

# Giáo trình

## KIẾN TRÚC MÁY TÍNH

*Biên soạn:* TS. Vũ Đức Lung

2009

### LỜI NÓI ĐẦU

Với mục tiêu đưa các môn học chuyên ngành công nghệ thông tin vào học ngay từ những học kỳ đầu trong trường Đại học Công nghệ thông tin, giáo trình ***Kiến trúc máy tính*** được biên soạn đặc biệt cho mục đích này và được định hướng cho sinh viên ngành công nghệ thông tin năm thứ nhất.

Giáo trình Kiến trúc máy tính này trình bày các vấn đề chung nhất, các thành phần cơ bản nhất cấu thành nên máy tính hiện đại nhằm trang bị cho sinh viên các nội dung chủ yếu trong 8 chương sau:

***Chương I:*** Trình bày lịch sử phát triển của máy tính cũng như các tích năng mới của máy tính trong từng giai đoạn, các thể hệ máy tính, định hướng phát triển của máy tính và cách phân loại máy tính.

***Chương II:*** Giới thiệu các nguyên lý hoạt động chung và các tính chất cơ bản của các bộ phận chính yếu trong máy tính như: bộ xử lý (CPU), bản mạch chính (Mainboard), các thiết bị lưu trữ dữ liệu, các loại bộ nhớ RAM, Card đồ họa, màn hình. Ngoài ra còn cho thấy được những hình dáng và sự tích hợp của các bộ phận với nhau nhằm giúp sinh viên có thể tự mua sắm, lắp ráp một máy tính cho mình.

***Chương III:*** Trình bày cách biến đổi cơ bản của hệ thống số (như hệ thập phân, hệ nhị phân, hệ bát phân, hệ thập lục phân), các cách cơ bản để biểu diễn dữ liệu, cách thực hiện các phép tính số học cho hệ nhị phân.

***Chương IV:*** Các cổng và đại số Boolean, các định lý trong đại số Boolean, cách đơn giản các hàm Boolean cũng như các mạch số, cách biểu diễn các mạch số qua các hàm Boolean và ngược lại, các mạch tổ hợp cơ bản, cách thiết kế các mạch đơn giản.

***Chương V:*** Trình bày nguyên lý hoạt động của các mạch lật, các flip-flop, qui trình thiết kế một mạch tuần tự và đưa ra ví dụ cụ thể cho việc thiết kế này.

**Chương VI:** Phân loại kiến trúc bộ lệnh, cách bố trí địa chỉ bộ nhớ, các cách mã hóa tập lệnh, các lệnh cơ bản của máy tính qua các lệnh hợp ngữ assembler

**Chương VII:** Giới thiệu cấu trúc của bộ xử lý trung tâm: tổ chức, chức năng và nguyên lý hoạt động của các bộ phận bên trong bộ xử lý như bộ tính toán logic số học, bộ điều khiển, tập các thanh ghi. Ngoài ra còn trình bày cách tổ chức đường đi dữ liệu, diễn biến quá trình thi hành lệnh và kỹ thuật ống dẫn.

**Chương VIII:** Trình bày các cấp bộ nhớ, thiết kế và nguyên lý hoạt động của các loại bộ nhớ. Phương pháp đánh giá hiệu năng của các cấp bộ nhớ. Các chiến thuật thay thế khối nhớ, trang nhớ cũng như các chiến thuật ghi vào bộ nhớ.

Như đã nói ở trên, giáo trình nhằm giảng dạy cho sinh viên năm thứ nhất do đó những kiến thức đưa ra chỉ là cơ bản. Để hiểu sâu hơn mọi vấn đề nên xem thêm trong các sách tham khảo ở cuối quyển giáo trình này.

Mặc dù đã cố gắng biên soạn rất công phu và kỹ lưỡng, tuy nhiên cũng khó tránh khỏi những thiếu sót. Chúng tôi mong được đón nhận các đóng góp ý kiến của các Thầy, các bạn đồng nghiệp, các bạn sinh viên và các bạn đọc nhằm chỉnh sửa giáo trình được hoàn thiện hơn.

Cuối cùng xin chân thành cảm ơn những góp ý quý giá của các đồng nghiệp khi biên soạn giáo trình này.

**Vũ Đức Lung.**

## Chương I: Giới thiệu

### 1.1. Lịch sử phát triển của máy tính

Trong quá trình phát triển của công nghệ máy tính, con người đã chế tạo ra hàng ngàn loại máy tính khác nhau. Rất nhiều trong số những máy tính này đã bị quên lãng đi, chỉ một số ít còn được nhắc lại cho đến ngày nay. Đó là các máy tính với những ý tưởng thiết kế và nguyên lý hoạt động độc đáo tạo nên một tầm ảnh hưởng lớn đến các máy tính thế hệ sau nó. Để giúp sinh viên có được những khái niệm cơ bản về máy tính và hiểu rõ hơn bằng cách nào mà con người đã phát minh ra những máy tính hiện đại, dễ sử dụng như ngày nay, trong phần này sẽ trình bày những chi tiết quan trọng về lịch sử quá trình phát triển của máy tính.

Máy tính thường được phân loại thành các thế hệ dựa trên nền tảng công nghệ phần cứng được sử dụng trong quá trình chế tạo. Lịch sử phát triển máy tính có thể được chia thành các thế hệ máy tính sau:

#### 1.1.1. Thế hệ zero – máy tính cơ học (1642-1945)

Mốc lịch sử máy tính phải nhắc đến đầu tiên là khi nhà bác học người Pháp Blaise Pascal (1626-1662) vào năm 1642 đã phát minh ra máy tính toán đầu tiên – máy tính cơ học với 6 bánh quay và bộ dẫn động bằng tay. Máy của ông chỉ cho phép thực hiện các phép tính cộng và trừ.

Sau 30 năm, vào năm 1672 một nhà bác học khác, Gotfrid Vilhelm Leibnits đã chế tạo ra máy tính với 4 phép tính cơ bản (+ - \* /) sử dụng 12 bánh quay. Từ khi còn là sinh viên cho đến hết cuộc đời, ông đã nghiên cứu các tính chất của hệ nhị phân và là người đã đưa ra các nguyên lý cũng như khái niệm cơ bản nhất cho hệ nhị



phân được dùng ngày nay trong máy tính điện tử.

Năm 1834 giáo sư toán học trường ĐH Cambridge (Anh), Charles Babbage (người phát minh ra đồng hồ công tơ mét) đã thiết kế ra máy tính với chỉ 2 phép tính + và – nhưng có một cấu trúc đáng để ý thời bấy giờ – máy tính có 4 bộ phận:

- bộ nhớ,
- bộ tính toán,
- thiết bị nhập để đọc các phiếu đục lỗ,
- thiết bị xuất để khoan lỗ lên các tấm đồng.

Chính ý tưởng của ông là tiền đề cho các máy tính hiện đại sau này.

Để máy tính hoạt động nó cần phải có chương trình, và ông đã thuê cô Ada làm chương trình cho máy tính này. Cô Ada chính là lập trình viên đầu tiên và để tưởng nhớ tới cô ta sau này Ada được đặt tên cho 1 ngôn ngữ lập trình. Tuy nhiên máy tính đã không hoạt động được vì đòi hỏi quá phức tạp và thời bấy giờ con người và kỹ thuật chưa cho phép.

Năm 1936 K. Zus (người Đức) đã thiết kế một vài máy đếm tự động trên cơ sở rơle (relay). Tuy nhiên ông không biết gì về máy tính của Babbage và máy tính của ông đã bị phá hủy trong một trận bom vào Berlin khi chiến tranh thế giới lần thứ 2 - 1944. Vì vậy những phát minh của ông ta đã không ảnh hưởng đến sự phát triển của kỹ thuật máy tính sau này.

Năm 1944 G. Iken (thuộc ĐH Havard Mỹ) đã đọc về công trình của Babbage và ông đã cho ra đời Mark I sau đó là Mark II. Máy Mark I ra đời với mục đích chính là phục vụ chiến tranh. Nó nặng 5 tấn, cao 2.4 m, dài 15 m, chứa 800 km dây điện. Tuy nhiên vào thời điểm đó máy tính relay đã qua thời và đã bắt đầu kỷ nguyên của máy tính điện tử.

#### 1.1.2. Thế hệ I – bóng đèn điện (1945-1955)

Chiến tranh thế giới thứ 2 bắt đầu và vào đầu thời kỳ chiến tranh tàu ngầm của Đức đã phá hủy nhiều tàu của Anh, nhờ những

tín hiệu mã hóa được chuyển đi bởi thiết bị ENIGMA mà quân đội Anh đã không thể giải mã được. Để giải mã đòi hỏi một số lượng tính toán rất lớn và mất nhiều thời gian, trong khi chiến tranh thì không cho phép chờ đợi. Vì vậy chính phủ Anh đã cho thành lập một phòng thí nghiệm bí mật nhằm chế tạo ra một máy tính điện phục vụ cho việc giải mã những thông tin này. Năm 1943 máy tính **COLOSSUS** ra đời với 2000 đèn chân không và được giữ bí mật suốt 30 năm và nó đã không thể trở thành cơ sở cho sự phát triển của máy tính. Một trong những người sáng lập ra COLOSSUS là nhà toán học nổi tiếng **Alain Turing**. Trong hình 1.1 là bức chân dung của Alain Turing và một bóng đèn chân không.



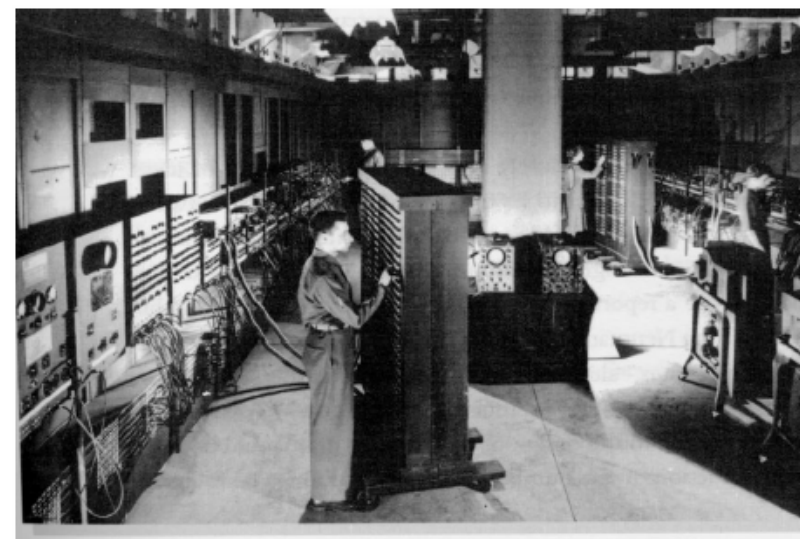
**Alain Turing**      Bóng đèn chân không

Hình 1.1. Alain Turing với bóng đèn chân không

Chiến tranh thế giới đã có ảnh hưởng lớn đến phát triển kỹ thuật máy tính ở Mỹ. Quân đội Mỹ cần các bảng tính toán cho pháo binh và hàng trăm phụ nữ đã được thuê cho việc tính toán này trên các máy tính tay (người ta cho rằng phụ nữ trong tính toán cẩn thận hơn nam giới). Tuy nhiên quá trình tính toán này vẫn đòi hỏi thời gian khá lâu và nhằm đáp ứng yêu cầu của BRL (Ballistics Research Laboratory – Phòng nghiên cứu đạn đạo quân đội Mỹ) trong việc tính toán chính xác và nhanh chóng các bảng số liệu đạn đạo cho từng loại vũ khí mới, dự án chế tạo máy **ENIAC** đã được bắt đầu vào năm 1943.

Máy **ENIAC** (*Electronic Numerical Integrator And Computer*), do **John Mauchly** và **John Presper Eckert** (đại học Pennsylvania, Mỹ) thiết kế và chế tạo, là chiếc máy số hoá điện tử đa năng đầu tiên trên thế giới (hình 1.2).

- **Số liệu kỹ thuật:** ENIAC là một chiếc máy khổng lồ với hơn 18000 bóng đèn chân không, nặng hơn 30 tấn, tiêu thụ một lượng điện năng vào khoảng 140kW và chiếm một diện tích xấp xỉ 1393 m<sup>2</sup>. Mặc dù vậy, nó làm việc nhanh hơn nhiều so với các loại máy tính điện cơ cùng thời với khả năng thực hiện 5000 phép cộng trong một giây đồng hồ.



Hình 1.2. Máy tính ENIAC

- **Điểm khác biệt giữa ENIAC & các máy tính khác:** ENIAC sử dụng hệ đếm thập phân chứ không phải nhị phân như ở tất cả các máy tính khác. Với ENIAC, các con số được biểu diễn dưới dạng thập phân và việc tính

toán cũng được thực hiện trên hệ thập phân. Bộ nhớ của máy gồm 20 "bộ tích lũy", mỗi bộ có khả năng lưu giữ một số thập phân có 10 chữ số. Mỗi chữ số được thể hiện bằng một vòng gồm 10 đèn chân không, trong đó tại mỗi thời điểm, chỉ có một đèn ở trạng thái bật để thể hiện một trong mười chữ số từ 0 đến 9 của hệ thập phân. Việc lập trình trên ENIAC là một công việc vất vả vì 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.

- **Hoạt động thực tế:** Máy ENIAC bắt đầu hoạt động vào tháng 11/1945 với nhiệm vụ đầu tiên không phải là tính toán đạn đạo (vì chiến tranh thế giới lần thứ hai đã kết thúc) mà để thực hiện các tính toán phức tạp dùng trong việc xác định tính khả thi của bom H. Việc có thể sử dụng máy vào mục đích khác với mục đích chế tạo ban đầu cho thấy tính đa năng của ENIAC. Máy tiếp tục hoạt động dưới sự quản lý của BRL cho đến khi được tháo rời ra vào năm 1955.

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

### Máy tính Von Neumann

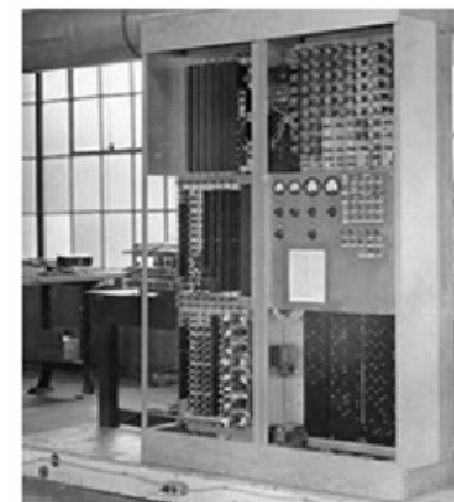
Như đã đề cập ở trên, việc lập trình trên máy ENIAC là một công việc rất tẻ nhạt và tốn kém nhiều thời gian. Công việc này có lẽ sẽ đơn giản hơn nếu chương trình có thể được biểu diễn dưới dạng thích hợp cho việc lưu trữ trong bộ nhớ cùng với dữ liệu cần xử lý. Khi đó máy tính chỉ cần lấy chỉ thị bằng cách đọc từ bộ nhớ, ngoài ra chương trình có thể được thiết lập hay thay đổi thông qua sự chỉnh sửa các giá trị lưu trong một phần nào đó của bộ nhớ.

Ý tưởng này, được biết đến với tên gọi "**khái niệm chương trình được lưu trữ**", do nhà toán học *John von Neumann* (Hình

1.3), một cố vấn của dự án ENIAC, đưa ra ngày 8/11/1945, trong một bản đề xuất về một loại máy tính mới có tên gọi **EDVAC** (*Electronic Discrete Variable Computer – do Eckert và Moyshly đã bắt đầu làm rồi ngừng lại đi thành lập công ty, sau này là Unisys Corporation*). Máy tính này cho phép nhiều thuật toán khác nhau có thể được tiến hành trong máy tính mà không cần phải nối dây lại như máy ENIAC nhờ vào khái niệm chương trình lưu trữ.



*John von Neumann*



**Máy EDVAC**

Hình 1.3. Von Neumann với máy tính EDVAC

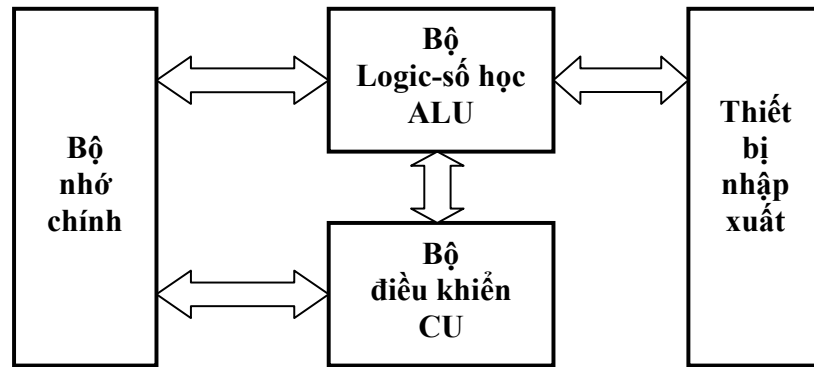
### *Máy IAS*

Tiếp tục với ý tưởng của mình, vào năm 1946, von Neuman cùng các đồng nghiệp bắt tay vào thiết kế một máy tính mới có chương trình được lưu trữ với tên gọi *IAS* (*Institute for Advanced Studies*) tại học viện nghiên cứu cao cấp Princeton, Mỹ. Mặc dù mãi đến năm **1952 máy IAS mới được hoàn tất, nó vẫn là mô hình cho tất cả các máy tính đa năng sau này.**

Cấu trúc tổng quát của máy IAS, như được minh họa trên hình 1.4, gồm có:

- Một bộ nhớ chính để lưu trữ dữ liệu và chương trình.
- Một bộ logic-số học (**ALU** – *Arithmetic and Logic Unit*) có khả năng thao tác trên dữ liệu nhị phân.
- Một bộ điều khiển chương trình có nhiệm vụ thông dịch các chỉ thị trong bộ nhớ và làm cho chúng được thực thi.
- Thiết bị nhập/xuất được vận hành bởi đơn vị điều khiển.

Hầu hết các máy tính hiện nay đều có chung cấu trúc và chức năng tổng quát như trên. Do vậy chúng còn có tên gọi chung là các máy von Neumann.



Hình 1.4. Cấu trúc của máy IAS

### 1.1.3. Thế hệ II – transistor (1955-1965)

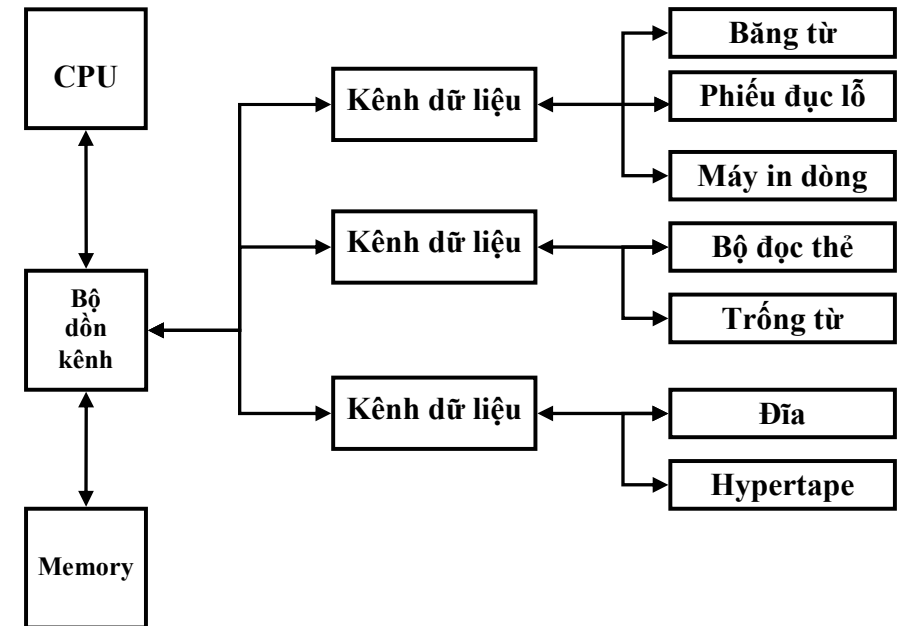
Sự thay đổi đầu tiên trong lĩnh vực máy tính điện tử xuất hiện khi có sự thay thế đèn chân không bằng đèn bán dẫn. Đèn bán dẫn nhỏ hơn, rẻ hơn, tỏa nhiệt ít hơn trong khi vẫn có thể được sử dụng theo cùng cách thức của đèn chân không để tạo nên máy tính. Không như đèn chân không vốn đòi hỏi phải có dây, có băng kim loại, có bao thủy tinh và chân không, đèn bán dẫn là một thiết bị ở trạng thái rắn được chế tạo từ silicon có nhiều trong cát trong tự nhiên.

Đèn bán dẫn là phát minh lớn của phòng thí nghiệm **Bell Labs** trong năm 1947 bởi **Bardeen, Brattain và Shockley**. Nó đã tạo ra một cuộc cách mạng điện tử trong những năm 50 của thế kỷ

20. Dù vậy, mãi đến cuối những năm 50, các máy tính bán dẫn hóa hoàn toàn mới bắt đầu xuất hiện trên thị trường máy tính.

Việc sử dụng đèn bán dẫn trong chế tạo máy tính đã xác định thế hệ máy tính thứ hai, với đại diện tiêu biểu là máy **PDP-1** của công ty **DEC (Digital Equipment Corporation)** và **IBM 7094** của IBM. DEC được thành lập vào năm 1957 và sau đó 4 năm cho ra đời sản phẩm đầu tiên của mình là máy PDP-1 như đã đề cập ở trên. Đây là chiếc máy mở đầu cho dòng máy tính mini của DEC, vốn rất phổ biến trong các máy tính thế hệ thứ ba.

Các máy IBM-709,7090,7094 có chu kỳ thời gian là 2 microsecond, bộ nhớ 32 K word 16 bit. Hình 1.5 mô tả một cấu hình với nhiều thiết bị ngoại vi của máy IBM 7094.



Hình 1.5 Một cấu trúc máy IBM 7094



Ở đây có nhiều điểm khác biệt so với máy IAS mà chúng ta cần lưu ý. Điểm quan trọng nhất trong số đó là việc sử dụng các **kênh dữ liệu**. Một kênh dữ liệu là một module nhập/xuất độc lập có bộ xử lý và tập lệnh riêng. Trên một hệ thống máy tính với các thiết bị như thế, CPU sẽ không thực thi các chỉ thị nhập/xuất chi tiết. Những chỉ thị đó được lưu trong bộ nhớ chính và được thực thi bởi một bộ xử lý chuyên dụng trong chính kênh dữ liệu. CPU chỉ khởi động một sự kiện truyền nhập/xuất bằng cách gửi tín hiệu điều khiển đến kênh dữ liệu, ra lệnh cho nó thực thi một dãy các chỉ thị trong máy tính. Kênh dữ liệu thực hiện nhiệm vụ của nó độc lập với CPU và chỉ cần gửi tín hiệu báo cho CPU khi thao tác đã hoàn tất. Cách sắp xếp này làm giảm nhẹ công việc cho CPU rất nhiều.

Một đặc trưng khác nữa là **bộ đa công**, điểm kết thúc trung tâm cho các kênh dữ liệu, CPU và bộ nhớ. Bộ đa công lập lịch các truy cập đến bộ nhớ từ CPU và các kênh dữ liệu, cho phép những thiết bị này hoạt động độc lập với nhau.

