, C_B không có đường phỏng điện, mức điện trên dây đọc bit ở mức thấp. Như vậy, tín hiệu (điện áp) trên dây đọc bit là tín hiệu đảo của bit lưu trữ trong C . Tín hiệu ra trên dây đọc bit phải qua bộ khuếch đại đọc và mạch đảo đưa đến đầu ra của bộ nhớ.

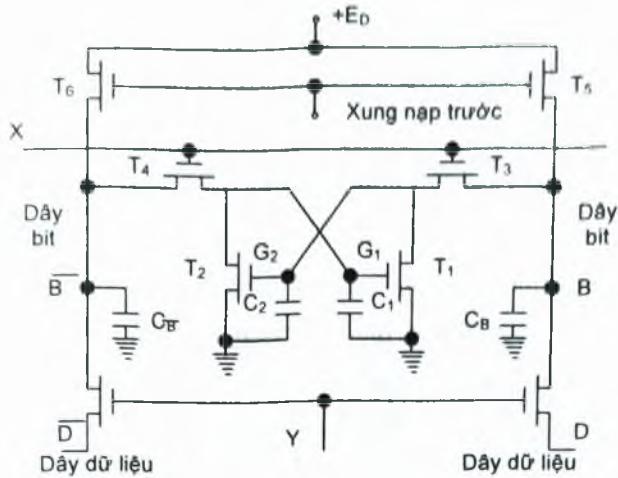
– Quá trình ghi thông tin: Khi truyền điện đến dây ghi tín hiệu cần ghi và tín hiệu trên dây chọn ghi ở mức cao, T_1 thông nên tụ C được nạp qua T_1 . Như vậy, ta đã ghi được thông tin vào phản tử nhớ. Mức điện áp đặt ở G (đặt ở tụ C) cùng mức điện áp trên dây ghi bit.

– Làm tươi bộ nhớ: Nếu có tín hiệu chọn đồng thời đến T_1 và T_3 thì thông tin đọc được sau đó được khuếch đại và qua mạch đảo đưa vào để ghi lại cho tụ C .

- *Phản tử nhớ động 4 transistor*

Dữ liệu (diện tích) lưu trữ trên C_1 và C_2 . Điện áp trên C_1 , C_2 điều khiển T_1 , T_2 thông hoặc ngắt.

Khi C_1 nạp điện tích (điện áp trên C_1 lớn hơn điện áp cắt của T_1) và C_2 không có điện tích (điện áp trên C_2 nhỏ hơn điện áp cắt của T_2) thì T_1 thông, T_2 ngắt – trường hợp này tương ứng với trạng thái 0 của phản tử nhớ.



Hình 4.13. Phần tử nhớ động MOS 4 bóng

Khi C_2 nạp điện tích (điện áp trên C_2 lớn hơn điện áp cát của T_2) và C_1 không có điện tích (điện áp trên C_1 nhỏ hơn điện áp cát của T_1) thì T_2 thông, T_1 ngắt – trường hợp này tương ứng với trạng thái 1 của phần tử nhớ.

T_3 và T_4 là những bóng để điều khiển sự nối thông phần tử nhớ với dây bit. T_5 và T_6 là mạch nạp trước của dây bit, dùng chung cho tất cả các phần tử nhớ cùng cột trong ma trận nhớ. Khi bắt đầu truy xuất bộ nhớ, trên cực công của T_5 và T_6 có xung nạp trước nên T_5 và T_6 nối thông, do đó trên các dây bit B và \bar{B} có mức điện áp cao vì thông đến nguồn E_D . Sau khi kết thúc xung nạp trước, T_5 và T_6 ngắt, các dây bit cách ly khỏi nguồn E_D . Nhưng do tác động của điện dung phân bố C_B và $C_{\bar{B}}$, khi đó mức điện áp cao của dây bit có thể duy trì thêm một khoảng thời gian nữa.

Trong khoảng thời gian này, giả sử tiến hành đọc dữ liệu, dây X có mức cao. T_3 và T_4 thông. Giả sử phần tử nhớ có trạng thái 0 (T_1 thông, T_2 ngắt) thì G_1 có mức cao, G_2 có mức thấp. Lúc này C_B phóng điện qua T_1 và T_3 . Do đó dây bit B biến thành có mức điện áp thấp. Do T_2 ngắt, dây bit \bar{B} vẫn ở mức cao. Vậy dữ liệu trong phần tử nhớ đã được đọc ra dây bit B và \bar{B} . Nếu lúc này dây Y cũng ở mức cao thì tín hiệu sẽ đưa đến đầu ra của RAM qua dây dữ liệu $D (= 0)$ và $\bar{D} (= 1)$.

Vậy, mạch điện nạp trước của dây bit có tác dụng gì? Trong khoảng thời gian T_3 , T_4 thông, nếu dây bit không được nạp điện trước thì mức cao có **được** của dây \bar{B} chỉ do C_1 phỏng qua T_4 nạp vào C_B . Nếu thế thì điện tích trên C_1 bị suy giảm, điện áp trên C_B thậm chí lớn hơn trên C_1 (vì có nhiều phần tử nhớ **nối** vào dây bit). Vậy G_1 có thể không giữ nguyên mức cao sau một lần đọc, tức là dữ liệu bị mất. Nhờ có mạch điện nạp trước, điện thế trên dây bit \bar{B} còn cao hơn điện thế G_1 một chút. Vậy khi đọc dữ liệu, điện tích trên C_1 không những không bị suy giảm mà còn được làm tươi nhờ sự nạp thêm cho C_1 qua T_4 .

Khi ghi dữ liệu, dây Y và dây X đều có mức điện áp cao, đầu vào dữ liệu của RAM sẽ làm thay đổi trạng thái phần tử nhớ thông qua dây dữ liệu và dây bit, tức là đưa dữ liệu vào trong phần tử nhớ.

Tóm lại: Ba loại phần tử nhớ động vừa trình bày ở trên đều có ưu, nhược điểm riêng. Mạch 4 bóng chiếm diện tích lớn trên chip, nhưng không cần mạch làm tươi riêng, quá trình đọc ra đồng thời là quá trình làm tươi, vì vậy mạch phụ đơn giản. Mạch 3 bóng giảm diện tích một chút, nhưng yêu cầu các dây chọn đọc, chọn ghi riêng và dây dữ liệu đều riêng biệt, cần mạch ngoài điều khiển phần hồi để làm tươi, vì vậy dây nối với mạch ngoài tương đối nhiều. Mạch 1 bóng là đơn giản nhất, nhưng yêu cầu bộ khuếch đại đọc ra phải có độ nhạy cao và yêu cầu làm tươi sau mỗi lần đọc dữ liệu, do vậy mạch điện phụ ngoài phức tạp. RAM tĩnh (SRAM) so với RAM động (DRAM): số lượng bóng bán dẫn của SRAM nhiều hơn, độ tích hợp thấp hơn, nhưng không phải làm tươi, mạch phụ đơn giản, tiện sử dụng.

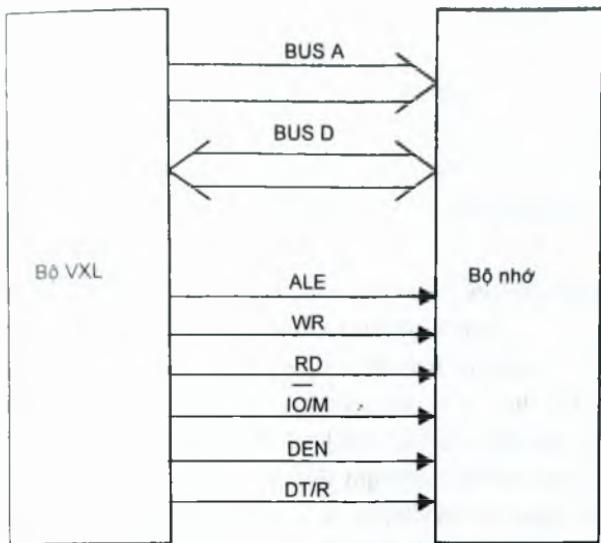
4.2. HỆ THỐNG NHỚ CHÍNH

4.2.1. Tổ chức phối ghép giữa CPU và bộ nhớ

Để trao đổi (đọc/ghi) dữ liệu với bộ nhớ, CPU cần phát ra tín hiệu địa chỉ để xác định ngăn nhớ cần đọc/ghi qua Bus địa chỉ (Bus A). Khi đã xác định được vị trí nhớ cần đọc/ghi, CPU thực hiện trao đổi dữ liệu với bộ nhớ qua Bus dữ liệu (Bus D) bằng việc sử dụng các tín hiệu điều khiển khác nhau (hình 4.14).

Các tín hiệu điều khiển bộ nhớ hoạt động như sau:

1. ALE (Address latch enable): Dùng sườn xuống của tín hiệu ALE để chốt địa chỉ nhớ. Địa chỉ này được chốt nên có thể tiến hành việc ghi/đọc cho đến khi ALE trở lại mức cao.



Hình 4.14. Ghép nối giữa CPU và bộ nhớ

2. WR/RD (Write/read): Khi địa chỉ được mở, bộ vi xử lý sẽ tác động hoặc vào tín hiệu WR (ghi) hoặc tín hiệu RD (đọc) để tiến hành quá trình ghi/đọc bộ nhớ. Tín hiệu 3 trạng thái có mức logic thấp (với loại thường là thông) hoặc mức logic cao (với loại thường là ngắn) và không thể mở cả hai ô nhớ cùng một lúc.

3. IO/M (Input – Output/Memory): Tín hiệu cho phép chọn hoặc là cổng vào/ra hoặc là bộ nhớ. Tín hiệu này là cần thiết vì cùng một địa chỉ có thể được dùng cho bộ nhớ hoặc cho một cổng I/O. Ví dụ, với CPU 8086 của Intel khi $\overline{IO/M} = 1$ thì CPU làm việc với bộ nhớ; với CPU 8088 của Intel khi $\overline{IO/M} = 0$ thì CPU làm việc với bộ nhớ.

4. DEN (Data Enable): Để phân định thời gian cho Bus, báo hiệu trên Bus dữ liệu đã ổn định và mở thông mạch đệm. Hầu hết các hệ thống nối với Bus dữ liệu đều phải qua mạch đệm. Tín hiệu DEN là tín hiệu 3 trạng thái hiệu lực (ở mức thấp hoặc cao tùy theo loại thiết bị 3 trạng thái) dùng để mở (enable) mạch đệm này.

5. DT/R (Data Transmit/Receive): Dùng để xác định chiều truyền dữ liệu. Mức cao khi bộ xử lý truyền dữ liệu và mức thấp khi nhận dữ liệu.

4.2.2. Kỹ thuật bộ nhớ ảo

Trong những ngày đầu tiên của kỹ nguyên máy tính Von Neumann, bộ nhớ của máy tính thường nhỏ và đắt tiền. Máy IBM 650 là một máy tính cho nghiên cứu khoa học dẫn đầu lúc đó (khoảng năm 1950) chỉ có bộ nhớ với dung lượng 4000 byte. Thời kỳ này các nhà lập trình thường phải dành phần lớn thời gian nhằm làm sao cho chương trình càng bé càng tốt để chương trình có thể lọt vào bộ nhớ rất nhỏ.

Một giải pháp để giải quyết khó khăn trên là sử dụng bộ nhớ phụ, thường là ổ đĩa. Người lập trình chia chương trình thành một số phần (gọi là Overlay), mỗi overlay có thể nằm lọt trong bộ nhớ. Để chạy chương trình, overlay thứ nhất được đọc vào, trước khi kết thúc nó sẽ đọc overlay tiếp theo và gọi overlay đó ra làm việc, cứ như thế cho đến khi chương trình hoàn thành. Người lập trình phải lo chia chương trình thành các overlay, phải quy định từng overlay được đặt ở đâu trong bộ nhớ phụ, phải tổ chức để vận chuyển các overlay giữa bộ nhớ chính và bộ nhớ phụ và nói chung là phải tự mình quản lý, tổ chức toàn bộ quá trình thực hiện chương trình theo phương pháp overlay.

Kỹ thuật này đã được sử dụng rộng rãi trong nhiều năm, nhưng nó đòi hỏi nhiều công sức của nhà lập trình cho việc quản lý và tổ chức overlay. Năm 1961, một nhóm nhà khoa học ở Manchester nước Anh đã đề xuất một phương pháp *thực hiện quá trình overlay một cách tự động*, thậm chí không đòi hỏi người lập trình phải biết điều gì đang xảy ra khi chương trình thực thi. Phương pháp này gọi là *bộ nhớ ảo – Virtual memory*.

Phương pháp bộ nhớ ảo được bắt đầu sử dụng trong một số máy tính vào những năm 1960, hầu hết liên quan đến dự án nghiên cứu về thiết kế hệ thống máy tính.

Ngày nay (từ những năm 1970), bộ nhớ ảo đã trở nên thông dụng trong hầu hết các máy tính.

Bộ nhớ ảo là một đặc trưng của hệ điều hành, nó cho phép một tiến trình sử dụng không gian địa chỉ nhớ độc lập với các tiến trình khác đang chạy trên cùng hệ thống và cho phép sử dụng một không gian địa chỉ lớn hơn dung lượng thực sự của bộ nhớ RAM hiện có, có sự chuyển tạm một phần nội dung tiến trình từ RAM ra một ô đĩa với dung lượng nhỏ hơn hoặc bằng dung lượng của tiến trình nói ở trên và sau đó sẽ chuyển vào RAM khi cần.

Bộ nhớ ảo cho phép mỗi một tiến trình thực thi được cấp một vùng nhớ vật lý trong RAM nhỏ hơn dung lượng của nó. Các địa chỉ mà mỗi tiến trình sử dụng để tham chiếu bộ nhớ (địa chỉ logic/địa chỉ ảo) được chuyển đổi bởi máy bộ nhớ ảo (Virtual memory mechanism) hay đơn vị quản lý bộ nhớ (Memory Management Unit – MMU) thành các giá trị địa chỉ khác nhau trong bộ nhớ vật lý (địa chỉ vật lý). Điều này cho phép các tiến trình khác nhau sử dụng cùng địa chỉ tham chiếu bộ nhớ – MMU sẽ chuyển giá trị địa chỉ tham chiếu này thành các giá trị địa chỉ vật lý khác nhau (hai tiến trình khác nhau cùng sử dụng một địa chỉ tham chiếu bộ nhớ như nhau. MMU sẽ chuyển đổi giá trị địa chỉ này thành hai giá trị địa chỉ vật lý khác nhau). Thông thường, một tiến trình không thể truy cập vào vùng nhớ vật lý được cấp của một tiến trình khác. Một tiến trình có thể sử dụng một không gian địa chỉ lớn hơn bộ nhớ vật lý có sẵn và mỗi một địa chỉ tham chiếu mà tiến trình sử dụng sẽ được chuyển đổi thành một địa chỉ vật lý có thực. Phần không gian địa chỉ vượt hơn dung lượng bộ nhớ có thực mà một tiến trình có thể địa chỉ hóa thực sự là dung lượng của vùng swap (vùng nhớ sử dụng trên đĩa cứng). Phần không gian có thể địa chỉ hóa thực sự cho tiến trình bao giờ cũng nhỏ hơn hoặc bằng không gian mà CPU có thể định địa chỉ.

Dung lượng của bộ nhớ ảo trên một hệ thống nhỏ hơn tổng của dung lượng RAM vật lý và dung lượng vùng swap, vì các trang nhớ đã được hoán đổi vẫn không bị xoá khỏi vùng swap, cho nên các trang này được nhân hai.

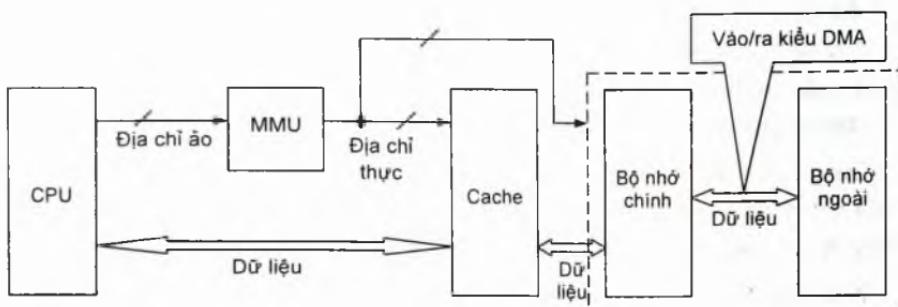
Như vậy, ta có thể đưa ra khái niệm bộ nhớ ảo như sau: *Bộ nhớ ảo* (Virtual memory) là việc sử dụng không gian trên đĩa cứng để giả lập phần thêm vào bộ nhớ chính. Hay nói cách khác, *bộ nhớ ảo* là bộ nhớ RAM không có thực, mà đúng hơn là nó bao gồm bộ nhớ RAM vật lý và một vùng nhớ trên đĩa cứng mà hệ điều hành coi nó như là một phần bộ nhớ RAM. Về bản chất, đây là một mạo được thực hiện bởi hệ điều hành, nhờ đó nâng cao hiệu năng của hệ thống.

Bộ nhớ ảo được thực thi trong Windows sử dụng kỹ thuật được gọi là trang nhớ. Khi một phần của một tiến trình hoặc dữ liệu chưa cần thiết nạp vào bộ nhớ RAM vật lý thì nó tự động được di chuyển tới file swap trên đĩa cứng theo các trang với dung lượng cho trước. Khi một phần chương trình hoặc dữ liệu cần thiết cho quá trình xử lý, nó tự động được di chuyển vào RAM vật lý. Trang nhớ được quản lý bởi hệ điều hành và trong suốt với người sử dụng.

Tóm lại: Kỹ thuật bộ nhớ ảo sử dụng hai loại địa chỉ (xem hình 4.15):

- Địa chỉ ảo (địa chỉ logic/địa chỉ chương trình): Khi chương trình được nạp vào thực thi (tiến trình), các giá trị địa chỉ tiến trình tham chiếu được tính từ 0 cho đến hết chương trình. Tập hợp tất cả các địa chỉ chương trình tạo thành bộ nhớ chương trình. Bộ nhớ chương trình bằng dung lượng chương trình đã được biên dịch. Mỗi chương trình có dung lượng bộ nhớ chương trình khác nhau.
- Địa chỉ vật lý hay còn gọi là địa chỉ thực trong bộ nhớ vật lý được tính từ 0 cho đến hết vùng nhớ RAM có thực.
- Ánh xạ từ địa chỉ ảo về địa chỉ vật lý do đơn vị MMU (Memory Management Unit) đảm nhiệm.

- Cơ chế hoạt động: Hoạt động hoán đổi dữ liệu giữa vùng swap và bộ nhớ vật lý ngày nay có nhiều phương pháp khác nhau (mục 4.2.3), song đều được điều khiển bằng phần mềm và vào/ra dữ liệu theo kiểu DMA.



Hình 4.15. Sơ đồ nguyên lý kỹ thuật bộ nhớ ảo

4.2.3. Tái định vị và bảo vệ chương trình

Chương trình muốn thực thi phải được hệ điều hành nạp vào bộ nhớ chính. Trong trường hợp kích thước chương trình lớn và dung lượng bộ nhớ vật lý (bộ nhớ chính) có hạn (phần bộ nhớ còn tự do có kích thước nhỏ hơn kích thước chương trình cần thực thi), để giải quyết vấn đề này, cần phải có các kỹ thuật thiết kế, tổ chức và sử dụng bộ nhớ hợp lý. Giải pháp *bộ nhớ ảo* đã được sử dụng và ngày nay ba phương pháp chủ yếu thường được dùng để phục vụ quản lý bộ nhớ ảo là *kỹ thuật tổ chức bộ nhớ theo phân đoạn*, *tổ chức bộ nhớ theo phân trang* hoặc *kết hợp cả hai kỹ thuật trên*.

Các kỹ thuật này cho phép chia chương trình thành nhiều phần và nạp một hoặc nhiều phần chương trình vào bộ nhớ vật lý, các phần còn lại không nhất thiết phải thi hành ngay thì được lưu trữ tạm ở một địa chỉ xác định trên ổ đĩa cứng (vùng swap) do hệ thống quản lý, khi phần chương trình được nạp đã thực hiện xong thì có thể được giải phóng khỏi bộ nhớ vật lý và đưa ra lưu trữ tạm trong ổ đĩa để tạo ra vùng nhớ vật lý tự do. Sau đó thực hiện nạp các phần chương trình còn lại để thực thi.

Có những phần chương trình (ví dụ như chương trình con) thường được gọi lặp đi lặp lại nhiều lần. Như vậy có thể phải hoán đổi phần chương trình này từ bộ nhớ ra ngoài ổ đĩa rồi lại đưa vào thực thi nhiều lần. Thao tác này gọi là *kỹ thuật tái định vị chương trình trong bộ nhớ*, hay còn gọi là *kỹ thuật Swapping*.

Một vấn đề đặt ra cho kỹ thuật quản lý bộ nhớ theo phân đoạn hoặc phân trang là làm thế nào để khi cung cấp các vùng nhớ nhỏ cho các phần của chương trình mà không bị chồng chéo nhau giữa các phần trong một chương trình hay giữa các chương trình đang cùng thực thi trong máy tính với nhau. Để giải quyết vấn đề này, trong mỗi kỹ thuật tổ chức và quản lý bộ nhớ, người ta phải thiết kế ra các thành phần phân cứng hỗ trợ cho việc quản lý các đoạn bộ nhớ (với kỹ thuật phân đoạn) hay các trang nhớ (với kỹ thuật phân trang) đã cấp phát cho chương trình cũng như các đoạn/các trang nhớ còn tự do có thể cấp phát cho chương trình. Nhờ các thành phần phân cứng này kết hợp với cơ chế quản lý theo từng kỹ thuật phân trang và phân đoạn mà các chương trình khác nhau đảm bảo được cấp phát các vùng nhớ tách biệt (không có sự ghi đè); chương trình ứng dụng này không thể truy nhập trái phép vào vùng nhớ đã cấp phát cho chương trình ứng dụng khác. Do vậy giúp bảo vệ tốt các chương trình đang thực thi trong hệ thống.

a) Kỹ thuật phân đoạn

Chương trình và dữ liệu được người lập trình chia thành các módun (chương trình chính và các chương trình con). Khi biên dịch, chương trình được biên dịch theo từng módun, mỗi módun được biên dịch riêng khi chương trình được gọi và sẽ được nạp vào một vùng nhớ riêng biệt trong bộ nhớ, hoặc được cắt trong vùng swap. Mỗi módun được gọi là một đoạn, được đánh số hiệu quản lý từ 0 và chiếm một đoạn không gian nhớ trong bộ nhớ khi được nạp. Đoạn nào được nạp vào trong bộ nhớ chính thì được quản lý theo số hiệu và địa chỉ đã cấp, địa chỉ này là địa chỉ đầu của đoạn bộ nhớ vừa cấp (còn gọi là địa chỉ đoạn hay địa chỉ segment).

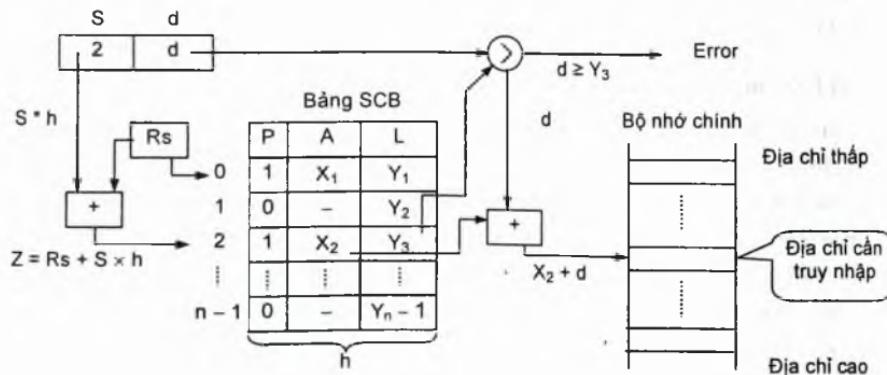
Hệ thống sử dụng một bảng quản lý phân đoạn (Segment Control Block – SCB) để quản lý trạng thái các đoạn trong chương trình vừa được nạp. Địa chỉ đầu của bảng được quản lý nhờ thanh ghi đoạn Rs. Bảng SCB có ba trường là P, A, L:

- P – trường dấu hiệu: Cho biết đoạn đã nạp vào bộ nhớ chính hay chưa?
 - Nếu $P = 0$: đoạn chưa được nạp, nằm ở vùng swap do hệ thống quản lý;
 - Nếu $P = 1$: đoạn đã được nạp vào bộ nhớ chính.
- A – trường địa chỉ: Cho biết địa chỉ đầu vùng nhớ mà đoạn đã được nạp, trường này chỉ có ý nghĩa khi $P = 1$.
- L – trường độ dài: Cho biết độ dài của đoạn, hay chính là dung lượng của môđun khi biên dịch.

Cơ chế truy nhập bộ nhớ:

- Để truy nhập vào một lệnh hay dữ liệu trong đoạn, cần biết được số hiệu của đoạn (ký hiệu là S) cần truy nhập và địa chỉ tương đối của lệnh hay dữ liệu trong đoạn (ký hiệu là d, d tính từ 0), địa chỉ tương đối này còn được gọi là *địa chỉ lệch* hay *địa chỉ offset*. Do vậy, hệ thống cần nhận địa chỉ truy nhập là bộ đôi giá trị $\langle S, d \rangle$, địa chỉ này được gọi là *địa chỉ logic*.

- Để tham chiếu được đến địa chỉ vật lý của lệnh hoặc dữ liệu trong đoạn cần có các thành phần phần cứng hỗ trợ như là các thanh ghi đoạn (Segment Register) để nạp địa chỉ đầu của SCB; các thanh ghi lệch (Offset Register) để nạp địa chỉ lệch trong đoạn; bộ cộng, bộ so sánh để ánh xạ địa chỉ logic ra địa chỉ cần truy nhập trong bộ nhớ vật lý.



Hình 4.16. Sơ đồ cơ chế truy nhập bộ nhớ quản lý kiểu phân đoạn

- Ta xét sơ đồ cơ chế truy nhập bộ nhớ quản lý kiểu phân đoạn theo hình 4.16, mô tả với ví dụ đoạn có số hiệu $S = 2$, độ dài $L = Y_3$, nạp tại địa chỉ $\Lambda = X_2$.
 - Để truy nhập dữ liệu, hệ thống nhận địa chỉ yêu cầu truy nhập $\langle S, d \rangle$.
 - Cần cù vào nội dung thanh ghi Rs , hệ thống tính ra địa chỉ phần tử S (quản lý trạng thái đoạn S) trong bảng SCB là $Z = Rs + S \times h$, trong đó h là độ rộng phần tử trong bảng SCB.
 - Hệ thống truy nhập phần tử S trong SCB, đọc L , kiểm tra:
 - Nếu $d \geq L$ thì hệ thống đưa ra thông báo lỗi (độ dài đoạn là $L = Y_3$ thì d chỉ nằm trong khoảng từ 0 đến $Y_3 - 1$) và tiến trình bị dừng.
 - Nếu $d < L$, hệ thống kiểm tra P :
 - Nếu $P = 0$, đoạn chưa được nạp, hệ thống tiến hành nạp đoạn từ vùng swap vào bộ nhớ chính, sửa $P = 1$ và $A =$ địa chỉ đầu vùng nhớ vừa nạp đoạn.
 - Nếu $P = 1$, hệ thống đọc trường A và xác định được địa chỉ vật lý cần truy nhập là $A + d$ nhờ bộ cộng (theo hình 4.16, địa chỉ vật lý là $X_2 + d$).

Trong kỹ thuật này ta cần lưu ý rằng, hệ thống luôn đảm bảo rằng, khi một đoạn có độ dài là Y_1 nào đó đã nạp tại địa chỉ X_1 nào đó, thì địa chỉ đầu cho đoạn tiếp theo sau được nạp (kể cả của một chương trình bất kỳ khác) không thể có giá trị trong khoảng $X_1 + Y_1$, vì như vậy hai đoạn sẽ bị cấp trùng cùng một vùng nhớ nào đó. Trong trường hợp hết bộ nhớ tự do mà có módun chương trình cần nạp, hệ thống phải xem xét các módun chương trình đã được nạp vào bộ nhớ chính để xác định módun chương trình có thể đưa tạm ra vùng swap với điều kiện dung lượng módun này lớn hơn hoặc bằng dung lượng módun chương trình cần nạp (Módun chương trình được đưa ra vùng swap phải tạo ra vùng nhớ tự do lớn hơn hoặc bằng dung lượng módun cần nạp), sau đó mới tiến hành nạp módun chương trình cần nạp, thao tác này gọi là *đổi đoạn*. Để làm được điều này hệ thống căn cứ vào các bảng SCB. Do vậy, với kỹ thuật quản lý kiểu phân đoạn cho phép hỗ trợ các hệ điều hành đa nhiệm cách ly và bảo vệ hệ điều hành khỏi những truy nhập trái phép của các chương trình ứng dụng; cách ly và bảo vệ chương trình ứng dụng này khỏi những truy nhập trái phép của các chương trình ứng dụng khác.

Như vậy, về kiến trúc phân cứng cần có các thành phần phân cứng hỗ trợ cho cơ chế này thực hiện được như là các thanh ghi để nạp địa chỉ bảng SCB, địa chỉ lệnh trong đoạn, các bộ cộng, bộ so sánh,...

b) Kỹ thuật phân trang

Bộ nhớ vật lý phải được chia thành nhiều trang vật lý (đánh số hiệu từ 0) mà kích thước của các trang vật lý là như nhau (bằng L), một trang vật lý thường được gọi là *khung trang*. Khi được gọi, chương trình được biên dịch thành một khối thống nhất, và cũng được chia thành các trang gọi là các trang logic, kích thước của các trang logic là bằng nhau (đánh số hiệu từ 0) và bằng kích thước của trang vật lý. Mỗi trang logic có thể đã được nạp vào một trang vật lý nào đó hoặc có thể được đưa vào vùng swap. Trang logic nào được nạp vào trong bộ nhớ chính thì đều được quản lý theo số hiệu của nó và số hiệu trang vật lý đã cấp cho nó.

Hệ thống sử dụng một bảng quản lý phân trang (Page Control Block – PCB) để quản lý trạng thái các trang trong chương trình vừa được nạp. Địa chỉ đầu của bảng PCB được quản lý nhờ thanh ghi Rp. Bảng PCB có hai trường là P, A:

- P – trường dấu hiệu: Cho biết trang đã nạp vào bộ nhớ chính hay chưa?
 - Nếu P = 0: trang chưa được nạp, nằm ở vùng swap do hệ thống quản lý.
 - Nếu P = 1: trang đã được nạp vào bộ nhớ chính.
- A – trường địa chỉ: Cho biết số hiệu trang vật lý đang chứa trang logic, trường này chỉ có ý nghĩa khi P = 1.

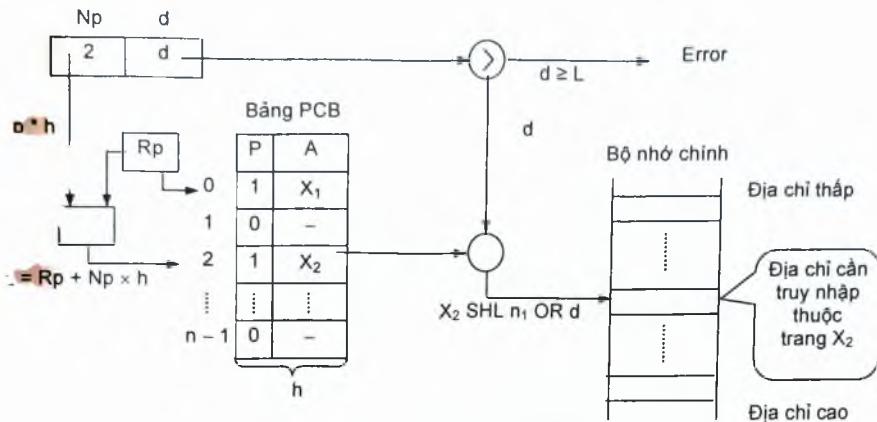
Cơ chế truy nhập bộ nhớ:

- Để truy nhập vào một lệnh hay dữ liệu trong trang nào đó, cần xác định được số hiệu của trang logic cần truy nhập (ký hiệu là Np) và địa chỉ tương đối (offset) của lệnh hay dữ liệu trong trang (ký hiệu là d, d tính từ 0). Do vậy hệ thống cần nhận địa chỉ truy nhập là bộ đôi giá trị $\langle Np, d \rangle$, địa chỉ này được gọi là *địa chỉ logic*.
- Để tham chiếu được đến địa chỉ vật lý của lệnh hoặc dữ liệu trong trang cần có các thành phần phần cứng hỗ trợ như là các thanh ghi quản lý trang Rp (Page Register) để nạp địa chỉ đầu của PCB; các thanh ghi lệch (Offset Register) để nạp địa chỉ lệch trong trang; bộ cộng, bộ dịch, bộ so sánh,... để ánh xạ địa chỉ logic ra địa chỉ cần truy nhập trong bộ nhớ vật lý.

- Ta xét sơ đồ cơ chế truy nhập bộ nhớ quản lý kiểu phân trang theo hình 4.17, mô tả với ví dụ Np = 2, số hiệu trang vật lý đang chứa trang Np là X₂.

- Để truy nhập dữ liệu, hệ thống đưa ra địa chỉ yêu cầu truy nhập $\langle Np, d \rangle$.

- Hệ thống kiểm tra: Nếu $d \geq L$ (trong đó L là độ dài trang) thì hệ thống đưa ra thông báo lỗi (độ dài của một trang là L thì d chỉ nằm trong khoảng từ 0 đến L) và tiến trình bị dừng; nếu $d < L$ thì tiếp tục thực hiện các bước tiếp sau.
- Căn cứ vào nội dung thanh ghi Rp , hệ thống tính ra địa chỉ phần tử Np quản lý trạng thái trang logic Np trong bảng PCB là $Z = Rp + Np \times h$, trong đó h độ rộng phần tử trong bảng PCB.



Hình 4.17. Sơ đồ cơ chế truy nhập bộ nhớ quản lý kiểu phân trang

- Truy nhập phần tử Np trong PCB, hệ thống kiểm tra P :
 - + Nếu $P = 0$, trang logic chưa được nạp, hệ thống tiến hành nạp trang từ vùng swap vào bộ nhớ chính, sửa $P = 1$ và $A =$ số hiệu trang vật lý vừa nạp trang logic.
 - + Nếu $P = 1$, hệ thống đọc trường A và xác định được trang vật lý đang chứa trang logic và xác định được địa chỉ truy nhập là $A \times L + d$ nhờ bộ ghép (theo hình 4.17, địa chỉ vật lý là $X_2 \times L + d$).

Bộ ghép thực hiện như sau: Hệ thống dùng n bit để đánh địa chỉ vật lý và $n = n_2 + n_1$. Trong đó: n_1 bit thấp để đánh địa chỉ lêch trong trang; n_2 bit cao còn lại dùng để đánh số hiệu các trang vật lý. Ta thấy số trang vật lý trong bộ nhớ chính là 2^{n_2} và dung lượng một trang $L = 2^{n_1}$. Do vậy, giá trị $A \times L = A \times 2^{n_1}$ tương đương với phép toán $A \text{ SHL } n_1$ (giá trị A dịch trái n_1 bit); giá trị $A \times L + d = A \text{ SHL } n_1 + d$, vì d được biểu diễn bởi n_1 bit thấp nên $A \text{ SHL } n_1 + d$

tương đương với phép toán: A SHL n_1 OR d. Vậy, bộ ghiệp thực hiện **hai** thao tác là dịch trái giá trị A đi n_1 bit thành giá trị A trong n_2 bit cao và OR giá trị A trong n_2 bit cao với giá trị d trong n_1 bit thấp.

Trong kỹ thuật này ta cần lưu ý rằng, hệ thống luôn đảm bảo **rắng**, khi một trang vật lý X nào đó đã nạp một trang Y_j logic nào đó, thì một trang logic Y_j bất kỳ nào khác (kể cả của tiến trình khác) cần nạp sẽ không thể nạp vào trang vật lý X nói trên nếu nội dung trang vật lý X này vẫn đang cần thực thi trong hệ thống. Trong trường hợp bộ nhớ vật lý hết các trang vật lý tự do, hệ thống sẽ xem xét trang vật lý nào chưa nội dung (trang logic) chưa cần thiết thực thi để đưa nội dung trang này ra vùng swap (hệ thống sẽ ghi nhớ và quản lý) rồi với nạp trang logic Y_j vào trang vật lý X – quá trình này gọi là *đổi trang*. Để làm được điều này, hệ thống căn cứ vào các bảng PCB. Do vậy, với kỹ thuật quản lý kiểu phân trang cho phép hỗ trợ các hệ điều hành đa nhiệm cách ly và bảo vệ hệ điều hành khỏi những truy nhập trái phép của các chương trình ứng dụng; cách ly và bảo vệ chương trình ứng dụng này khỏi những truy nhập trái phép của các chương trình ứng dụng khác.

Như vậy, về kiến trúc phân cứng, cần có các thành phần phân cứng hỗ trợ cho cơ chế này thực hiện được như là các thanh ghi để nạp địa chỉ bảng PCB; địa chỉ lệnh trong trang; các bộ cộng, bộ ghép (mạch dịch và mạch OR),...

c) Kỹ thuật phân trang – phân đoạn kết hợp

Qua nghiên cứu kỹ thuật phân đoạn ta thấy nhược điểm là, khi đổi đoạn nhiều lần sẽ xuất hiện nhiều vùng nhớ tự do với dung lượng nhỏ (do một módun bị đưa ra vùng swap lớn hơn módun cần nạp, nên khi nạp đoạn sẽ thừa ra một vùng nhớ tự do với dung lượng nhỏ) không liền kề nhau (gọi là bộ nhớ bị phân mảnh). Chính vì vậy, có hiện tượng tổng dung lượng nhớ tự do lớn, song không thể nạp một módun chương trình nào vì dung lượng mỗi vùng nhớ tự do nhỏ hơn dung lượng módun cần nạp. Hiện tượng này gây lãng phí bộ nhớ.

Kỹ thuật phân trang khắc phục nhược điểm của kỹ thuật phân đoạn, song vẫn còn gây ra hiện tượng lãng phí bộ nhớ tự do khi trang logic cuối cùng của chương trình được nạp (ví dụ, kích thước một khung trang là 4 byte, chương trình dung lượng 18 byte, chương trình sẽ có 5 trang logic, trang cuối chỉ chiếm 2 byte, như vậy khi nạp sẽ có 2 byte bị dư thừa lãng phí). Để giảm thiểu sự lãng phí bộ

nhỏ, bộ nhớ vật lý được chia thành các trang vật lý với dung lượng nhỏ đi, điều này dẫn đến làm tăng số trang logic (với cùng một chương trình). Khi biên dịch chương trình, hệ thống tạo ra bảng PCB và nạp toàn bộ bảng PCB vào bộ nhớ, hiển nhiên bảng PCB này sẽ có số phần tử tăng do số trang logic tăng, điều này lại gây tốn bộ nhớ.

Để tối ưu hóa trong sử dụng bộ nhớ, người ta đưa ra giải pháp kỹ thuật phân trang kết hợp với phân đoạn.

Với kỹ thuật phân trang – phân đoạn kết hợp, bộ nhớ vật lý được chia thành nhiều trang vật lý (đánh số từ 0, độ dài L) như kỹ thuật phân trang. Khi biên dịch, chương trình được biên dịch theo từng módun và cũng sử dụng một bảng SCB để quản lý trạng thái của từng módun như kỹ thuật phân đoạn. Mỗi módun lại được chia thành nhiều trang logic, kích thước một trang logic bằng một trang vật lý. Khi một módun được nạp, hệ thống sinh ra một bảng PCB để quản lý trạng thái của từng trang logic trong módun, tương tự như kỹ thuật phân trang. Khi đoạn được giải phóng khỏi bộ nhớ chính (đưa ra vùng swap) thì bảng PCB tương ứng với nó cũng được giải phóng khỏi bộ nhớ. Chính vì vậy, hiện tượng phân mảnh do kỹ thuật phân đoạn gây nên không còn và tại mỗi thời điểm, tổng kích thước các bảng PCB trong bộ nhớ chính nhỏ hơn kích thước bảng PCB duy nhất được tạo ra khi dùng kỹ thuật phân trang.

Hệ thống sử dụng một bảng quản lý phân đoạn (Segment Control Block – SCB) để quản lý trạng thái các đoạn trong chương trình vừa được nạp. Địa chỉ đầu của bảng được quản lý nhờ thanh ghi đoạn Rs. Bảng SCB có ba trường là P_s , Ap, Ls:

- P_s – trường dấu hiệu: Cho biết đoạn đã nạp vào bộ nhớ chính hay chưa?
 - Nếu $P_s = 0$: đoạn chưa được nạp, nằm ở vùng swap do hệ thống quản lý.
 - Nếu $P_s = 1$: đoạn đã được nạp vào bộ nhớ chính.
- Ap – trường địa chỉ: Cho biết địa chỉ đầu của bảng PCB tương ứng với đoạn được nạp, trường này chỉ có ý nghĩa khi $P_s = 1$.
- Ls – trường độ dài: Cho biết độ dài của đoạn, chính là số trang logic của đoạn.

Với mỗi đoạn khi được nạp, hệ thống tạo ra một bảng PCB để quản lý trạng thái các trang trong đoạn vừa được nạp. Địa chỉ đầu của bảng được ghi trong trường Ap của bảng SCB. Bảng PCB có hai trường là P, A:

- P – trường dấu hiệu: Cho biết trang đã nạp vào bộ nhớ chính hay chưa?

- Nếu $P = 0$: trang chưa được nạp, nằm ở vùng swap do hệ thống quản lý.
- Nếu $P = 1$: trang đã được nạp vào bộ nhớ chính.
- A – trường địa chỉ: Cho biết số hiệu trang vật lý đang chứa trang logic, trường này chỉ có ý nghĩa khi $P = 1$.