### ➤ Máy PDP-1

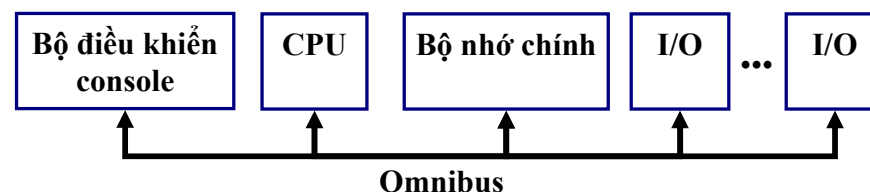
Máy **PDP-1** có gần 4 K word, 1 (word=18 bit) và thời gian cho 1 chu kỳ là 5 microsecond. Thông số này lớn hơn gần gấp 2 lần so với máy cùng dòng với nó IBM-709, nhưng PDP-1 là máy tính nhỏ gọn nhanh nhất thời bấy giờ và có giá bán 120000\$, còn IBM-7090- có giá bán tới 1 triệu USD. Máy PDP-1 với màn hình kính cỡ 512 điểm được cho đến ĐH công nghệ *Massachuset* và từ đây các sinh viên đã viết trò chơi máy tính đầu tiên – chò trôi chiến tranh giữa các vì sao .

Sau một vài năm DEC cho ra đời một hiện tượng khác trong ngành công nghiệp máy tính. Đó là máy **PDP-8**, máy tính 12 bit. Vào lúc một máy tính cỡ trung cũng đòi hỏi một phòng có điều hòa không khí, máy PDP-8 đủ nhỏ để có thể đặt trên một chiếc ghế dài vốn thường gặp trong phòng thí nghiệm hoặc để kết hợp vào trong các thiết bị khác. Nó có thể thực hiện mọi công việc của một máy

### **Omnibus**

tính lớn với giá chỉ có 16000 đô la Mỹ, so với số tiền lên đến hàng trăm ngàn đô la để mua được một chiếc máy System/360 của IBM. Tương phản với kiến trúc chuyển trung tâm được IBM sử dụng cho các hệ thống 709, các kiểu sau này của máy PDP-8 đã sử dụng một cấu trúc rất phổ dụng hiện nay cho các máy mini và vi tính: **cấu trúc đường truyền**. Hình 1.6 minh họa cấu trúc này.

Đường truyền PDP-8, được gọi là **Omnibus**, gồm 96 đường tín hiệu riêng biệt, được sử dụng để mang chuyển tín hiệu điều khiển, địa chỉ và dữ liệu. Do tất cả các thành phần hệ thống đều dùng chung một tập hợp các đường tín hiệu, việc sử dụng chúng phải được CPU điều khiển. Kiến trúc này có độ linh hoạt cao, cho phép các module được gắn vào đường truyền để tạo ra rất nhiều cấu hình khác nhau. Cấu trúc kiểu này của DEC đã được sử dụng trong tất cả các máy tính ngày nay. DEC đã bán được 50000 chiếc PDP-8 và trở thành nhà cung cấp máy tính mini đứng đầu thế giới lúc bấy giờ.



Hình 1.6 Cấu trúc đường truyền PDP-8

Một máy tính cũng đáng chú ý nữa trong giai đoạn này là vào năm 1964, khi công ty CDC (Control Data Corporation) cho ra đời máy tính 6600. Máy này có tốc độ cao hơn gấp nhiều lần IBM-7094 và điểm đặc biệt của máy tính này là sử lý song song mà sau này trong các siêu máy tính hay sử dụng.

#### 1.1.4. Thế hệ III – mạch tích hợp (1965-1980)

Một đèn bán dẫn đơn lẻ thường được gọi là một **thành phần rời rạc**. Trong suốt những năm 50 và đầu những năm 60 của thế kỷ 20, các thiết bị điện tử phần lớn được kết hợp từ những thành phần rời rạc – đèn bán dẫn, điện trở, tụ điện, v.v... Các thành phần rời rạc được sản xuất riêng biệt, đóng gói trong các bộ chứa riêng, sau đó được dùng để nối lại với nhau trên những bảng mạch. Các bảng này lại được gắn vào trong máy tính, máy kiểm tra dao động, và các thiết bị điện tử khác nữa.

Bất cứ khi nào một thiết bị điện tử cần đến một đèn bán dẫn, một ống kim loại nhỏ chứa một mẫu silicon sẽ phải được hàn vào một bảng mạch. Toàn bộ quá trình sản xuất, đi từ đèn bán dẫn đến bảng mạch, là một quá trình tốn kém và không hiệu quả. Các máy tính thế hệ thứ hai ban đầu chứa khoảng 10000 đèn bán dẫn. Con số này sau đó đã tăng lên nhanh chóng đến hàng trăm ngàn, làm cho việc sản xuất các máy mạnh hơn, mới hơn gặp rất nhiều khó khăn. Để giải quyết những vấn đề khó khăn này, năm 1958 Jack Kilby và Robert Noyce đã cho ra đời một công nghệ mới, công nghệ mạch tích hợp (**Integrated circuit - IC** hay vi mạch - CHIP).

Sự phát minh ra mạch tích hợp vào năm 1958 đã cách mạng hóa điện tử và bắt đầu cho kỷ nguyên vi điện tử với nhiều thành tựu rực rỡ. Mạch tích hợp chính là yếu tố xác định thế hệ thứ ba của máy tính. Với công nghệ này nhiều transistor được cho vào trong một chip nhỏ.

Đối với nhà sản xuất máy tính, việc sử dụng nhiều IC được đóng gói mang lại **những điểm có ích** như sau:

- Giá chip gần như không thay đổi trong quá trình phát triển nhanh chóng về mật độ của các thành phần trên

chip. Điều này có nghĩa là giá cả cho các mạch nhớ và luận lý giảm một cách đáng kể.

- Vì những thành phần luận lý và ô nhớ được đặt gần nhau hơn trên các chip nên khoảng cách giữa các nguyên tử ngắn hơn dẫn đến việc gia tăng tốc độ chung cho toàn bộ.
- Máy tính sẽ trở nên nhỏ hơn, tiện lợi hơn để bố trí vào các loại môi trường khác nhau.
- Có sự giảm thiểu trong những yêu cầu về bộ nguồn và thiết bị làm mát hệ thống.
- Sự liên kết trên mạch tích hợp đáng tin cậy hơn trên các nối kết hàn. Với nhiều mạch trên mỗi chip, sẽ có ít sự nối kết liên chip hơn.

#### ➤ Máy IBM System/360

Máy **IBM System/360** được IBM đưa ra vào năm 1964 là họ máy tính công nghiệp đầu tiên được sản xuất một cách có kế hoạch. Khái niệm họ máy tính bao gồm các máy tính tương thích nhau là một khái niệm mới và hết sức thành công. Đó là chuỗi các máy tính với cùng một ngôn ngữ **Assembler**. Chương trình viết cho máy này có thể được dùng cho máy khác mà không phải viết lại, đây chính là ưu điểm nổi bật của nó. Ý tưởng thành lập họ máy tính trở thành rất phổ biến trong rất nhiều năm sau đó. Trong bảng 1.1 cho ta thấy những thông số chính của một trong những đời đầu tiên của họ IBM-360.

Họ máy IBM System/360 không những đã **quyết định tương lai về sau của IBM mà còn có một ảnh hưởng sâu sắc đến toàn bộ ngành công nghiệp máy tính**. Nhiều đặc trưng của họ máy này đã trở thành tiêu chuẩn cho các máy tính lớn khác.



Thông số	Model 30	Model 40	Model 50	Model 60
Tốc độ so sánh giữa chúng	1	3.5	10	21
Thời gian 1 chu kỳ, nano giây	1000	625	500	250
Dung lượng bộ nhớ tối đa, Kbyte	64	256	256	512
Số byte lấy từ bộ nhớ mỗi chu kỳ	1	2	4	16
Số kênh dữ liệu tối đa	3	3	4	6

Bảng 1.1. Các thông số họ IBM - 360

Một số cột mốc đáng chú ý nữa trong giai đoạn này là:

- 1975 máy tính cá nhân đầu tiên (*Portable computer*) IBM 5100 (hình 1.7) ra đời, tuy nhiên máy tính này đã không gặt hái được thành công nào. Những thông số chính của nó như sau:
  - Bộ nhớ dùng băng từ
  - Nặng 23 Kg
  - Có giá 10000\$
  - Khả năng lập trình trên Basic
  - Màn hình 16 dòng, 64 ký tự
  - Bộ nhớ  $\leq 64$ Kbyte
- 1979 chương trình Sendmail ra đời bởi 1 sinh viên ĐHTH California, Berkely university cho ra đời BSD UNIX (Berkely Software Distribution).



Hình 1.7. Máy tính IBM 5100

#### 1.1.5. Thế hệ IV – máy tính cá nhân (1980-đến nay)

Sự xuất hiện của mạch tích hợp tỷ lệ cao ***Very Large Scale Integrated (VLSI) circuit*** vào những năm 80 cho phép ghép hàng triệu transistor trên một bản mạch. Điều đó dẫn đến khả năng thiết kế những máy tính cỡ nhỏ, nhưng với tốc độ cao.

Trong phần tiếp theo, hai thành tựu tiêu biểu về công nghệ của máy tính thế hệ thứ tư sẽ được giới thiệu một cách tóm lược.

##### ➤ Bộ nhớ bán dẫn

Vào khoảng những năm 50 đến 60 của thế kỷ này, hầu hết bộ nhớ máy tính đều được chế tạo từ những vòng nhỏ làm bằng vật liệu sắt từ, mỗi vòng có đường kính khoảng 1/16 inch. Các vòng này được treo trên các lưới ở trên những màn nhỏ bên trong máy tính. Khi được từ hóa theo một chiều, một vòng (gọi là một **lỗi**) biểu thị giá trị 1, còn khi được từ hóa theo chiều ngược lại, lỗi sẽ đại diện cho giá trị 0. Bộ nhớ lỗi từ kiểu này làm việc khá nhanh. Nó chỉ cần một phần triệu giây để đọc một bit lưu trong bộ nhớ. Nhưng nó rất đắt tiền, cồng kềnh, và sử dụng cơ chế hoạt động loại trừ: một thao tác đơn giản như đọc một lỗi sẽ xóa dữ liệu lưu trong lỗi đó. Do vậy cần phải cài đặt các mạch phục hồi dữ liệu ngay khi nó được lấy ra ngoài.

Năm 1970, *Fairchild* chế tạo ra bộ nhớ bán dẫn có dung lượng tương đối đầu tiên. Chip này có kích thước bằng một lỗi đơn, có thể lưu 256 bit nhớ, hoạt động không theo cơ chế loại trừ và nhanh hơn bộ nhớ lỗi từ. Nó chỉ cần 70 phần tử giây để đọc ra một bit dữ liệu trong bộ nhớ. Tuy nhiên giá thành cho mỗi bit cao hơn so với lỗi từ.

Kể từ năm 1970, bộ nhớ bán dẫn đã đi qua 11 thế hệ: 1K, 4K, 16K, 64K, 256K, 1M, 4M, 16M, 64M, 256M và giờ đây là 1G bit trên một chip đơn ( $1K = 2^{10}$ ,  $1M = 2^{20}$ ). Mỗi thế hệ cung cấp khả năng lưu trữ nhiều gấp bốn lần so với thế hệ trước, cùng với sự giảm thiểu giá thành trên mỗi bit và thời gian truy cập.

### ➤ Bộ vi xử lý

Vào năm 1971, hãng **Intel** cho ra đời chip 4004, chip đầu tiên có chứa tất cả mọi thành phần của một CPU trên một chip đơn. Kỷ nguyên bộ vi xử lý đã được khai sinh từ đó. Chip 4004 có thể cộng hai số 4 bit và nhân bằng cách lập lại phép cộng. Theo tiêu chuẩn ngày nay, chip 4004 rõ ràng quá đơn giản, nhưng nó đã đánh dấu sự bắt đầu của một quá trình tiến hóa liên tục về dung lượng và sức mạnh của các bộ vi xử lý. Bước chuyển biến kế tiếp trong quá trình tiến hóa nói trên là sự giới thiệu chip Intel 8008 vào năm 1972. Đây là bộ vi xử lý 8 bit đầu tiên và có độ phức tạp gấp đôi chip 4004.

Đến năm 1974, Intel đưa ra chip 8080, bộ vi xử lý đa dụng đầu tiên được thiết kế để trở thành CPU của một máy vi tính đa dụng. So với chip 8008, chip 8080 nhanh hơn, có tập chỉ thị phong phú hơn và có khả năng định địa chỉ lớn hơn.

Cũng trong cùng thời gian đó, các bộ vi xử lý 16 bit đã bắt đầu được phát triển. Mặc dù vậy, mãi đến cuối những năm 70, các bộ vi xử lý 16 bit đa dụng mới xuất hiện trên thị trường. Sau đó đến năm 1981, cả Bell Lab và Hewlett-packard đều đã phát triển các bộ

vi xử lý đơn chip 32 bit. Trong khi đó, Intel giới thiệu bộ vi xử lý 32 bit của riêng mình là chip 80386 vào năm 1985.

- ❖ Điểm đáng lưu ý nhất trong giai đoạn này là vào năm 1981 ra đời máy IBM PC trên cơ sở CPU Intel 8088 và dùng hệ điều hành MS-DOS của Microsoft (hình 1.8).



Hình 1.8. Máy tính IBM PC đầu tiên

## 1.2. Khối các nước XHCN và Việt Nam

Nhắc đến lịch sử phát triển của máy tính, chúng ta cũng cần hướng tầm nhìn đến các máy tính của khối XHCN phát minh mà hầu như trong các tài liệu ít khi đề cập đến. Vào khoảng giữa thế kỷ 20, các nước XHCN cũng đã cho ra đời hàng loạt các máy tính với các tính năng tương đương với các loại máy tính của khối Tư bản. Chiếc máy tính đầu tiên bắt đầu được xây dựng có thể kể đến đó là vào năm 1950 tại trường Cơ khí chính xác và quang học ( Nay là trường Đại học Công nghệ thông tin và quang học). Một năm sau đó tại đây đã cho ra đời máy tính toán điện cơ lớn đầu tiên ra đời với mục đích giải quyết các bài toán khoa học và kỹ thuật phức tạp.

Trong khoảng từ năm 1959 đến 1966 đã cho ra đời 4 thế hệ tiếp theo của máy này với khả năng tính toán lên đến 10000 phép tính/giây. Cũng trong khoảng thời gian này tại một trường khác là Trường Đại học Toán thuộc Viện Hàn lâm Khoa Học Liên Xô cũng tiến hành xây dựng một máy tính khác gọi là Strela (Mũi tên) và cuối năm 1953 đã cho ra đời máy này. Sau đó các dòng họ của máy này cũng đã được sản xuất hàng loạt.

Một số các máy tính của khối XHCN thời bấy giờ có tầm ảnh hưởng lớn được liệt kê trong bảng 1.2. Trong đó máy EC-1840, 1841 tương đương với máy 8086, còn máy EC-1842, 1843 tương đương với 80286. Các thế hệ sau đó sử dụng bộ vi xử lý của Intel.

PC	Năm bắt đầu SX	Năm kết thúc SX	Số lượng
EC-1840	1986	1989	7461
EC-1841	1987	1995	83937
EC-1842	1988	1996	10193
EC-1843	1990	1993	3012
EC-1849	1990	1997	4966
EC-1851	1991	1997	3142
EC-1863	1991	1997	3069
BM2001	1994	-	1074

Bảng 1.2. Các máy tính tiêu biểu của Liên Xô

Một trong những máy tính có ảnh hưởng lớn đến Việt Nam là máy MINSK-22 cũng được ra đời trong khoảng thời gian này. Máy Minsk có nhiều thế hệ như Minsk-1, Minsk-11, Minsk-12, Minsk-14, Minsk-22,... với khả năng tính toán lên đến 6000 phép tính/giây.

Lịch sử phát triển máy tính của Việt Nam có thể kể đến bắt đầu từ năm 1968 khi chiếc máy tính điện tử do Liên Xô tặng về đến Việt Nam và được đặt tại Ủy Ban Khoa học và Kỹ thuật Nhà nước, ở 39 Trần Hưng Đạo, Hà Nội.

Máy tính điện tử Minsk-22 là loại máy tính thuộc thế hệ thứ hai, được chế tạo từ các linh kiện bán dẫn (chưa phải là mạch tổ hợp), kết hợp kỹ thuật cơ khí chính xác, nhưng vào thời điểm đó Minsk-22 là một trong những máy tính hiện đại nhất của Liên Xô và các nước Đông Âu. Vì vậy cần phải có một diện tích khoảng 100m<sup>2</sup> để đặt máy. Hệ thống máy tính điện tử Minsk-22 gồm một thiết bị trung tâm có bộ nhớ khoảng 68K bites với lõi Ferit từ, tốc độ tính toán 5000 phép tính/giây. Bộ nhớ ngoài bằng băng từ, thiết bị vào số liệu dùng bìa và băng giấy. Ngôn ngữ sử dụng là ngôn ngữ máy (nếu so sánh chức năng máy tính điện tử Minsk-22 chỉ tương đương một phần nhỏ của một dàn máy vi tính hiện đại).

Nhắc đến việc khai thác Minsk-22 không thể nào quên các nhóm cán bộ nghiên cứu của các Viện, Trường, Bộ, ngành mà tên của họ gắn bó thân thiết với Minsk-22 như Trần Bình, Lại Huy Phương, Nguyễn Tri Niên, Nguyễn Đức Hiếu, Hoàng Kiếm, Nguyễn Bá Hào, Trịnh Văn Thư, Mai Anh, Bùi Khương...

Máy tính Minsk-22 đã được dùng để tính toán phương án sửa cầu Long Biên sau khi mấy nhịp cầu bị bom Mỹ đánh sập, được dùng trong bài toán dự báo thời tiết ngắn hạn, giải hệ phương trình khí nhiệt động học đầy đủ bằng phương pháp hệ thức tích phân; giải các bài toán tính toán dự báo thủy triều; phân tích đánh giá quan hệ giữa phân bố mưa với các điều kiện hoàn lưu khí quyển; tính hàm phân bố gió bão... Về sau, dự báo thời tiết tiếp tục được xử lý trên máy tính Minsk-32 của quân đội và đạt kết quả ngày càng tốt hơn.

### 1.3. Khuynh hướng hiện tại

Việc chuyển từ thế hệ thứ tư sang thế hệ thứ 5 còn chưa rõ ràng. Người Nhật đã và đang đi tiên phong trong các chương trình nghiên cứu để cho ra đời thế hệ thứ 5 của máy tính, thế hệ của những máy tính thông minh, dựa trên các ngôn ngữ trí tuệ nhân tạo như LISP và PROLOG,... và những giao diện người - máy thông minh. Đến thời điểm này, các nghiên cứu đã cho ra các sản phẩm bước đầu và gần đây nhất (2004) là sự ra mắt sản phẩm người máy thông minh gần giống với con người nhất: **ASIMO** (*Advanced Step Innovative Mobility*: Bước chân tiên tiến của đổi mới và chuyển động). Với hàng trăm nghìn máy móc điện tử tối tân đặt trong cơ thể, ASIMO có thể lên/xuống cầu thang một cách uyển chuyển, nhận diện người, các cử chỉ hành động, giọng nói và đáp ứng một số mệnh lệnh của con người. Thậm chí, nó có thể bắt chước cử động, gọi tên người và cung cấp thông tin ngay sau khi bạn hỏi, rất gần gũi và thân thiện. Hiện nay có nhiều công ty, viện nghiên cứu của Nhật thuê Asimo tiếp khách và hướng dẫn khách tham quan như: Viện Bảo tàng Khoa học năng lượng và Đổi mới quốc gia, hãng IBM Nhật Bản, Công ty điện lực Tokyo. Hãng Honda bắt đầu nghiên cứu ASIMO từ năm 1986 dựa vào nguyên lý chuyển động bằng hai chân. Cho tới nay, hãng đã chế tạo được 50 robot ASIMO.

Các tiến bộ liên tục về mật độ tích hợp trong VLSI đã cho phép thực hiện các mạch vi xử lý ngày càng mạnh (8 bit, 16 bit, 32 bit và 64 bit với việc xuất hiện các bộ xử lý RISC năm 1986 và các bộ xử lý siêu vô hướng năm 1990). Chính các bộ xử lý này giúp thực hiện các máy tính song song với từ vài bộ xử lý đến vài ngàn bộ xử lý. Điều này làm các chuyên gia về kiến trúc máy tính tiên đoán thế hệ thứ 5 là thế hệ các máy tính xử lý song song.

Đó là việc của tương lai xa, còn hiện tại thì các công ty sản xuất máy tính đang định hướng phát triển các máy tính với nhiều bộ xử lý nhằm giải quyết các bài toán song song để tăng cường tốc độ xử lý chung của máy tính. Thành quả của việc này là hàng loạt

các bộ xử lý đa lõi đã ra đời và các siêu máy tính với tốc độ không tưởng đã cũng đã được các công ty đua nhau giới thiệu.

Một số cột mốc đáng chú ý trong quá trình chuyển sang CPU đa lõi như sau:

- 1999 – CPU 2 lõi kép đầu tiên ra đời (IBM Power4 cho máy chủ)
- 2001 – bắt đầu bán ra thị trường Power4
- 2002 – AMD và Intel cùng thông báo về việc thành lập CPU đa lõi của mình.
- 2004 – CPU lõi kép của Sun ra đời UltraSPARS IV
- 2005 – Power5
- 03/2005 – CPU Intel lõi kép x86 ra đời, AMD – Opteron, Athlon 64X2
- 20-25/05/2005 – AMD bắt đầu bán Opteron 2xx, 26/05 Intel Pentium D, 31/05 AMD – bán Athlon 64X2

Từ các cột mốc đó cho thấy sự cạnh tranh gay gắt giữa hai công ty sản xuất CPU hàng đầu.

Một thành quả ngày nay nữa sẽ được ứng dụng trong tương lai gần là việc cho ra đời các siêu máy tính. Một trong những siêu máy tính hàng đầu của thế giới ngày nay là máy tính Blue Gene của IBM với 8192 CPU và cho tốc độ tính toán lên đến 7,3 Tfops. Tuy nhiên chẳng bao lâu sau nó cũng chỉ là chú rùa khi mà kế hoạch của IBM sản xuất **supercomputer Blue Gene/L** với 128 dãy, 130 ngàn CPU, 360 Tfops, với giá dự định 267 triệu USD đã thành công vào 26/06/2007.

Tuy nhiên siêu máy tính của IBM vẫn chưa được gọi là nhanh nhất thế giới vì vào 06/2006 viện nghiên cứu của Nhật RIKEN thông báo cho ra đời máy tính **MDGRAPE-3** với tốc độ lên đến 1 Petaflop, tức nhanh hơn tới 3 lần máy Blue Gene/L nhưng chỉ dùng 40.314 CPU. Máy tính này không được sắp vào TOP500 vì nó không được dùng cho mục đích chung mà phục vụ cho việc

ngiên cứu và mô phỏng các hệ thống phức tạp trong một chương trình chung của các công ty Riken, Hitachi, Intel, and NEC subsidiary SGI Japan.

### 1.3. Phân loại máy tính

Dựa vào kích thước vật lý, hiệu suất, giá tiền và lĩnh vực sử dụng, thông thường máy tính được phân thành bốn loại chính như sau:

**a) Các siêu máy tính (Super Computer):** là các máy tính đắt tiền nhất và tính năng kỹ thuật cao nhất. Giá bán một siêu máy tính từ vài triệu USD. Các siêu máy tính thường là các máy tính vector hay các máy tính dùng kỹ thuật vô hướng và được thiết kế để tính toán khoa học, mô phỏng các hiện tượng. Các siêu máy tính được thiết kế với kỹ thuật xử lý song song với rất nhiều bộ xử lý (hàng ngàn đến hàng trăm ngàn bộ xử lý trong một siêu máy tính).

**b) Các máy tính lớn (Mainframe)** là loại máy tính đa dụng. Nó có thể dùng cho các ứng dụng quản lý cũng như các tính toán khoa học. Dùng kỹ thuật xử lý song song và có hệ thống vào ra mạnh. Giá một máy tính lớn có thể từ vài trăm ngàn USD đến hàng triệu USD.

**c) Máy tính mini (Minicomputer)** là loại máy cỡ trung, giá một máy tính mini có thể từ vài chục USD đến vài trăm ngàn USD.

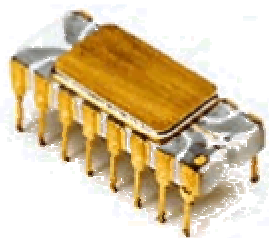
**d) Máy vi tính (Microcomputer)** là loại máy tính dùng bộ vi xử lý, giá một máy vi tính có thể từ vài trăm USD đến vài ngàn USD.

### 1.4. Các dòng Intel

Do ở thị trường Việt Nam chủ yếu sử dụng bộ vi xử lý của hãng này, nên ở phần này sẽ trình bày kỹ hơn về quá trình phát triển các bộ xử lý của *Intel*.

*Intel* là nhà tiên phong trong việc sản xuất bộ vi xử lý (BVXL) khi tung ra Intel 4004 vào năm 1971. Khả năng tính toán của Intel 4004 chỉ dừng lại ở hai phép toán: cộng hoặc trừ và nó chỉ có thể tính toán được 4 bits tại một thời điểm. Điều đáng kinh ngạc ở đây là toàn bộ "cỗ máy" tính toán được tích hợp "nằm" gọn trên một chip đơn duy nhất (hình 1.9). Trước khi cho ra đời Intel 4004, các kỹ sư đã chế tạo ra máy tính hoặc là từ một tổ hợp nhiều chip hoặc là từ các thành phần linh kiện rời rạc.

Thế nhưng BVXL đầu tiên "đặt chân" vào ngôi nhà số của chúng ta hiện nay lại không phải là Intel 4004 mà là BVXL thế hệ kế tiếp của nó - Intel 8080, một máy tính 8-bit hoàn hảo trên một chip duy nhất, được giới thiệu vào năm 1974. Trong khi đó, Intel 8088 mới là thế hệ BVXL đầu tiên "loé sáng" thực sự trên thị trường. Được giới thiệu năm 1979 và sau đó được tích hợp vào các máy tính cá nhân IBM xuất hiện trên thị trường vào năm 1982, Intel 8088 có thể được xem như "người tiền nhiệm chính" của các bộ xử lý thế hệ tiếp theo: Intel 80286, 80386, 80486 rồi đến Intel Pentium, Pentium Pro, Pentium II, III và IV. Do tất cả đều được cải tiến dựa trên thiết kế cơ bản của Intel 8088. Ngày nay, BVXL Intel Pentium 4 có thể thực hiện bất kỳ đoạn mã nào đã chạy trên BVXL Intel 8088 nguyên thủy nhưng với tốc độ nhanh hơn gấp nhiều nghìn lần.



Hình 1.9. Bộ vi xử lý 4004 đầu tiên của Intel

Để có cái nhìn bao quát hơn, chúng ta xem **Quá trình phát triển của CPU Intel** trong các thời kỳ như sau:

**Năm 1971:** Bộ vi xử lý 4004

4004 là bộ vi xử lý đầu tiên của Intel. Phát minh đột phá này nhằm tăng sức mạnh cho máy tính Busicom và dọn đường cho khả năng nhúng trí thông minh của con người vào trong các thiết bị vô tri cũng như các hệ thống máy tính cá nhân.

Số lượng bóng bán dẫn: 2.300

Tốc độ: 108KHz

**Năm 1972:** Bộ vi xử lý 8008

Bộ vi xử lý 8008 mạnh gấp đôi bộ vi xử lý 4004. Thiết bị Mark-8 được biết đến như là một trong những hệ thống máy tính đầu tiên dành cho người sử dụng gia đình – một hệ thống mà theo các tiêu chuẩn ngày nay thì rất khó để xây dựng, bảo trì và vận hành.

Số lượng bóng bán dẫn: 3.500

Tốc độ: 200KHz

**Năm 1974:** Bộ vi xử lý 8080

Bộ vi xử lý 8080 đã trở thành bộ não của hệ thống máy tính cá nhân đầu tiên – Altair.

Số lượng bóng bán dẫn: 6.000

Tốc độ: 2MHz

**Năm 1978:** Bộ vi xử lý 8086-8088

Một hợp đồng cung cấp sản phẩm quan trọng cho bộ phận máy tính cá nhân mới thành lập của IBM đã biến bộ vi xử lý 8088 trở thành bộ não của sản phẩm chủ đạo mới của IBM—máy tính IBM PC.

Số lượng bóng bán dẫn: 29.000

Tốc độ: 5MHz, 8MHz, 10MHz

**Năm 1982:** Bộ vi xử lý 286

Bộ vi xử lý 286, còn được biết đến với cái tên là 80286, là bộ vi xử lý Intel đầu tiên có thể chạy tất cả các phần mềm được viết cho những bộ vi xử lý trước đó. Tính tương thích về phần mềm này vẫn luôn là một tiêu chuẩn bắt buộc trong họ các bộ vi xử lý của Intel

Số lượng bóng bán dẫn: 134.000

Tốc độ: 6MHz, 8MHz, 10MHz, 12,5MHz

**Năm 1985:** Bộ vi xử lý Intel 386

Bộ vi xử lý Intel 386 có 275.000 bóng bán dẫn – nhiều hơn 100 lần so với bộ vi xử lý 4004 ban đầu. Đây là một chip 32 bit và có khả năng xử lý “đa tác vụ”, nghĩa là nó có thể chạy nhiều các chương trình khác nhau cùng một lúc.

Số lượng bóng bán dẫn: 275.000

Tốc độ: 16MHz, 20MHz, 25MHz, 33MHz

**Năm 1989:** Bộ vi xử lý CPU Intel 486 DX

Thế hệ bộ vi xử lý 486 thực sự có ý nghĩa khi giúp chúng ta thoát khỏi một máy tính phải gõ lệnh thực thi và chuyển sang điện toán chỉ và nhấn (point-and-click).

Số lượng bóng bán dẫn: 1,2 triệu

Tốc độ: 25MHz, 33MHz, 50MHz



**Năm 1993:** Bộ vi xử lý Pentium®

Bộ vi xử lý Pentium® cho phép các máy tính dễ dàng hơn trong việc tích hợp những dữ liệu ‘thế giới thực’ như giọng nói, âm thanh, ký tự viết tay và các ảnh đồ họa

Số lượng bóng bán dẫn: 3,1 triệu

Tốc độ: 60MHz, 66MHz

**Năm 1997:** Bộ vi xử lý Pentium® II

Bộ vi xử lý Pentium® II có 7,5 triệu bóng bán dẫn này được tích hợp công nghệ Intel MMX, một công nghệ được thiết kế đặc biệt để xử lý các dữ liệu video, audio và đồ họa một cách hiệu quả.

Số lượng bóng bán dẫn: 7,5 triệu

Tốc độ: 200MHz, 233MHz, 266MHz, 300MHz

**Năm 1999:** Bộ vi xử lý Pentium® III

Bộ vi xử lý Pentium® III có 70 lệnh xử lý mới – những mở rộng Internet Streaming SIMD – giúp tăng cường mạnh mẽ hiệu suất hoạt động của các ứng dụng xử lý ảnh tiên tiến, 3-D, streaming audio, video và nhận dạng giọng nói. Bộ vi xử lý này được giới thiệu sử dụng công nghệ 0,25 micron.

Số lượng bóng bán dẫn: 9,5 triệu

Tốc độ: 650MHz tới 1,2GHz

**Năm 2000:** Bộ vi xử lý Pentium® 4

Bộ vi xử lý này được giới thiệu với 42 triệu bóng bán dẫn và các mạch 0,18 micron. Bộ vi xử Pentium® 4 có tần số hoạt động là 1,5 gigahertz (1,5 tỷ hertz), nhanh hơn gấp 10 nghìn lần so với bộ vi xử lý đầu tiên của Intel, bộ vi xử lý 4004, chạy ở tốc độ 108 kilohertz (108.000 hertz).

Số lượng bóng bán dẫn: 42 triệu

Tốc độ: 1,30GHz, 1,40GHz, 1,50GHz, 1,70GHz, 1,80Ghz

**Tháng 8 năm 2001:** Bộ vi xử lý Pentium 4 đạt mốc 2 GHz

**Tháng 11 năm 2002:** Bộ vi xử lý Intel Pentium 4 hỗ trợ Công nghệ Siêu phân luồng

Intel giới thiệu Công nghệ Siêu phân luồng đột phá cho bộ vi xử lý Intel® Pentium® 4 mới có tốc độ 3,06 GHz. Công nghệ Siêu phân luồng có thể tăng tốc hiệu suất hoạt động của máy tính lên tới 25%. Intel đạt mốc tốc độ mới cho máy tính với việc giới thiệu bộ vi xử lý Pentium 4 tốc độ 3,06 GHz. Đây là bộ vi xử lý thương mại đầu tiên có thể xử lý 3 tỷ chu trình một giây và được hiện thực hóa thông qua việc sử dụng công nghệ sản xuất 0,13 micron tiên tiến nhất của ngành công nghiệp.

**Tháng 11 năm 2003:** Bộ vi xử lý Intel® Pentium® 4 Extreme Edition hỗ trợ Công nghệ Siêu phân luồng tốc độ 3,20 GHz được giới thiệu. Sử dụng công nghệ xử lý 0,13 micron của Intel, bộ vi xử lý Intel Pentium 4 Extreme Edition có bộ nhớ đệm L2 dung lượng 512 kilobyte, một bộ nhớ đệm L3 dung lượng 2 megabyte và một kênh truyền hệ thống tốc độ 800 Mhz. Bộ vi xử lý này tương thích với họ chipset hiện tại Intel® 865 và Intel® 875 cũng như bộ nhớ hệ thống chuẩn.

**Tháng 6 năm 2004:** Bộ vi xử lý Intel Pentium 4 hỗ trợ Công nghệ Siêu phân luồng đạt mốc 3,4 GHz

**Tháng 4 năm 2005:** Giới thiệu nền tảng sử dụng bộ vi xử lý hai nhân đầu tiên của Intel gồm bộ vi xử lý Intel® Pentium® Extreme Edition 840 chạy ở tốc độ 3,2 GHz và một chipset Intel® 955X Express. Các bộ vi xử lý hai nhân hoặc đa nhân được phát triển bằng cách đưa hai hay nhiều nhân xử lý hoàn chỉnh vào trong một bộ vi xử lý đơn nhất giúp quản lý đồng thời nhiều tác vụ.

**Tháng 5 năm 2005:** Bộ vi xử lý Intel® Pentium® D với hai nhân xử lý – hay còn gọi là “các bộ não” – được giới thiệu cùng với họ chipset Intel® 945 Express có khả năng hỗ trợ những tính năng của các thiết bị điện tử tiêu dùng như âm thanh vòm, video có độ phân giải cao và các khả năng xử lý đồ họa tăng cường.

**Tháng 5 năm 2006:** Nhân hiệu Intel Core 2 Duo được công bố ra thế giới và sau đó 5 tháng chính hãng Intel đã đến Việt Nam để quảng bá cho sản phẩm mới này.

**Tháng 7 năm 2006:** Tập đoàn Intel công bố 10 bộ vi xử lý mới Intel Core 2 Duo và Core Extreme cho các hệ thống máy tính để bàn và máy tính xách. Những bộ vi xử lý mới này nâng cao tới 40% hiệu suất hoạt động và nhiều hơn 40% hiệu quả tiết kiệm điện năng so với bộ vi xử lý Intel® Pentium® tốt nhất. Các bộ vi xử lý Core 2 Duo có 291 triệu bóng bán dẫn.

Bảng dưới đây (Bảng 1.3) sẽ giúp chúng ta hiểu được sự khác biệt giữa các bộ xử lý mà Intel đã giới thiệu qua các năm:

Tên gọi	Năm giới thiệu	Số lượng Transistors	Microns	Tốc độ đồng hồ	Data width	MIPS
8080	1974	6,000	6	2 MHz	8 bits	0.64
8088	1979	29,000	3	5 MHz	16 bits 8-bit bus	0.33
80286	1982	134,000	1.5	6 MHz	16 bits	1
80386	1985	275,000	1.5	16 MHz	32 bits	5
80486	1989	1,200,000	1	25 MHz	32 bits	20
Pentium	1993	3,100,000	0.8	60 MHz	32 bits 64-bit bus	100
Pentium II	1997	7,500,000	0.35	233 MHz	32 bits 64-bit bus	~300
Pentium III	1999	9,500,000	0.25	450 MHz	32 bits 64-bit bus	~510
Pentium 4	2000	42,000,000	0.18	1.5 GHz	32 bits 64-bit bus	~1,700
Pentium 4 "Prescott"	2004	125,000,000	0.09	3.6 GHz	32 bits 64-bit bus	~7,000

Bảng 1.3. Tổng quan về CPU Intel

**Micros:** là chiều rộng, tính bằng Microns, của dây dẫn nhỏ nhất trên chip. Để dễ hình dung, chúng ta hãy liên tưởng đến tóc người có độ dày là 100 microns. Và như chúng ta thấy thì kích thước đặc trưng của các phần tử giảm xuống thì số lượng transistor sẽ được tăng lên.

**Data Width:** là chiều rộng của bộ tính toán Logic-Số học ALU. Một ALU 8 bit có thể cộng/trừ/nhân/... 2 số 8 bit, trong khi một ALU 32 bit có thể tính toán các số 32 bit. Một ALU 8 bit sẽ phải thực hiện 4 chỉ lệnh để cộng hai số 32 bit, trong khi một ALU 32 bit có thể làm việc này chỉ với một chỉ lệnh duy nhất. Trong đa số trường hợp, tuyến dữ liệu ngoại có cùng độ rộng với ALU, nhưng không phải lúc nào cũng vậy. Trong khi các CPU Pentium mới tìm nạp dữ liệu 64 bit tại cùng một thời điểm cho các ALU 32 bit của chúng

**MIPS:** viết tắt của cụm "millions of instructions per second", là thước đo tương đối cho hiệu năng của CPU. Các CPU thế hệ mới hiện nay có thể làm rất nhiều việc khác nhau khiến việc đánh giá bằng các giá trị MIPS mất dần ý nghĩa của chúng. Thay thế bằng MIPS, ngày nay người ta dùng MFLOPS (*Mera Floating Point Operations Per Second*) hoặc TFLOPS (*Tera Floating Point Operations Per Second*) để đánh giá hiệu năng của máy tính. Tuy nhiên, chúng ta có thể có được phán đoán chung về sức mạnh tương đối của các CPU từ cột cuối trong bảng 1.2.

## CÂU HỎI VÀ BÀI TẬP CHƯƠNG I

1. Dựa vào tiêu chuẩn nào người ta phân chia máy tính thành các thế hệ?
2. Hãy điểm qua các cột mốc quan trọng và đặc trưng cơ bản của các máy tính thế hệ thứ nhất?
3. Hãy nêu điểm đặc biệt của máy tính ENIAC so với các máy tính ra đời trước nó. Máy tính Von Neumann khác ENIAC ở điểm nào chính?
4. Hãy điểm qua các cột mốc quan trọng và đặc trưng cơ bản của các máy tính thế hệ thứ hai?
5. Hãy điểm qua các cột mốc quan trọng và đặc trưng cơ bản của các máy tính thế hệ thứ ba?
6. Hãy nêu một vài ưu điểm của công nghệ mạch tích hợp. Điểm đặc biệt trong các máy tính IBM System/360 là gì?
7. Hãy điểm qua các cột mốc quan trọng và đặc trưng cơ bản của các máy tính thế hệ thứ tư?
8. Khuynh hướng phát triển của máy tính điện tử ngày nay là gì?
9. Việc phân loại máy tính dựa vào tiêu chuẩn nào? có mấy loại máy tính?
10. Hiện nay bộ vi xử lý nào của Intel đang được bán rộng rãi ở thị trường Việt Nam? Hãy đưa ra một số loại CPU Intel thông dụng nhất ngày nay.

## Chương II: Các bộ phận cơ bản của máy tính

Vì tính phức tạp của các bộ phận cơ bản trong máy tính, trong phần này tôi chỉ giới thiệu sơ qua hình dáng bên ngoài, vị trí nằm trong máy tính, chức năng làm việc với mục đích nắm bắt được các đặc tính chính, giúp ta có thể tháo gỡ, lắp ráp một máy tính để bàn và hiểu được nguyên lý hoạt động cơ bản, cũng như liên kết giữa các thiết bị trong máy tính.

### 2.1. Bộ xử lý (CPU)

Bộ vi xử lý **CPU** (*Central Processing Unit*) là cốt lõi của một máy vi tính. Những bộ vi xử lý tương thích của các hãng như **AMD** và **Cyrix** có cách phân bố chân vi mạch và hoạt động tương thích với xử lý của Intel, vì thế chúng ta sẽ chỉ nói đến vi xử lý của Intel, hãng chiếm thị phần lớn nhất thế giới về CPU.

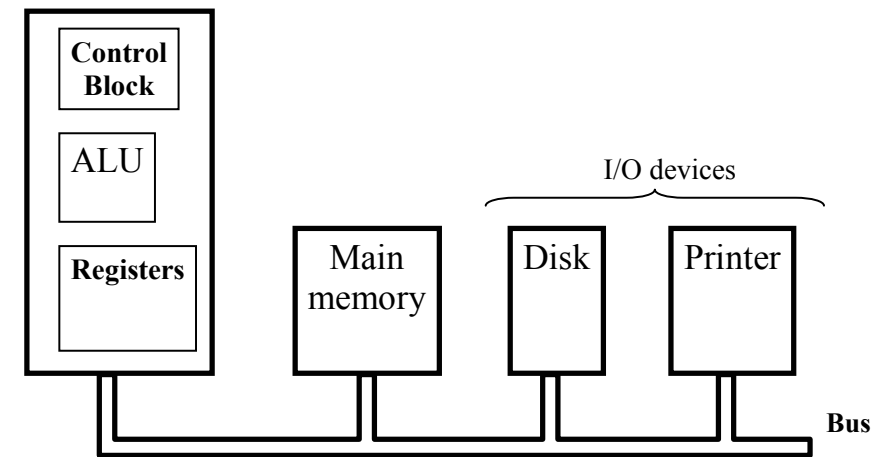
Trong hình 2.1 minh họa tổ chức máy tính theo hướng **BUS** đơn giản. CPU là bộ não của máy tính, nó đóng vai trò thi hành chương trình lưu trong bộ nhớ chính bằng cách nạp lệnh, kiểm tra chúng rồi thi hành lần lượt từng lệnh.

Bộ điều khiển (**control block**) chịu trách nhiệm tìm nạp lệnh từ bộ nhớ chính và định loại.

CPU chứa bộ nhớ nhỏ có tốc độ cao, dùng để lưu trữ kết quả tạm thời và thông tin điều khiển. Bộ nhớ này gồm các thanh ghi (**register**), mỗi thanh ghi có một chức năng cụ thể. Thanh ghi quan trọng nhất là bộ đếm chương trình (**PC- program counter**) chỉ đến lệnh sẽ thi hành tiếp theo.

**ALU**-bộ xử lý logic-số học, thực hiện các phép tính số học như phép cộng (+) và các luận lý logic như logic AND, OR.

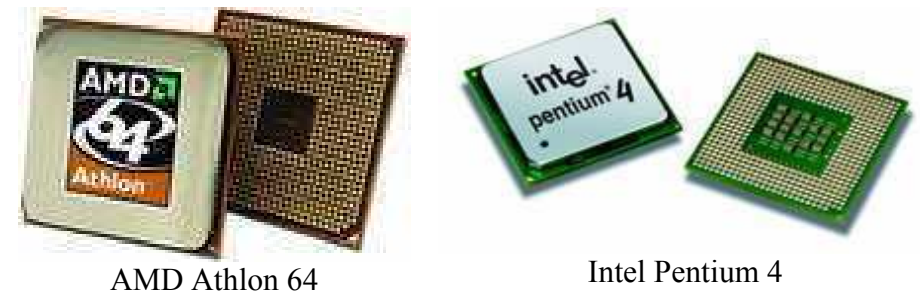
### Central Processing Unit - CPU



Hình 2.1. Tổ chức máy tính theo hướng BUS đơn giản

Phụ thuộc vào số bit trong các thanh ghi mà ta có CPU 8 bit, 16 bit, 32 bit, 64 bit. Các máy tính hiện đại ngày nay là loại CPU 64 bit.

Một thông số quan trọng khi lựa chọn mua CPU là tốc độ được đo bằng **MOPS** (*Millions of Operations Per Second*) hay ngày nay hay dùng là **TFOPS** (*Tera Floating Point Operations Per Second*), tuy nhiên trong thực tế chúng ta lại hay dựa vào tần số ghi kèm để nói đến tốc độ tương đối của CPU. Hình dáng bên ngoài của các CPU hiện đại ngày nay đều có dạng như hình 2.2.



AMD Athlon 64

Intel Pentium 4

Hình 2.2. Hình dáng bên ngoài CPU.

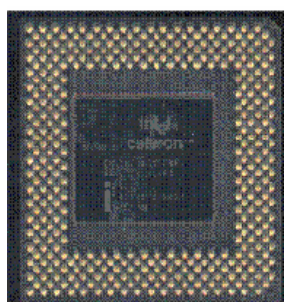
Các thông số quan trọng của bộ vi xử lý:

**a) Hãng sản xuất và model (Processor make and model)**

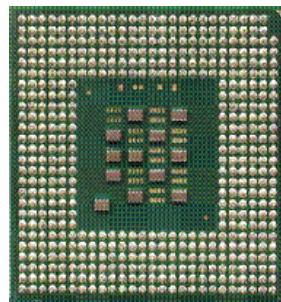
Trên thị trường máy tính cá nhân hiện nay chủ yếu có 2 hãng sản xuất CPU chiếm hầu hết thị phần là AMD và Intel. Tuy các CPU của 2 hãng này có những đặc tính và tốc độ gần như nhau, nhưng không thể cài đặt một AMD-CPU vào một bo mạch chính (Motherboard) dùng cho Intel-CPU và ngược lại.

**b) Dạng Socket (Socket type)**

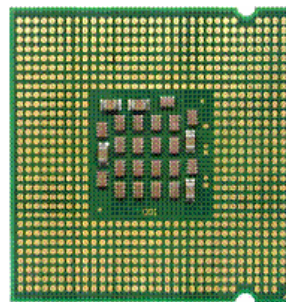
Tính chất này xác định số lượng, hình dạng, cũng như cách sắp xếp các chân và như vậy mỗi loại CPU phải được gắn vào bo mạch chính có socket loại đó hay nói cách khác là loại khe cắm của CPU. Trong bảng 2.1 cho thấy các loại CPU nào dùng với loại Socket nào và loại nào có thể nâng cấp (upgrade) được, còn hình 2.3 cho thấy một số bộ vi xử lý với các dạng Socket khác nhau.



Socket 370



Socket 478



Socket 775

Hình 2.3. Một số loại Socket

**c) Tốc độ đồng hồ xung (Clock Speed - CS)**

Tốc độ đồng hồ xung của CPU thường được tính bằng megahertz (MHz) hoặc gigahertz (GHz). Chúng ta thường dùng thông số này để nói đến tốc độ xử lý của CPU. Tuy nhiên, không phải lúc nào CS của CPU nào lớn hơn thì CPU đó cũng mạnh hơn. Ví dụ, một 3.0 GHz Celeron CPU sẽ chậm hơn 2.6

GHz Pentium 4, bởi vì Celeron có bộ nhớ đệm cache L2 nhỏ hơn và tốc độ của kênh truyền chủ (host-bus) thấp hơn.

Đặc biệt là giữa AMD và Intel có sự khác biệt lớn, AMD-CPU chạy với CS thấp hơn Intel, nhưng làm khoảng 50% công việc nhiều hơn Intel trong một xung đồng hồ (clock tick). Do đó một AMD Athlon 64 chạy ở 2.0 GHz sẽ tương đương với Intel P4 chạy ở 3.0 GHz. Chính vì CS của AMD-CPU luôn thấp hơn của intel, nên AMD mới có các ký hiệu model như 3000+ để chỉ ra rằng tốc độ của nó tương đương với 3.0 GHz của Intel.