Cơ chế truy nhập bộ nhớ:

- Để truy nhập vào một lệnh hay dữ liệu trong trang nào đó, cần biết được số hiệu của đoạn (ký hiệu là S), số hiệu trang logic trong đoạn (ký hiệu Np) và địa chỉ tương đối (offset) của lệnh hay dữ liệu trong trang (ký hiệu là d, d tính từ 0). Do vậy, hệ thống cần nhận địa chỉ truy nhập là bộ ba giá trị $\langle S, Np, d \rangle$, địa chỉ này được gọi là *địa chỉ logic*.

- Để tham chiếu được đến địa chỉ vật lý của lệnh hoặc dữ liệu trong đoạn cần có các thành phần phần cứng hỗ trợ như là các thanh ghi đoạn (Segment Register) để nạp địa chỉ đầu của SCB; các thanh ghi lệch (Offset Register) để nạp địa chỉ lệch trong trang; bộ cộng, bộ so sánh, bộ ghép,... để ánh xạ địa chỉ logic ra địa chỉ cần truy nhập trong bộ nhớ vật lý.

- Ta xét sơ đồ cơ chế truy nhập bộ nhớ quản lý kiểu phân trang – phân đoạn kết hợp theo hình 4.18, mô tả với ví dụ địa chỉ là bộ ba tham số $\langle 2, 3, d \rangle$, nghĩa là truy nhập ô nhớ có địa chỉ offset = d trong trang logic Np = 3 thuộc đoạn có số hiệu S = 2.

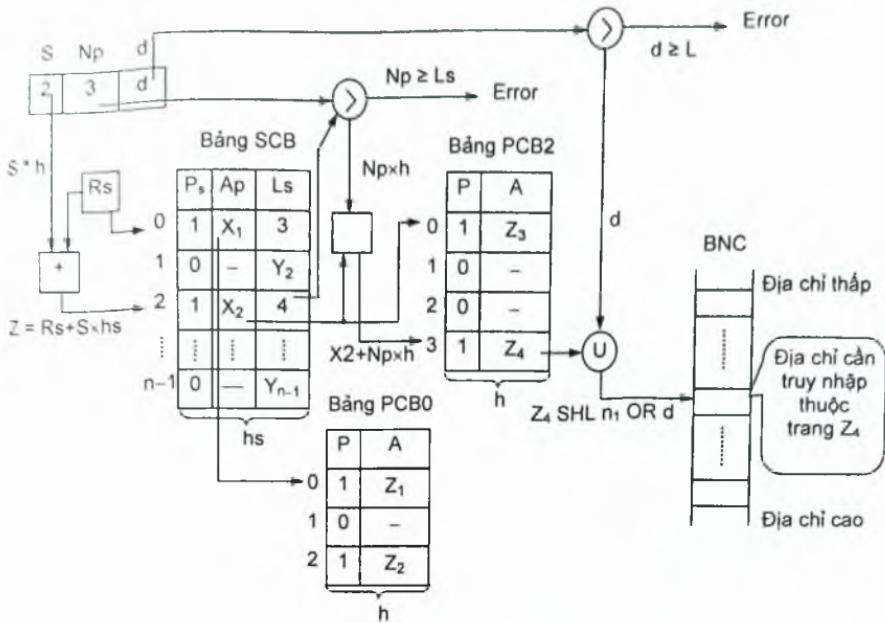
– Để truy nhập dữ liệu, hệ thống nhận địa chỉ yêu cầu truy nhập $\langle S, Np, d \rangle$.

- Hệ thống kiểm tra: Nếu $d \geq L$ thì đưa ra thông báo lỗi (độ dài trang là L thì giá trị d chỉ nằm trong khoảng từ 0 đến $L - 1$) và tiến trình dừng; ngược lại, $d < L$ thì hệ thống thực hiện tiếp các bước sau.

- Căn cứ vào nội dung thanh ghi Rs, hệ thống tính ra địa chỉ phần tử S (quản lý trạng thái đoạn S) trong bảng SCB là $Z = Rs + S \times hs$, trong đó hs là độ rộng phần tử trong bảng SCB.

– Hệ thống truy nhập phần tử S trong SCB, đọc Ls, kiểm tra:

- + Nếu $Np \geq Ls$ thì hệ thống đưa ra thông báo lỗi (số trang logic trong đoạn là Ls = 4 thì Np chỉ nằm trong khoảng từ 0 đến 3 – giá trị Np = 3 và bảng SCB trong hình 4.18 thì không sinh lỗi) và tiến trình bị dừng.



Hình 4.18. Sơ đồ cơ chế truy nhập bộ nhớ quản lý kiểu phân trang – phân đoạn kết hợp

+ Nếu $Np < L_s$, hệ thống kiểm tra P :

* Nếu $P = 0$, đoạn chưa được nạp, hệ thống phải tiến hành nạp đoạn từ vùng swap vào bộ nhớ chính bằng cách tạo ra bảng PCB để quản lý các trang trong đoạn, sửa $P = 1$ và $A_p =$ địa chỉ đầu của bảng PCB vừa tạo ra.

* Nếu $P = 1$, hệ thống đọc trường A_p kết hợp với Np và xác định được địa chỉ vật lý phân tử Np cần truy nhập trong PCB là $A_p + Np \times h$ (trong hình 4.18, $A_p = X_2$, h là độ rộng phân tử bảng PCB) nhờ bộ cộng.

+ Truy nhập phân tử Np trong PCB, hệ thống kiểm tra P :

* Nếu $P = 0$, trang logic chưa được nạp, hệ thống tiến hành nạp trang Np từ vùng swap vào bộ nhớ chính, sửa $P = 1$ và $A =$ số hiệu trang vật lý vừa nạp trang logic.

* Nếu $P = 1$, hệ thống đọc trường A và xác định được trang vật lý đang chứa trang logic và xác định được địa chỉ truy nhập là $A \times L + d$ nhờ bộ ghép (theo hình 4.18, địa chỉ vật lý là $Z_4 \times L + d$).

+ Bộ ghép thực hiện như sau: Hệ thống dùng n bit để đánh địa chỉ vật lý và $n = n_2 + n_1$. Trong đó: n_1 bit thấp để đánh địa chỉ lêch trong trang; n_2 bit cao còn lại dùng để đánh số hiệu các trang vật lý. Ta thấy số trang vật lý trong bộ nhớ chính là 2^{n_2} và dung lượng một trang là $L = 2^{n_1}$. Do vậy, giá trị $A \times L = A \times 2^{n_1}$ tương đương với phép toán $A \text{ SHL } n_1$ (giá trị A dịch trái n_1 bit). Phép $A \times L + d = A \text{ SHL } n_1 + d$, vì d được biểu diễn bởi n_1 bit thấp nên $A \text{ SHL } n_1 + d$ tương đương với phép toán $A \text{ SHL } n_1 \text{ OR } d$. Vậy, bộ ghép thực hiện hai thao tác là dịch trái giá trị A đi n_1 bit để được giá trị A trong n_2 bit cao và OR giá trị A trong n_2 bit cao với giá trị d trong n_1 bit thấp.

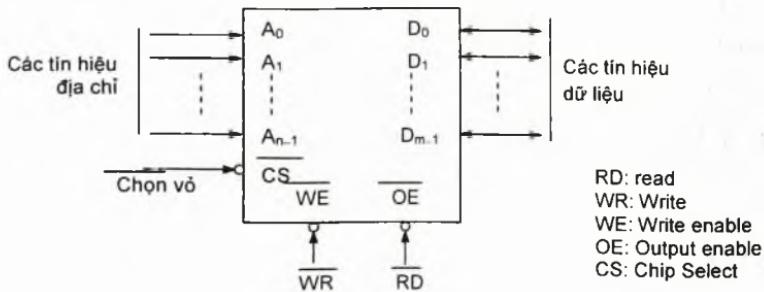
Trong kỹ thuật này thao tác đổi trang, đổi đoạn thực hiện như trong kỹ thuật phân trang và phân đoạn.

Như vậy, về kiến trúc phần cứng cần có các thành phần phần cứng hỗ trợ cho cơ chế này thực hiện được như là các thanh ghi để nạp địa chỉ bằng SCB; địa chỉ lêch trong đoạn; các bộ cộng, bộ so sánh, bộ ghép,...

4.2.4. Mở rộng bộ nhớ và tổ chức bank nhớ

a) Sơ đồ khối chung của vi mạch nhớ

Một vi mạch nhớ thường có sơ đồ khối chung như hình 4.19.



Hình 4.19. Sơ đồ khối chung của vi mạch nhớ

Theo sơ đồ khối hình 4.19, ta có các nhóm tín hiệu sau:

➤ Nhóm tín hiệu địa chỉ

– Các tín hiệu địa chỉ có tác dụng chọn ra một ngăn nhớ để ghi/ đọc.

- Các ngăn nhớ có độ dài khác nhau, tuỳ theo nhà sản xuất: 1, 4 hoặc 8 bit. Ngày nay thường là 8 bit và gọi là ô nhớ.

Số đường tín hiệu địa chỉ liên quan đến dung lượng nhớ. Với 1 mạch nhớ có n bit địa chỉ thì dung lượng của mạch nhớ đó là 2^n ngăn nhớ.

➤ Nhóm tín hiệu dữ liệu

Số đường dây dữ liệu quy định độ dài ngăn nhớ của vi mạch nhớ, tức là nếu có m đường dây dữ liệu vào/ra thì ngăn nhớ có độ rộng là m bit. Thông thường người ta hay nói rõ dung lượng vi mạch nhớ với đơn vị là bit bằng cách biểu diễn qua số lượng ngăn nhớ và độ dài ngăn nhớ cùng một lúc qua công thức $2^n \times m$ (bit).

Ví dụ: Mạch nhớ $1K \times 8$ (có nghĩa là dung lượng 1KB, độ dài ô nhớ 8 bit).

Mạch nhớ $16K \times 8$ (có nghĩa là dung lượng 16KB, độ dài ô nhớ 8 bit).

➤ Nhóm tín hiệu chọn vi mạch (chọn vỏ)

Các tín hiệu chọn vỏ là:

\overline{CS} (chip Select – chọn chip) – thường ký hiệu cho các mạch RAM;

\overline{CE} (Chip Enable – chấp nhận chip) – thường ký hiệu cho các mạch ROM.

Chúng thường được nối với đầu ra của bộ giải mã địa chỉ và dùng để chọn ra vi mạch nhớ cụ thể để ghi/đọc.

Khi vi mạch nhớ không được chọn, thì Bus dữ liệu của nó bị treo (ở trạng thái chờ kháng cao).

➤ Nhóm tín hiệu điều khiển

– Với mạch nhớ ROM:

Thường có một đầu vào điều khiển \overline{OE} (Output Enable) để cho phép dữ liệu được đưa ra Bus dữ liệu. Một vi mạch nhớ không được mở bởi \overline{OE} thì Bus dữ liệu của nó bị treo.

– Với mạch nhớ RAM:

+ Nếu chỉ có một tín hiệu điều khiển thì thường là \overline{R}/W để điều khiển quá trình ghi/đọc.

– Nếu có hai tín hiệu điều khiển thì thường là:

+ \overline{WE} để điều khiển ghi;

+ \overline{OE} để điều khiển đọc.

b) Sơ đồ khối chung của vi mạch giải mã địa chỉ bộ nhớ

Mỗi mạch nhớ nối ghép với CPU cần phải được CPU quy chiếu tới một cách chính xác khi thực hiện các thao tác ghi/doc. Do đó cần phải có **mạch giải mã địa chỉ** để gán địa chỉ cụ thể cho mạch nhớ nhờ một xung chọn vỏ lấy từ mạch giải mã địa chỉ.

Một cách tổng quát, bộ giải mã địa chỉ thường có sơ đồ khối chung như hình 4.20.



Hình 4.20. Sơ đồ khối vi mạch giải mã địa chỉ bộ nhớ

Theo sơ đồ khối hình 4.20, ta có các nhóm tín hiệu sau:

➤ Các tín hiệu địa chỉ

Gồm các bit địa chỉ có quan hệ nhất định với các tín hiệu chọn vỏ đầu ra. Nếu có k bit địa chỉ vào thì có $q = 2^k$ tín hiệu chọn vỏ đầu ra, và vi mạch giải mã địa chỉ này được gọi là vi mạch giải mã địa chỉ vào k ra q .

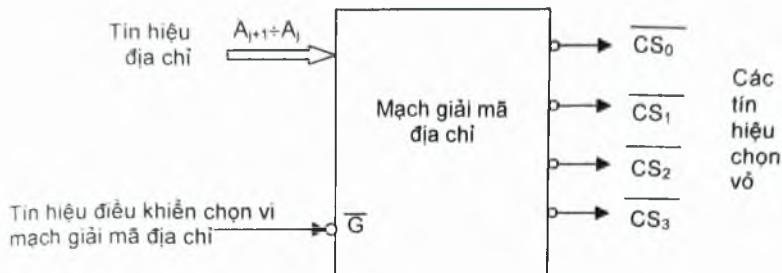
Để giải mã địa chỉ đầy đủ cho một mạch nhớ đòi hỏi phải đưa đến đầu vào của mạch giải mã địa chỉ các tín hiệu địa chỉ sao cho các tín hiệu ở đầu ra của nó chỉ chọn riêng một vi mạch nhớ đã định. Yêu cầu phải dùng tổ hợp đầy đủ các đầu vào địa chỉ tương ứng để chọn được mạch nhớ, nếu bỏ bớt đi 1 bit địa chỉ nào thì sẽ xảy ra trường hợp giải mã thiếu cho mạch nhớ, vì ngoài việc chọn ra vùng nhớ đã định, có thể phải chọn ra các vi mạch nhớ ở các vùng nhớ khác nữa.

Thông thường, khi thiết kế mạch giải mã địa chỉ, người ta thường tính dôi ra một chút để dự phòng sao cho sau này nếu có sự thay đổi do cần phải tăng thêm dung lượng của bộ nhớ thì vẫn có thể sử dụng được mạch giải mã đã thiết kế.

➤ Các tín hiệu điều khiển

Thường là tín hiệu $\overline{IO/M}$ (hoặc IO/\overline{M}) – tuỳ theo từng bộ vi xử lý, nó dùng để phân biệt đối tượng mà CPU chọn làm việc là bộ nhớ hay thiết bị ngoại vi.

Ví dụ: Cho sơ đồ khối mạch giải mã địa chỉ bộ nhớ vào 2 ra 4 (hình 4.21).



Hình 4.21. Sơ đồ khối mạch giải mã địa chỉ bộ nhớ vào 2 ra 4

Và bảng chân lý của nó (hình 4.22):

\bar{G}	A_{j+1}	A_j	\bar{CS}_0	\bar{CS}_1	\bar{CS}_2	\bar{CS}_3
0	0	0	0	1	1	1
	0	1	1	0	1	1
	1	0	1	1	0	1
	1	1	1	1	1	0
1	-	-	1	1	1	1

Hình 4.22. Bảng chân lý của mạch giải mã địa chỉ bộ nhớ vào 2 ra 4

Theo bảng chân lý, khi $\bar{G} = 0$, tại một thời điểm (một giá trị tổ hợp bởi A_{j+1} và A_j) chỉ có một giá trị $\bar{CS}_i = 0$, còn các giá trị \bar{CS}_j khác đều bằng 1, với $\forall j \neq i$. Điều này có nghĩa là, khi giải mã địa chỉ được chọn, tại một thời điểm chỉ có một vi mạch nhớ được chọn trong số các vi mạch nối với giải mã địa chỉ

c) Xây dựng chip nhớ có độ dài ngắn nhớ mong muôn

Từ các chip nhớ 1 bit (độ dài ngắn nhớ 1 bit), ta có thể ghép nối, xây dựng các chip nhớ có độ dài ngắn nhớ 2 bit, 4 bit, 8 bit,... Trong máy tính, đơn vị lưu trữ và vào/ra cơ sở là byte. Do vậy, để xây dựng bộ nhớ, người ta phải phối ghép bộ nhớ thành bộ nhớ bao gồm các ngăn nhớ 8 bit (gọi là 1 ô nhớ).

Khi mở một ô nhớ để đọc hoặc ghi, thì 8 bit nhớ trên 8 chip nhớ với độ rộng ngăn nhớ là 1 bit phải đồng thời được xác định. Do vậy, 8 chip nhớ đưa vào ghép nối phải có chung tín hiệu địa chỉ (ghép chung vào các đường dây tín hiệu địa chỉ).

Đối với các đường dây dữ liệu phải đảm bảo dù 8 đường dây dữ liệu vào/ra, tức là mỗi bit nhớ đưa ra một đường dây dữ liệu riêng biệt, và được sắp xếp đánh số theo thứ tự nhất định (từ D7 – D0), rồi ghép lại thành đường Bus dữ liệu 8 bit.

Khi tín hiệu điều khiển đọc hoặc ghi phát ra, 8 bit nhớ phải đồng thời bị tác động để có thể đồng thời đưa ra hoặc ghi dữ liệu vào 8 bit nhớ. Do vậy, tín hiệu điều khiển đọc/ghi cần phối ghép chung.

Trên bộ nhớ, có nhiều ô nhớ, tại một thời điểm CPU chỉ đọc/ghi một ô nhớ. Do vậy, 8 bit nhớ tạo nên một ô nhớ phải có chung một tín hiệu chọn vỏ (CS – Chip Select) từ bộ giải mã địa chỉ gửi đến.

Ví dụ: Cho các vi mạch nhớ $1K \times 1$ (bit), hãy xây dựng vi mạch nhớ $1K \times 2$ bit.

Giai:

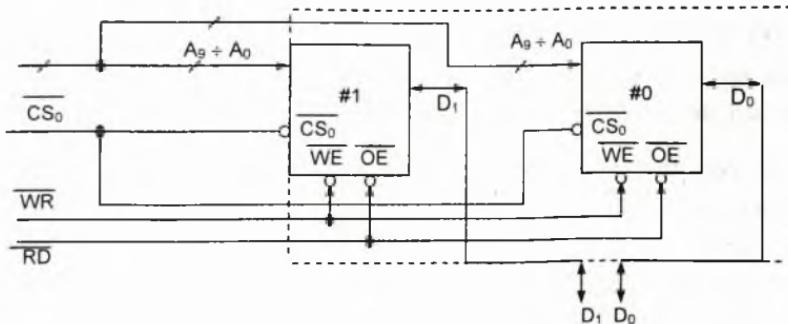
Theo lập luận như trên ta thấy cần có hai chip nhớ $1K \times 1$ bit.

Hai chip nhớ này cần chung tín hiệu địa chỉ, với số đường địa chỉ là $n = 10$ vì số ngăn nhớ là $1K = 2^{10}$. Ký hiệu các đường địa chỉ là $A_9 \div A_0$.

Chip nhớ khi xây dựng xong là chip nhớ $1K \times 2$ bit, nên độ rộng ngăn nhớ là 2 bit, do vậy có hai đường dữ liệu vào/ra (lấy từ hai chip nhớ).

Các tín hiệu điều khiển đọc/ghi chung. Hai ngăn nhớ trên hai chip $1K \times 1$ bit phải đồng thời được chọn, nên chung tín hiệu chọn vỏ CS.

Ta có sơ đồ ghép nối mạch như hình 4.23.



Hình 4.23. Vi mạch $1K \times 2$ bit phối ghép từ hai vi mạch $1K \times 1$ bit

d) Xây dựng módun nhớ có số lượng ngăn nhớ (dung lượng) mong muốn

Từ các chip nhớ 8 bit có dung lượng hạn chế, muốn nâng cao dung lượng của chip nhớ, người ta xây dựng và ghép nối các chip nhớ này (vi mạch nhớ cơ sở) lại

với nhau thành módun nhớ với dung lượng lớn hơn. Cứ như thế người ta được các thành RAM (ví dụ RAM khe cắm SIMM hoặc DIMM).

Các bước tiến hành thiết kế và ghép nối như sau:

- Bước 1: Tính toán số chip nhớ cơ sở cần cho phôi ghép và chọn mạch giải mã địa chỉ.

- Bước 2: Xây dựng bảng chân lý địa chỉ cho vi mạch nhớ cơ sở.

- Bước 3: Xây dựng bảng chân lý địa chỉ cho vi mạch nhớ định xây dựng và ghép nối với máy tính tại địa chỉ xác định (ngầm định là bắt đầu ở địa chỉ 0).

- Bước 4: Đưa ra bảng chân lý cho bộ giải mã địa chỉ.

- Bước 5: Tính toán số linh kiện điện tử cần thiết, như là bộ giải mã địa chỉ loại gì và số lượng là bao nhiêu, số lượng đường dây địa chỉ, số lượng các đường dây dữ liệu, các mạch công phụ trợ khác,... Phân tích dữ liệu đã có, để có thể đưa ra sơ đồ phôi ghép các vi mạch nhớ.

- Bước 6: Vẽ sơ đồ mạch phôi ghép các vi mạch nhớ.

- Bước 7: Xây dựng phôi ghép theo sơ đồ phôi ghép và các linh kiện đã có.

Ví dụ 1: Cho vi mạch nhớ $1K \times 8$ bit, xây dựng vi mạch nhớ $4K \times 8$ bit và phôi ghép với CPU 8086, giải mã địa chỉ tự chọn.

Giai:

- Bước 1: Để có vi mạch nhớ $4K \times 8$ bit, ta phải cần 4 vi mạch nhớ $1K \times 8$ bit. Vì vậy, ta cần một mạch giải mã địa chỉ vào 2 ra 4 để chọn một trong bốn vi mạch nhớ cơ sở tại một thời điểm.