Socket	Khả năng nâng cấp	CPU gốc	CPU có thể nâng cấp
Slot 1	không	Pentium II/III, Celeron	không có
Slot A	không	Athlon	không có
370	có, nhưng rất hạn chế	Celeron, Pentium III, VIA	Celeron, Pentium III
423	không	Pentium 4	không có
462	có	Athlon, Athlon XP, Sempron	Sempron
478	có	Celeron, Celeron D, Pentium 4	Celeron D, Pentium 4
754	tốt	Sempron, Athlon 64	Sempron, Athlon 64
775	rất tốt	Celeron D, Pentium 4	Celeron D, Pentium 4, Pentium D
939	rất tốt	Athlon 64, Athlon 64/FX	Athlon 64, Athlon 64/FX, Athlon 64 X2
940	rất tốt	Athlon 64 FX, Opteron	Athlon 64 FX, Opteron

Bảng 2.1. Các loại socket và CPU tương ứng

**d) Tốc độ đường truyền chủ (host-bus speed)**

Hay còn gọi là *front-side bus (FSB) speed*, hay *FSB speed*, hay chỉ đơn giản là *FSB* để chỉ ra tốc độ truyền dữ liệu giữa CPU và các vi mạch (chipset). Tốc độ FSB giúp tăng hiệu suất của CPU ngay cả khi CPU có cùng một CS. AMD và intel thực hiện truyền dữ liệu giữa bộ nhớ và cache khác nhau, nhưng bản chất đều là số lượng lớn nhất của một gói dữ liệu có thể được truyền trong một giây. Theo cách tính này thì một máy tính với FSB là 100 MHz, nhưng trong một chu kỳ xung đồng hồ lại truyền được 4 lần thì tương đương với một máy tính cùng CPU nhưng FSB hoạt động ở FSB là 400 MHz.

**e) Kích thước bộ nhớ đệm (Cache size)**

Cache là một loại bộ nhớ có tốc độ cao hơn rất nhiều so với bộ nhớ chính (main memory). Các CPU dùng hai loại bộ nhớ cache L1 (Level 1) và L2 (Level 2) để tăng hiệu suất của CPU bằng cách tạm thời lưu trữ các dữ liệu cần truyền giữa CPU và bộ nhớ chính vào trong cache. Cache L1 là cache nằm trong CPU và nó không thể thay đổi nếu không thiết kế lại CPU. Cache L2 là cache nằm ngoài nhân CPU, có nghĩa là có thể chế tạo CPU với kích thước L2 khác nhau. Như vậy cache càng lớn thì càng tốt, càng giúp cho tốc độ xử lý chung của máy tính nhanh hơn.

➤ **Ví dụ:**

P4 2.8Ghz (511)/Socket 775/ Bus 533/ 1024K/ Prescott CPU có nghĩa là:

- P4, viết tắt của từ Pentium 4, tức là tên của loại CPU. Đây là CPU của hãng Intel. 2.8 Ghz, chỉ tốc độ xung đồng hồ của vi xử lý. Con số này là một trong những thước đo sức mạnh của vi xử lý, tuy vậy nó không phải là tất cả. Đôi lúc chỉ là một con số nhằm so sánh tương đối sức mạnh của CPU. Con số 511 phía sau con số thể hiện chất lượng và vị thế của con CPU trong toàn bộ các sản phẩm

thuộc cùng dòng. Con số này là một quy ước của hãng Intel. Số càng cao chứng tỏ CPU càng tốt.

- Socket 775, chỉ loại khe cắm của CPU. Đây là đặc tính để xét sự tương hợp giữa vi xử lý và mainboard. Bo mạch chủ phải hỗ trợ loại socket này thì vi xử lý mới có thể hoạt động được.

- Bus 533, chỉ tốc độ "lỗi" của đường giao tiếp giữa CPU và mainboard. Một CPU được đánh giá nhanh hay chậm tùy thuộc khá lớn vào giá trị này. Vi xử lý chạy được bus 533 thì đương nhiên hơn hẳn so với vi xử lý chỉ chạy được bus 400 Mhz.

- 1024K, chỉ bộ nhớ đệm của vi xử lý. Đây là vùng chứa thông tin trước khi đưa vào cho vi xử lý trung tâm (CPU) thao tác. Thường thì tốc độ xử lý của CPU sẽ rất nhanh so với việc cung cấp thông tin cho nó xử lý, cho nên, không gian bộ nhớ đệm (cache) càng lớn càng tốt vì CPU sẽ lấy dữ liệu trực tiếp từ vùng này. Một số Vi xử lý còn làm bộ nhớ đệm nhiều cấp. Số 1024 mà bạn thấy đó chính là dung lượng bộ nhớ đệm cấp 2, 1024 KB = 1 MB.

- Prescott chính là tên một dòng vi xử lý của Intel. Dòng vi xử lý này có khả năng xử lý video siêu việt nhất trong các dòng vi xử lý cùng công nghệ của Intel. Tuy nhiên, đây là dòng CPU tương đối nóng, tốc độ xung đồng hồ tối đa đạt 3.8 Ghz.

➤ **Sự khác biệt cơ bản giữa AMD và Intel**

**a) Cách đặt tên**

**AMD**

Được gọi theo tên và không hề xuất hiện xung nhịp thực của CPU, thay vào đó là các con số để so sánh nó tương đương với thế hệ Intel Pentium tương ứng. Ví dụ trong tên gọi của CPU AMD Athlon 64 3000+, không hề xuất hiện xung nhịp thực của CPU. Đây là điều hơi khác lạ đối với người Việt Nam vì thường quen



đánh giá khả năng của CPU theo tên gọi “có xung nhịp kèm theo”, ví dụ như 1 mẫu đối thoại sau:

A: Máy nhà B dùng CPU gì vậy ? Máy tôi dùng Pentium 4 2GHz.

B: Máy của tôi dùng CPU Pentium 4 3GHz.

A: Vậy là máy bạn nhanh hơn máy tôi rồi.

Cách nghĩ và gọi tên như vậy là do thói quen dùng CPU Intel. Cách so sánh hiệu năng như trên sẽ đúng nếu 2 CPU đó được sản xuất theo cùng 1 công nghệ, vì khi đó, CPU nào có xung nhịp cao hơn sẽ có hiệu năng tốt hơn. Nhưng nếu ta so sánh 2 CPU của 2 hãng khác nhau, công nghệ chế tạo khác nhau thì hiệu năng không còn đi đôi với xung nhịp.

Ví dụ như khi so sánh 2 CPU AMD và Intel có cùng tốc độ 1,8GHz, CPU AMD có hiệu năng vượt trội hoàn toàn so với CPU Intel.

Chính từ điều trên mà hãng AMD đã không còn đặt tên CPU của mình dựa theo xung nhịp nữa. Bắt đầu từ dòng Athlon XP của thế hệ K7 trở đi, AMD đã đặt tên sản phẩm của mình là tên sản phẩm cộng với 1 con số phía sau.

Vd: AMD Athlon XP 2500+ : con số 2500+ có ý nghĩa là CPU Athlon XP này có hiệu năng tương đương 1 CPU 2500MHz cùng cấp của Intel.

Tương tự như vậy, CPU Athlon 64 3000+ 1800MHz được AMD xác định là có hiệu năng tương đương CPU 3000MHz của Intel. Sự tương đương ở đây được đánh giá trên nhiều mặt và có giá trị tương đối.

### Intel

Sau một thời gian AMD đưa ra cách đặt tên mới cho dòng CPU để bán, Intel cũng đã nhận ra khuyết điểm về tên gọi CPU có kèm theo xung nhịp. Khuyết điểm đó là họ không thể đưa ra thị trường các CPU có tốc độ ngày càng cao được. Vì kiến trúc NetBurst được Intel áp dụng cho dòng CPU Pentium 4 có thể áp dụng để sản xuất các CPU có xung nhịp cao như 4-5GHz hoặc hơn

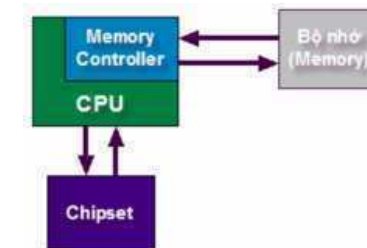
nữa nhưng xung nhịp cao luôn đi đôi với vấn đề như lượng điện năng tiêu thụ, hiệu năng không tỉ lệ thuận với mức xung tăng thêm, và đặc biệt là vấn đề tản nhiệt.

Khi tung ra dòng CPU Pentium 4 dùng đế cắm LGA 775, Intel đã không còn kèm theo xung nhịp trong tên gọi CPU nữa. Họ đặt tên CPU theo từng serie như hãng xe hơi BMW thường làm. Ví dụ như Pentium 4 630. 630 là tên 1 model CPU thuộc serie 6xx.

### b) Các công nghệ tiêu biểu

#### AMD

- Tích hợp Memory Controller (Hình 2.4) : Trong hầu hết các CPU mới, Memory Controller nằm trong nhân CPU, có cùng xung nhịp với CPU (CPU có tốc độ 1,8GHz thì Memory Controller cũng có tốc độ 1,8GHz). Dữ liệu từ RAM sẽ được truyền trực tiếp vào CPU, độ trễ thấp, không còn hiện tượng thất cổ chai nữa. Lúc này người dùng càng sử dụng RAM tốc độ cao thì càng có lợi.

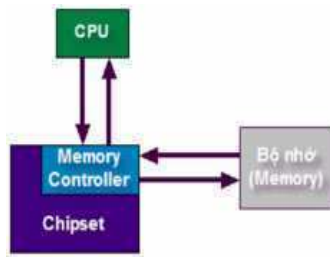


Hình 2.4. Bố trí memory kiểu AMD

- Công nghệ HyperTransport : đây là công nghệ kết nối trực tiếp theo kiểu điểm-điểm, kết nối với RAM và chipset bằng HyperTransport bus (HTT) có băng thông rất lớn và được mở đồng thời 2 chiều (như hình minh họa 2.4).

### Intel

- Intel vẫn sử dụng kiểu thiết kế Memory Controller nằm tại chipset (Hình 2.5), Memory Controller này có tốc độ nhất định, có tên là Front Side Bus. Dữ liệu từ RAM bắt buộc phải đến chipset rồi mới vào được CPU. Độ trễ của thiết kế này lớn và luôn tồn tại nút thắt cổ chai tại chipset.



Hình 2.5. Bố trí memory kiểu Intel

- Công nghệ Hyper Threading Sử dụng công nghệ này giúp tận dụng hiệu quả hơn tài nguyên dư thừa của CPU, CPU Intel có Hyper Threading sẽ chạy nhanh hơn CPU Intel không có Hyper Threading khoảng từ 10%-20%. CPU 1 nhân có Hyper Threading sẽ được hệ điều hành nhận diện thành 2 CPU (1 physical, 1 logical) nhưng đó vẫn là 1 CPU đơn luồng, tại 1 thời điểm thì CPU chỉ thực hiện được duy nhất 1 tác vụ.

### c) Tỏa nhiệt

Đây là một thông số mà ở Việt Nam đáng được quan tâm vì điều kiện khí hậu nước ta rất nóng. Các bộ CPU của AMD trước đây thường tỏa nhiệt nhiều hơn và không thích hợp cho khí hậu nóng như ở nước ta. Có thể chính vì điểm này mà AMD không có đầu tư quảng bá sản phẩm ở Việt Nam. Tuy nhiên từ AMD K8 với công nghệ 90nm hiện nay rất mát, không còn nóng như thế hệ K6, K7. CPU Athlon 64 3000+ cũng không là ngoại lệ.

Trong khi đó do Intel chú trọng việc tăng xung tần đã làm cho các CPU của mình tỏa ra một nhiệt độ không thể chấp nhận được. Trong thời gian gần đây Intel cũng đã nhận ra điều này và đang đầu tư nhiều vào giải quyết vấn đề này.

Trong điều kiện khí hậu nhiệt đới Việt Nam, 36°C của AMD là một nhiệt độ rất lý tưởng, CPU tỏa nhiệt ít, người dùng không phải lo lắng về tiếng ồn, về vấn đề quạt tản nhiệt một khi sử dụng CPU AMD. Lúc nào hệ thống dùng AMD cũng mát và tĩnh lặng.

Còn đối với Intel, nhiệt độ CPU cao góp phần làm nhiệt độ thùng máy và môi trường tăng lên. Người sử dụng cũng phải lưu ý đến vấn đề quạt tản nhiệt vì quạt tản nhiệt của Intel quay với tốc độ cao, đặc biệt là khi hoạt động vào ban đêm, tiếng ồn do hệ thống dùng Intel phát ra sẽ gây khó chịu đối với người dùng.

✦ *Tóm lại khi mua CPU thì ngoài việc cần chú ý các thông số về giá cả, công nghệ, tốc độ xử lý thì còn cần lưu ý đến loại socket để đảm bảo sự tương thích của các thiết bị khi lắp ráp. Vấn đề tỏa nhiệt ở Việt Nam là quan trọng cho nên cũng cần chú ý.*

## **2.2. Bản mạch chính (Bo mạch chủ, Mainboard)**

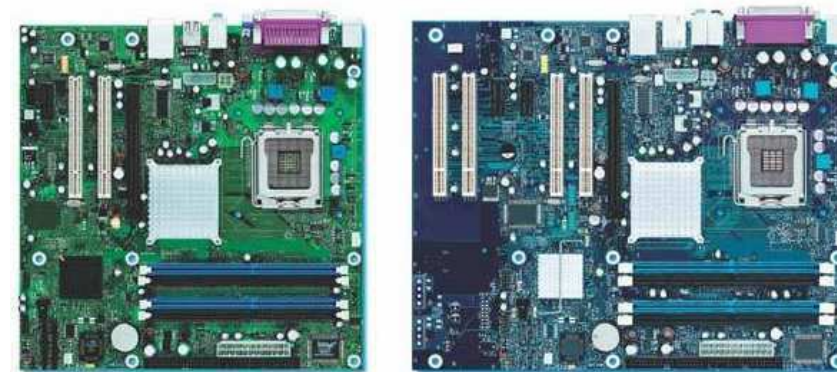
Mainboard là trung tâm điều khiển mọi hoạt động của một máy tính và đóng vai trò là trung gian giao tiếp giữa CPU và các thiết bị khác của máy tính. Bản mạch chính là nơi để chứa đựng (cắm) những linh kiện điện tử và những chi tiết quan trọng nhất của một máy tính cá nhân như: bộ vi xử lý CPU (central processing unit), các thành phần của CPU, hệ thống bus, bộ nhớ, các thiết bị lưu trữ (đĩa cứng, ổ CD,...), các card cắm (card màn hình, card mạng, card âm thanh) và các vi mạch hỗ trợ. Do các vai trò của nó như vậy nên bản mạch chính cần thỏa mãn nhiều điều kiện về cấu trúc và đặc tính điện khắt khe như: gọn, nhỏ và ổn định với nhiễu từ bên ngoài.

Cũng như nhiều loại máy điện, điện tử khác, mainboard và vỏ máy phải tuân thủ theo các quy định chung về an toàn điện, an toàn nhiễu điện từ (đặc biệt do tần số làm việc của máy vi tính nằm trong dải tần sóng viba nên rất dễ gây nhiễu cho các máy móc khác xung quanh). Bo mạch chủ được sản xuất bằng công nghệ mạch in PCB (Printed Circuit Board). Do số chân nổi của vi mạch ngày càng nhiều (Core 2 Duo 775 chân) nên số lượng dây dẫn trên bản mạch ngày càng nhiều khiến diện tích bản mạch cũng tăng theo nếu không thay đổi công nghệ. Số chân nổi và độ phức tạp gia tăng khiến việc thiết kế bản mạch thêm rắc rối. Để giải quyết vấn đề này, người ta dùng mạch in nhiều lớp (multi layer PCB) cho máy vi tính hiện đại. Bản mạch chính được sản xuất theo lối xếp chồng (sandwich) tương tự công nghệ chế tạo vi mạch và ngày nay có từ 4 đến 8 lớp. Một công nghệ nữa góp phần thu nhỏ kích thước bản mạch chính là công nghệ gắn chip tiết SMT (surface mounted technology). Công nghệ này cho phép dán trực tiếp vi mạch lên bản mạch chính, giảm bớt công nghệ khoan bản mạch và giảm đáng kể kích thước vỏ vi mạch.

### ➤ Các đặc tính quan trọng trong mainboard:

#### a) Form factor

Đặc tính này qui định kích thước của mainboard cũng như cách bố trí nó trong thân máy tính (case). Chuẩn thống trị hiện nay trên máy tính để bàn nói chung chính là **ATX** (*Advanced Technology Extended*) 12V, được thiết kế bởi Intel vào năm 1995 và đã nhanh chóng thay thế chuẩn AT cũ bởi nhiều ưu điểm vượt trội. Nếu như với nguồn **AT**, việc kích hoạt chế độ bật được thực hiện qua công tắc có bốn điểm tiếp xúc điện thì với bộ nguồn ATX bạn có thể bật tắt bằng phần mềm hay chỉ cần nối mạch hai chân cắm kích nguồn (dây xanh lá cây và một trong các dây Ground đen). Các nguồn ATX chuẩn luôn có công tắc tổng để có thể ngắt hoàn toàn dòng điện ra khỏi máy tính. Ngoài ra còn có microATX có kích thước nhỏ hơn ATX. Hình 2.6 cho thấy một dạng của 2 loại mainboard này.



Hình 2.6. Mainboard microATX (bên trái) và ATX (bên phải)

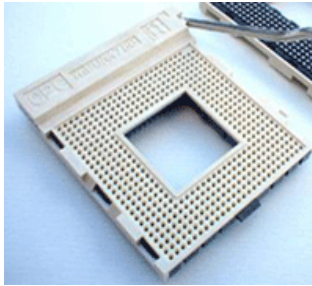
- ❖ **BTX** – Vào năm 2004, Intel bắt đầu sản xuất loại mainboard BTX (Balanced Technology eXtended). BTX và thùng máy mới sẽ sử dụng ít quạt hơn nên máy tính chạy êm hơn và có khả năng nhiệt độ cũng thấp hơn những hệ thống dùng chuẩn ATX (Advanced Technology Extended) hiện nay. Do vậy, bo mạch BTX có nhiều thay đổi đáng kể trong cách bố trí các thành phần và thiết kế tản nhiệt.

#### b) Giao tiếp với CPU

Để gắn CPU lên trên bo mạch chủ ta dùng hai dạng cơ bản là dạng khe cắm (slot) hoặc chân cắm (socket). Dạng khe cắm là một rãnh dài nằm ở khu vực giữa mainboard dùng cho các máy tính đời cũ như PII, PIII. Hiện nay hầu như người ta không sử dụng dạng khe cắm này nữa.

Dạng chân cắm (socket) là một khối hình vuông gồm nhiều chân (hình 2.7). Hiện nay đang sử dụng socket 478, 775 cho dòng CPU Intel và 939, 940, AM2 cho dòng CPU của hãng AMD. Con số chỉ ra trong socket tương ứng với số chân của CPU.





Hình 2.7. CPU socket

### c) Khe cắm card màn hình AGP (Array Graphic Adapter)

Dùng để cắm card đồ họa vào mainboard và có dạng như trong hình 2.8. AGP lại chia làm nhiều loại với các tốc độ khác nhau như 1x, 2x, 4x và đang thịnh hành hiện nay là 8x. Giữa các loại AGP cũng có sự khác nhau ở dạng khe cắm, do đó khi mua card đồ họa ta phải chú ý đến điểm này. Tuy nhiên các máy tính hiện đại ngày nay có xu hướng không dùng khe cắm AGP cho card đồ họa nữa mà thay vào đó là loại khe cắm PCI Express 16x với băng thông lớn hơn rất nhiều lần.



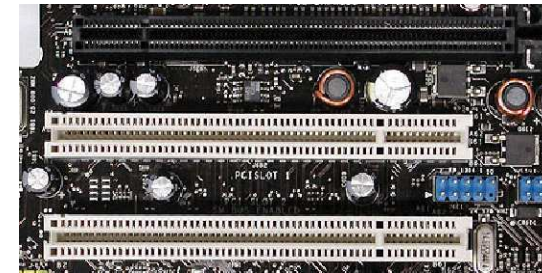
Hình 2.8. AGP slot

### d) Khe cắm PCI Express:

Hầu hết các máy tính cao cấp hiện nay đều được trang bị khe cắm mở rộng PCI Express (PCIe) cùng với các khe cắm PCI tiêu chuẩn. Khe cắm chuẩn PCI Express hỗ trợ băng thông cao hơn 30 lần so với chuẩn PCI và thực sự có khả năng thay thế hoàn toàn khe cắm PCI lẫn AGP.

Khe cắm PCI Express có nhiều độ dài khác nhau, tùy thuộc vào dung lượng dữ liệu có thể hỗ trợ. Khe cắm PCI Express x1 thay cho khe PCI tiêu chuẩn, có chiều dài khoảng 1" (hay 26mm) và có khả năng hỗ trợ đến 250 MBps dữ liệu vào/ra tại cùng thời điểm.

Khe cắm PCI Express x16, giống như khe PCI thông thường, có khả năng thay cho khe cắm card đồ họa AGP có chiều dài 90 mm (khoảng 3,5"). Một khe PCI Express x16 có thể truyền dữ liệu nhanh hơn 16 lần so với khe x1, khoảng 4 GBps dữ liệu vào/ra cùng lúc. Trên hình 2.9 cho thấy hai loại khe cắm PCI và PCI Express x16.



Hình 2.9. PCI (màu trắng) và PCI Express x16 (màu đen)

### e) Giao diện cắm ổ cứng

Để nối ổ cứng với mainboard thường dùng các loại chuẩn IDE, SCSI, ATA, SATA. Mỗi loại có giao diện riêng và không thể cắm ổ cứng loại dùng SATA vào mainboard chỉ có loại IDE.

#### ■ IDE (Intergrated Drive Electronics)

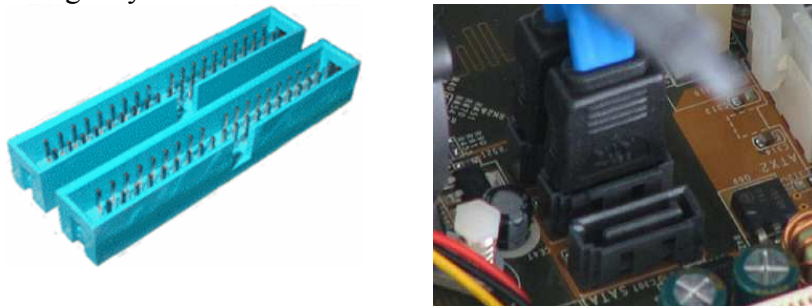
Đầu cắm có 40 chân dạng đinh trên mainboard để cắm các loại ổ cứng, CD, DVD (hình 2.10). Mỗi mainboard thường có 2 IDE và thường dùng chân cắm chính IDE1, để cắm dây cáp nối với ổ cứng chính, còn chân cắm phụ IDE2 để cắm dây cáp nối với ổ cứng thứ 2 hoặc các ổ CD, DVD...

#### ■ Serial ATA (SATA):

Thay thế cho chuẩn ATA song song có tốc độ chậm hơn (hay còn gọi là PATA hoặc EIDE), được sử dụng từ trước đến nay để nối đĩa cứng và ổ quang với Mainboard. Cổng SATA xuất hiện lần đầu trên các Mainboard cách đây vài năm và nhiều Mainboard hiện nay hỗ trợ đồng thời SATA và PATA.

Đầu nối SATA có kích thước nhỏ hơn so với đầu nối PATA và chỉ hỗ trợ một ổ đĩa. Do vậy, ta không cần quan tâm đến các jumper để thiết lập đĩa master hoặc slave như trong trường hợp sử dụng chuẩn PATA. Cáp SATA nhỏ hơn nên ít gây lộn xộn bên trong thùng máy như khi dùng cáp PATA và quan trọng nhất là cáp nhỏ hơn giảm thiểu nguy cơ gây ra tình trạng "quá nóng" bên trong thùng máy (cáp PATA to hơn nên có thể cản trở dòng không khí lưu thông trong thùng máy). Hơn thế nữa, đầu nối SATA dễ dàng kéo dài ra ngoài thùng máy để sử dụng với các đĩa cứng và ổ quang gắn ngoài.