- Bước 2: Xây dựng bảng chân lý cho một vi mạch nhớ cơ sở.

+ Mỗi vi mạch nhớ cơ sở có dung lượng là $1KB = 2^{10}$. Vậy, cần sử dụng 10 bit địa chỉ để quản lý được 2^{10} ô nhớ, tức là 1 KB, miền địa chỉ từ $000h \div 3FFh$.

+ Ta có bảng chân lý không gian địa chỉ bộ nhớ như sau:

A_{j+9}	A_{j+8}	A_{j+7}	A_{j+6}	A_{j+5}	A_{j+4}	A_{j+3}	A_{j+2}	A_{j+1}	A_j
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	1	0
...
1	1	1	1	1	1	1	1	0	1
1	1	1	1	1	1	1	1	1	0
1	1	1	1	1	1	1	1	1	1

- Bước 3: Xây dựng bảng chân lý cho vi mạch nhớ 4KB tại địa chỉ 0 (ngầm định).

+ Vi mạch sau phôi ghép có dung lượng là $4KB = 2^2 \times 2^{10} = 2^{12}$. Vậy, cần sử dụng 12 bit địa chỉ để quản lý được 2^{12} ô nhớ, tức là 4 KB, miền địa chỉ từ 000h ÷ FFFh.

+ Ta có bảng chân lý không gian địa chỉ bộ nhớ như sau:

A₁₁	A₁₀	A₉	A₈	A₇	A₆	A₅	A₄	A₃	A₂	A₁	A₀
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	1	0
...
0	0	1	1	1	1	1	1	1	1	0	1
0	0	1	1	1	1	1	1	1	1	1	0
0	0	1	1	1	1	1	1	1	1	1	1
0	1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	0	1
0	1	0	0	0	0	0	0	0	0	1	0
0	1
0	1	1	1	1	1	1	1	1	1	0	1
0	1	1	1	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	1	1	1	1
1	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1
1	0	0	0	0	0	0	0	0	0	1	0
1	0
1	0	1	1	1	1	1	1	1	1	0	1
1	0	1	1	1	1	1	1	1	1	1	0
1	0	1	1	1	1	1	1	1	1	1	1
1	1	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	1
1	1	0	0	0	0	0	0	0	0	1	0
1	1
1	1	1	1	1	1	1	1	1	1	0	1
1	1	1	1	1	1	1	1	1	1	1	0
1	1	1	1	1	1	1	1	1	1	1	1

- Bước 4: Xây dựng giải mã địa chỉ bộ nhớ.

+ So sánh dài địa chỉ bộ nhớ và dài địa chỉ trong mỗi vi mạch nhớ ta thấy giá trị các bit địa chỉ từ A_9, \dots, A_0 trùng với giá trị các bit địa chỉ A_{j+9}, \dots, A_j lặp đi lặp lại, vậy các bit địa chỉ A_9, \dots, A_0 đưa vào các vi mạch nhớ cơ sở.

+ Giá trị trong các bit địa chỉ A_{11}, A_{10} hoàn toàn phù hợp giá trị các bit địa chỉ vào giải mã địa chỉ vào 2 ra 4 đã chọn, do vậy các bit địa chỉ A_{11}, A_{10} được đưa vào vào bộ giải mã địa chỉ để xác định ra chip nhớ cơ sở duy nhất cần làm việc tại một thời điểm.

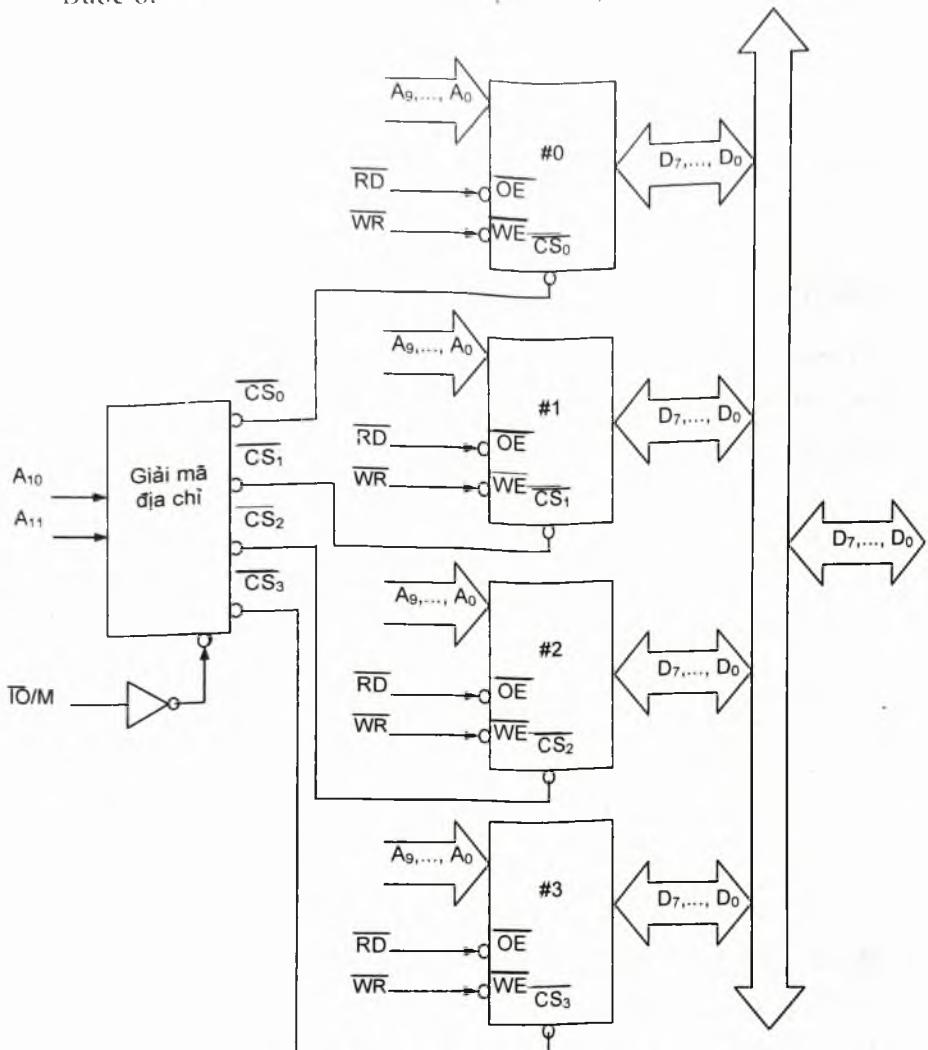
+ Vì mạch giải mã địa chỉ bộ nhớ này chỉ được chọn khi tín hiệu $\overline{IO/M} = 1$, nên ta đưa tín hiệu này vào giải mã địa chỉ phải qua mạch đảo.

Ta có bảng chân lý cho giải mã địa chỉ vào 2 ra 4 như sau:

$\overline{G} = \overline{IO/M}$	A_{11}	A_{10}	\overline{CS}_0	\overline{CS}_1	\overline{CS}_2	\overline{CS}_3
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0
1	-	-	1	1	1	1

- Bước 5: Số đường dây địa chỉ vào vi mạch $4K \times 8$ bit là 12 bit, trong đó: A_{11}, A_{10} vào giải mã địa chỉ; A_9, \dots, A_0 cùng vào các vi mạch nhớ cơ sở, số đường dây dữ liệu vào/ra là 8 bit (nối song song) cho các vi mạch nhớ cơ sở, số mạch giải mã địa chỉ vào 2 ra 4 là 1, số đường dây chọn vỏ cho 4 vi mạch nhớ cơ sở là 4 (mỗi vi mạch nhớ cơ sở 1 đường) nối từ 4 đầu ra giải mã địa chỉ sang các vi mạch nhớ cơ sở, đường dây chọn vỏ cho giải mã địa chỉ (cho vi mạch $4K \times 8$ bit) là 1, số đường dây điều khiển đọc là 1 cùng vào 4 vi mạch nhớ cơ sở, đường dây điều khiển ghi là 1 cùng vào 4 vi mạch nhớ cơ sở. Trường hợp này chỉ cần thêm 1 mạch đảo.

- Bước 6: Vẽ sơ đồ mạch phối ghép (hình 4.24).



Hình 4.24. Sơ đồ mạch phối ghép

Ví dụ 2: Thiết kế mạch ghép nối giữa 8086 và các IC nhớ SRAM 1K × 8 bit để thành bộ nhớ có dung lượng mở rộng thêm 4K × 8 bit. Địa chỉ bắt đầu của vùng nhớ là AD000h, giải mã địa chỉ tự chọn. Tín hiệu chọn bộ nhớ để làm việc của CPU là $\overline{IO/M} = 1$.

Giải:

- Bước 1: Xác định số chip nhớ cơ sở cần mắc nối tiếp là

$$n = \frac{\text{Dung lượng theo yêu cầu}}{\text{Dung lượng của 1 IC nhớ}} = \frac{4K}{1K} = 4$$

Để chọn một trong bốn vi mạch cần làm việc tại một thời điểm, ta chọn giải mã địa chỉ vào 2 ra 4.

- Bước 2: Xây dựng bảng chân lý cho một vi mạch nhớ cơ sở.

+ Mỗi vi mạch nhớ cơ sở có dung lượng là $1KB = 2^{10}$. Vậy cần sử dụng 10 bit địa chỉ để quản lý được 2^{10} ô nhớ, tức 1KB, miền địa chỉ từ $000h$ đến $3FFh$.

- Ta có bảng chân lý không gian địa chỉ tương đối cho vi mạch nhớ cơ sở như sau:

A_{j+9}	A_{j+8}	A_{j+7}	A_{j+6}	A_{j+5}	A_{j+4}	A_{j+3}	A_{j+2}	A_{j+1}	A_j
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	1	0
...
1	1	1	1	1	1	1	1	0	1
1	1	1	1	1	1	1	1	1	0
1	1	1	1	1	1	1	1	1	1

- Bước 3: Xác định dài địa chỉ của bộ nhớ.

+ Địa chỉ tương đối lớn nhất của vùng bộ nhớ 4K sẽ phải ghép thêm là $FFFh$.

+ Địa chỉ vật lý bắt đầu của vùng nhớ 4K phải ghép thêm là $AD000h$.

+ Địa chỉ vật lý kết thúc của vùng nhớ 4K phải ghép thêm là $ADFFFh$.

Vậy tổng số đường địa chỉ cho phôi ghép là 20 đường (từ $A19$ đến $A0$).

Ta có bảng chân lý cho vùng nhớ 4K phôi ghép thêm là

- Bước 4: Xây dựng bảng chân lý giải mã địa chỉ bộ nhớ.
 - + So sánh dài địa chỉ bộ nhớ và dài địa chỉ trong mỗi vi mạch nhớ ta thấy giá trị các bit địa chỉ từ A_9, \dots, A_0 trùng với A_{j+9}, \dots, A_j lặp đi lặp lại. Vậy ta đưa các bit địa chỉ từ A_9, \dots, A_0 vào các vi mạch nhớ cơ sở để đánh địa chỉ tương đối trong các vi mạch nhớ này.
 - + Hai giá trị trong các bit địa chỉ A_{11}, A_{10} hoàn toàn phù hợp giá trị địa chỉ vào giải mã địa chỉ vào 2 ra 4 đã chọn. Do vậy 2 bit A_{11}, A_{10} là giá trị địa chỉ đưa vào bộ giải mã địa chỉ để tạo ra 4 tín hiệu đầu ra đèn chân chọn chip trên các chip nhớ cơ sở.
 - + Cuối cùng vùng nhớ 4K phôi ghép thêm này phải đảm bảo chỉ được chọn khi tập các bit địa chỉ $A_{19}, \dots, A_{12} = 1010\ 1101b$, tức là giải mã địa chỉ vào 2 ra 4 mà ta dùng để chọn ra 1 chip nhớ cơ sở cho phôi ghép này chỉ được chọn khi các bit A_{18}, A_{16}, A_{13} đồng thời bằng 0 và các bit $A_{19}, A_{17}, A_{15}, A_{14}, A_{12}$ đồng thời bằng 1. Vậy các tín hiệu A_{18}, A_{16}, A_{13} sẽ qua các mạch đảo rồi đưa vào mạch NAND cùng với các tín hiệu $A_{19}, A_{17}, A_{15}, A_{14}, A_{12}$ và $\overline{IO/M}$ để tạo ra tín hiệu chọn chip cho mạch giải mã địa chỉ này.

Ta có bảng chân lý cho giải mã địa chỉ vào 2 ra 4 như sau:

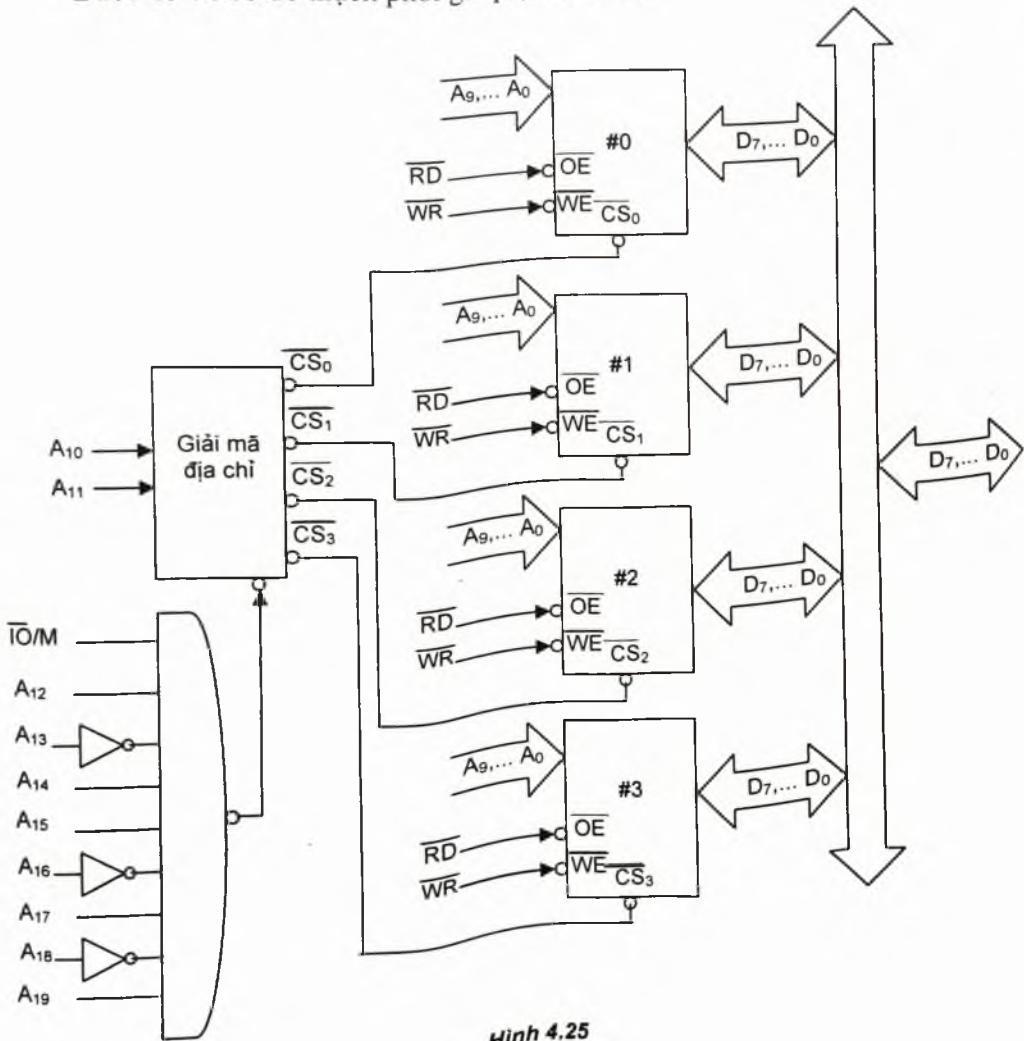
Đầu vào mạch NAND ($ra = \overline{G}$)									A		Chọn vòi vi mạch nhớ			
$\overline{IO/M}$	A_{19}	$\overline{A_{18}}$	A_{17}	$\overline{A_{16}}$	A_{15}	A_{14}	$\overline{A_{13}}$	A_{12}	A_{11}	A_{10}	\overline{CS}_0	\overline{CS}_1	\overline{CS}_2	\overline{CS}_3
1	1	1	1	1	1	1	1	1	0	0	0	1	1	1
1	1	1	1	1	1	1	1	1	0	1	1	0	1	1
1	1	1	1	1	1	1	1	1	1	0	1	1	0	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0	-	-	-	-	-	-	-	-	-	-	1	1	1	1

- Bước 5: Phân tích.

Số đường dây địa chỉ vào vi mạch $4K \times 8$ bit là 12 bit, trong đó A_{11}, A_{10} vào giải mã địa chỉ; A_9, \dots, A_0 cùng vào các vi mạch nhớ cơ sở, số đường dây dữ liệu

vào/ra là 8 bit (nội song song) cho các vi mạch nhớ cơ sở, số mạch giải mã địa chỉ vào 2 ra 4 là 1, số đường dây chọn vò cho 4 vi mạch nhớ cơ sở là 4 (mỗi vi mạch nhớ cơ sở 1 đường) nối từ 4 đầu ra giải mã địa chỉ sang các vi mạch nhớ cơ sở, đường dây chọn vò cho giải mã địa chỉ (cho vi mạch $4K \times 8$ bit) là 1, số đường dây điều khiển đọc là 1 cùng vào 4 vi mạch nhớ cơ sở, đường dây điều khiển ghi là 1 cùng vào 4 vi mạch nhớ cơ sở. Trường hợp này cần 1 mạch NAND và 3 mạch đảo trên các đường dây địa chỉ A_{18}, A_{16}, A_{13} trước khi đưa vào mạch NAND.

- Bước 6: Vẽ sơ đồ mạch phối ghép (hình 4.25).



Hình 4.25

e) Tổ chức bank nhớ cho bộ nhớ

Trong thực tế, máy tính có các đường dây dữ liệu không phải là độ rộng 8 bit mà thường là 16 bit, 32 bit,... Nếu là máy tính có Bus D là 16 bit thì cho phép ghép nối vào/ra tại một thời điểm với 2 byte trong bộ nhớ, nếu máy tính có Bus D là 32 bit dữ liệu cho phép ghép nối vào/ra tại một thời điểm với 4 byte trong bộ nhớ,...

Tổ chức bộ nhớ thành các *bank* nhớ là một phương pháp hữu hiệu cho việc tổ chức vào/ra dữ liệu giữa Bus dữ liệu và bộ nhớ.

Bộ nhớ RAM của hệ thống được chia thành 2 hoặc 4 bank (hoặc hơn). Việc đánh địa chỉ được đan xen lùn lượt giữa các bank.

Nếu tổ chức làm 2 bank thì sẽ có bank#0 và bank#1, địa chỉ các ô nhớ trên bank#0 là các giá trị địa chỉ chẵn đánh số là $2 \times i$ [$i = 0 \div (n - 1)$, n là số ô nhớ trong một bank], địa chỉ các ô nhớ trong bank#1 là các giá trị địa chỉ lẻ được đánh số là $2 \times i + 1$. Như vậy, các byte dữ liệu sẽ được cắt giữ lùn lượt liên tiếp trong các bank là byte 0 trong bank#0, byte 1 trong bank#1, byte 2 trong bank#0, byte 3 trong bank#1, byte 4 trong bank#0,....

Tương tự như trên, nếu tổ chức làm 4 bank thì sẽ có bank#0, bank#1, bank#2 và bank#3. Địa chỉ các ô nhớ trên bank#0 là các giá trị địa chỉ chẵn đánh số là $4 \times i$ [$i = 0 \div (n - 1)$, n là số ô nhớ trong một bank], địa chỉ các ô nhớ trong bank#1 là các giá trị địa chỉ lẻ được đánh số là $4i + 1$, địa chỉ các ô nhớ trên bank#2 là các giá trị địa chỉ chẵn đánh số là $4 \times i + 2$, địa chỉ các ô nhớ trong bank#3 là các giá trị địa chỉ lẻ được đánh số là $4 \times i + 3$. Như vậy, các byte dữ liệu sẽ được cắt giữ lùn lượt liên tiếp trong các bank là byte 0 trong bank#0, byte 1 trong bank#1, byte 2 trong bank#2, byte 3 trong bank#3, byte 4 trong bank#0, byte 5 trong bank#1, byte 6 trong bank#2, byte 7 trong bank#3,...

Khi truy nhập bộ nhớ, CPU sẽ bước lùn lượt từ bank này sang bank kia. Khi CPU đang đọc bank này thì bank kia được truy cập trước để không phải chờ. Trường hợp dữ liệu cần truy cập không nằm theo thứ tự, thì phải chèn thêm các trạng thái chờ cho đến khi bắt đầu chạy được theo thứ tự luân phiên mới. Bộ nhớ xen nhau tăng tốc độ hoạt động của hệ lên đến 75%, đồng thời sử dụng được các vi mạch nhớ loại cũ.