Ổ đĩa SATA yêu cầu phải có đầu nối cáp điện đặc biệt thay cho đầu nối 5V tiêu chuẩn vẫn dùng cho ổ đĩa IDE. Nhiều máy tính mới có kèm theo một đầu nối điện SATA nhưng thường không có ở những máy đời cũ.



Hình 2.10. Khe cắm IDE và SATA

#### f) Khe cắm cho RAM (Ram slot)

Trên mainboard thường có hai hoặc 4 khe để cắm các thanh RAM và mainboard. Trên mỗi khe cắm RAM luôn có cần gạt ở 2 đầu để kẹp chặt thanh RAM lên mainboard và giữ cho các mối nối bền vững hơn. Tùy vào loại RAM (SDRAM, DDRAM, RDRAM) mà giao diện khe cắm sẽ khác nhau, cho nên khi cần thay RAM hoặc gắn thêm RAM mới cần để ý tới điểm này.

Các máy tính cũ thường dùng SDRAM có 168 chân và có hai khe cắt ở phần chân cắm, do đó khe cắm RAM trên mainboard sẽ là một khe cắm được chia thành ba phần. Trong khi DDRAM có 184 chân và chỉ có một khe cắt ở giữa phần chân cắm, tương ứng với khe cắm trên mainboard chia thành hai phần. DDRAM2 cũng

chia làm hai phần nhưng không dùng được loại khe cắm cho DDRAM. Một loại RAM đời mới nữa là RDRAM mà khe cắm cho nó cũng được chia làm 3 phần như SDRAM, nhưng cách chia khác nhau và chúng không dùng chung của nhau được.

#### Ví dụ:

Mainboard :ASUS Intel 915GV P5GL-MX, Socket 775/ s/p 3.8Ghz/ Bus 800/ Sound& Vga, Lan onboard/PCI Express 16X/ Dual 4DDR400/ 3 PCI/ 4 SATA/ 8 USB 2.0. có nghĩa là:

- ASUS Intel 915GV P5GL-MX, đơn giản, đây chỉ là tên của loại bo mạch chủ của hãng Asus.
- Socket 775 như đã nói ở trên, là loại khe cắm cho CPU
- s/p 3.8 Ghz đó chính là tốc độ xung đồng hồ tối đa của CPU mà bo mạch chủ hỗ trợ.
- Bus 800, chỉ tần số hoạt động tối đa của đường giao tiếp dữ liệu của CPU mà bo mạch chủ hỗ trợ. Thường thì bus tốc độ cao sẽ hỗ trợ luôn các CPU chạy ở bus thấp hơn.
- PCI Express 16X là tên của loại khe cắm card màn hình và bo mạch chủ. Khe PCI Express là loại khe cắm mới nhất, hỗ trợ tốc độ giao tiếp dữ liệu nhanh nhất hiện nay giữa bo mạch chủ và Card màn hình. Con số 16X thể hiện một cách tương đối bằng thông giao tiếp qua khe cắm, so với AGP 8X, 4X mà bạn có thể thấy trên một số bo mạch chủ cũ. Tuy bằng thông giao tiếp trên lý thuyết là gấp X lần, thế nhưng tốc độ hoạt động thực tế không phải như vậy mà còn phụ thuộc vào rất nhiều yếu tố khác như lượng RAM trên card, loại GPU (CPU trung tâm của card màn hình).

- Sound& Vga, Lan onboard: bo mạch chủ này đã được tích hợp sẵn card âm thanh, card màn hình và card mạng phục vụ cho việc kết nối giữa các máy tính với nhau.

- Dual 4DDR400: trên bo mạch chủ này có 4 khe cắm Bộ nhớ (RAM), hỗ trợ tốc độ giao tiếp 400 Mhz. Dựa vào thông số này, bạn có thể lựa chọn loại bộ nhớ (RAM) với tốc độ thích hợp để nâng cao tính đồng bộ và hiệu suất của máy tính. Chữ Dual là viết tắt của Dual Chanel, tức là bo mạch chủ hỗ trợ chế độ chạy 2 thanh RAM song song. Với công nghệ này, có thể nâng cao hiệu suất và tốc độ chuyển dữ liệu của RAM.

- 3PCI, 4SATA, 8 USB 2.0: trên bo mạch chủ có 3 khe cắm PCI dành để lắp thêm các thiết bị giao tiếp với máy tính như card âm thanh, modem gắn trong... 4SATA là 4 khe cắm SATA, một loại chuẩn giao tiếp dành cho đĩa cứng. SATA thì nhanh hơn và ổn định hơn so với chuẩn IDE. 8 cổng cắm USB 2.0 được hỗ trợ trên bo mạch chủ. USB 2.0 thì nhanh hơn USB 1.1. USB 2.0 thì tương thích luôn với các thiết bị chỉ có USB 1.1.

### 2.3. Ổ mềm (FDD)

Cùng với sự xuất hiện của máy tính cá nhân thì một vấn đề nan giải cũng xuất hiện. Đó là làm thế nào để phổ biến những chương trình ứng dụng đến người dùng? để giải quyết vấn đề này, đầu tiên con người đã phát minh ra đĩa mềm (floppy disk)(hình 2.11).



Hình 2.11. ổ đĩa FDD

Hãng IBM đã nghĩ ra công nghệ này đầu tiên. Ổ đĩa mềm bao gồm phần cơ khí và phần điện tử điều khiển tự động cũng như bộ phận đọc/ghi và giải mã. Ổ đĩa phải đảm bảo độ quay chính xác (300 hoặc 360 vòng/phút với sai số 1 đến 2%). Khả năng định vị của đầu từ cũng rất chính xác đến vài micromet chỉ trong thời gian vài miligiây rất ngắn. Đĩa mềm có các tính chất chung rất giống với HDD. Điểm khác nhau đặc biệt là đầu từ của HDD di chuyển trên bề mặt đĩa nhờ một đệm không khí, trong khi trên đĩa mềm thì đầu từ trực tiếp trượt trên bề mặt đĩa. Kết quả là cả đầu từ và đĩa bị ma sát làm cho nhanh chóng bị hỏng. Chính vì thế nên khi không có đòi hỏi đọc/ghi lên đĩa thì đầu từ được cất đi và đĩa dừng lại không quay như trong trường hợp HDD. Điều này làm ảnh hưởng lớn đến tốc độ của đĩa vì phải mất một khoảng thời gian để kích hoạt đĩa quay trở lại khi cần thiết.

Có 2 loại đĩa mềm: 5,25 inch và 3,5 inch. Cả hai đều có thể tích hợp mật độ ghi thấp (Low Density - LD), hoặc cao (High Density - HD). Những thông số chính của 4 loại đĩa mềm đưa ra trong bảng 2.2.

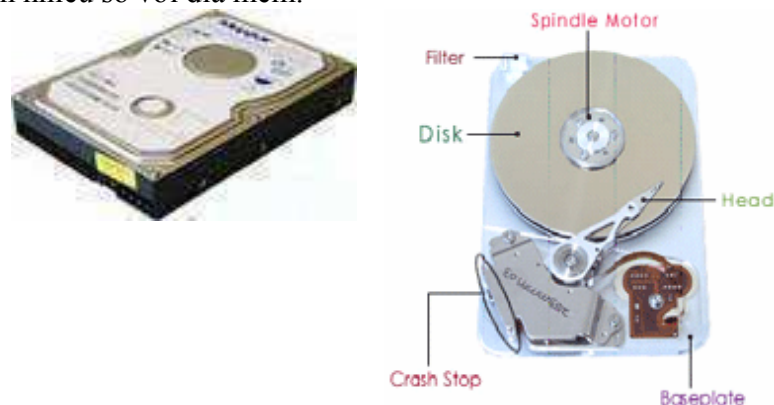
Đặc tính	LD 5,25	HD 5,25	LD 3,5	HD 3,5
Kính thước	5,25	5,25	3,5	3,5
Dung lượng	360Kbyte	1,2 MB	720 Kbyte	1,44MB
Số đường	40	80	80	80
Số sector trong 1 đường	9	15	9	18
Số đầu đọc	2	2	2	2
Số vòng quay/ 1 phút	300	300	300	300
Tốc độ truyền dữ liệu Kbit/s	250	500	250	500

Bảng 2.2. Các đặc tính của đĩa mềm



## 2.4. Ổ cứng (HDD)

Nguyên tắc hoạt động của đĩa cứng (hình 2.12) hoàn toàn tương tự đĩa mềm. Điểm khác nhau căn bản là đĩa cứng được cài đặt ngay trong ổ đĩa, có cấu tạo bền và có dung lượng lưu trữ lớn hơn nhiều so với đĩa mềm.



Hình 2.12. Bên ngoài và bên trong HDD

Đĩa cứng được làm từ vật liệu nền cứng như nhôm, thủy tinh hay gốm. Lớp vật liệu nền được phủ một lớp tiếp xúc bám (nickel) phía trên lớp tiếp xúc bám là màng từ lưu trữ dữ liệu (Cobalt). Bề mặt trên cùng được phủ một lớp chống ma sát (graphit hay saphia). Do cấu tạo cơ học bền, đĩa cứng có thể quay với tốc độ lớn (7200 vòng/phút), nhanh gấp 20 lần đĩa mềm. Một ổ đĩa cứng thường có hai hay nhiều đĩa. Tốc độ máy nhập đĩa cứng nhanh hơn nhiều lần so với đĩa mềm, thời gian truy nhập được phân loại như sau:

- Chậm:  $t > 40\text{ms}$ ,
- Trung bình:  $28\text{ms} < t < 40\text{ms}$ .
- Nhanh:  $18\text{ms} < t < 28\text{ms}$ .
- Cực nhanh:  $t < 18\text{ms}$ .

Mật độ lưu trữ trên đĩa cứng rất lớn (10000 bit/inch), vì thế vật liệu từ như ôxyt sắt không dùng được cho đĩa cứng và được thay thế bởi một lớp kim loại từ như cobalt hay Nicken. Các ổ đĩa cứng hiện đại ngày nay có mật độ thông tin vào khoảng 100 đến

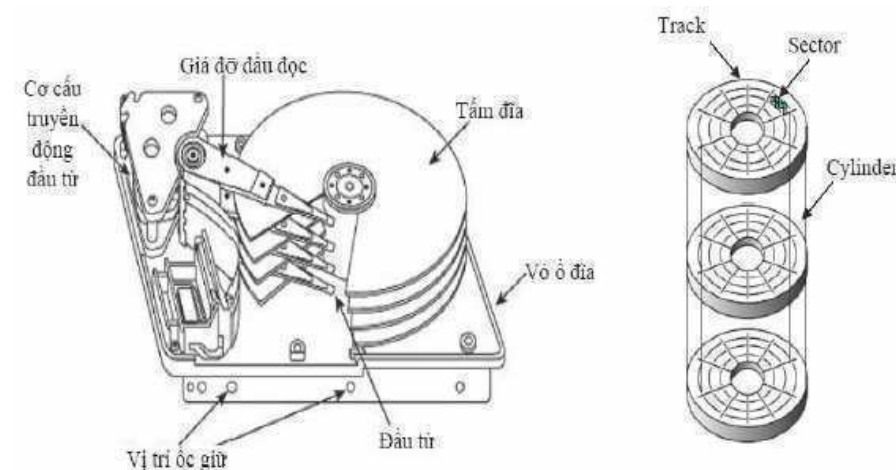
300 Mbit trong một inch vuông. Hai yếu tố quan trọng quyết định đến mật độ lưu trữ cao là:

- Cấu trúc hạt của vật liệu từ thật nhỏ,
- Bề mặt đĩa thật phẳng để giữ khoảng cách giữa đầu đọc và mặt đĩa tại giá trị tối thiểu.

Khác với đĩa mềm, do tốc độ quay nhanh, đầu đọc/ghi không được tiếp xúc với bề mặt đĩa cứng. Đầu đọc được giữ cách xa mặt đĩa qua một lớp đệm không khí. Lớp đệm không khí này được hình thành khi đĩa quay với tốc độ cao.

Khoảng cách giữa đầu từ và mặt đĩa chỉ vào khoảng vài micrômét, nhỏ hơn rất nhiều một hạt bụi khói trung bình. Vì thế phía bên trong ổ đĩa cứng cần được giữ thật sạch. Người sử dụng không được phép mở ổ đĩa trong môi trường bình thường. Để sản xuất hoặc sửa chữa đĩa cứng người ta cần đến môi trường siêu sạch như thường gặp trong công nghiệp vi điện tử.

HDD được làm từ một hay nhiều đĩa nhôm (platter) với một lớp từ (hình 2.13). Ban đầu nó có kích thước 50cm, còn bây giờ từ 3 đến 12 cm, còn ở máy xách tay thì nhỏ hơn 3cm, kích thước này vẫn ngày càng được thu nhỏ. Mỗi platter được chia thành từng rãnh (track), mỗi rãnh lại được chia thành từng sector.



Hình 2.13. Cấu tạo HDD

Khi mua đĩa cứng ta cần xem xét các thông số chính:

- Tốc độ quay: hiện nay thông dụng loại 7200 vòng/1 phút (loại chậm hơn - 5400 vòng hoặc 3600 vòng)
- Dung lượng: Đối với máy tính để bàn thì thông dụng loại 80-160 GB, tuy nhiên nếu muốn lưu trữ thông tin nhiều thì có thể dùng ổ > 200GB (loại 250 GB hiện nay cũng đang bán rất chạy)
- Tốc độ đọc/ghi: tính bằng MB/s, ngày nay khoảng trên 12MB/s

Ví dụ những thông số chính của 1 đĩa cứng như trong bảng 2.3.

Characteristics	Seagate ST31401N Elite-2 SCSI Drive
Disk diameter (inches)	5.25
Formatted data capacity (GB)	2.8
Cylinders	2627
Tracks per cylinder	21
Sectors per track	≈ 99
Bytes per sector	512
Rotation speed (RPM)	5400
Average seek in ms (random cylinder to cylinder)	11.0
Minimum seek in ms	1.7
Maximum seek in ms	22.5
Data transfer rate in MB/sec	≈ 4.6

Bảng 2.3. Ví dụ các thông số cơ bản của HDD

### Các chuẩn giao tiếp đĩa cứng thông dụng

- **Intergrated Drive Electronics (IDE)**: giao diện bộ điều khiển ổ cứng kết hợp với bộ điều khiển điện tử trên board của ổ cứng. Giao tiếp EIDE là một phát triển gần nhất của IDE.
- **Small Computer System Interface (SCSI)**: Là một loại chuẩn giao tiếp thường được dùng để kết nối PC đến thiết bị khác như là ổ cứng, máy in, scanner và CD-ROM.
- **Serial ATA (SATA)** là một bước phát triển của giao diện lưu trữ vật lý song song ATA, thay thế cáp chuẩn 40 sợi và đầu kết nối IDE thành cáp 7 sợi và đầu kết nối SATA. HDD SATA có tốc độ truyền dữ liệu rất cao (hiện nay là 150 Mbyte/s và còn sẽ được nâng lên cao hơn nữa) và có giá cũng tương đương với HDD IDE.

### 2.5. Ổ CD và DVD

Tương tự như đĩa từ, đĩa quang là môi trường lưu trữ dữ liệu ngay cả khi mất nguồn điện. Điểm khác nhau giữa đĩa quang và đĩa từ nằm ở phương pháp lưu trữ vật lý. Thông tin được lưu trữ trên đĩa quang dưới dạng thay đổi tính chất quang trên bề mặt đĩa. Tính chất này được phát hiện qua chất lượng phản xạ một tia sáng của bề mặt đĩa. Tia sáng này thường là một tia **LASER** với bước sóng cố định (790nm đến 850nm). Bề mặt đĩa được thay đổi khi ghi để có thể phản xạ tia laser tốt hoặc kém. Tia laser được hội tụ vào một điểm rất nhỏ trên mặt đĩa, vì thế đĩa quang có dung tích lưu trữ lớn hơn nhiều lần so với đĩa từ. Hai nhược điểm chính của đĩa quang là:

- Chỉ ghi được một lần (nay đã được khắc phục với đĩa CD-WR),
- Tốc độ đọc chậm hơn đĩa từ.

Đĩa quang được chia ra thành bốn loại chính:

- **CD-ROM** (*compact disk read only memory*): thông tin được lưu trữ ngay khi sản xuất đĩa. Dữ liệu tồn tại dưới dạng mặt phẳng (land) và lỗ (pit). Người sản xuất dùng khuôn để đúc ra nhiều phiên bản CD-ROM.
- **CD-R** (*RECORDABLE COMPACT DISK*) được đọc từ ổ đĩa CD-ROM bình thường. Đĩa này có đặc điểm là ghi được. Đĩa trống được phủ một lớp chất nhạy màu. Dưới tác dụng của tia laser, lớp này đổi màu và dùng đặc điểm đó để lưu trữ dữ liệu. Loại đĩa này còn có tên là WORM (write once read many).
- **CD-WR** (*writeable/readable compact disk*) cũng dùng laser để đọc và ghi dữ liệu. Điểm khác nhau cơ bản là bề mặt đĩa được phủ một lớp kim loại mỏng. Trạng thái lớp kim loại được thay đổi dưới tác dụng tia laser.
- **DVD** (*Digital Versatile Disc hay Digital Video Disc*) cũng giống như CD nhưng có mật độ ghi cao hơn rất nhiều do đó lưu trữ được nhiều thông tin hơn. Đặc biệt là ở một số định dạng có khả năng ghi được nhiều lớp và dùng được cả hai mặt. DVD cũng có nhiều loại như DVD-ROM, DVD-R (Digital Versatile Disc – Recordable), DVD-RAM (Digital Versatile Disc – Random Access Memory), DVD-RW,...

Laser dùng để đọc và ghi đĩa quang là laser bán dẫn. Năng lượng của tia laser rất thấp, khoảng 5 mw. Với năng lượng này, tia laser không nguy hiểm đến mắt. Mặc dù vậy cần tránh nhìn trực tiếp **vào tia laser khi sửa chữa và bảo trì ổ đĩa CD-ROM**. Nguồn laser luôn được tắt khi đưa đĩa vào ổ, vì thế ổ đĩa laser rất an toàn cho người sử dụng. Để đọc được thông tin phản xạ từ tia laser, Ổ đĩa quang còn được trang bị diốt cảm quang:

1. Diốt kiểm tra cường độ tia laser. Diốt này đo cường độ laser để hiệu chỉnh nếu công suất phát sáng giảm theo thời gian.

2. Diốt đọc dùng để hiện tín hiệu quang thành tín hiệu điện để xử lý tiếp. Đĩa quang áp dụng nguyên tắc mã hoá tương tự như đĩa từ. Mã hay dùng nhất là mã RLL vì nó tiết kiệm điện tích và tự định thời. Điểm khác nhau duy nhất giữa đĩa quang và đĩa từ là đĩa quang cần kiểm tra và sửa lỗi nhiều hơn. Thông tin rất dễ bị nhiễu chẳng hạn khi một hạt bụi nằm giữa nguồn laser và nơi cần đọc trên đĩa. Đĩa quang vì thế cần nhiều thông tin CRC hơn đĩa từ. Lỗi đọc phải được phát hiện và sửa lại dùng mã CRC đi kèm theo dữ liệu.

✚ *Một đặc tính quan trọng của các ổ đĩa quang mà khi mua đĩa cần biết là tốc độ đọc/ghi. Các tốc độ đọc ghi dữ liệu thông dụng ngày nay là 24X, 32X, 48X, 52X.*

## 2.6. Bộ nhớ RAM và ROM

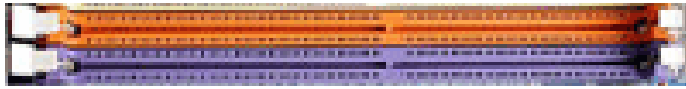
### a) Các khái niệm về bộ nhớ

#### ➤ **Các tế bào nhớ (storage cell):**

Bộ nhớ lưu giữ thông tin dưới dạng một dãy các con số nhị phân 1 và 0, trong đó 1 là đại diện cho sự có mặt của điện áp tín hiệu, và 0 đại diện cho sự vắng mặt. Vì mỗi bit được đại diện bởi một mức điện áp, nên điện áp đó phải được duy trì trong mạch điện tử nhớ, gọi là tế bào nhớ. Nội dung lưu giữ trong tế bào nhớ có thể được sao chép ra bus hoặc các linh kiện chờ khác, gọi là đọc ra (reading). Một số tế bào nhớ cũng cho phép sao chép vào bản thân mình những mức tín hiệu mới lấy từ bus ngoài, gọi là ghi vào (writing). Bằng cách sắp xếp liên kết tế bào nhớ thành các hàng và cột (ma trận), người ta có thể xây dựng nên các mạch nhớ nhiều triệu bit. Các ma trận tế bào nhớ được chế tạo trên một chip silic nhỏ giống như các mạch tích hợp.

➤ **RAM slot** (hình 2.14) Dùng để cắm RAM vào main mà ta có thể nhận dạng ở đầu khe cắm RAM luôn có cần gạt ở 2 đầu. Tùy loại RAM (SDRAM, DDRAM, RDRAM) mà giao diện khe cắm

khác nhau => **Mua RAM cho máy thì phải biết máy có slot cho loại nào.**



Hình 2.14. Slot để cắm RAM

➤ **Interface:** là cấu trúc bên ngoài của memory. Khi mua RAM chúng ta cần phải xem nó có phù hợp với (ăn khớp) RAM slot của máy mình không. Hình 2.15 là hình dạng của một vài loại RAM



Hình 2.15. Hình dáng bên ngoài một số loại RAM

#### ➤ **RAM và ROM:**

Có hai dòng bộ nhớ phổ biến có tên gọi tắt là RAM và ROM. Mạch nhớ truy cập ngẫu nhiên (**random - access memory - RAM**) là bộ nhớ chính (main memory) bên trong máy tính, nơi lưu trữ tạm thời các dữ liệu và lệnh chương trình để Bộ xử lý (BXL) có thể truy cập nhanh chóng. Thuật ngữ "truy cập ngẫu nhiên" có ý nhấn mạnh một tính chất kỹ thuật quan trọng: mỗi vị trí lưu trữ trong RAM đều có thể truy cập trực tiếp. Nhờ đó các thao tác truy tìm và cất trữ có thể thực hiện nhanh hơn nhiều so với các thiết bị lưu trữ tuần tự như ổ đĩa hay ổ băng từ. Nội dung lưu giữ trong RAM là không cố định - có nghĩa phải luôn có nguồn nuôi để duy trì nội dung nhớ đó, mất điện là mất thông tin.

Kích thước của RAM thường đo bằng đơn vị megabyte (MB). Bao nhiêu RAM thì đủ? Đây là câu hỏi chắc chắn ta sẽ đặt ra khi mua sắm hay nâng cấp máy tính. Windows XP SP2 chỉ chạy

với 128MB RAM, nhưng đạt được hiệu năng tốt nhất với 256MB RAM trở đi.

Dòng thứ hai là bộ nhớ chỉ đọc ra (**read-only memory - ROM**). Nội dung trong ROM chỉ có thể được đọc ra trong quá trình hoạt động bình thường của máy tính. Bộ nhớ ROM là loại cố định (nonvolatile), nên nó vẫn duy trì nội dung nhớ khi không có điện. Nhờ tính năng này, người ta dùng ROM để lưu giữ các chương trình BIOS không thay đổi.

#### **b) Các loại bộ nhớ :**

- **RAM tĩnh (static RAM - SRAM)** lưu giữ các bit trong những tế bào của mình dưới dạng chuyển mạch điện tử. Tế bào SRAM mở mạch điện (logic 1) hoặc tắt mạch (logic 0) để phản ánh trạng thái của tế bào. Thực tế đó là các mạch flip-flop trong tình trạng set hoặc reset. Mạch flip-flop sẽ giữ nguyên mẫu trạng thái cho đến khi được thay đổi bởi thao tác ghi tiếp theo hoặc ngắt điện. Tuy nhiên SRAM có kích thước lớn và tốn điện, hiện nay thường được chế tạo sẵn trong giới hạn 512K. Mặc dù có tốc độ nhanh, nhưng phức tạp và đắt tiền, SRAM chỉ được sử dụng trong các bộ phận cần tốc độ như bộ nhớ cache chẳng hạn.
- **RAM động (dynamic RAM - DRAM)** lưu giữ các bit dưới dạng điện tích chứa trong các tụ điện cực nhỏ, đó là các điện dung của bản thân transistor MOS đóng vai trò chuyển mạch hoặc phần tử điều khiển. Có hoặc không có điện tích trong tụ điện này tương ứng với logic 1 hoặc logic 0. Do tụ điện nhỏ nên điện tích được nạp và phóng rất nhanh, cỡ chục nanô giây. Bởi kích thước nhỏ và hầu như không tiêu thụ điện nên DRAM có mật độ lưu trữ khá cao và giá rẻ. Nhược điểm duy nhất của DRAM là không giữ được thông tin lâu quá vài miligiây, nên phải thường xuyên nạp lại năng lượng cho nó gọi là làm tươi hay hồi phục (refresh), thực chất là làm đầy lại điện tích cho các tụ điện nhớ tí hon.



- **DDR SDRAM** (Double Data Rate SDRAM)

SDRAM là tên gọi chung của một dòng bộ nhớ máy tính, nó được phân ra SDR (Single Data Rate) và DDR (Double Data Rate). Do đó nếu gọi một cách chính xác, chúng ta sẽ có hai loại RAM chính là SDR SDRAM và DDR SDRAM. Cấu trúc của hai loại RAM này tương đối giống nhau, nhưng DDR có khả năng truyền dữ liệu ở cả hai điểm lên và xuống của tín hiệu nên tốc độ nhanh gấp đôi. Trong thời gian gần đây xuất hiện chuẩn RAM mới dựa trên nền tảng DDR là DDR-II, DDR-III có tốc độ cao hơn nhờ cải tiến thiết kế.

- **Bộ nhớ ROM** thực chất là một tổ chức ghép nối sẵn các mạch điện để thể hiện các trạng thái có nối (logic 0) hoặc không nối (logic 1). Cách bố trí các trạng thái 1 và 0 như thế nào là tùy yêu cầu, và được chế tạo sẵn trong ROM khi sản xuất. Khi vi mạch ROM được chế tạo xong thì nội dung của nó không thể thay đổi nữa. ROM dùng trong hệ BIOS cũ thuộc loại này cho nên khi bật máy tính là các chương trình chứa sẵn trong đó được lấy ra để chạy khởi động máy (bao gồm các bước kiểm tra chẩn đoán, hỗ trợ phần mềm cơ sở và hợp nhất các bộ phận trong hệ thống máy). Ta không muốn và cũng không thể thay đổi bất cứ điều gì đối với các chương trình cốt tử này. Tuy nhiên khi phát hiện có một lỗi trong ROM hoặc cần đưa vào một thông số BIOS mới để phù hợp với thiết bị ngoại vi mới thì thật là tai họa. Gần đây có một giải pháp là dùng flash BIOS, nó thay một phần ROM bằng loại EEPROM, đó là vi mạch ROM có thể lập trình và xóa bằng điện (Electrically Erasable Programmable ROM). Phương pháp này cho phép chỉ xóa ở một số địa chỉ, không phải toàn bộ trong khi vi mạch vẫn giữ nguyên trên board.

- **SIMM (single in-line memory module).** Đây là loại mô đun nhớ một hàng chân ra để dễ cắm vào các ổ cắm thích hợp trên board mẹ. SIMM gồm nhiều vi mạch nhỏ DRAM được gắn trên một tấm mạch in nhỏ, để tổ chức thành các loại mô đun từ 1MB

đến 16MB hoặc hơn. SIMM loại cũ có 30 chân, phổ biến hiện nay là 72 chân nên các nhà thiết kế có nhiều phương án cấu hình hơn. Đây là loại thuận lợi nhất cho việc nâng cấp bộ nhớ của ta.

- ✦ *Cần lưu ý là có rất nhiều loại RAM khác nhau, do đó khi mua RAM thì phải biết loại nào có thể dùng được cho máy của mình và tốc độ BUS tối đa cho RAM mà mainboard hỗ trợ là bao nhiêu thì chỉ nên lựa loại RAM có tốc độ đó là đủ.*

**c) Thời gian truy cập:**

Một bộ nhớ lý tưởng phải đưa dữ liệu được chọn ngay tức khắc lên các đường dữ liệu của vi mạch nhớ đó. Tuy nhiên trong thực tế luôn tồn tại một thời gian trễ giữa thời điểm tín hiệu địa chỉ lối vào có hiệu lực và thời điểm dữ liệu có mặt trên các đường dữ liệu, gọi là thời gian truy cập (access time). Mặc dù thời gian này được tính bằng nanô giây nhưng cũng làm chậm tốc độ hoạt động chung của toàn hệ thống, nên bộ xử lý phải đợi, có khi đến 4 hoặc 5 xung nhịp.

Các máy PC loại cũ có thể sử dụng các chip DRAM có thời gian truy cập trong vòng 60-80 nanô giây. Các máy tính hiện nay dùng loại nhanh hơn 60 nanô giây. Thời gian truy cập càng nhanh thì DRAM càng đắt.

## 2.7. Bàn phím (keyboard)

Thành phần cơ bản của bàn phím là phím ấn. Phím ấn có tác dụng như một cảm biến lực và được dùng để chuyển lực ấn thành một đại lượng điện. Đại lượng điện này sẽ được xử lý tiếp thành một tín hiệu số để truyền đến máy vi tính cá nhân. Vì vậy phím ấn được phân loại tùy theo nguyên tắc cảm biến như sau:

- Phím cảm biến điện trở (thay đổi về điện trở),
- Phím cảm biến điện dung (thay đổi về điện dung),

- Phím cảm biến điện từ (thay đổi về dòng điện theo hiệu ứng Hall),

Bàn phím thông dụng nhất cho các loại máy vi tính cá nhân tương thích IBM là loại MF101 hay MF102. Số 101 và 102 chỉ ra số phím trên bàn, số phím này thường giao động trong khoảng 90-104. tuy nhiên cũng có những bàn phím trên 130 nút.

Bàn phím hiện đại ngày nay cho ngôn ngữ tiếng Anh lại theo một loại mới gọi là QWERTY, được lấy từ 6 ký tự đầu tiên trên bàn phím.

## 2.8. Chuột (mouse)

Chuột đóng một vài trò và tầm ảnh hưởng rất lớn trong công việc hằng ngày của những ai sử dụng máy tính. Con chuột đầu tiên được Douglas Engelbart phát minh vào năm 1964. Cùng với sự phát triển của các công nghệ vi mạch, vi xử lý, công nghệ lưu trữ,... công nghệ chế tạo chuột cũng đã trải qua nhiều thời kỳ với rất nhiều cải tiến cả về kiểu dáng lẫn công nghệ cảm ứng. Chuột ngày nay có độ nhạy và nhiều tính năng tốt hơn rất nhiều so với một vài năm trước đây.

Thiết bị nhận dữ liệu vào dưới dạng vị trí điểm tương đối được gọi là con chuột (mouse). Ta gọi cách xác định tọa độ của con chuột là tương đối vì chuột là một thiết bị đo vận tốc di chuyển con trỏ. Từ giá trị vận tốc tương đối này, hàm ngắt của hệ điều hành sẽ tính ra vị trí mới của con trỏ (cursor) trên màn hình. Nguyên tắc này hoàn toàn khác phương pháp xác định vị trí tuyệt đối của bút quang hay một điểm vẽ trong bảng vẽ vector. Mỗi chuột có từ hai đến năm phím nhấn để đưa tín hiệu chọn vị trí hiện hành.

Có hai cách phân loại chuột:

- Theo nguyên tắc đo vận tốc chuyển động hay cơ chế cảm ứng
- Theo giao diện với máy tính

Theo loại giao diện chuột ta có:

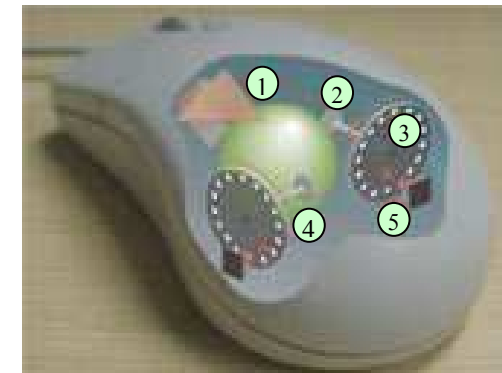
- Chuột song song (nối với máy vi tính qua cổng song song LPT1 hoặc LPT2),

- Chuột nối tiếp (nối hữu tuyến với cổng COM1 hoặc COM2, nối vô tuyến với cổng tia hồng ngoại hay nối qua vi điều khiển 8042 như chuột PS/2)

### ➤ Nguyên tắc đo vận tốc chuyển động:

#### a) Chuột cơ

Chuột cơ dùng viên bi sắt phủ cao su để đo chuyển động. Nguyên lý đầu tiên của chuột chính là loại này và nó được áp dụng kéo dài hàng chục năm sau đó và hiện vẫn có thể tìm thấy các loại chuột bi ở cửa hàng. Chuột máy tính đầu tiên xuất hiện trên thế giới có kích thước khá lớn với hai bánh xe vuông góc với nhau. Để dùng nó phải sử dụng cả hai tay để điều khiển: một tay cầm chuột và tay kia cầm một bàn phím nhỏ có 5 nút bấm. Tới năm 1970, kỹ sư Bill English của Xerox PARC đã thay thế bánh xe cổ điển bằng một viên bi nổi tiếng mà chúng ta đều biết. Viên bi này có thể chuyển động theo mọi hướng (1) (xem hình 2.16), chuyển động này sẽ được hai bánh xe nhỏ bên trong chuột ghi nhận (2), trên bánh xe có các khe hở nhỏ (3) cho phép một tia sáng phát qua tới đầu cảm ứng bên kia, mỗi lần ngắt sẽ báo hiệu chuột di chuyển (4).



Hình 2.16. Cấu tạo chuột cơ

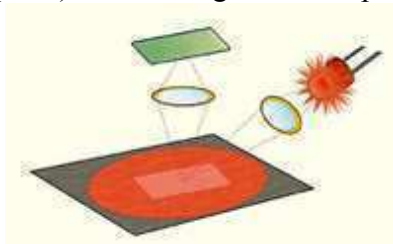
Cuối cùng, một thiết bị cảm ứng (5) sẽ thu thập tín hiệu và tổng kết thành giá trị tọa độ tương ứng của chuột trên màn hình. Kiểu thiết kế này đã đạt được thành công rực rỡ và được sử dụng



rộng rãi trong suốt mấy chục năm cuối của thế kỷ 20. Tuy nhiên nó cũng có một số nhược điểm là hệ thống cơ sẽ bị ăn mòn và các bụi dơ bẩn dễ bám vào làm sai lệch thông tin về tọa độ.

### **b) Chuột quang**

Việc phát minh ra chuột quang nhằm khắc phục những khuyết điểm ở chuột cơ và là một bước tiến đáng kể trong chế tạo chuột. Nó loại bỏ hoàn toàn thành phần cơ học (bi và bánh xe), thay bằng một thiết bị bắt hình siêu nhỏ. Thiết bị này sẽ liên tục "chụp" lại bề mặt mà người dùng di chuột và thông qua phép so sánh giữa những bức hình này, bộ xử lý trong chuột sẽ tính toán được tọa độ. Chuột bị sử dụng đầu cảm ứng quang để bắt chuyển động của viên bi còn chuột quang sử dụng thiết bị ghi hình để bắt chuyển động của bề mặt nhờ sự phản xạ của các tia từ bàn để chuột. Trên thực tế, để tính toán chính xác thì hình ảnh chụp phải tốt. Vì thế, nhiệm vụ quan trọng đầu tiên là soi sáng bề mặt và một đèn LED đỏ được sử dụng cho việc này. Khi chiếu sáng bề mặt, tia sáng sẽ bị phản chiếu và hội tụ thông qua một thấu kính trước khi chạm vào bộ cảm ứng (xem hình 2.17). Nhờ thế, hình ảnh sẽ rất chi tiết. Đôi khi, chuột quang học sử dụng đèn LED bị hiểu nhầm là chuột laser (đề cập sau) do ánh sáng đỏ mà nó phát ra.



Hình 2.17. Nguyên lý cảm ứng trong chuột quang

Ưu điểm của thể hệ chuột quang học là không có các bộ phận cơ nên hoàn toàn không sợ hỏng hóc do ăn mòn hay bụi bẩn. Việc bảo trì cũng rất đơn giản (chỉ cần lau mắt đọc là xong). Thêm vào đó là độ chi tiết và độ nhạy của cơ chế cảm ứng quang cũng tốt hơn rất nhiều. Tuy nhiên, chuột quang không thể làm việc trên các bề mặt bóng hoặc trong suốt, còn các bề mặt sần sùi thì chuột hoạt động không chính xác. Điều này đúng với những loại chuột quang

thuộc thế hệ đầu tiên. Ngoài ra, một số loại chuột rẻ tiền có hệ thống xử lý hình ảnh kém sẽ không đủ khả năng tính toán khi người dùng di chuyển chuột với tốc độ nhanh (chuột cao cấp có thể theo được tốc độ di chuyển lên tới hơn 1m mỗi giây). Điểm yếu cuối của chuột quang là nó "ngốn" điện nhiều hơn chuột cơ: 25mA so với chỉ khoảng 5mA.

### **c) Chuột laser**

Chuột sử dụng cảm ứng laser là công nghệ mới nhất và tiên tiến nhất hiện nay. Không chỉ thừa hưởng đầy đủ ưu điểm quang học mà chuột laser còn có nhiều đặc điểm ưu việt khác. Được giới thiệu lần đầu tiên vào năm 2004 dưới sự hợp tác của Logitech và Agilent Technologies, Logitech MX1000 là đại diện đầu tiên của thế hệ chuột laser xuất hiện trên thị trường. Chuột này sử dụng một tia laser nhỏ thay vì đèn LED đỏ thông thường. Công nghệ laser cho phép tia sáng có độ tập trung cao hơn và đặc biệt ổn định. Nhờ thế chuột có thể tăng độ chi tiết của hình ảnh "chụp" tới 20 lần trên lý thuyết.

## **2.9. Card màn hình (VGA Card)**

Trong máy tính cá nhân thế hệ trước, nội dung màn hình được bộ vi xử lý trực tiếp quản lý. Nội dung màn hình được truy nhập trực tiếp qua địa chỉ bộ nhớ. Tài nguyên xử lý không bị ảnh hưởng nhiều nếu máy làm việc trong chế độ văn bản (ví dụ như trên hệ điều hành MS-DOS). Máy tính hiện đại làm việc trong chế độ đồ họa (ví dụ như hệ điều hành Windows). Số điểm ảnh và số màu trong chế độ này rất lớn và đòi hỏi được truy nhập nhanh. Nếu không có trợ giúp từ bên ngoài, bộ vi xử lý sẽ phải dùng phần lớn tài nguyên của nó để điều hợp hiển thị đồ họa. Bảng 2.4 cho thấy lịch sử phát triển của các chuẩn thẻ điều hợp hiển thị.

Để giải quyết vấn đề này, nhiều nhà sản xuất cho ra thị trường thẻ điều hợp hiển thị có tên là bộ gia tốc (*accelerator*). Những thẻ này có bộ vi điều khiển của nó, các phép tính liên quan đến điều hợp hiển thị được tiến hành trên thẻ, giảm gánh nặng cho bộ vi xử lý. Thay vì phải tính toán bộ các điểm ảnh cần hiển thị, bộ vi xử lý chỉ cần gửi một lệnh ngắn về thẻ điều hợp hiển thị, phần

còn lại được bộ vi xử lý đồ họa **GPU(Graphics Processing Unit)** của thẻ thực hiện. Vi xử lý của thẻ điều hợp hiển thị được thiết kế đặc biệt cho nhiệm vụ này nên làm việc hiệu quả hơn nhiều bộ vi xử lý

Năm	Chuẩn	Ý nghĩa	Kích thước	Số màu
1981	CGA	Colour Graphics Adaptor	640 x 200 160 x 200	Không 16
1984	EGA VGA	Enhanced Graphics Adaptor Video Graphics Array	640 x 350 640 x 480 320 x 200	6 16 256
1987	XGA	Extended Graphics Array	800 x 600 1024x768	16.7 triệu 65536
1990	SXGA UXGA Ultra XGA	Super Extended Graphics Array	1280x 1024 1600 x 1200	65,536 65,536

Bảng 2.4. Quá trình phát triển thẻ điều hợp hiển thị

### Bộ nhớ video

Bộ nhớ video (VRAM) chứa nội dung hình ảnh được hiển thị và các thông tin liên quan đến nó. Chỉ riêng các điểm ảnh một màn hình 1600x1200 màu thực đã cần đến 8MB bộ nhớ (xem bảng 2.5). Nhu cầu về bộ nhớ hiển thị khiến phải cắm thêm bộ nhớ video dành riêng cho mục đích này.

Bộ nhớ video còn được gọi là bộ đệm khung (**frame buffer**). Một số máy vi tính có vi mạch Chipset trên bản mạch chính và dùng một phần bộ nhớ chính làm bộ nhớ video, phương pháp này làm giảm đáng kể khả năng hiển thị nhưng rẻ hơn thẻ cắm đồ họa. Từ thế hệ Pentium, bộ vi xử lý có cổng gia tốc đồ họa **AGP** (accelerated graphics port). Cổng này cho phép bộ vi xử lý đồ họa truy nhập trực tiếp bộ nhớ hệ thống cho các phép tính đồ họa nhưng

vẫn có bộ nhớ video riêng để lưu trữ nội dung các điểm ảnh màn hình. Phương pháp này cho phép sử dụng bộ nhớ hệ thống mềm dẻo hơn mà không làm ảnh hưởng đến tốc độ máy tính. Cổng AGP ngày nay trở thành chuẩn trong các máy vi tính hiện đại.

Dung lượng bộ nhớ	Kích thước màn hình	Chiều sâu màu	số màu
1 Mb	1024x768	8-bit	256
	800 x 600	16-bit	65,536
2Mb	1024 x 768	8-bit	256
	1284 x 1024	16-bit	65,536
	800x600	24-bit	16.7 million
4Mb	1024x768	24-bit	16.7 million
6Mb	1280x1024	24-bit	16.7 million
8Mb	1600x1200	32-bit	16.7 million

Bảng 2.5. Dung lượng bộ nhớ video và khả năng hiển thị màn hình

Ngoài ra công nghệ sản xuất bộ nhớ video khác nhau cũng sẽ cho các đặc tính của bộ nhớ khác nhau. Bảng 2.6 cho ta thấy một số khác biệt giữa các bộ nhớ video.

Loại bộ nhớ	EDO	VRAM	WRAM	SDRAM	SGRAM	RDRAM
Tốc độ truyền cao nhất (MBps)	400	400	960	800	800	600
Cổng kép hay đơn	Sing	dual	dual	single	single	single
Chiều rộng dữ liệu	64	64	64	64	64	8
Thời gian truy cập	50-60ns	50-60ns	50-60ns	10-15ns	8-10ns	3ns

Bảng 2.6. So sánh các loại bộ nhớ dành cho bộ nhớ video

**Bộ chuyển đổi từ tín hiệu số sang tín hiệu tương tự** (DAC - Digital to Analog Converter). Bộ chuyển đổi này còn được gọi là RAMDAC, có nhiệm vụ biến đổi hình ảnh thành tín hiệu analog để màn hình có thể hiển thị. Một vài card đồ họa có nhiều hơn một bộ RAMDAC, do đó tăng tốc độ xử lý và hỗ trợ hiển thị nhiều màn hình.

## 2.10. Màn hình (Monitor)

Cùng với bàn phím và chuột, màn hình là một thiết bị không thể thiếu được trong máy vi tính. Công nghệ chế tạo và ứng dụng của màn hình rất đa dạng. Chương trình này chỉ đề cập kỹ đến các loại màn hình thông dụng:

- Màn hình tia âm cực (CRT- cathode ray tube),
- Màn hình tinh thể lỏng (LCD - liquid crystal display),
- Màn hình plasma (plasma display),

### a) Các thông số cơ bản của các loại màn hình

#### ➤ Vùng hiển thị hình ảnh (Viewable area):

Vùng hiển thị trên màn hình mà người dùng có thể nhìn thấy được.

#### ➤ Độ phân giải của màn hình (Resolution):

Độ phân giải của màn hình, tính bằng số lượng các điểm ảnh trên đường ngang (row) và đường dọc (column). Ví dụ màn hình hỗ trợ các độ phân giải 640x480, 1024x768, 1280x1024,...

- **Điểm ảnh (Pixel):** là điểm ảnh, điểm sáng hiển thị màu trên màn hình.
- **Khoảng cách giữa tâm các điểm ảnh (Dot pitch):** khoảng cách này càng nhỏ màn hình có độ phân giải càng cao, hình ảnh hiển thị càng sắc nét. Ví dụ: 0.31mm, 0.28mm, 0.27mm, 0.26mm, 0.25mm, ...
- **Độ sâu của màu (Colour Depth):** số lượng màu hiển thị trên 1 điểm ảnh. Ví dụ: 16,8 triệu màu, 65.000 màu,...

- **Refresh Rate:** tốc độ làm tươi hình ảnh hay gọi là tần số quét của màn hình, là số lần "vẽ lại" hình ảnh trong 1 giây từ trên xuống dưới cho tất cả các điểm ảnh. Chất phosphor giữ cho độ sáng điểm ảnh vừa đủ để mắt người không cảm nhận được sự thay đổi này. Thông số này rất quan trọng, nó càng cao thì mắt người dùng không bị mỏi. Mỗi loại màn hình có thể hỗ trợ các tần số quét khác nhau (50 Hz, 60 Hz, 72 Hz, 85 Hz, 90 Hz, 100 Hz...).
- **Respect ratio:** tỉ số giữa chiều rộng và chiều cao của màn hình giúp hình ảnh không bị kéo dãn khi được thể hiện ở những khung hình khác nhau, thông thường tỉ số này là 4:3.
- **Power Consumption:** công suất tiêu thụ điện của màn hình

Phần tử nhỏ nhất của một ảnh hay một thiết bị hiển thị ảnh gọi là điểm ảnh pixel (picture element). Khái niệm này xuất hiện trong quá trình nghiên cứu và phát triển màn hình ống tia âm cực. Kích thước một điểm ảnh trên màn hình CRT phụ thuộc vào các tham số

- Kích thước chùm tia điện tử,
- Kích thước hạt photpho,
- Chiều dày lớp photpho.

Kích thước ngang và dọc với đơn vị là một điểm ảnh được gọi là kích thước màn hình. Màn hình VGA cơ bản có kích thước 640x480 điểm ảnh.

Độ phân giải được định nghĩa là kích thước chi tiết nhỏ nhất và đo được của một thiết bị hiển thị. Một tham số để đo độ phân giải là số điểm ảnh trên một đơn vị chiều dài (inch hay centimet), được gọi là mật độ điểm ảnh. Mật độ điểm ảnh thường gặp được tính theo số điểm ảnh trên một inch, viết tắt là dpi (dot per inch). Ta cần tránh nhầm lẫn giữa kích thước màn hình và độ phân giải. Độ phân giải được phân loại như sau:

- Phân giải thấp (<50 dpi).
- Phân giải trung bình (51dpi - 70dpi).