Ví dụ: Cho vi mạch nhớ $1K \times 8$ bit, tổ chức bộ nhớ phôi ghép với CPU 8086 thành 2 bank nhớ với tổng dung lượng là $4K \times 8$ bit. Hãy giải mã địa chỉ tự chọn.

Giai:

– Bước 1: Xác định số vi mạch nhớ cơ sở.

+ Ta thấy cần 4 vi mạch nhớ cơ sở $1K \times 8$ bit để có bộ nhớ sau phôi ghép là $4K \times 8$ bit. Để tổ chức thành 2 bank nhớ thì mỗi bank nhớ có dung lượng là $2K \times 8$ bit. Như vậy, mỗi bank nhớ sẽ được phôi ghép trước thành vi mạch nhớ $2K \times 8$ bit từ 2 vi mạch nhớ cơ sở $1K \times 8$ bit và mỗi vi mạch $2K \times 8$ bit sẽ được bố trí thành một bank nhớ, ký hiệu là bank#0 và bank#1.

+ Việc phôi ghép 2 vi mạch nhớ $1K \times 8$ bit để thành vi mạch $2K \times 8$ bit này cần một giải mã địa chỉ vào 1 ra 2. Do vậy, để có 2 vi mạch $2K \times 8$ bit từ các vi mạch $1K \times 8$ bit ta cần 2 mạch giải mã vào 1 ra 2. Và mỗi vi mạch $2K \times 8$ bit sẽ có 11 đường địa chỉ vào, trong đó 10 đường địa chỉ vào vi mạch nhớ cơ sở còn một đường địa chỉ cao nhất vào mạch giải mã địa chỉ để đưa ra tín hiệu chọn vi mạch nhớ cơ sở.

– Bước 2: Bảng chân lý cho địa chỉ bộ nhớ.

+ Bộ nhớ sau phôi ghép có dung lượng là $4KB = 2^2 \times 2^{10} = 2^{12}$. Vậy cần sử dụng 12 bit địa chỉ để quản lý được 2^{12} ô nhớ, tức là 4KB, miền địa chỉ từ $000h \div FFFh$.

+ Ta có bảng chân lý không gian địa chỉ bộ nhớ như sau:

A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	0	1	1
***	***	***	***	***	***	***	***	***	***	***	***
1	1	1	1	1	1	1	1	1	1	0	0
1	1	1	1	1	1	1	1	1	1	0	1
1	1	1	1	1	1	1	1	1	1	1	0
1	1	1	1	1	1	1	1	1	1	1	1

– Bước 3: Xác định đường địa chỉ vào mỗi bank nhớ và giải mã địa chỉ chọn bank nhớ.

Tại một thời điểm CPU chỉ truy cập (đọc/ghi) một ô nhớ trên một bank nhớ nào đó, tức là chỉ có một bank nhớ được chọn làm việc. Vậy ta phải chọn giải mã địa chỉ vào 1 ra 2 để phát ra tín hiệu chọn bank nhớ làm việc; đồng thời ta thấy bit A_0 quyết định tính chẵn lẻ của giá trị địa chỉ. Nếu $A_0 = 0$ thì giá trị địa chỉ chẵn, ngược lại $A_0 = 1$ thì giá trị địa chỉ là lẻ. Như vậy, ta đưa tín hiệu A_0 vào mạch giải mã địa chỉ chọn bank nhớ thì đảm bảo việc đánh địa chỉ đan xen là các ô nhớ trên bank#0 luôn có giá trị địa chỉ chẵn, các ô nhớ trên bank#1 luôn có giá trị địa chỉ lẻ.

– Chân chọn vỏ của bộ giải mã địa chỉ này được nối với tín hiệu điều khiển $\overline{IO/M}$ từ CPU 8086 đã qua mạch đảo.

– Ta có bảng chân lý của mạch giải mã địa chỉ vào 1 ra 2 cho chọn bank như sau:

$\overline{G} = \overline{IO/M}$	A_0	\overline{BE}_0	\overline{BE}_1
0	0	0	1
0	1	1	0
1	-	1	1

Trong đó BE_0 , BE_1 là chân tín hiệu chọn vỏ tương ứng cho bank#0 và bank#1 (BE – Bank Enable).

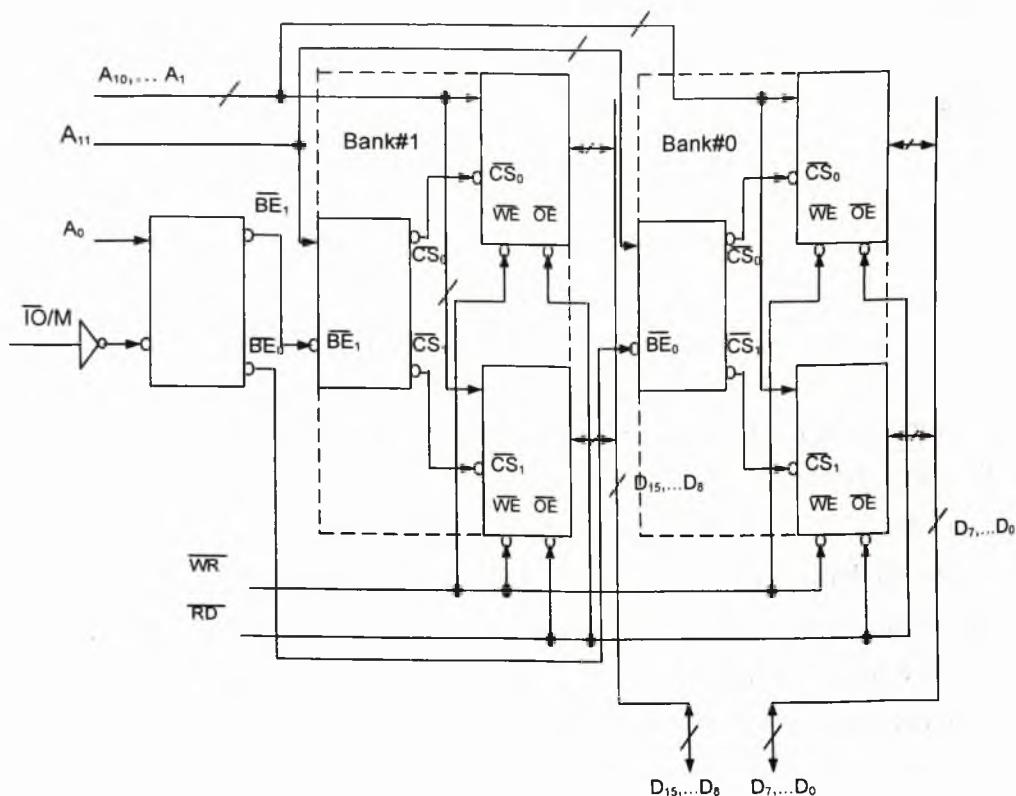
– Nhìn vào bảng chân lý không gian địa chỉ bộ nhớ ở trên và bảng chân lý cho giải mã địa chỉ chọn bank nhớ, các chân tín hiệu địa chỉ còn lại từ A_{11} , ..., A_1 sẽ được đưa vào các bank nhớ, trong đó A_{10} , ..., A_1 được đưa vào các chip nhớ cơ sờ $1K \times 8$ bit, còn đường A_{11} đưa vào giải mã vào 1 ra 2 để đưa ra tín hiệu chọn vỏ cho các chip nhớ cơ sờ trong bank.

– Bước 4: Xác định các đường dây ghép nối.

Tổng số đường dây địa chỉ cần là 12 đường. A_0 đưa vào giải mã địa chỉ vào 1 ra 2 để chọn bank nhớ, A_{11} đưa vào giải mã địa chỉ vào 1 ra 2 để chọn vi mạch nhớ cơ sờ $1K \times 8$ bit trong mỗi bank nhớ, các đường dây địa chỉ còn lại được đưa

vào các vi mạch nhớ cơ sở $1K \times 8$ bit để xác định ô nhớ cần trao đổi dữ liệu. Số đường dây dữ liệu vào/ra là 16 bit (ghép song song hai đường 8 bit ra từ các bank nhớ). Số mạch giải mã địa chỉ vào 1 ra 2 là 3, số đường dây chọn vỏ cho 4 vi mạch nhớ cơ sở là 4 (mỗi vi mạch nhớ cơ sở 1 đường) nối từ các đầu ra của 2 bộ giải mã địa chỉ sang các vi mạch nhớ cơ sở, đường dây chọn vỏ để chọn bank nhớ (nối từ đầu ra của giải mã địa chỉ bank nhớ tới chân tín hiệu chọn vỏ của giải mã địa chỉ thuộc từng bank nhớ) là 2. Số đường dây điều khiển đọc là 1 cùng vào 4 vi mạch nhớ cơ sở, đường dây điều khiển ghi là 1 cùng vào 4 vi mạch nhớ cơ sở. Trường hợp này không cần các mạch công phụ trợ, chỉ cần một mạch NOT.

– Bước 5: Vẽ sơ đồ mạch phối ghép (hình 4.26).



Hình 4.26

Ghi chú: Các linh kiện trong phần kèn nét đứt thuộc về một bank nhớ.

4.3. CÁC VẤN ĐỀ VỀ THIẾT KẾ BỘ NHỚ

Trong thiết kế bộ nhớ, các vấn đề cần được quan tâm là: tốc độ bộ nhớ so với tốc độ CPU, vùng địa chỉ bộ nhớ, tốc độ và giá thành.

4.3.1. Tốc độ bộ nhớ so với tốc độ CPU

Tốc độ của bộ nhớ thường nhỏ hơn rất nhiều so với CPU, do vậy khi trao đổi dữ liệu, CPU thường phải chèn thêm các chu kỳ đợi. Nếu tốc độ bộ nhớ càng thấp thì chu kỳ đợi với khoảng thời gian đợi càng dài, hiệu năng làm việc của CPU càng thấp.

Nếu bộ nhớ chính được chế tạo từ các chip nhớ SRAM thì tốc độ vào/ra dữ liệu với CPU tương đối phù hợp, song ta thấy khả năng tích hợp các chip nhớ gấp nhiều khó khăn do một phần tử nhớ SRAM (1 bit) sử dụng nhiều transistor.

Nếu sử dụng các chip nhớ DRAM làm bộ nhớ chính thì khả năng tích hợp các chip nhớ trên vi mạch nhớ dễ dàng hơn. Cho phép dung lượng chip nhớ cao hơn nhiều lần, tuy nhiên tốc độ bộ nhớ chính lại chậm đi nhiều lần.

Do vậy, để đáp ứng được vấn đề tốc độ bộ nhớ so với CPU và nâng cao dung lượng bộ nhớ nên một giải pháp đã được đưa ra là đặt bộ nhớ cache (chế tạo từ các phần tử nhớ SRAM) giữa CPU và bộ nhớ chính (chế tạo từ các phần tử nhớ DRAM).

4.3.2. Vùng địa chỉ bộ nhớ

Vùng địa chỉ bộ nhớ phải phù hợp với không gian nhớ mà CPU có thể quản lý được, tức là dung lượng bộ nhớ không được phép vượt quá không gian nhớ do CPU quản lý. Mặt khác, khi thiết kế phôi ghép các loại bộ nhớ trong máy tính phải đảm bảo giá trị địa chỉ nhớ tăng tuyến tính, địa chỉ của mỗi đối tượng truy cập là duy nhất và không có sự trùng lặp một giá trị địa chỉ cho hai đối tượng.

4.3.3. Tốc độ và giá thành

Nếu sử dụng bộ nhớ SRAM làm bộ nhớ chính thì tuy tốc độ cao, song dung lượng lại không cho phép lớn và giá thành đắt. Nếu sử dụng bộ nhớ DRAM làm bộ nhớ chính thì cho phép xây dựng bộ nhớ dung lượng cao, giá thành hạ, song tốc độ lại chậm, dẫn đến hiệu năng của hệ thống bị hạn chế.

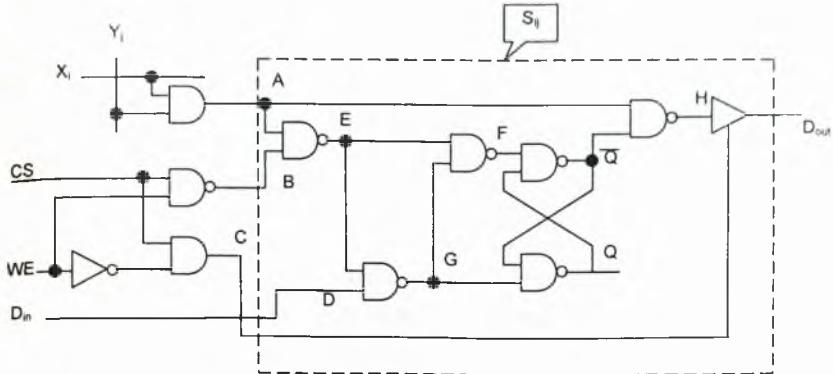
Chính vì vậy, giải pháp trung hoà là đưa bộ nhớ cache (chế tạo từ SRAM) làm bộ nhớ đệm với bộ nhớ chính (chế tạo từ DRAM).

CÂU HỎI VÀ BÀI TẬP CHƯƠNG 4

1. Cho chip nhớ $1K \times 1$ bit, xây dựng chip nhớ $1K \times 4$ bit.
2. Cho vi mạch nhớ $1K \times 4$ bit, xây dựng chip nhớ $1K \times 8$ bit.
3. Thiết kế mạch ghép nối giữa 8086 và các IC nhớ SRAM $4K \times 4$ bit để thành bộ nhớ $4K \times 8$ bit. Địa chỉ đầu của bộ nhớ là 7C000h.
4. Thiết kế mạch ghép nối giữa 8086 và các IC nhớ SRAM $4K \times 8$ bit để thành bộ nhớ $4K \times 16$ bit. Địa chỉ đầu của bộ nhớ là 7C000h. Giải mã địa chỉ tự chọn.
5. Cho vi mạch nhớ $2K \times 8$ bit, xây dựng chip nhớ $6K \times 8$ bit, mạch giải mã địa chỉ và các linh kiện khác tự chọn.
6. Cho chip nhớ $2K \times 1$ bit, xây dựng chip nhớ $6K \times 8$ bit, mạch giải mã địa chỉ và các linh kiện khác tự chọn.
7. Thiết kế mạch ghép nối giữa 8086 và chip nhớ RAM $1K \times 8$ bit để thành chip nhớ $6K \times 8$ bit, biến địa chỉ bắt đầu là ABC00h. Giải mã địa chỉ tự chọn.
8. Cho vi mạch nhớ $8K \times 8$ bit, xây dựng phôi ghép bộ nhớ $64K \times 8$ bit với CPU 8086 và tổ chức thành 4 bank nhớ.
9. Cho vi mạch nhớ $8K \times 8$ bit, xây dựng phôi ghép bộ nhớ $192K \times 8$ bit với CPU 8086 và tổ chức thành 6 bank nhớ.
10. CPU có 24 bit địa chỉ, bộ nhớ chính 256KB chia làm 512 block nhớ, bộ nhớ cache có dung lượng 8KB. Khi CPU được lệnh phát ra địa chỉ truy nhập bộ nhớ là B7281Ah, BF0825h, 2F1025h, 2F0825h. Hãy trình bày chi tiết phương pháp đọc cache theo kỹ thuật ánh xạ trực tiếp cho hai trường hợp phát ra địa chỉ trên và địa chỉ ô nhớ cần truy cập trong block theo từng trường hợp.
11. CPU có 24 bit địa chỉ, bộ nhớ chính 256KB chia làm 512 block nhớ, bộ nhớ cache có dung lượng 8KB. Khi CPU được lệnh phát ra địa chỉ truy nhập bộ nhớ là 7280Ah, F0812h. Hãy trình bày chi tiết phương pháp đọc cache theo kỹ thuật ánh xạ liên kết hoàn toàn cho hai trường hợp phát ra địa chỉ trên và chỉ ra địa chỉ ô nhớ cần truy cập trong bộ nhớ theo từng trường hợp.
12. CPU có 24 bit địa chỉ, bộ nhớ chính 256KB chia làm 512 block nhớ, bộ nhớ cache có dung lượng 8KB chia làm 4 set. Khi CPU được lệnh phát ra địa chỉ truy nhập bộ nhớ là 95418h, 132415h, 72426h. Hãy trình bày chi tiết phương pháp đọc cache cho ba trường hợp phát ra địa chỉ trên và địa chỉ vật lý của ô nhớ cần truy nhập tương ứng.

Cho phân tử nhớ sau (hình 4.27). Chứng minh rằng khi:

- $Y_j = 1, X_i = 1, WE = 1$ và $CS = 1$ thì $Q = D_{in}$, đồng thời H và ngắt mạch.
- $Y_j = 1, X_i = 1, WE = 0/1$ và $CS = 0$ thì Q không phụ thuộc D_{in} , đồng thời H và D_{out} ở trạng thái trở kháng cao (ngắt mạch).



Hình 4.27

14. Trình bày phương pháp tổ chức bank nhớ cho bộ nhớ.

15. Tại sao phải dùng bộ nhớ ảo. Sự khác biệt giữa cache và bộ nhớ ảo.

16. Giả sử bộ nhớ vật lý có dung lượng 64MB, chương trình gồm 4 módun, xác định địa chỉ truy cập trong chiến lược phân trang – phân đoạn, với:

SCB			PCB ₀		PCB ₁		PCB ₂		PCB ₃	
P	A	L	P _p	A _p						
1	400400h	4	1	405h	1	407h	1	40Ah	1	40Ch
1	400800h	4	0	-	1	408h	1	40Bh	1	40Dh
1	400C00h	4	1	406h	0	-	0	-	1	40Fh
1	401000h	4	0	-	1	409h	0	-	0	-

Biết kích thước 1 trang là 4KB, địa chỉ đầu của bảng phân đoạn $Rs = 400000h$.

Xác định địa chỉ vật lý cần truy cập theo địa chỉ logic sau:

<0, 0, 2Bh>; <0, 1, 7Ah>.

17. Trình bày các nguyên nhân chính gây thất bại khi truy nhập cache.
18. Trình bày phương pháp tăng độ dài ngăn nhớ cho vi mạch nhớ – có ví dụ.
19. Trình bày phương pháp tăng số lượng ô nhớ cho vi mạch nhớ (mở rộng dung lượng bộ nhớ).
20. Giả sử bộ nhớ vật lý có dung lượng 64MB, chương trình gồm 4 módun, xác định địa chỉ truy cập trong chiến lược phân trang – phân đoạn, với:

SCB			PCB ₀		PCB ₁		PCB ₂		PCB ₃	
P	A	L	P _p	A _p						
1	400400h	4	1	405h	1	407h	1	40Ah	1	40Ch
1	400800h	4	0	-	1	408h	1	40Bh	1	40Dh
1	400C00h	4	1	406h	0	-	0	-	1	40Fh
1	401000h	4	0	-	1	409h	0	-	0	-

Biết kích thước 1 trang là 4KB, địa chỉ đầu của bảng phân đoạn Rs = 400000h.

Xác định địa chỉ vật lý cần truy cập theo địa chỉ logic sau:

<1, 0, 46h>; <1, 1, 1001h>; <2, 1, 1000h>; <4, 1, 200h>; <3, 4, 57h>.

21. Giả sử bộ nhớ vật lý có kích thước 32 byte, chia làm 8 trang vật lý. Chương trình có kích thước 16 byte có nội dung '0123456789ABCDEF'. Và bảng quản lý trang (PCB) có nội dung sau:

$$N_p = 2 \quad d = 3.$$

Nhàm thuộc của một trang
L = 32 byte / 8 trang = 4.

$$N_p = 2 \rightarrow P = d, A = 3.$$

Tính NP L số trang logic
 $\Leftrightarrow (2 \times 4) (16/4)$

Số Sanh D < L (3 < 4)

P	A
1	7
0	-
1	3
1	5

Ghép A số nr DR d
d = A * L + d
= 3 * 4 + 3 = 15

Nội dung ô nhớ là B

– Xác định địa chỉ vật lý truy nhập tương ứng với các địa chỉ logic sau:

<2, 4>; <1, 2>; <0, 3>.

– Cho biết nội dung các ô nhớ tương ứng với các địa chỉ trên.

22. Giả sử bộ nhớ vật lý có kích thước 32 byte, chia làm 8 trang vật lý. Chương trình có kích thước 16 byte có nội dung '0123456789ABCDEF'. Và bảng quản lý trang (PCB) có nội dung sau:

P	A
1	7
1	3
0	-
1	5

- Xác định địa chỉ vật lý truy nhập tương ứng với các địa chỉ logic sau:
 $\langle 4, 2 \rangle; \langle 1, 2 \rangle; \langle 2, 3 \rangle.$

- Cho biết nội dung các ô nhớ tương ứng với các địa chỉ trên.

23. Giả sử bộ nhớ vật lý có dung lượng 512MB, chương trình gồm 5 módun, xác định địa chỉ vật lý tương ứng với các địa chỉ logic phát ra là $\langle 2, 43h \rangle, \langle 3, 4FFh \rangle, \langle 4, 600h \rangle, \langle 5, 100h \rangle$, cho biết bảng quản lý phân đoạn như sau:

P	A	L
1	100h	200h
0	-	400h
1	300h	700h
0	-	500h
1	A00h	600h

Giai: $s = 432, d = 43.h$

$s = 2$ ~~700~~ ~~400~~ ~~300~~ ~~200~~ ~~100~~ có $P = 1, A = 300h, L = 700h.$

so sánh $d < L$ ($43h < 700h$) thỏa mãn.