- Phân giải cao (71dpi - 120dpi ).
- Phân giải siêu cao (>120 dpi)

Kích thước điểm ảnh không còn là tham số đối với loại màn hình ma trận điểm (dot matrix display) như màn hình LCD ngày nay. Điểm ảnh của các màn hình này luôn là hình vuông và kích thước màn hình thường là 640x480, 800x600, 1024x768, 1280x1024,... Kích thước điểm ảnh cần được thiết kế để tỷ lệ chiều ngang và chiều dọc của màn hình là 4:3.

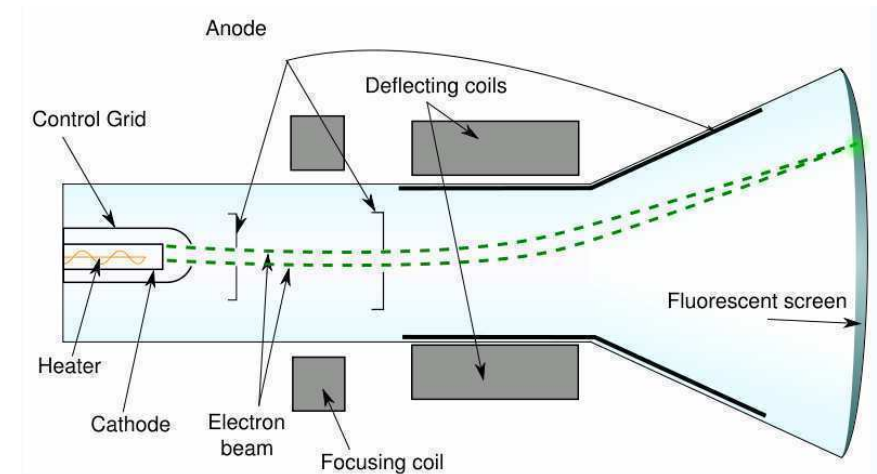
Một màu bất kỳ có thể biểu diễn qua ba màu cơ bản: đỏ, xanh lục, xanh nước biển tùy theo độ đậm nhạt (gray scale). Độ sâu màu (color depth) là số màu có thể hiển thị được cho một điểm ảnh. Tùy theo số bit được dùng để hiển thị màu ta phân loại màn hình theo màu như sau:

- Đen trắng 1 bit (2 màu),
- Màu CGA 4 bit (16 màu),
- Màu giả (pseudo color) 8 bit (256 màu),
- Màu (high color) 16 bit,
- Màu thật (true color) 24 bit
- Màu siêu thật (highest color) 32 bit

Tốc độ quét màn hình còn gọi là tần số làm tươi (refresh rate) là một tham số quan trọng và đòi hỏi nhiều vấn đề khó giải quyết từ công nghệ màn hình cũng như công nghệ bộ điều khiển màn hình. Để mắt thường phân biệt được thay đổi tự nhiên trên màn hình, toàn bộ màn hình ít nhất phải được thể hiện lại ít nhất 30 lần một giây. Điều này có nghĩa là màn hình cần có tần số làm tươi tối thiểu là 30Hz. Tần số làm tươi của màn hình VGA nằm trong khoảng 30 đến 60Hz, thời gian tồn tại một ảnh nhỏ hơn 33 ms. Tần số này không cao lắm nhưng đã là thách thức lớn cho màn hình, nhất là các loại chậm như LCD. Một điểm ảnh LCD cần từ 50 đến 250 ms để thay đổi trạng thái

## b) Màn hình tia âm cực CRT

Được phát minh bởi nhà vật lý người Đức *Karl Ferdinand Braun* vào năm 1879. Cấu tạo cơ bản của màn hình CRT như trong hình 2.18, bao gồm một súng phóng điện tử, một hệ thống tạo từ trường để biến đổi quỹ đạo electron, và một màn huỳnh quang. Ống phóng điện tử dựa theo hiệu ứng phát xạ nhiệt electron. Khi cung cấp năng lượng cho mẫu kim loại dưới dạng nhiệt, các electron sẽ được truyền năng lượng để bật ra khỏi liên kết mạng tinh thể kim loại. Các electron này sau khi bật ra được tăng tốc bởi một điện trường. Sau khi được tăng tốc bởi điện trường, electron có quỹ đạo thẳng hướng về phía màn huỳnh quang. Trước khi đập vào màn huỳnh quang, electron sẽ phải bay qua một vùng từ trường được tạo bởi hai cuộn dây, một cuộn tạo từ trường ngang và một cuộn tạo từ trường dọc. Tùy theo cường độ của hai từ trường này, quỹ đạo của electron trong từ trường sẽ bị lệch đi và đập vào màn huỳnh quang tại một điểm được định trước. Tọa độ của điểm này trên màn hình có thể được điều khiển bởi việc điều chỉnh cường độ dòng điện trong hai ống dây, qua đó điều chỉnh cường độ từ trường tác dụng lên electron. Electron đập vào màn huỳnh quang (thường là ZnS) sẽ khiến điểm đó phát sáng. Để tạo ra ba màu cơ bản trong hệ màu RGB, người ta sử dụng ba súng phóng điện tử riêng, mỗi súng tương ứng với một màu



Hình 2.18. Cấu tạo màn hình CRT

### MÀN HÌNH TINH THỂ LỎNG

Tinh thể lỏng được một nhà thực vật học người Áo, Friedrich Reinitzer, phát hiện vào cuối thế kỷ 19. Một thời gian ngắn sau, khái niệm tinh thể lỏng được nhà vật lý học người Đức Otto Lehmann nhắc đến lần đầu tiên.

Từ năm 1971, màn hình tinh thể lỏng được ứng dụng trong nhiều lĩnh vực: TV, máy ảnh số, màn hình máy tính .v.v. Ngày nay, màn hình tinh thể lỏng để bàn hay màn hình máy tính xách tay được chế tạo từ hai nguyên tắc chính:

- DSTN (dual-scan twisted nematic)
- TFT (thin film transistor)

Tinh thể lỏng LCD (liquid crystal display) là chất lỏng hữu cơ mà phân tử của nó có khả năng phân cực ánh sáng dẫn đến thay đổi cường độ sáng. Trường tĩnh điện được dùng để điều khiển hướng phân tử tinh thể lỏng.

Do hình ảnh được mã hoá và hiển thị dưới dạng bản đồ ma trận điểm ảnh, nên màn hình LCD cũng phải được cấu tạo từ các điểm ảnh. Mỗi điểm ảnh trên màn hình LCD sẽ hiển thị một điểm ảnh của khung hình. Trong mỗi điểm ảnh trên màn hình LCD, có ba điểm ảnh con (subpixel), mỗi điểm ảnh hiển thị một trong ba màu: đỏ, xanh lá, xanh lam. Để nắm được nguyên lý hoạt động của màn hình LCD, ta xét một số khái niệm sau:

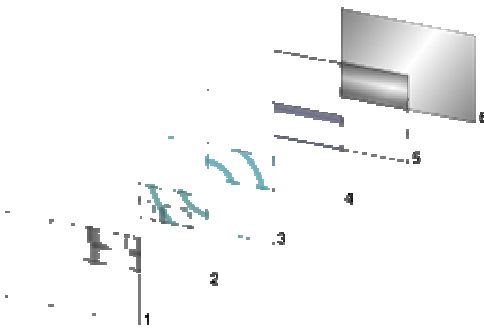
• **Ánh sáng phân cực:** theo lý thuyết sóng ánh sáng của Huyghen, Fresnel và Maxwell, ánh sáng là một loại sóng điện từ truyền trong không gian theo thời gian. Phương dao động của sóng ánh sáng là phương dao động của từ trường và điện trường (vuông góc với nhau). Dọc theo phương truyền sóng, phương dao động của ánh sáng có thể lệch nhau một góc tùy ý. Xét tổng quát, ánh sáng bình thường có vô số phương dao động khác nhau. Ánh sáng phân cực là ánh sáng chỉ có một phương dao động duy nhất, gọi là phương phân cực.

• **Kính lọc phân cực:** là loại vật liệu chỉ cho ánh sáng phân cực đi qua. Lớp vật liệu phân cực có một phương đặc biệt gọi là quang trục phân cực. Ánh sáng có phương dao động trùng với quang trục phân cực sẽ truyền toàn bộ qua kính lọc phân cực. Ánh sáng có phương dao động vuông góc với quang trục phân cực sẽ bị chặn lại. Ánh sáng có phương dao động hợp với quang trục phân cực một góc  $0 < \varphi < 90$  sẽ truyền một phần qua kính lọc phân cực. Cường độ ánh sáng truyền qua kính lọc phân cực phụ thuộc vào góc hợp bởi phương phân cực của ánh sáng và quang trục phân cực của kính lọc phân cực.

• **Tinh thể lỏng:** tinh thể lỏng không có cấu trúc mạng tinh thể cố định như các vật rắn, mà các phân tử có thể chuyển động tự do trong một phạm vi hẹp như một chất lỏng. Các phân tử trong tinh thể lỏng liên kết với nhau theo từng nhóm và giữa các nhóm có sự liên kết và định hướng nhất định, làm cho cấu trúc của chúng có phần giống cấu trúc tinh thể. Vật liệu tinh thể lỏng có một tính chất đặc biệt là có thể làm thay đổi phương phân cực của ánh sáng truyền qua nó, tùy thuộc vào độ xoắn của các chùm phân tử. Độ xoắn này có thể điều chỉnh bằng cách thay đổi điện áp đặt vào hai đầu tinh thể lỏng

Màn hình tinh thể lỏng được cấu tạo bởi các lớp xếp chồng lên nhau như trong hình 2.19. Lớp dưới cùng (lớp 6) là đèn nền, có tác dụng cung cấp ánh sáng nền (ánh sáng trắng). Đèn nền dùng trong các màn hình thông thường, có độ sáng dưới  $1000\text{cd/m}^2$  thường là đèn huỳnh quang. Đối với các màn hình công cộng, đặt ngoài trời, cần độ sáng cao thì có thể sử dụng đèn nền xenon. Đèn nền xenon về mặt cấu tạo khá giống với đèn pha bi-xenon sử dụng trên các xe hơi cao cấp. Đèn xenon không sử dụng dây tóc nóng sáng như đèn Vonfram hay đèn halogen, mà sử dụng sự phát sáng bởi nguyên tử bị kích thích, theo định luật quang điện và mẫu nguyên tử Bo. Bên trong đèn xenon là hai bản điện cực, đặt trong khí trơ xenon trong một bình thủy tinh thạch anh. Khi đóng nguồn, cấp cho hai điện cực một điện áp rất lớn, cỡ  $25\,000\text{V}$ . Điện áp này

vượt ngưỡng điện áp đánh thủng của xenon và gây ra hiện tượng phóng điện giữa hai điện cực. Tia lửa điện sẽ kích thích các nguyên tử xenon lên mức năng lượng cao, sau đó chúng sẽ tự động nhảy xuống mức năng lượng thấp và phát ra ánh sáng theo định luật bức xạ điện từ. Điện áp cung cấp cho đèn xenon phải rất lớn, thứ nhất để vượt qua ngưỡng điện áp đánh thủng để sinh ra tia lửa điện, thứ hai để kích thích các nguyên tử khí trơ lên mức năng lượng đủ cao để ánh sáng do chúng phát ra khi quay trở lại mức năng lượng thấp có bước sóng ngắn.



Hình 2.19. Các lớp cấu tạo màn hình LCD

Lớp thứ hai (lớp 5) là lớp kính lọc phân cực có quang trục phân cực ngang, kế đến là một lớp tinh thể lỏng (lớp 3) được kẹp chặt giữa hai tấm thủy tinh mỏng (lớp 4 và 2), tiếp theo là lớp kính lọc phân cực có quang trục phân cực dọc (lớp 1). Mặt trong của hai tấm thủy tinh kẹp tinh thể lỏng có phủ một lớp các điện cực trong suốt. Ta xét nguyên lý hoạt động của màn hình LCD với một điểm ảnh con: ánh sáng đi ra từ đèn nền là ánh sáng trắng, có vô số phương phân cực. Sau khi truyền qua kính lọc phân cực thứ nhất, chỉ còn lại ánh sáng có phương phân cực ngang. Ánh sáng phân cực này tiếp tục truyền qua lớp tinh thể lỏng. Nếu giữa hai đầu lớp tinh thể lỏng không được đặt một điện áp, các phân tử tinh thể lỏng

sẽ ở trạng thái tự do, ánh sáng truyền qua sẽ không bị thay đổi phương phân cực. Ánh sáng có phương phân cực ngang truyền tới lớp kính lọc thứ hai có quang trục phân cực dọc sẽ bị chặn lại hoàn toàn. Lúc này, điểm ảnh ở trạng thái tắt.

Nếu đặt một điện áp giữa hai đầu lớp tinh thể lỏng, các phân tử sẽ liên kết và xoắn lại với nhau. Ánh sáng truyền qua lớp tinh thể lỏng được đặt điện áp sẽ bị thay đổi phương phân cực. Ánh sáng sau khi bị thay đổi phương phân cực bởi lớp tinh thể lỏng truyền đến kính lọc phân cực thứ hai và truyền qua được một phần. Lúc này, điểm ảnh được bật sáng. Cường độ sáng của điểm ảnh phụ thuộc vào lượng ánh sáng truyền qua kính lọc phân cực thứ hai. Lượng ánh sáng này lại phụ thuộc vào góc giữa phương phân cực và quang trục phân cực. Góc này lại phụ thuộc vào độ xoắn của các phân tử tinh thể lỏng. Độ xoắn của các phân tử tinh thể lỏng phụ thuộc vào điện áp đặt vào hai đầu tinh thể lỏng.

Như vậy, có thể điều chỉnh cường độ sáng tại một điểm ảnh bằng cách điều chỉnh điện áp đặt vào hai đầu lớp tinh thể lỏng. Trước mỗi điểm ảnh con có một kính lọc màu, cho ánh sáng ra màu đỏ, xanh lá và xanh lam. Với một điểm ảnh, tùy thuộc vào cường độ ánh sáng tương đối của ba điểm ảnh con, dựa vào nguyên tắc phối màu phát xạ, điểm ảnh sẽ có một màu nhất định. Khi muốn thay đổi màu sắc của một điểm ảnh, ta thay đổi cường độ sáng tỉ đối của ba điểm ảnh con so với nhau. Muốn thay đổi độ sáng tỉ đối này, phải thay đổi độ sáng của từng điểm ảnh con, bằng cách thay đổi điện áp đặt lên hai đầu lớp tinh thể lỏng.

Một nhược điểm của màn hình tinh thể lỏng, đó chính là tồn tại một khoảng thời gian để một điểm ảnh chuyển từ màu này sang màu khác (thời gian đáp ứng – response time). Nếu thời gian đáp ứng quá cao có thể gây nên hiện tượng bóng ma với một số cảnh có tốc độ thay đổi khung hình lớn. Khoảng thời gian này sinh ra do sau khi điện áp đặt lên hai đầu lớp tinh thể lỏng được thay đổi, tinh thể lỏng phải mất một khoảng thời gian mới có thể chuyển từ trạng thái xoắn ứng với điện áp cũ sang trạng thái xoắn ứng với điện áp mới. Thông qua việc tái tạo lại màu sắc của từng điểm ảnh, chúng ta có thể tái tạo lại toàn bộ hình ảnh.



### **MÀN HÌNH TFT**

Màn hình LCD màu hay còn gọi là màn hình ma trận chấm (dot matrix display) có điện cực và bộ lọc màu riêng cho từng điểm tinh thể lỏng. Mỗi điểm ảnh sẽ bao gồm ba điểm màu riêng biệt. Màn hình ma trận chủ động (active matrix display) tối ưu hoá quá trình định địa chỉ và nạp từng điểm ảnh. Màn hình ma trận chủ động dùng một transistor màng mỏng TFT (thin-film transistor) làm công tắc chuyển mạch cho từng điểm màu. Transistor đóng mạch rất nhanh (trong vài micro giây), tụ điện mắc song song với nó sẽ giữ trạng thái dòng mạch lâu hơn trong khi transistor của các dòng khác tiếp tục đóng mạch. Màn hình TFT được sản xuất theo công nghệ vi điện tử và chứa vi mạch điều khiển ngay trên màn hình.

### **MÀN HÌNH PLASMA**

Nguyên tắc màn hình plasma giống nguyên tắc đèn Neon. Màn hình plasma thường có màu đặc trưng là xanh hay vàng đỏ. Màn hình plasma gồm nhiều ô khí trơ được hàn kín tương ứng với các điểm ảnh. Mỗi ô khí trơ có hai điện cực. Khi hiệu điện thế vượt quá một giới hạn nhất định, khí trơ sẽ ion hóa và phát sáng. Nguyên tắc điều khiển màn hình loại này đơn giản hơn màn hình LCD. Nhược điểm của màn hình loại này là thời gian làm việc ngắn, tiêu thụ nhiều năng lượng. Độ tương phản vào khoảng 10:1. Màn hình plasma từng được dùng cho máy tính xách tay của Toshiba và Compaq. Ngày nay chúng hầu như không thể cạnh tranh được với màn hình tinh thể lỏng tiên tiến. Màn hình LCD có thể dùng trong máy chiếu ảnh (có vai trò như phim trong máy chiếu bóng) để có được hình trên màn ảnh rộng.

## **2.11. Card mạng (Network adapter) và Modem**

Thiết bị đầu tiên cần để xây dựng mạng là card mạng (Network Interface – Card NIC). Mỗi máy tính trong mạng cần một NIC. NIC cùng với driver thực hiện 2 chức năng chính: truyền và nhận thông tin ở dạng data frame. Trên NIC phải có một đầu nối với

cáp mạng, thông dụng nhất là loại BNC (viết tắt của bayonet connector) dùng cho cáp đồng trục và RJ45, dùng cho cáp dạng dây xoắn đôi. NIC còn phụ thuộc vào loại mạng dùng mà thông dụng nhất ngày nay là loại Ethernet (10Mbit/s), Fast Ethernet (100Mbit/s) và Gigabit Ethernet (1000Mbit/s).

### **Modem**

Cùng với đà sử dụng máy tính trong những năm qua, việc máy tính này cần giao tiếp với máy tính khác là chuyện hiển nhiên xảy ra. Modem ra đời với việc ứng dụng mạng lưới điện thoại có sẵn để kết nối các máy tính đặt xa nhau. Ví dụ, kết nối một máy tính cá nhân tại nhà với máy tính đặt tại nơi làm việc, với hệ thống ngân hàng trong nước, hoặc với bảng thông báo điện tử, mà phổ biến nhất ngày nay là truy cập Internet.

Những modem hiện đại truyền dữ liệu với tốc độ từ 28800 bit/s đến 57600 bit/s, tốc độ này hiện nay được coi là hơi chậm, do vậy mới ra đời modem với công nghệ ADSL .



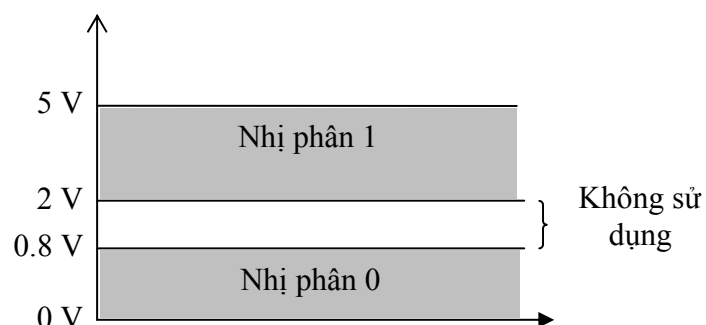
## **CÂU HỎI VÀ BÀI TẬP CHƯƠNG II**

1. CPU có các thành phần chủ yếu nào? hãy mô tả hoạt động của các thành phần đó. Một máy tính đơn giản được tổ chức như thế nào?
2. Hãy nêu ra sự khác biệt cơ bản giữa các bộ vi xử lý của hãng Intel và của hãng AMD.
3. Tần số của CPU có phải là tốc độ xử lý của nó không? giải thích rõ sự khác nhau về tần số của CPU hãng Intel và hãng AMD.
4. Cho biết các loại bản mạch chính đang thông dụng ngày nay ngoài thị trường. Cho biết trên mainboard được cắm những bộ phận gì?
5. Mô tả vận hành của ổ đĩa cứng. Cách lưu trữ thông tin trong ổ đĩa cứng.
6. Nguyên tắc vận hành của đĩa quang. Ưu khuyết điểm của các loại đĩa quang.
7. Cho biết sự khác biệt giữa bộ nhớ RAM và ROM. Liệt kê một số loại RAM thông dụng và các đặc tính kỹ thuật của nó
8. Bộ xử lý đồ họa GPU khác CPU ở điểm gì? điểm đặc biệt của bộ nhớ video so với các loại bộ nhớ khác.
9. Hãy cho biết các thể loại màn hình và các đặc tính cơ bản của từng loại

## Chương III: Biểu diễn dữ liệu

### 3.1. Khái niệm thông tin

Để mã hóa thông tin trong máy tính, người ta dùng các tín hiệu điện thế. Thường tín hiệu trong khoảng  $0 \rightarrow 0.8V$  đại diện cho một giá trị (nhị phân 0) và tín hiệu có mức điện thế bất kỳ trong khoảng  $2 \rightarrow 5V$  đại diện cho giá trị kia (nhị phân 1). (xem hình 3.1.)



Hình 3.1. Biểu diễn trị nhị phân qua điện thế

Trong hình này, chúng ta quy ước có hai trạng thái có ý nghĩa: trạng thái thấp khi hiệu điện thế thấp hơn  $0.8V$  và trạng thái cao khi hiệu điện thế lớn hơn  $2V$ . Để có thông tin, ta phải xác định thời điểm ta quan sát trạng thái của tín hiệu. Thí dụ, tại thời điểm  $t_1$  thì tín hiệu ở trạng thái thấp và tại thời điểm  $t_2$  thì tín hiệu ở trạng thái cao.

### 3.2. Lượng thông tin và sự mã hoá thông tin

Thông tin được đo lường bằng đơn vị thông tin mà ta gọi là bit. Lượng thông tin được định nghĩa bởi công thức:

$$I = \log_2(N)$$

Trong đó:

I: là lượng thông tin tính bằng bit

N: là số trạng thái có thể có

Vậy một bit ứng với một trạng thái trong hai trạng thái có thể có. Hay nói cách khác, một bit có thể biểu diễn hai trạng thái 0 hoặc 1. Ví dụ, để biểu diễn một trạng thái trong 8 trạng thái có thể có, ta cần một số bit ứng với một lượng thông tin là:

$$I = \log_2(8) = 3 \text{ bit}$$

Tám trạng thái được ghi nhận nhờ 3 số nhị phân (mỗi số nhị phân có thể có giá trị 0 hoặc 1).

Như vậy lượng thông tin là số con số nhị phân cần thiết để biểu diễn số trạng thái có thể có. Do vậy, một con số nhị phân được gọi là một bit. Một từ n bit có thể tượng trưng một trạng thái trong tổng số  $2^n$  trạng thái mà từ đó có thể tượng trưng.

Ví dụ: Nếu dùng 3 bit ( $A_2, A_1, A_0$ ) để biểu diễn thông tin, ta sẽ có được 8 trạng thái khác nhau như trong bảng 3.1.

Trạng thái	$A_2$	$A_1$	$A_0$
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

Bảng 3.1. Ba bit biểu diễn được 8 trạng thái

Như vậy trong máy tính thì mọi thứ đều được biểu diễn dưới dạng hai con số là 0 và 1. Nhưng ở thế giới thực của chúng ta thì thông tin lại là các khái niệm như con số, chữ cái, hình ảnh, âm thanh,... Cho nên để đưa các thông tin vào máy tính thì ta cần chuyển đổi thông tin thực thành những con số 0 và 1. Công việc này ta gọi là **sự mã hóa thông tin**

Để biểu diễn dữ liệu trong máy tính chúng ta cần có các quy tắc “gắn kết” các khái niệm trong thế giới thật với một dãy gồm các con số 0 và 1.