Tính địa chỉ vật lý: $A + d = 300h + 43h = 343h.$

Chương 5

KỸ THUẬT ĐƯỜNG ỐNG VÀ RISC

TÓM TẮT: – Lý thuyết: 6 tiết;
– Bài tập lớn: 6 tiết.

Mục tiêu cần đạt được	Các mục chính	Bài tập bắt buộc	Bài tập làm thêm
<ul style="list-style-type: none">- Cung cấp, giới thiệu một số công nghệ, kỹ thuật liên quan đến kiến trúc máy tính hiện đại ngày nay như: kỹ thuật đường ống (pipeline), kiến trúc RISC.- Kết thúc chương, sinh viên cần nắm tư tưởng, kỹ thuật pipeline. Nắm được các đặc điểm, lợi thế của kiến trúc RISC so với kiến trúc CISC.	<ul style="list-style-type: none">4.1. Kỹ thuật đường ống.4.2. Mạch xử lý vectơ ống.4.3. Máy tính với tập lệnh đơn giản hóa.	<ul style="list-style-type: none">Câu hỏi và bài tập cuối chương: 1.	<p>Không</p>

5.1. KỸ THUẬT ĐƯỜNG ỐNG

Đường ống (Pipeline) là một trong các kỹ thuật để tăng tốc của máy tính, tận dụng tối đa thời gian CPU. Kỹ thuật này dựa trên ý tưởng: chia một lệnh ra làm nhiều công đoạn và cho xử lý các công đoạn gói lên nhau (giống như một dây truyền sản xuất). Để thực hiện điều này, cần xây dựng phần cứng bao gồm một số đơn vị chức năng và liên kết chúng thành một "đường ống". Hình 5.1 mô tả ý tưởng triển khai kỹ thuật pipeline và hình 5.2 mô tả mô hình kiến trúc phần cứng thực thi kỹ thuật pipeline.

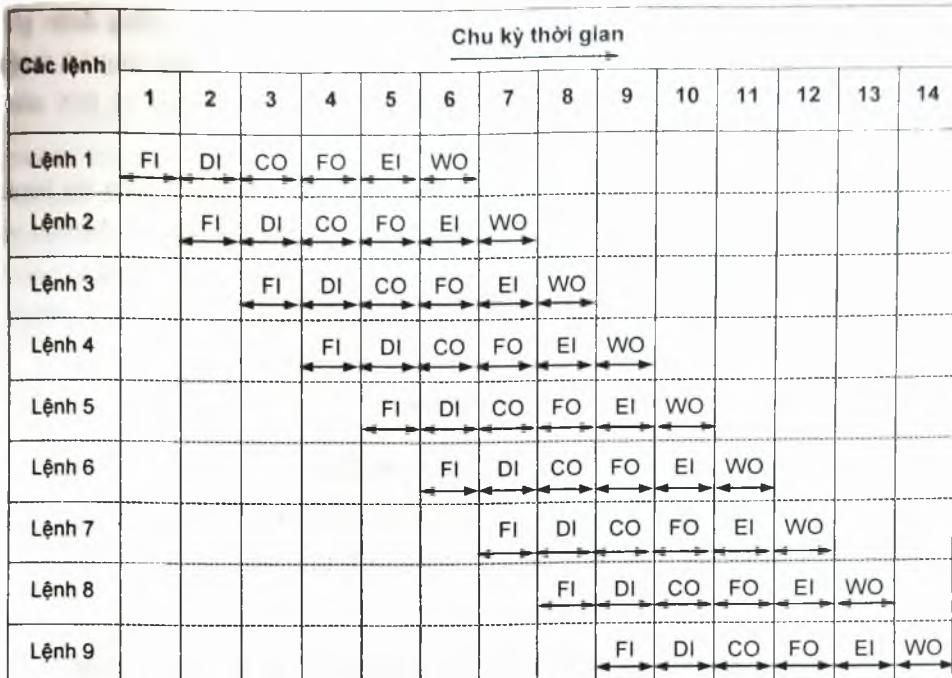
Giả sử một lệnh được chia thành các công đoạn như sau:

- Nhận lệnh (Fetch Instruction – FI): Đọc lệnh cần thi hành vào bộ đệm.
- Giải mã lệnh (Decode Instruction – DI): Xác định thao tác (opcode) của lệnh và toán hạng tham gia trong lệnh.
- Tính địa chỉ toán hạng (Calculate Operands – CO): Tính toán địa chỉ vật lý của từng toán hạng trong lệnh.

hận toán hạng (Fetch Operands – FO): Nhận từng toán hạng từ bộ nhớ,
các toá. trong các thanh ghi thì không cần nhận.

– **Thi hành lệnh (Execute Instruction – EI):** Thực hiện thao tác đã xác định và
lưu trữ kết quả vào toán hạng đích nếu vị trí toán hạng đích đã được xác định.

– **Ghi toán hạng (Write Operand – WO):** Ghi kết quả ra bộ nhớ.

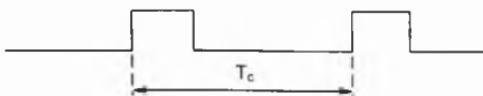


Hình 5.1. Một đường ống gồm 6 công đoạn

Trên hình 5.1 cho thấy nếu chỉ thị 1 được lấy về trong chu kỳ 1 thì nó được giải mã lệnh ở chu kỳ 2; đồng thời tại chu kỳ 2, lệnh 2 cũng được đọc vào. Tại chu kỳ 5, lệnh 1 được thi hành và lệnh 2 đang ở công đoạn lấy toán hạng,... Trong điều kiện tối ưu, trong mỗi chu kỳ tiếp theo sẽ có một chỉ thị được nạp, một chỉ thị được giải mã,..., một chỉ thị được thi hành. Tức là tại một chu kỳ máy có 6 bộ dữ liệu thuộc 6 giai đoạn xử lý khác nhau được đưa vào đường ống và 6 thao tác này được thực hiện đồng thời. Nhờ vậy, nếu tính từ khi thực thi xong lệnh đầu tiên, cứ sau mỗi chu kỳ máy lại có một lệnh được hoàn thành và một lệnh mới được nhập. Một cách trực quan ta thấy kỹ thuật đường ống tăng tốc độ thực hiện chương trình lên nhiều lần.

Ta xét xem tốc độ thực hiện chương trình trong kỹ thuật pipeline nhanh gấp bao nhiêu lần so với thực hiện chương trình ở dạng thực hiện tuần tự từng lệnh. Nếu các công đoạn không thực hiện trong thời gian như nhau thì các phần mạch thực hiện các công đoạn khác nhau sẽ xảy ra hiện tượng chờ đợi lẫn nhau. Do vậy, nếu lý tưởng thì các công đoạn có thời gian thực hiện bằng nhau. Trong thực tế, thời gian thực hiện các công đoạn không bằng nhau. Thêm nữa không phải tất cả các lệnh đều chiếm 6 công đoạn, ví dụ như lệnh *load* không cần công đoạn ghi toán hạng ra bộ nhớ (WO). Tuy nhiên, để đơn giản, phần cứng đáp ứng kỹ thuật pipeline được xây dựng đáp ứng cho mọi lệnh yêu cầu 6 công đoạn và thời gian thực hiện mỗi công đoạn là như nhau.

Trong hình 5.1 cho thấy một pipeline 6 công đoạn giảm thời gian thi hành cho 9 lệnh từ 54 đơn vị thời gian (nếu thực hiện tuần tự) xuống còn 14 đơn vị thời gian. Do vậy, chọn thời gian thực hiện chung cho mọi công đoạn là T_c .



Với $T_c = \max(T_1, T_2, T_3, \dots, T_m)$, m là số công đoạn.

Giả sử kỹ thuật pipeline chia một lệnh làm m công đoạn, thời gian thực hiện một công đoạn là T_c , thì thực hiện xong n lệnh sẽ mất thời gian là

$$T_p = m \times T_c + (n - 1) \times T_c.$$

Trong khi đó, với kỹ thuật thông thường thì thời gian thực hiện n lệnh tuần tự là

$$T_t = n \times m \times T_c.$$

Ta thấy thời gian T_p nhỏ hơn T_t rất nhiều, nhất là khi số lệnh thực hiện càng lớn. Tức là kỹ thuật pipeline đã nâng cao tốc độ thực hiện chương trình trong hệ thống máy tính lên xấp xỉ m lần.

Có hai loại pipeline trên CPU là pipeline lệnh và pipeline số học.

5.1.1. Kỹ thuật đường ống đơn vị số học

Sử dụng các mạch số học và logic để thực hiện các công đoạn thao tác số học khác nhau. Để đáp ứng được theo kỹ thuật đường ống thì phải tăng số lượng các mạch có cùng chức năng.

Kỹ thuật đường ống đơn vị số học chỉ đơn thuần phải tăng các linh kiện điện tử, còn quá trình thực hiện thì xử lý tương đối đơn giản.

5.1.2. Kỹ thuật đường ống đơn vị lệnh

Quá trình thực hiện lệnh chia thành các công đoạn như nhận lệnh, giải mã lệnh, tính địa chỉ toán hạng, nhận toán hạng, điều khiển xử lý lệnh, cắt kết quả.

Quá trình thực hiện kỹ thuật đường ống đơn vị lệnh có thể xảy ra ba loại xung đột sau:

- *Xung đột về cấu trúc*: Tại 1 thời điểm xử lý liền n lệnh, sẽ có thể xảy ra trường hợp có một số công đoạn của các lệnh khác nhau cần dùng chung một tài nguyên phần cứng. Ví dụ như trong hình 5.1, tại chu kỳ 6 thì lệnh 1 thực hiện công đoạn cắt kết quả, lệnh 3 thực hiện công đoạn lấy toán hạng, lệnh 6 thực hiện công đoạn nhận lệnh. Như vậy, cả 3 lệnh này đều cần chung một loại tài nguyên là đường truyền. Khắc phục vấn đề xung đột này bằng cách tăng số tài nguyên lên.

- *Xung đột về mặt dữ liệu*: Ví dụ, khi thực hiện 2 lệnh sau:

$$\text{ADD R1, R2, R3 ; R3} = \text{R1} + \text{R2}$$

$$\text{ADD R3, R4, R5 ; R5} = \text{R3} + \text{R4}$$

Ta thấy lệnh thứ 2 muốn thực hiện thì lệnh thứ nhất phải hoàn thành xong, song 2 lệnh nằm sát cạnh nhau nên lệnh thứ 2 ở công đoạn nhận toán hạng (cần R3) thì lệnh thứ nhất mới ở công đoạn xử lý lệnh. Như vậy, lệnh 2 phải chờ, hiệu quả thực hiện pipeline giảm đi.

Để khắc phục xung đột này, nếu thiền về phần mềm, các lập trình viên cần lưu ý đặt các lệnh cạnh nhau có ít sự phụ thuộc lẫn nhau, tức là đặt các lệnh phụ thuộc nhau ở cách xa nhau một số lệnh, điều này là không thực tế vì rất khó cho người lập trình, nhất là với ngôn ngữ lập trình bậc cao. Một giải pháp khác là chương trình dịch sẽ sắp lại thứ tự lệnh để các lệnh gần nhau không ảnh hưởng lẫn nhau nếu việc sắp lại đó không ảnh hưởng đến sự thực thi của chương trình.

- *Xung đột về mặt điều khiển*: Khi gặp lệnh rẽ nhánh có điều kiện (IF ... THEN ... ELSE) thì khi chưa kịp giải mã lệnh, không biết lệnh rẽ theo nhánh nào. Do vậy lại phải chờ,... Để khắc phục tình trạng này, có thể xử lý bằng cách cho phép hệ thống được nạp lệnh tiếp theo cho cả 2 nhánh, khi giải mã xong điều kiện nhảy sẽ chỉ lấy kết quả thi hành của lệnh thoả mãn điều kiện, hoặc hệ

thống phải có khả năng dự đoán rẽ nhánh chính xác. Tuy nhiên, xử lý vẫn đề xung đột này không phải đơn giản.

Ví dụ 1: Ở 486 pipeline được chia thành 5 công đoạn.

1. Nhận lệnh;
2. Giải mã 1;
3. Giải mã 2;
4. Thực hiện lệnh;
5. Ghi trở lại bộ nhớ.

Như là đối với lệnh: ADD EAX, [EBX + ECX * 8 + 200]

	1	2	3	4	5	6	7	8	9	10	(T _c)
I1	PF	D1	D2	EX	WB						
I2		PF	D1	D2	EX	WB					
I3			PF	D1	D2	EX	WB				
I4				PF	D1	D2	EX	WB			
I5					PF	D1	D2	EX	WB		

PE: Prefetch

D1: Decode 1

D2: Decode2

EX: Execute

WB: Write Back

Hình 5.2. Cấu trúc pipeline của bộ vi xử lý 80486

Ví dụ 2: Bộ vi xử lý Pentium có *cấu trúc Superscalar* (cấu trúc siêu hướng), có 2 đơn vị thực hiện song song bên trong CPU (2 đường ống). Mỗi đơn vị thực hiện lại được tổ chức theo kiến trúc đường ống dẫn gồm nhiều công đoạn nên tăng được tốc độ xử lý lên nhiều lần. Như vậy khi Pentium thực hiện lệnh, tại mỗi thời điểm có 2 lệnh cùng chuyển đến 2 đơn vị thực hiện khác nhau. Tuy nhiên để thực hiện được điều này nếu các lệnh là độc lập hay nói cách khác là chỉ khi việc thực hiện lệnh này không phụ thuộc vào lệnh kia.

Ví dụ: ADD EAX, EBX ; EAX = EAX + EBX

NOT EAX ; EAX = số bù 2(EAX)

INC DI

MOV [DI], EBX

Ö i lệnh trên, lệnh ADD và NOT không thể chuyên đến 2 đơn vị thực hiện được, **vì** ở lệnh sau EAX là kết quả của lệnh trước. Các lệnh trên có thể sắp xếp lại mà không ảnh hưởng tới kết quả thực hiện các lệnh, chẳng hạn:

ADD EAX, EBX

INC DI

NOT EAX

MOV [DI], EBX

Nếu trình tự các lệnh đã được lập như trên, thì các lệnh có thể được gửi đồng thời tới các đơn vị thực hiện, điều đó cho phép thực hiện song song cả 2 lệnh ở 2 đơn vị khác nhau của CPU. Việc sắp xếp lại trình tự các lệnh sẽ tận dụng lợi điểm của 2 đơn vị thực hiện của Pentium. Đó chính là công việc của bộ chương trình dịch và được gọi là *lập bảng lệnh*. Hiện nay các bộ chương trình dịch được xây dựng để tạo lập bảng lệnh nhằm loại trừ sự phụ thuộc (của dữ liệu). Việc xử lý 2 lệnh ở 2 đơn vị thực hiện thường được xem là *tạo cặp lệnh*.

Pentium có 2 đơn vị thực hiện các số nguyên được gọi là pipeline "U" và pipeline "V". Mỗi pipeline có 5 tầng. Pipeline U có thể thực hiện bất cứ lệnh nào của họ 8086, trong khi đó pipeline V chỉ thực hiện các lệnh đơn giản như INC, DEC, ADD, SUB, MUL, DIV, NOT, AND, OR, EXOR và NEG. Những lệnh đơn giản được thực hiện trong 1 nhịp đồng hồ khi các toán hạng là "REG, REG" hoặc "REG, IMM" và có thanh ghi không phụ thuộc. Ví dụ ở lệnh "ADD EAX, EBX", "SUB ECX, 2000" và "MOV EDX, 1500" là những lệnh đơn giản đòi hỏi 1 nhịp đồng hồ; còn các lệnh "ADD DWORD PTR [EBX + EDI + 500], EAX" thì khác, sẽ cần 3 nhịp đồng hồ.

Để cập nhật vấn đề tăng tốc độ hoạt động của bộ vi xử lý, các nhà thiết kế quan tâm tới hai giải pháp sau:

➤ **Superpipeline** (Kiến trúc siêu đường ống dẫn) được thực hiện bằng cách tăng số công đoạn trong từng lệnh (tăng số thành phần mạch phần cứng trong một đường ống): Có thể lên 8 hoặc thậm chí 10 công đoạn trong một đường ống. Các nhà thiết kế đã không đi theo hướng này.

➤ **Superscalar** (Kiến trúc siêu hướng): Tăng số đường ống lên. Ví dụ, số đường ống bằng 2 thì tại một thời điểm sẽ có 2 lệnh đồng thời được nạp, 2 lệnh đồng thời được giải mã,... 2 lệnh đồng thời được thi hành. Trong khi xây dựng Pentium, các nhà thiết kế đã chọn giải pháp Superscalar, chứ không đi theo giải pháp Superpipeline. Sở dĩ thực hiện được điều này là nhờ có những thành quả vượt bậc của công nghệ vi điện tử đã tăng được một số lượng rất lớn bóng bán dẫn trong 1 chip vi mạch có độ tích hợp cực cao.

Một đặc điểm mới nữa của Pentium đó là khả năng *dự đoán nhánh* (Branch prediction), tức là bộ vi xử lý có khả năng suy đoán và nhận trước mã lệnh khi có lệnh nhảy hoặc lệnh gọi chương trình con CALL. Ví dụ ở máy 486 đối với lệnh JNZ, khi thực hiện lệnh nhảy này thì trước hết nội dung đang có ở pipeline cần phải được giải phóng và sau đó được thay bằng nội dung mới liên quan tới lệnh nhảy này. Quá trình này đòi hỏi mất một thời gian. Ở Pentium lệnh này thực hiện mất 3 nhịp đồng hồ. Bộ vi xử lý Pentium có khả năng suy đoán và nhận trước các mã lệnh (của cả lệnh nhảy và lệnh ngay sau lệnh nhảy) và các lệnh này vẫn được thực hiện mà không phải đợi. Khả năng suy đoán rẽ nhánh đã làm giảm một phần tổng thời gian thực hiện lệnh. Xem ví dụ sau:

Ví dụ: So sánh tổng số nhịp đồng hồ khi thực hiện các lệnh sau ở 486 và Pentium.

	Pentium	486
AGAIN: MOV EAX, DWORD PTR[SI]		1
ADD SI, 4	} 1 nhịp đồng hồ	1
MOV [DI], EAX		2
ADD DI, 4	} 1 nhịp đồng hồ	1
DEC CX		1
JNZ AGAIN	} <u>1 nhịp đồng hồ</u>	<u>3</u>
Tổng cộng	3	9

Trong ví dụ trên các thanh ghi là không phụ thuộc, điều đó có nghĩa là có thể nhóm đôi các lệnh để thực hiện đồng thời ở 2 đơn vị thực hiện trong Pentium. Kết quả mỗi nhịp đồng hồ Pentium thực hiện được 2 lệnh.

Lệnh JNZ ở 486 thực hiện mất 3 nhịp đồng hồ, song đối với Pentium thì chỉ cần 1 nhịp đồng hồ là đủ, bởi vì CPU đã dự kiến rẽ nhánh, nhận lệnh rẽ nhánh; còn các lệnh ở nhãn AGAIN thì vẫn được thực hiện trong đường ống dẫn. Không phụ thuộc vào trạng thái của JNZ, cả 2 lệnh là lệnh sau JNZ và lệnh đầu tiên ở nhãn AGAIN vẫn được đưa tới 2 đường ống dẫn độc lập. Nếu ZF = 0 một đường ống xem như bỏ, còn nếu ZF = 1 (tức kết thúc lệnh nhảy) có nghĩa là lệnh sau JNZ được thực hiện và quá trình dự đoán rẽ nhánh kết thúc.

5.1.3. Đơn vị chức năng định thời biểu

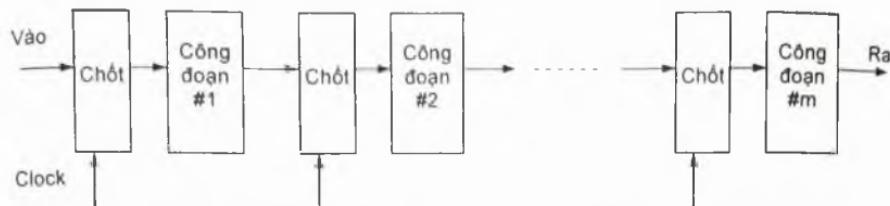
Đơn vị chức năng định thời biểu dùng để thống nhất xung nhịp thực hiện các công đoạn với thời gian bằng nhau.

Đơn vị này thường dùng 2 thanh ghi: Thanh ghi lưu trữ dùng để lưu trữ số xung nhịp đồng hồ cố định tương ứng với thời gian T_c . Và thanh ghi thứ hai là thanh ghi đếm, giá trị trong thanh ghi lưu trữ được copy vào thanh ghi đếm, thanh ghi đếm thực hiện đếm lùi về 0, khi giá trị trong thanh ghi đếm bằng 0, đơn vị thời biểu phát ra tín hiệu ngắt báo kết thúc một chu kỳ công đoạn. Đồng thời giá trị trong thanh ghi lưu trữ lại được copy vào thanh ghi đếm và quá trình cứ như vậy tiếp diễn,... (xem hình 5.3 là đơn vị phát ra xung nhịp clock).

5.2. MẠCH XỬ LÝ VECTƠ ỐNG

Ý tưởng đầu tiên có lẽ là "ống dẫn". Đây là một kỹ thuật nhằm chia nhỏ quá trình thực thi một lệnh thành nhiều công đoạn, và các bước khác nhau của các lệnh khác nhau có thể được thực thi đồng thời. Một bộ xử lý thông thường đọc một lệnh, giải mã nó, đọc những vùng nhớ lệnh đó cần, thực thi lệnh, trả kết quả về. Với kỹ thuật "ống dẫn", bộ xử lý có thể đọc một lệnh ngay sau khi nó đọc xong lệnh trước đó, tức là nó vừa giải mã một lệnh, vừa đọc lệnh kế tiếp, tới chu kỳ tiếp theo bộ xử lý sẽ làm việc với ba lệnh cùng lúc, và cứ thế tiếp tục. Dù thực tế không có lệnh nào được thực thi nhanh hơn, nhưng do lệnh theo sau sẽ hoàn thành ngay sau khi lệnh trước hoàn tất, nên đây là một giải pháp rất hiệu quả nhằm tận dụng tối đa tài nguyên của các vi xử lý.

Mạch xử lý vectơ ống đáp ứng cho ý tưởng này và nó được thiết kế theo mô hình như hình 5.3 (giống như các thiết bị trong một dây truyền sản xuất công nghiệp).



Hình 5.3. Mô hình kiến trúc phần cứng thực thi kỹ thuật pipeline

5.3. MÁY TÍNH VỚI TẬP LỆNH ĐƠN GIẢN HÓA

5.3.1. Đại cương

Từ những năm 1960, ở tất cả các máy tính lớn và máy tính mini, với mỗi vi lệnh của CPU, các nhà thiết kế có gắng xây dựng được càng nhiều lệnh càng tốt. Một số lệnh trong đó có khả năng thực hiện được những phép tính phức tạp. Một ví dụ là việc hiệu chỉnh kết quả của phép cộng thập phân để nhận dữ liệu dạng nible theo mã BCD. Thực ra các nhà thiết kế bộ vi xử lý đã theo sự dẫn dắt của các nhà thiết kế máy tính lớn và máy tính mini. Kể từ khi các bộ vi xử lý này sử dụng một lượng lớn các lệnh và trong đó nhiều lệnh thực hiện được những tính toán có mức phức hợp cao, thì các bộ vi xử lý này được gọi là bộ vi xử lý CISC (Complex Instruction Set Computer – Máy tính có tập lệnh phức hợp). Phù hợp với các nghiên cứu ở những năm 70 của thế kỷ XX, nhiều lệnh phức hợp đã được tạo ngay trong CPU mà người lập trình hay bộ biên dịch không bao giờ sử dụng đến. Kinh phí rất lớn cho việc gắn một số lượng lớn các lệnh vào CPU (một số lệnh trong đó là phức tạp) cộng với việc là trên 60% transistor trên chip đều được các bộ giải mã lệnh sử dụng, đã đặt vấn đề để các nhà thiết kế phải suy nghĩ nhằm đơn giản hóa và giảm bớt các lệnh. Khi CPU dạng đó được phát triển, người ta gọi là RISC (Reduced Instruction Set Computer – Máy tính với tập lệnh đơn giản hóa).

5.3.2. Cạnh tranh giữa RISC và CISC

a) Định hướng thiết kế trước thời RISC

Những ngày đầu của ngành công nghiệp máy tính, trình biên dịch chưa xuất hiện. Công việc lập trình được thực hiện hoặc bằng ngôn ngữ máy (mã nhị phân),

hoặc bằng hợp ngữ. Để việc lập trình đơn giản, các vi xử lý được thêm những lệnh có thể biểu diễn trực tiếp những cấu trúc của ngôn ngữ lập trình cấp cao. Lúc đó thiết kế phần cứng dễ hơn nhiều so với thiết kế trình dịch, vì thế mọi phức tạp đều dồn vào phần cứng.

Một nguyên nhân khác thúc đẩy sự ra đời của những lệnh phức hợp là sự thiếu thốn bộ nhớ. Do bộ nhớ quá nhỏ, do đó sẽ có lợi hơn nhiều nếu tăng mật độ tập trung thông tin trong mã lệnh. Khi mà mỗi byte bộ nhớ còn quá đắt, bộ nhớ chính của toàn bộ hệ thống ở thời kỳ này chỉ vài KB, ngành công nghiệp vi xử lý bị thúc đẩy phải mã hóa thật cao mã lệnh, mã lệnh có thể có kích thước thay đổi, một lệnh có thể thực hiện nhiều phép toán hoặc một lệnh có thể vừa chuyển dữ liệu, vừa xử lý dữ liệu. Lúc đó việc đưa ra một lệnh nén thật tốt được ưu tiên hơn là đưa ra một lệnh dễ giải mã.

Bộ nhớ được sản xuất bằng công nghệ từ, do đó nó không những đỡ ít mà còn chậm. Đây cũng là một lý do để tăng mật độ thông tin trong một mã lệnh. Một mã lệnh với nhiều thông tin sẽ giảm được rất nhiều lần phải truy xuất nguồn bộ nhớ chậm chạp này.

Những CPU thời kỳ này chứa ít thanh ghi vì những lý do sau:

- Một bit trong CPU bao giờ cũng đắt hơn rất nhiều so với một bit ở bộ nhớ ngoài. Với công nghệ tích hợp ở thời kỳ này, muốn có thêm thanh ghi bắt buộc phải có thêm vùng trống trên board hoặc trên chip.
- Một lượng lớn thanh ghi cũng sẽ cần một lượng lớn các bit trong mã lệnh để xác định các thanh ghi đó.

Vì những lý do trên, những nhà thiết kế vi xử lý *có gắng để mỗi lệnh có thể thực hiện càng nhiều chức năng càng tốt*. Điều này dẫn đến một lệnh sẽ làm tất cả công việc như nạp hai số cần cộng, cộng chúng lại, và cuối cùng lưu trả lại vào bộ nhớ. Cũng lệnh đó lại có thể đọc một số từ thanh ghi và số còn lại từ bộ nhớ sau đó lưu kết quả vào bộ nhớ. Khuynh hướng thiết kế vi xử lý này được gọi là *Complex Instruction Set Computer – CISC* (máy tính với tập lệnh phức hợp).

Mục đích chung của thời kỳ này là mỗi lệnh hỗ trợ càng nhiều phương pháp đánh địa chỉ càng tốt, đây chính là lý thuyết trực giao. Điều này dẫn đến một số

phức tạp cho CPU, mặc dù theo lý thuyết mỗi lệnh có thể được tối ưu riêng rẽ, làm quá trình thiết kế nhanh hơn nếu người lập trình sử dụng các lệnh đơn giản.

Sự tương phản rõ ràng nhất đến từ hai loại vi xử lý là 6502 và VAX. Chip 6502 có giá \$25 với 1 thanh ghi duy nhất, và bằng cách tối ưu giao tiếp bộ nhớ nó có thể hoạt động ở tốc độ cao hơn so với thiết kế ban đầu. VAX vốn là một máy tính nhỏ (minicomputer), nó cần tới một tá các thiết bị phụ và được đặc biệt chú ý bởi một lượng rất lớn các kiểu đánh địa chỉ mà nó hỗ trợ, cũng như lệnh nào của nó cũng đều hỗ trợ các kiểu đánh địa chỉ đó.

b) Định hướng thiết kế RISC

Ý tưởng bắt đầu khi người ta nhận thấy rất nhiều tính năng trong các bộ vi xử lý vốn được thiết kế nhằm giúp công việc lập trình trở nên dễ dàng hơn lại thường bị các phần mềm bỏ sót. Những tính năng này thông thường cần vài chu kỳ máy để thực thi. Cộng thêm sự cách biệt về hiệu suất giữa các CPU và bộ nhớ chính đã dẫn đến nhiều kỹ thuật, hoặc nhằm tổ chức lại quá trình thực thi trong bộ xử lý, hoặc nhằm giảm bớt số lần truy xuất bộ nhớ.

Những năm cuối của thập niên 1970, các nhà nghiên cứu của IBM (và cả một số dự án khác) đã chứng minh rằng, phần lớn các phương pháp đánh địa chỉ trực giao thường bị các chương trình bỏ qua. Đây chính là kết quả không mong đợi do sử dụng các *trình biên dịch* cấp cao thay vì sử dụng *hợp ngữ*. Các trình dịch tại thời điểm đó không đủ khả năng để tận dụng hết tính năng của các bộ vi xử lý CISC, chủ yếu là do sự khó khăn trong thiết kế trình dịch. Trình biên dịch càng trở nên phô biến thì các tính năng này lại càng bị bỏ quên.

Một nghiên cứu khác cũng chỉ ra rằng, những tính năng này ít được dùng vì thực ra chúng thực thi chậm hơn một nhóm lệnh cùng thực hiện tác vụ đó. Đây giống như một nghịch lý của quá trình thiết kế vi xử lý, người thiết kế không có đủ thời gian để tối ưu cho tất cả các lệnh, do đó họ chỉ chú trọng đến những lệnh thường được sử dụng nhiều nhất. Ví dụ cụ thể nhất có lẽ là lệnh INDEX của CPU máy VAX, sẽ nhanh hơn từ 45% đến 60% nếu lệnh này được thay bằng một nhóm các lệnh VAX đơn giản khác.

Cũng trong thời gian này, CPU bắt đầu hoạt động nhanh hơn bộ nhớ. Thậm chí trong thập niên 1970, người ta cho rằng, điều này sẽ còn tiếp tục không dưới

I tháp i nữa, và tới lúc đó CPU sẽ nhanh hơn bộ nhớ hàng chục tới hàng trăm lần. Có k ã đến lúc CPU cần thêm nhiều thanh ghi và bộ nhớ đệm cache để có thể hoạt động ở tốc độ cao hơn. Những *thanh ghi* và *bộ nhớ đệm* mới sẽ cần khoảng trống trên *bo mạch* hoặc trên *chip* được tạo ra nếu giảm sự phức tạp của CPU.

Tới lúc này, một phần đóng góp cho kiến trúc RISC đến từ thực tế đó đặc những chương trình trong thế giới thực. Andrew Tanenbaum từ tổng kết rất nhiều **đo đạc** này đã chỉ ra rằng, hầu hết những CPU lúc bấy giờ đều được thiết kế thừa **quá mức**. Ví dụ, ông cho rằng, 98% các *hằng* hoàn toàn có thể biểu diễn bằng 13 bit, trong khi đó các CPU được thiết kế theo bội số của 8 thường là 8, 16 hoặc 32 bit. Do đó nếu CPU cho phép các *hằng* được lưu trong những bit dư của mã lệnh sẽ làm giảm đi rất nhiều lần truy xuất bộ nhớ. Thay vì phải đọc từ bộ nhớ hay từ thanh ghi, các *hằng* đã ở ngay đó khi CPU cần, vì thế quá trình thực thi sẽ nhanh hơn. Tất nhiên, điều lại này yêu cầu mã lệnh phải thật nhỏ để những lệnh 32 bit có thể chứa được những *hằng* tương đối lớn.

Những chương trình trong thực tế thường tồn phần lớn thời gian để thực hiện một số tác vụ đơn giản, do đó một số nhà nghiên cứu hướng tới việc tối ưu hoá những tác vụ này. Do *xung nhịp* (clock rate) của CPU bị giới hạn bởi thời gian thực hiện lệnh chậm nhất, nên nếu tối ưu lệnh này (có thể bằng cách giảm số phương pháp đánh địa chỉ mà nó hỗ trợ) sẽ khiến cho toàn bộ tập lệnh được thực thi nhanh hơn nhiều. Mục tiêu của RISC chính là đơn giản hoá các lệnh, để mỗi lệnh có thể được thực thi chỉ trong 1 chu kỳ máy. Việc tập trung đơn giản hoá các lệnh đã cho ra đời các loại "Máy tính với tập lệnh được đơn giản hoá" – RISC.

Rất tiếc, cụm từ "Máy tính với tập lệnh được đơn giản hoá" thường bị hiểu sai là máy tính với tập lệnh ít lệnh hơn các máy tính khác. Thực ra, RISC lại thường có tập lệnh với số lượng lệnh rất lớn. Cũng từ khuynh hướng đơn giản hoá đó, một số thiết kế thú vị ra đời như MISC (Minimal Instruction Set Computer – Máy tính với tập lệnh tối thiểu) hay OISC (One Instruction Set Computer) với những máy tính như Transport Triggered Architectures.

Điểm khác biệt thực sự giữa RISC so với CISC là nguyên tắc thực hiện mọi thứ trong các thanh ghi, đọc và lưu dữ liệu vào các thanh ghi. Do đó để tránh hiểu lầm nhiều nhà nghiên cứu thích dùng thuật ngữ *load – store*.

Giờ đây dễ thực hiện cùng một công việc, chương trình được viết với những lệnh đơn giản thay vì với một lệnh phức tạp, tông số các lệnh phải đọc từ bộ nhớ nhiều hơn, do đó cũng mất nhiều thời gian hơn. Tại thời điểm đó người ta chưa biết khuyết điểm này có còn đảm bảo sự ưu việt hơn về hiệu suất của RISC hay không, và hầu như đó cũng đã là một cuộc chiến dai dẳng về khái niệm RISC.

c) Cạnh tranh giữa RISC và CISC

Trong khi những khái niệm về RISC đang dần được hoàn thiện thì những ý tưởng nhằm cải tiến hiệu suất cho các CPU cũng bắt đầu ra đời.

Những năm đầu thập niên 1980, người ta sợ rằng, về lý thuyết, công nghệ thiết kế vi xử lý đã đạt đến giới hạn. Sự cải tiến chỉ còn có thể thực hiện với công nghệ bán dẫn, như giảm kích thước của các transistor hoặc dây nối trên chip. Dù độ phức tạp của chip không đổi, nhưng với kích thước nhỏ hơn nó vẫn có thể hoạt động ở tốc độ cao hơn. Cũng có nhiều cố gắng để thiết kế các chip xử lý song song. Thay vì làm cho chip nhanh hơn người ta làm cho nhiều chip có khả năng cùng chia sẻ các tác vụ. Tuy nhiên lịch sử đã chứng minh rằng, nhận định lúc đầu là sai lầm, rất nhiều ý tưởng ra đời cuối thập niên 1980 đã cải tiến một cách toàn diện hiệu suất của các vi xử lý thời kỳ này.

Ý tưởng đầu tiên có lẽ là "ống dẫn" (pipeline). Đây là một kỹ thuật nhằm chia nhỏ quá trình thực thi một lệnh thành nhiều bước, và các bước khác nhau của các lệnh khác nhau có thể được thực thi đồng thời. Một bộ xử lý thông thường đọc một lệnh, giải mã nó, đọc những vùng nhớ lệnh đó cần, thực thi lệnh, trả kết quả về. Với kỹ thuật "kênh dẫn", bộ xử lý có thể đọc một lệnh ngay sau khi nó đọc xong lệnh trước đó, tức là nó vừa giải mã một lệnh vừa đọc lệnh kế tiếp, tới chu kỳ tiếp theo bộ xử lý sẽ làm việc với ba lệnh cùng lúc, và cứ thế tiếp tục. Dù thực tế không có lệnh nào được thực thi nhanh hơn, nhưng do lệnh theo sau sẽ hoàn thành ngay sau khi lệnh trước hoàn tất, nên đây là một giải pháp rất hiệu quả nhằm tận dụng tối đa tài nguyên của các vi xử lý.

Một phương án khác là dùng nhiều đơn vị xử lý song song trong cùng một bộ xử lý. Thay vì thực thi một lệnh, bộ xử lý sẽ tìm cách thực thi đồng thời lệnh kế tiếp trong một đơn vị xử lý khác. Tuy nhiên, đây là một phương án khó vì nhiều lệnh đôi khi lại phụ thuộc vào kết quả của lệnh trước nó.

Cả hai phương pháp trên hướng tới việc cài tiến bằng cách tăng độ phức tạp của CPU. Vì không gian trên chip là có hạn, do đó để thêm những tính năng mới này người ta cần bỏ đi những tính năng khác. Vì thế RISC là kè được hưởng lợi trước tiên do về cấu trúc nó đơn giản hơn CISC rất nhiều. Những thiết kế RISC đầu tiên nhanh chóng được thêm những tính năng mới giúp chúng vượt qua những chip CISC tương ứng. Lúc này người ta bắt đầu tính đến việc thêm những tính năng này vào những chip CISC trong khi vẫn đảm bảo kích thước của chúng, công việc này kéo dài suốt những năm cuối thập niên 1980 và đầu thập niên 1990.

Dù ở bất kỳ cấp độ nào, đơn vị logic của một chip RISC bao giờ cũng cần ít transistor hơn so với của một chip CISC. Điều này giúp người thiết kế có rất nhiều sự linh hoạt, ví dụ họ có thể:

1. Tăng số lượng thanh ghi.
2. Sử dụng các phương pháp tối ưu để tăng mức độ xử lý song song bên trong CPU (pipeline, superscalar).
3. Tăng kích thước cache.
4. Thêm các tính năng như I/O, timer...
5. Thêm các bộ xử lý vector.
6. Tận dụng các dây chuyền công nghệ cũ, trong khi với CISC điều này rất khó khăn do kích thước chip lớn hơn.
7. Cung cấp những chip cho những ứng dụng có yêu cầu cao về thời gian sử dụng pin hoặc về kích thước chip.

Một trong những điểm yếu của các vi xử lý RISC thế hệ đầu tiên là hiệu ứng *branch delay slot*. Hiệu ứng này xảy ra khi có 1 lệnh nhảy có điều kiện, lúc đó dù có thực hiện nhảy hay không thì một hoặc một số lệnh sau vẫn được thực thi do các lệnh này đã được đưa vào pipeline trong lúc lệnh nhảy đang được xử lý. Điều này gây ra một khoảng thời gian trễ khi thực thi lệnh nhảy có điều kiện. Branch delay slot không những xuất hiện trong những vi xử lý RISC như MIPS, PA-RISC và SPARC mà còn cả trong các DSP như μPD77230 hoặc TMS320C3x. Tuy nhiên, ngày nay trong các thiết kế RISC hiện đại người ta đã có thể loại bỏ được hiệu ứng này.

Trong những năm đầu, các dự án chủ yếu chỉ được biết đến trong các trường đại học. Đến năm 1986, tất cả các dự án về RISC bắt đầu cho ra đời sản phẩm. Ngày nay, hầu hết các chip RISC đều được thiết kế dựa trên kiến trúc RISC-II của Berkeley.

Berkeley không thương mại hóa dự án của mình, tuy nhiên hầu hết các công ty sau này đều sử dụng kiến trúc RISC-II như Sun Microsystems với SPARC, hoặc Pyramid Technology. Chính Sun là công ty đầu tiên chứng minh sức mạnh của RISC là có thật trong những hệ thống mới của mình, và cũng nhờ đó họ nhanh chóng chiếm lĩnh thị trường Workstation lúc bấy giờ.

John Hennessy tạm thời rời Standford để thành lập MIPS Computer Systems nhằm thương mại hóa dự án MIPS. Thiết kế đầu tiên của họ là chip R2000, đây là thế hệ tiếp theo của chip MIPS. MIPS nhanh chóng trở thành chip phổ biến nhất khi nó được sử dụng trong PlayStation và Nintendo 64 game consoles. Ngày nay chúng là một trong những chip được sử dụng phổ biến trong các ứng dụng nhúng high-end.

Rút kinh nghiệm từ thất bại của RT - PC, IBM thiết kế RS/6000 dựa trên kiến trúc POWER mới. Họ chuyển họ chip AS/400 thành các chip có kiến trúc POWER, và nhận thấy ngay cả 1 lệnh phức tạp nhất cũng được thực thi nhanh hơn một cách đáng kể. Kết quả đó là sự ra đời của họ iSeries. Kiến trúc POWER cũng được sử dụng trong các chip PowerPC nhưng ở cấp độ thấp hơn. Ngày nay PowerPC là một trong những họ vi xử lý được sử dụng phổ biến trong xe hơi (một số xe có thể sử dụng trên 10 chip loại này). Đây cũng là họ vi xử lý được sử dụng trong máy tính Apple Macintosh cho tới năm 2006 (từ tháng 2/2006 Apple chuyển qua sử dụng vi xử lý của Intel).

Những nghiên cứu ở Anh cũng cho ra đời các dòng vi xử lý như INMOS Transputer, Acorn Archimedes và Advanced RISC Machine. Những công ty vốn sản xuất CISC trước đây cũng nhanh chóng tham gia. Intel với i860 và i960 vào cuối những năm 1980 nhưng không đạt được thành công như mong muốn. Motorola với chip 88000 nhưng cũng không thành công và họ nhanh chóng từ bỏ để hợp tác cùng IBM sản xuất PowerPC. AMD cho ra đời vi xử lý 29000 và trở thành vi xử lý phổ biến nhất những năm đầu thập kỷ 90 của thế kỷ XX.

Ngày nay, CPU RISC (và microcontrollers) chiếm một lượng lớn trong số CPU được sử dụng. Kỹ thuật thiết kế RISC đem đến sức mạnh ngay cả ở những kích thước nhỏ. Do đó nó nhanh chóng chiếm lĩnh hoàn toàn thị trường CPU nhưng công suất thấp. Đây là một thị trường cực kỳ lớn của CPU, có thể tìm thấy chúng trong xe hơi, điện thoại di động, thậm chí một số thiết bị khác có thể chứa hàng tá CPU loại này. RISC cũng chiếm lĩnh thị trường Workstation trong hầu hết những năm 90 của thế kỷ XX. Sau khi Sun cho ra đời SPARCstation, các hãng khác cũng vội vã hoàn thành các hệ thống dựa trên RISC của mình. Thậm chí ngày nay trên thế giới các mainframe cũng thiết kế và xây dựng theo kiến trúc RISC.

Mặc dù vậy, thị trường PC và server lại không phải là nơi dành cho RISC, đây là nơi họ x86 của Intel chiếm ưu thế tuyệt đối (đối thủ chính của Intel là AMD, nhưng chip của AMD lại cũng được xây dựng dựa trên nền x86). Có ba nguyên nhân chính như sau: Thứ nhất, nền tảng rất lớn của các ứng dụng trên PC đều được viết cho x86, trong khi đó chưa hề nền tảng tương tự cho RISC, điều đó đồng nghĩa với việc người dùng chỉ có một lựa chọn là x86. Thứ hai, cho dù kiến trúc RISC có thể mở rộng nhanh và rẻ, nhưng Intel, với thị trường khổng lồ của mình, lại có thể đầu tư hàng đồng tiền vào công nghệ phát triển chip. Intel cũng đã bỏ thời gian để cai tiến các quá trình thiết kế, sản xuất như bất kỳ các nhà sản xuất vi xử lý RISC khác. Thứ ba, những người thiết kế x86 nhận ra rằng, họ hoàn toàn có thể vận dụng RISC trong các vi xử lý của mình. Ví dụ lõi P6 của bộ xử lý PentiumPro và những vi xử lý kế tiếp, có những đơn vị đặc biệt nhằm bù hụt hết các lệnh CISC thành nhiều quá trình RISC. Như vậy, bản thân các bộ xử lý sử dụng lõi P6 là các CPU RISC mô phỏng kiến trúc CISC.

Người dùng thật ra chỉ quan tâm đến tốc độ, giá cả và tính tương thích với các phần mềm có sẵn hơn là chi phí để phát triển những chip mới. Cùng với sự phức tạp của CPU tăng lên, chi phí thiết kế và sản xuất cũng tăng lên nhanh chóng. Lợi nhuận thu được từ RISC trở nên quá nhỏ bé so với chi phí đầu tư để phát triển các CPU mới. Do đó ngày nay chỉ có những nhà sản xuất lớn mới có đủ khả năng phát triển những CPU mạnh. Kết quả là hầu hết những nền tảng RISC (ngoại trừ IBM POWER/PowerPC) đều thu hẹp quy mô (SPARC và MIPS) hoặc thậm chí từ bỏ

(Alpha và PA – RISC) phát triển các CPU mạnh. Năm 2004, CPU nhanh nhất trong các tác vụ với số nguyên là x86 (benchmark với SPECint) và với số thực đầu chấm động là IBM Power 5 (benchmark với SPECfp).

5.3.3. Tổng quan về kỹ thuật cài đặt RISC

a) Các đặc điểm chung trong kỹ thuật cài đặt RISC

Dưới đây là một số nét đặc trưng của RISC. Cần chú ý là các bộ vi xử lý CISC hiện nay như Pentium đã sử dụng một số đặc điểm này trong thiết kế.

➤ Đặc điểm 1

Các bộ vi xử lý RISC có kích thước lệnh cố định hay còn gọi là *định dạng chuẩn mã lệnh* (ví dụ lệnh có chiều dài cố định, các bit của mã lệnh luôn nằm ở vị trí cố định trong mã lệnh) sẽ làm quá trình giải mã lệnh đơn giản hơn. Ở các bộ vi xử lý CISC như 8086, các lệnh có thể là 1, 2 thậm chí là 6 byte. Xem ví dụ với các lệnh sau đây:

```
CLC  
SUB DX,DX  
ADD EAX, (SI + 8)  
JMP FAR
```

Kích thước các lệnh thay đổi làm cho công việc của bộ giải mã lệnh khó khăn hơn nhiều do kích thước của các lệnh không được biết trước. Đổi với bộ vi xử lý RISC kích thước của các lệnh là *cố định 4 byte (32 bit)*. Với các lệnh không đòi hỏi đến 32 bit thì các byte còn lại được điền giá trị 0. Vì vậy, CPU có thể giải mã các lệnh một cách nhanh chóng. Việc này cũng tương tự như việc xếp các lớp gạch có kích thước gạch giống nhau hoặc khác nhau. Tất nhiên, nếu gạch có cùng một kích thước thì việc sắp xếp sẽ hiệu quả hơn.

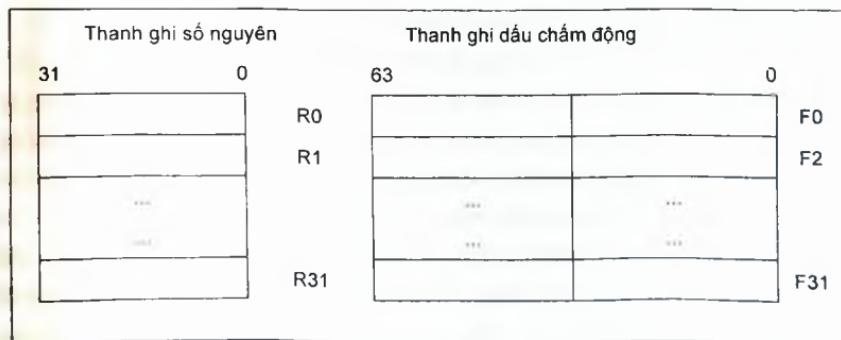
➤ Đặc điểm 2

RISC sử dụng kiến trúc nạp/lưu giữ (load/store). Ở các bộ vi xử lý CISC, dữ liệu có thể được xử lý khi ở trong bộ nhớ. Lấy ví dụ các lệnh của 8086 như "ADD [BX], AL", bộ vi xử lý phải chuyển nội dung của các ô nhớ do BX xác định vào CPU rồi cộng với AL, sau đó chuyển kết quả trở lại bộ nhớ được trả bởi

Đối với RISC các nhà thiết kế đã từ bỏ dạng lệnh này và để thực hiện lệnh ~~đã~~ trước hết cần dùng lệnh nạp để chuyển toán hạng bộ nhớ vào thanh ghi, sau đó thực hiện phép toán trên các thanh ghi bên trong của CPU và cuối cùng lại sử dụng lệnh lưu gửi trở lại kết quả tính được ở thanh ghi. Đây là phương pháp không trực tiếp thực hiện các lệnh số học và logic giữa các thanh ghi và nội dung của bộ nhớ. Ý tưởng này đầu tiên được thực hiện ở siêu máy tính CRAY 1 (năm 1976) và thường gọi là *kiến trúc nạp/lưu giữ*.

➤ Đặc điểm 3

Một đặc điểm quan trọng của kiến trúc RISC là có số lượng thanh ghi lớn. Các thanh ghi đồng nhất, do đó chúng có thể được sử dụng thay thế nhau trong mọi tình huống (tuy nhiên các thanh ghi dành cho số nguyên và số thực dấu chấm động vẫn phân biệt nhau). Tất cả các bộ vi xử lý RISC có 32 thanh ghi 32 bit R0-R31. Hình 5.4 là ví dụ các thanh ghi của Intel RISC 860, chỉ có một số thanh ghi được gán cho chức năng riêng. Ví dụ, R0 được gán tự động giá trị 0 và không thể có giá trị nào khác được ghi vào. Một thuận lợi khi số thanh ghi nhiều là tránh được việc dùng ngăn xếp để lưu các tham số. Mặc dù ngăn xếp có thể được thực hiện ở bộ vi xử lý RISC, nhưng nó không đóng vai trò cần thiết như đối với CISC, bởi vì ở đây có quá nhiều thanh ghi khả dụng khác. Cần lưu ý là các bộ vi xử lý RISC, bên cạnh các thanh ghi đa năng 32 bit, còn có 32 thanh ghi khác cho các phép toán dấu chấm động. Các thanh ghi dấu chấm động có thể có dạng 64 bit để lưu giữ các toán hạng độ chính xác kép.



Hình 5.4. Các thanh ghi số nguyên và dấu chấm động của Intel RISC 860

➤ Đặc điểm 4

Cách đánh địa chỉ đơn giản. Để có những phương pháp đánh địa chỉ phức tạp cần kết hợp với các phép toán số học.

➤ Đặc điểm 5

Các bộ vi xử lý RISC có một tập lệnh nhỏ. Chúng chỉ có các lệnh cơ sở như ADD, SUB, MUL, DIV, LOAD, STORE, AND, OR, EXOR, SHR, CALL và JMP. Không có các lệnh như INC, DEC, NOT, NEG, DAA, DAS.... Do RISC có rất ít lệnh nên công việc của người lập trình (chương trình biên dịch) là phải thực hiện những lệnh này thông qua các lệnh khác của RISC. Lấy ví dụ lệnh nạp trực tiếp như "MOV AX, 25" là không có ở Intel RISC 860. Có thể sử dụng lệnh khác, ví dụ lệnh OR để thực hiện lệnh MOV nêu trên. Lệnh sau được viết cho Intel RISC 860.

OR 25, R0, R8 ; OR 25 với R0 và lưu kết quả vào R8

Theo cú pháp của Intel RISC 860 thì thanh ghi đích là thanh ghi cuối R8. Vì R0 luôn luôn bằng 0 nên OR với bất cứ số nào cũng sẽ cho kết quả là chính số đó. Ví dụ trên sẽ đặt 25 vào R8. Một ví dụ khác về sử dụng lệnh ADD thay cho lệnh INC không có trong tập lệnh.

ADD 1, R15, R15 ; ADD 1 với R15 và gửi kết quả vào R15

Số lượng lệnh có hạn là một trong những hạn chế của bộ vi xử lý RISC, bởi vì nó làm cho công việc của người lập trình hợp ngữ khó khăn hơn nhiều so với CISC. Đó là lý do tại sao RISC thường được dùng ở môi trường ngôn ngữ bậc cao như C nhiều hơn so với môi trường hợp ngữ. Đáng chú ý để ghi nhận là những thành viên bảo vệ CISC vẫn gọi đó là "máy tính với tập lệnh đầy đủ" thay cho khái niệm "máy tính với tập lệnh phức hợp", bởi vì nó có một tập đầy đủ của mọi dạng lệnh. Có bao nhiêu lệnh được sử dụng lại là vấn đề khác. Số lệnh bị hạn chế của RISC sẽ làm cho chương trình bị lớn lên nhiều, như minh họa ở ví dụ 5.1. Do đó đòi hỏi phải dùng nhiều bộ nhớ hơn, song hiện nay DRAM khá rẻ, nên đó cũng không còn là vấn đề nữa. Tuy nhiên, trước khi tiến tới sử dụng bộ nhớ bán dẫn vào năm 1960, các nhà thiết kế CISC đã cố gắng xây dựng được càng nhiều phép toán trong 1 lệnh càng tốt.

Ví dụ 5.1. Ở các bộ vi xử lý 8086, lệnh NOT được thực hiện bằng cách tính 1 , nhưng bộ vi xử lý RISC không có lệnh như vậy. Tính số bù 1 trong RISC
• thực hiện như thế nào? Xác định mã của phép toán bù 1 của $25H$ ở Intel
360 và 8086.

Giải:

• vì xử lý RISC có lệnh EXOR. Nếu chúng ta EXOR toán hạng với toàn số
• được số đảo (số bù 1).

OR 25H,R0,R8 ; OR $25H$ với $R0$ và đặt kết quả vào $R8$ ($R8 = 25H$)

OR FFH,R0,R5 ; OR FFH với $R0$ và đặt kết quả vào $R5$ ($R5 = FFH$)

EXOR R8,R5,R9 ; XOR $R8$ với $R5$ và đặt kết quả vào $R9$ ($R9 = DAH$)

Mỗi lệnh chiếm 4 byte (32 bit), 3 lệnh chiếm hết 12 byte trong bộ nhớ.
Đối với 8086, một dạng của bộ vi xử lý CISC, chúng ta có các lệnh như sau và chỉ
chiếm tổng cộng 4 byte trong bộ nhớ (có thể kiểm tra bằng DEBUG).

MOV AL,25H

NOT AL

➤ **Đặc điểm 6**

Hỗ trợ rất ít kiểu dữ liệu (một số chip CISC có thể có cả các lệnh thao tác với chuỗi, xử lý số phức hoặc ma trận, những lệnh như thế chẳng bao giờ tồn tại trong một chip RISC lý tưởng).

➤ **Đặc điểm 7**

Đến lúc này người ta có thể hỏi: Với một loạt khó khăn khi lập trình RISC
như vậy thì đâu là điểm mạnh của bộ vi xử lý này? Đặc tính quan trọng nhất của
bộ xử lý RISC đó chính là trên 95% các lệnh đều được thực hiện chỉ với 1 nhịp
đồng hồ, điều này hoàn toàn khác với của CISC. Thậm chí với 5% lệnh RISC còn
lại mà cần 2 nhịp đồng hồ thì vẫn có thể dùng thủ thuật (đặt lại trình tự lệnh) để
chi mất 1 nhịp đồng hồ thực hiện. Các nhà thiết kế đã làm gì với những transistor
tiết kiệm được khi tuân theo nguyên tắc của RISC. Trường hợp với Intel RISC
860, những transistor dư ra được sử dụng để xây dựng bộ đồng xử lý, một cache
mạnh, bộ điều khiển cache và một bộ xử lý đồ họa rất mạnh – và tất cả đều trên
một chip. Ở nhiều máy tính, ví dụ 386, các chức năng trên đều được tổ chức trên
từng chip riêng rẽ.

➤ **Đặc điểm 8**

Bộ vi xử lý RISC có các Bus riêng cho dữ liệu và riêng cho mã. Đôi với họ 8086 cũng như các máy tính CISC khác, chỉ có 1 tập hợp Bus cho địa chỉ (ví dụ: A0 – A24 ở 80286) và tập hợp các Bus khác cho dữ liệu (ví dụ: D0 – D15 ở 80286) để chuyển mã lệnh và toán hạng vào/ra CPU. Để truy nhập được mọi phần của bộ nhớ trừ những vùng đang chứa mã lệnh hay toán hạng, có thể sử dụng các kênh địa chỉ và dữ liệu. Ở các bộ vi xử lý RISC có 4 tập hợp Bus: (1) Tập hợp Bus dữ liệu để chuyển dữ liệu (các toán hạng) vào/ra CPU; (2) Tập hợp các Bus địa chỉ để truy nhập các toán hạng dữ liệu; (3) Tập hợp các Bus để chuyển mã lệnh; (4) Tập hợp các Bus địa chỉ để truy nhập mã lệnh.

Việc sử dụng Bus riêng biệt cho các toán hạng mã và dữ liệu thường được gọi là *kiến trúc Harvard*. Điều này giúp cho quá trình đọc dữ liệu và mã lệnh có thể xảy ra đồng thời, do đó có khả năng nâng cao hiệu suất của vi xử lý.

b) **Những RISC đầu tiên**

Vi xử lý RISC được biết đến đầu tiên là siêu máy tính CDC 6600, do Jim Thornton và Seymour Cray thiết kế năm 1964, nó có 74 mã lệnh (8086 có 400 mã lệnh) cộng với 12 máy tính đơn giản được gọi là "bộ xử lý ngoại vi" để xử lý I/O. CDC 6600 sử dụng kiến trúc load – store, nó hỗ trợ hai phương pháp đánh địa chỉ, có 11 đơn vị được "kênh dẫn hoá", 5 đơn vị đọc dữ liệu và 2 đơn vị để lưu dữ liệu (bộ nhớ của nó được tổ chức theo bank, do đó các đơn vị đọc/ghi có thể hoạt động đồng thời). Tốc độ xung đồng hồ/lệnh nhanh hơn 10 lần so với tốc độ truy xuất bộ nhớ.

Một máy tính khác được thiết kế với kiến trúc load – store là Data General Nova. Đây là một máy tính nhỏ 16 bit, được thiết kế năm 1968 bởi một công ty của Mỹ với tên là Data General.

Tuy nhiên, vi xử lý RISC được biết nhiều nhất lại đến từ một dự án được tài trợ bởi chương trình VLSI (Very Large – Scale Integration) của DARPA (Defense Advanced Research Projects Agency). Chương trình trên đã cho ra đời rất nhiều cải tiến liên quan đến thiết kế, sản xuất chip và cả đồ họa máy tính.

Dự án RISC của Đại học California, Berkeley bắt đầu năm 1980 dưới sự hướng dẫn của David Patterson, với mục đích nâng cao hiệu suất của các vi xử lý

và kỹ thuật pipeline và register windows. Một vi xử lý thông thường có khá ít thanh ghi, các chương trình có thể tùy ý sử dụng các thanh ghi đó bất cứ lúc nào. Còn đối với các vi xử lý sử dụng kỹ thuật register windows, có rất nhiều thanh ghi trong vi xử lý, nhưng chương trình chỉ sử dụng cùng lúc một tập hợp nhỏ các thanh ghi. Vì thông thường mỗi lần gọi một chương trình con, vi xử lý cần lưu lại giá trị một số thanh ghi và sau đó hồi phục lại các thanh ghi đó khi thực hiện lệnh return. Vì vậy, bằng cách chuyển từ tập thanh ghi này sang tập thanh ghi khác (chuyển cửa sổ) chương trình có thể thực hiện các lệnh gọi hàm hoặc lệnh trả về một cách nhanh chóng.

Dự án RISC cho ra đời vi xử lý RISC-I năm 1982. Vi xử lý này chứa 44420 transistor (so với 100000 transistor cho 1 vi xử lý CISC), với 32 lệnh nhưng hoàn toàn vượt xa các vi xử lý đơn chip cùng thời. Vi xử lý RISC-II ra đời năm 1983 với 39 lệnh, chứa 40760 transistor và nhanh gấp 3 lần RISC-I.

Cũng khoảng thời gian đó, John L. Hennessy thực hiện dự án MIPS ở Đại học Stanford năm 1981. MIPS hầu như chỉ tập trung vào kỹ thuật pipeline nhằm tận dụng tối đa khả năng của các vi xử lý. Cho dù đã được sử dụng trước đó, nhưng với MIPS, kỹ thuật này đã thực sự được cải tiến vượt bậc. Nhưng vấn đề quan trọng nhất ở đây và có lẽ cũng phiền toái nhất, là nó đòi hỏi tất cả các lệnh bắt buộc phải được thực thi trong 1 chu kỳ máy. Nếu đáp ứng được yêu cầu này, pipeline có thể hoạt động ở tốc độ rất cao và đây hầu như là yếu tố quyết định đến tốc độ của vi xử lý. Tuy nhiên, nó cũng có mặt trái là phải bỏ đi rất nhiều lệnh có ích như nhân, chia.

Hầu hết các dự án kể trên đều nhằm mục đích cài tiến các kỹ thuật hiện có, phải chờ tới năm 1975, dự án đầu tiên nhằm cho ra đời chip RISC hoàn chỉnh mới được thực hiện ở IBM. Được đặt tên theo số của ngôi nhà nơi dự án được thực hiện, dự án này đã cho ra đời họ vi xử lý IBM 801 vốn được sử dụng rộng rãi trong các phần cứng của IBM.

c) Các hệ thống và các RISC phổ biến

- Họ MIPS trong các máy tính SGI, PlayStation và Nintendo 64 game consoles.

- Họ POWER trong các Super Computers/mainframes của IBM.
- Freescale (trước đây là Motorola SPS) và IBM's PowerPC trong Nintendo Gamecube, Microsoft Xbox 360, Nintendo Wii and Sony PlayStation 3 game consoles, và cho tới gần đây là Apple Macintosh.
- SPARC và UltraSPARC trong tất cả các hệ thống của Sun.
- Hewlett - Packard PA - RISC.
- DEC Alpha.
- ARM - Palm, Inc. Ban đầu sử dụng (CISC) Motorola 680x0 trong những PDA đầu tiên, nhưng hiện tại là (RISC) ARM: Nintendo sử dụng 1 chip ARM7 trong Game Boy Advance và Nintendo DS. Nhiều nhà sản xuất điện thoại di động, như Nokia cũng dựa trên kiến trúc của ARM.

CÂU HỎI VÀ BÀI TẬP CHƯƠNG 5

1. Trình bày kỹ thuật đường ống đơn vị lệnh (pipeline lệnh).
2. Giải sú một lệnh được chia làm 5 công đoạn: nhận lệnh, giải mã lệnh, nhận toán hạng, xử lý, cắt kết quả. So sánh thời gian thực hiện n lệnh giữa kỹ thuật pipeline và kỹ thuật tuần tự.
3. Trình bày các đặc điểm trong kỹ thuật cài đặt RISC.
4. Cho biết ý nghĩa của cụm từ RISC (Reduced Instruction Set Computer).
5. Trình bày ngắn gọn sự khác biệt giữa kiến trúc CISC và kiến trúc RISC.