### 3.3. Hệ Thống Số

#### Khái niệm hệ thống số:

Cơ sở của một hệ thống số định nghĩa phạm vi các giá trị có thể có của một chữ số. Ví dụ: trong hệ thập phân, một chữ số có giá trị từ 0-9, trong hệ nhị phân, một chữ số (một bit) chỉ có hai giá trị là 0 hoặc 1.

Dạng tổng quát để biểu diễn giá trị của một số:

$$V_k = \sum_{i=-m}^{n-1} b_i \cdot k^i$$

Trong đó:

$V_k$ : Số cần biểu diễn giá trị

$m$ : số thứ tự của chữ số phần lẻ (phần lẻ của số có  $m$  chữ số được đánh số thứ tự từ -1 đến - $m$ )

$n-1$ : số thứ tự của chữ số phần nguyên (phần nguyên của số có  $n$  chữ số được đánh số thứ tự từ 0 đến  $n-1$ )

$b_i$ : giá trị của chữ số thứ  $i$

$k$ : hệ số đếm ( $k=10$ : hệ thập phân;  $k=2$ : hệ nhị phân;  $k=8$ : hệ bát phân; ...).

**Ví dụ:** biểu diễn số  $541.25_{10}$

$$541.25_{10} = 5 \cdot 10^2 + 4 \cdot 10^1 + 1 \cdot 10^0 + 2 \cdot 10^{-1} + 5 \cdot 10^{-2}$$

$$= (500)_{10} + (40)_{10} + (1)_{10} + (2/10)_{10} + (5/100)_{10}$$

**Các hệ thống số cơ bản gồm:**

- **Thập phân (Decimal)**

Dùng 10 chữ số 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 để biểu diễn số. Ví dụ số 235.3 trong hệ thập phân biểu diễn một đại lượng:

$$\begin{array}{ccccccc} 2 & 1 & 0 & & -1 & & \leftarrow \text{trọng số} \\ 2 & 3 & 5 & . & 3 & & \\ & & & & & & = 2 \cdot 10^2 + 3 \cdot 10^1 + 5 \cdot 10^0 + 3 \cdot 10^{-1} \end{array}$$

- **Nhị phân (Binary)**

Dùng hai chữ số 0 và 1 để biểu diễn số. Ví dụ số  $m = 1101,011$  ở hệ nhị phân biểu diễn một đại lượng:

$$m_2 = 1.2^3 + 1.2^2 + 0.2^1 + 1.2^0 + 0.2^{-1} + 1.2^{-2} + 1.2^{-3}$$

Ở đây để tránh nhầm lẫn chúng ta dùng ký hiệu số nhỏ phía bên dưới để biểu diễn con số đó ở hệ nào, như  $m_2$  – số  $m$  ở hệ nhị phân,  $530_{10}$  – số 530 ở hệ thập phân.

- **Bát phân (Octal)**

Để biểu diễn số dùng 8 chữ số 0, 1, 2, 3, 4, 5, 6, 7. Ví dụ:

$$M = (6327,4051)_8 = 6.8^3 + 3.8^2 + 2.8^1 + 7.8^0 + 4.8^{-1} + 0.8^{-2} + 5.8^{-3} + 1.8^{-4}$$

▪ **Chú ý:** Mỗi con số ở hệ bát phân được biểu diễn bằng tập hợp 3 số ở hệ nhị phân (ví dụ  $5_8 = 101_2$ )

- **Thập lục phân (Hexadecimal)**

Để biểu diễn số dùng 16 chữ số: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E. Trong đó tương đương với hệ 10 thì A=10, B=11, C=12, D=13, E=14, F=15.

▪ **Chú ý:** Mỗi con số ở hệ thập lục phân được biểu diễn bằng tập hợp 4 số ở hệ nhị phân (ví dụ  $5_{16} = 0101_2$ )

Bảng 3.2 cho ta các đặc tính chính của các hệ đếm cơ bản.

Như vậy có nhiều hệ đếm khác nhau được dùng để biểu diễn dữ liệu và chúng ta sẽ xem xét cách chuyển đổi giữa các hệ này với nhau như thế nào sau đây.

HỆ ĐẾM	CƠ SỐ	KÍ HIỆU CHỮ SỐ	TRỌNG SỐ	VÍ DỤ
Nhị phân	2	0, 1	$2^i$	1001,1101
Bát phân	8	0,1,2,3,4,5,6,7	$8^i$	3567,24
Thập phân	10	0,1,2,3,4,5,6,7,8,9	$10^i$	1369,354
Thập lục phân	16	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F	$16^i$	3FA9,6B

Bảng 3.2. Các hệ đếm cơ bản

#### a) Chuyển đổi từ hệ cơ số 10 sang $b$

**Quy tắc:** Chia số cần đổi cho  $b$ , lấy kết quả chia tiếp cho  $b$  cho đến khi **kết quả** bằng 0. Số ở cơ số  $b$  chính là các số dư (của phép chia) viết ngược.

Ví dụ 1: Chuyển số 41 ở hệ 10 sang hệ 16

$$41 \div 16 = 2 \text{ dư } 9$$

$$2 \div 16 = 0 \text{ dư } 2$$

$$\Rightarrow 41_{10} = 29_{16}$$

Ví dụ 2: Chuyển số 41 ở hệ 10 sang hệ 2

$$41 \div 2 = 20 \text{ dư } 1$$

$$20 \div 2 = 10 \text{ dư } 0$$

$$10 \div 2 = 5 \text{ dư } 0$$

$$5 \div 2 = 2 \text{ dư } 1$$

$$2 \div 2 = 1 \text{ dư } 0$$

$$1 \div 2 = 0 \text{ dư } 1$$

$$\Rightarrow 41_{10} = 101001_2 \text{ (chú ý!!! viết ngược từ dưới lên)}$$

Vì chúng ta cần biểu diễn dữ liệu ở hệ nhị phân, nên việc **Chuyển đổi hệ 10 sang Nhị phân** cần được đặc biệt lưu ý riêng như sau:

**Quy tắc:** Người ta chuyển đổi từng phần nguyên và lẻ theo quy tắc sau:

**Phần nguyên:** Chia liên tiếp phần nguyên cho 2 giữ lại các số dư, số nhị phân được chuyển đổi sẽ là dãy số dư liên tiếp tính từ lần chia cuối về lần chia đầu tiên.

**Phần lẻ:** Nhân liên tiếp phần lẻ cho 2, giữ lại các phần nguyên được tạo thành. Phần lẻ của số Nhị phân sẽ là dãy liên tiếp phần nguyên sinh ra sau mỗi phép nhân **tính từ lần nhân đầu đến lần nhân cuối**.

Ví dụ 3: Chuyển sang hệ Nhị phân số: 13,625

Thực hiện:

Phần nguyên:

$$13:2 = 6 \text{ dư } 1$$

$$6:2 = 3 \text{ dư } 0$$

$$3:2 = 1 \text{ dư } 1$$

$$1:2 = 0 \text{ dư } 1$$

Phần nguyên của số Nhị phân là 1101

Phần lẻ:

$$0,6875 \times 2 = 1,375 \text{ Phần nguyên là } 1$$

$$0,375 \times 2 = 0,750 \text{ Phần nguyên là } 0$$

$$0,750 \times 2 = 1,500 \text{ Phần nguyên là } 1$$

$$0,5 \times 2 = 1,00 \text{ Phần nguyên là } 1$$

Phần lẻ của số Nhị phân là: 0,1011

Ta viết kết quả là:  $(13,625)_{10} = (1101,1011)_2$

**Chú ý:** việc chuyển đổi từ hệ thập phân sang hệ Nhị phân không phải luôn được gọn gàng chính xác, trong trường hợp phép tính chuyển đổi kéo dài, thì tùy theo yêu cầu về độ chính xác mà ta có thể dùng phép tính ở mức độ cần thiết thích hợp.

**Ví dụ 4:** Chuyển số  $(3287,5100098)_{10}$  sang Cơ số 8.

Phần nguyên:

$$\begin{aligned} 3287:8 &= 410 \text{ dư } 7 \\ 410:8 &= 51 \text{ dư } 2 \\ 51:8 &= 6 \text{ dư } 3 \\ 6:8 &= 0 \text{ dư } 6 \\ \text{Vậy } (3287)_{10} &= (6327)_8 \end{aligned}$$

Phần lẻ:

$$\begin{aligned} 0,5100098 \times 8 &= 4,0800784 \quad \text{phần nguyên là 4} \\ 0,0800784 \times 8 &= 0,6406272 \quad \text{phần nguyên là 0} \\ 0,6406270 \times 8 &= 5,1250176 \quad \text{phần nguyên là 5} \\ 0,1250176 \times 8 &= 1,0001408 \quad \text{phần nguyên là 1} \\ \text{Vậy } (0,5100098)_{10} &= (0,4051)_8 \end{aligned}$$

Kết quả chung là:  $(3287,5100098)_{10} = (6327,4051)_8$

**b) Chuyển đổi từ hệ cơ số  $b$  sang 10**

Việc chuyển đổi từ một hệ cơ số bất kỳ sang hệ 10 thì đơn giản hơn và cách làm như trong trường hợp định nghĩa đại lượng của số đó.

**Ví dụ 1:** số 235.3 trong hệ 8 chuyển sang hệ thập phân có giá trị như sau:

$$\begin{array}{ccccccc} 2 & 1 & 0 & & -1 & & \leftarrow \text{trọng số} \\ 2 & 3 & 5 & . & 3 & & \\ & & & & = 2 \cdot 8^2 + 3 \cdot 8^1 + 5 \cdot 8^0 + 3 \cdot 8^{-1} = 157.375_{10} \end{array}$$

➤ **Chuyển đổi Hệ 2 sang hệ 10**

**Quy tắc:** Muốn chuyển đổi một số biểu diễn trong hệ Nhị phân sang hệ thập phân ta lập Tổng theo trọng số của từng bit Nhị phân, Kết quả của tổng sẽ là biểu diễn Thập phân của số đó.

**Ví dụ 2:** Chuyển đổi sang hệ Thập phân số:  $m = 1101,011$

Thực hiện: Ta lập tổng theo trọng số của từng Bit nhị phân:  
 $m = 1.2^3 + 1.2^2 + 0.2^1 + 1.2^0 + 0.2^{-1} + 1.2^{-2} + 1.2^{-3}$

$$\begin{aligned} m &= 8 + 4 + 0 + 1 + 0 + 1/4 + 1/8 \\ m &= 13,375 \end{aligned}$$

**c) Chuyển đổi cơ số 2-8-16**

**Quy tắc:** Từ phải sang trái, gom 3 chữ số nhị phân thành một chữ số **bát phân** hoặc gom 4 chữ số nhị phân thành một chữ số **thập lục phân**.

$$\begin{array}{ccccccccc} 1 & 5 & 7 & 1 & 4 & 3 & & & \text{Hệ 8} \\ \{001\} & \{101\} & \{111\} & \{001\} & \{100\} & \{11 & 00\} & \{11\} \\ & D & E & 6 & 3 & & & \text{Hệ 16} \end{array}$$

Bảng 3.3 cho ta các chuyển đổi tương ứng từ các hệ số với nhau. Để làm bài tốt và học tốt các môn học liên quan đến kỹ thuật số, hệ thống số, vi xử lý,... sinh viên cần thuộc lòng bảng này.

Hệ 2 (Base 2)	Hệ bát phân (Base 8)	Hệ thập phân (Base 10)	Hệ thập lục phân (Base 16)
0000	0	0	0
0001	1	1	1
0010	2	2	2
0011	3	3	3
0100	4	4	4
0101	5	5	5
0110	6	6	6
0111	7	7	7
1000	10	8	8
1001	11	9	9
1010	12	10	A
1011	13	11	B
1100	14	12	C

1101	15	13	D
1110	16	14	E
1111	17	15	F

Bảng 3.3. Tương quan giữa các hệ thống số

**Ví dụ 1:** Chuyển số  $M = (574,321)_8$  sang biểu diễn nhị phân.

Thực hiện: Thay mỗi chữ số bằng nhóm nhị phân 3 bit tương ứng:

$$M = \begin{array}{ccc} 101 & 111 & 100 \\ 5 & 7 & 4 \end{array}, \begin{array}{ccc} 011 & 010 & 001 \\ 3 & 2 & 1 \end{array}$$

$$\Rightarrow M_2 = 101111100,011010001$$

**Ví dụ 2:** Chuyển số  $M = (1001110,101001)_2$  sang cơ số 8.

$$\begin{array}{l} \text{Thực hiện: } M = \begin{array}{ccc} 1 & 001 & 110 \\ & 6 & \end{array}, \begin{array}{cc} 101 & 001 \\ & 5 & 1 \end{array} \\ \Rightarrow M = (116,51)_8 \end{array}$$

### 3.4. Các phép tính số học cho hệ nhị phân

Các phép tính Cộng, Trừ, Nhân, Chia cũng được sử dụng trong số học Nhị phân, việc tính toán cụ thể được thực hiện theo quy tắc sau:

#### 3.4.1. Phép cộng hai số nhị phân không dấu:

Cộng nhị phân được thực hiện theo quy tắc ở bảng 3.4

Chú ý:

- Khi cộng, thực hiện từ bit có trọng số thấp đến bit có trọng số cao.
- Nếu có số nhớ thì số nhớ sinh ra được cộng vào bit có trọng số cao hơn liền kề

SỐ HẠNG 1	SỐ HẠNG 2	TỔNG	SỐ NHỚ	KẾT QUẢ
-----------	-----------	------	--------	---------

0	0	0	0	0
0	1	1	0	1
1	0	1	0	1
1	1	0	1	10

Bảng 3.4. Quy tắc Cộng Nhị phân cho 2 số 1 bit.

**Ví dụ:** Thực hiện các phép Cộng Nhị phân:

$$\begin{array}{r} 1011 \\ +1100 \\ \hline 10111 \end{array}$$

#### 3.4.2. Phép trừ hai số nhị phân không dấu:

Phép trừ nhị phân được thực hiện theo quy tắc trình bày ở Bảng 3.5

SỐ BỊ TRỪ	SỐ TRỪ	HIỆU SỐ	SỐ VAY
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Bảng 3.5. Quy tắc trừ Nhị phân cho 2 số 1 bit.

Chú ý:

- Phép tính được thực hiện từ Bit có trọng số thấp đến Bit có trọng số cao.
- Số vay sẽ được trừ vào Bit có trọng số cao hơn ở liền kề.

**Ví dụ:** Thực hiện các tính Trừ Nhị phân sau:

$$1011$$



$$\frac{-0110}{0101}$$

Tuy nhiên trong thực tế, máy tính không tính toán kiểu đó mà chuyển đổi phép trừ thành phép cộng với **số bù 2** của nó. Phương pháp này trong máy tính được cho là hiệu quả hơn và dễ dàng thiết kế phần cứng cho nó hơn. Số bù có hai loại thường dùng là số bù 1 và số bù 2.

### 3.4.3. Phép nhân và chia hai số nhị phân không dấu:

- **Phép nhân** nhị phân được thực hiện như nhân thập phân.

Ví dụ: Có phép tính: 1001 nhân với 1101

$$\begin{array}{r} \text{Ta thực hiện:} \quad 1001(\text{Số bị nhân-Multiplicant}) \\ \times \quad 1101(\text{Số nhân-Multiplier}) \\ \hline 1001 \\ 0000 \\ + 1001 \\ \hline 1001 \\ \hline \text{Kết quả là:} \quad 1110101 \end{array}$$

- **Phép chia** nhị phân được thực hiện như chia thập phân.

Ví dụ: Có phép tính: 1110101 chia cho 1001

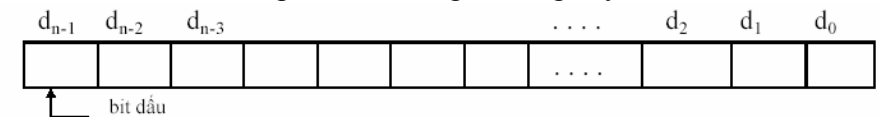
$$\begin{array}{r} \text{Ta thực hiện:} \quad 1110101 : 1001 \\ - \quad 1001 \quad 1101 \\ \hline 01011 \\ - \quad 1001 \\ \hline 001001 \\ \quad 1001 \\ \hline \quad 0000 \\ \hline \text{Kết quả:} \quad 1111010 : 1001 = 1101 \end{array}$$

### 3.4.4. Biểu diễn số nguyên có dấu

Có ba cách để biểu diễn một số nguyên  $n$  bit có dấu đó là biểu diễn bằng trị tuyệt đối và dấu, biểu diễn bằng số bù 1, biểu diễn bằng số bù 2.

Cách thông thường nhất là biểu diễn bằng trị tuyệt đối và dấu, trong trường hợp này thì bit cao nhất luôn tượng trưng cho dấu.

Khi đó, bit dấu có giá trị là 0 thì số đó là nguyên dương, bit dấu có giá trị là 1 thì số đó là nguyên âm. Tuy nhiên, cách biểu diễn dấu này không đúng trong trường hợp số được biểu diễn bằng số thừa  $K$  mà ta sẽ xét ở phần sau trong chương này.



Số nguyên có bit  $d_{n-1}$  là bit dấu và có trị tuyệt đối biểu diễn bởi các bit từ  $d_0$  tới  $d_{n-2}$ .

Ví dụ: dùng 8 bit biểu diễn số +25 và -25 dưới dạng dấu và trị tuyệt đối.

Ta biết  $25_{10} = 11001_2$ . Dùng 8 bit để biểu diễn số, như vậy 1 bit biểu diễn dấu, còn 7 bits biểu diễn giá trị của số đó.

$$\text{Vậy } +25 = \quad 0 \ 0011001$$

Bit dấu

$$\text{Vậy } -25 = \quad 1 \ 0011001$$

Bit dấu

Như vậy, theo cách này thì:

$$+25_{10} = 00011001_2$$

$$-25_{10} = 10011001_2$$

- Một Byte (8 bit) có thể biểu diễn các số có dấu từ -127 tới +127.

- Có hai cách biểu diễn số không là 0000 0000 (+0) và 1000 0000 (-0).

Hai cách biểu diễn còn lại ta sẽ xem xét trong phần tiếp dưới đây.

### 3.4.5. Số bù của một số

Số bù được dùng trong máy tính số giúp đơn giản phép toán trừ và các thao tác logic. Trong hệ cơ số (hệ đếm)  $r$  có hai dạng số bù: Số bù  $r$  và số bù  $(r-1)$ . Như vậy trong hệ 10 sẽ có bù 10 và bù 9, trong hệ 8 có bù 8 và bù 7, trong hệ 16 có bù 16 và bù 15, trong hệ 2 có bù 2 và bù 1.

#### Số bù $r-1$ của một số:

Giả sử  $N$  là một số có  $n$  ký số trong hệ thống số  $r$  thì bù  $r-1$  của  $N = (r^n - 1) - N$ .

Ví dụ đối với hệ thập phân ta có bù  $r-1$  = bù 9 của số thập phân  $N$  là  $(10^n - 1) - N$ . Trong đó  $10^n$  là một số gồm số 1 và theo sau là  $n$  chữ số 0 (ví dụ  $10^4 = 10000$ ). Vậy  $(10^n - 1)$  là gồm  $n$  số 9 (ví dụ  $10^4 - 1 = 9999$ ). Vậy bù 9 của  $N$  là một con số nhận được bằng cách lấy trừ 9 cho từng ký số của  $N$ .

Ví dụ:

Bù 9 của 43520 là  $9999 - 43520 = 56479$

#### Số bù $r$ của một số:

Số bù  $r$  của một số được tính bằng bù  $r-1$  cộng với 1. Như vậy bù 10 của 43520 là  $56478 + 1 = 56480$ . Từ đây ta có thể tính nhanh bù 10 theo qui tắc:

- Giữ nguyên các ký số 0 bên phải cho đến khi gặp ký số khác 0
- Lấy 10 trừ đi ký số đầu tiên khác 0
- Lấy 9 trừ đi các số còn lại

Ví dụ: Bù 10 của 347200 là 652800

Chúng ta sẽ chủ yếu làm việc với hệ nhị phân, do máy tính làm việc với hệ này. Trong hệ nhị phân sẽ có bù 1 và bù 2 mà ta sẽ xem xét sau đây.

### Số bù 1 của một số:

Số bù 1 của một số nhị phân (hay còn gọi là số invert) là một số nhị phân có được bằng cách đổi các bit 1 thành 0 và bit 0 thành 1.

Ví dụ:

Số cần đổi	10110101	11001110
Số bù 1 của nó	10001010	0011001

**Số bù 2 của một số:** Số bù hai của một số là số bù 1 của số đó cộng thêm 1.

Ví dụ:

Số:	01001110	00110101
Số bù một của nó là:	10110001	11001010
Cộng thêm 1	+1	+1
Bù hai của nó là:	10110010	11001011

### Quy tắc chung tìm bù hai của một số:

- Muốn tìm bù 2 của một số ta đi từ bit có trọng số nhỏ nhất ngược lên.
- Khi nào gặp được bit 1 đầu tiên thì giữ nguyên các số 0 bên phải số 1 đó và cả số 1 đó nữa, còn tất cả các bit bên trái số 1 đó thì đảo lại.

Ví dụ:

Số:	01100100	10010010	1101000	01100111
Số bù 2 là:	10011100	01101110	0011000	10011001

Sau khi ta đã biết về số bù 1 và bù 2 của một số nhị phân, ta xem cách biểu diễn một số nguyên có dấu theo hai cách này.

- Số nguyên có dấu được biểu diễn ở dạng bù 1 là:

- Đối với số dương thì biểu diễn giống dấu và trị tuyệt đối
- Đối với số âm thì được biểu diễn dưới dạng bit dấu và giá trị của số đó ở dạng bù 1.

Ví dụ: Dùng 8 bit biểu diễn số +25 và -25 dưới dạng bù 1.

Ta biết  $25_{10} = 11001_2$ . Dùng 8 bit để biểu diễn số, như vậy 1 bit biểu diễn dấu, còn 7 bits biểu diễn giá trị của số đó, nếu là số âm thì lấy bù 1 các bit này.

Vậy +25 = **0** 0011001

Bit dấu

Vậy -25 = **1** 1100110

Bit dấu

Ta cũng có thể hiểu là số âm được biểu diễn bằng cách lấy bù 1 của số dương kể cả bit dấu.

▪ **Số nguyên có dấu được biểu diễn ở dạng bù 2 là:**

- Đối với số dương thì biểu diễn giống dấu và trị tuyệt đối
- Đối với số âm thì được biểu diễn dưới dạng bit dấu và giá trị của số đó ở dạng bù 2.

Ví dụ: Dùng 8 bit biểu diễn số +25 và -25 dưới dạng bù 2.

Ta biết  $25_{10} = 11001_2$ . Dùng 8 bit để biểu diễn số, như vậy 1 bit biểu diễn dấu, còn 7 bits biểu diễn giá trị của số đó, nếu là số âm thì lấy bù 2 các bit này.

Vậy +25 = **0** 0011001

Bit dấu

Vậy -25 = **1** 1100111

Bit dấu

▪ **Chú ý: Số dương biểu diễn ở cả 3 cách là như nhau, chỉ khác nhau khi đó là số âm**

**3.4.6. Phép cộng trừ nhị phân dùng bù 1**

Số có dấu được biểu diễn bằng bù 1 theo qui tắc sau:

- Bit lớn nhất (MSB) là bit dấu, trong đó 0 là số dương và 1 là số âm
- Các bit còn lại biểu diễn trị thực của số dương hoặc trị bù 1 của số âm.

Ví dụ: Dùng 6 bit gồm cả bit dấu để biểu diễn số

17 = 010001    26 = 011010

-17 = 101110    -26 = 100101

Thực hiện phép cộng giống như cộng các số nhị phân không dấu, cộng cả bit dấu. Cần lưu ý một điểm nhỏ là cộng số nhớ của bit lớn nhất vào bit cuối cùng (LSB).

Ví dụ:

13	001101	-13	110010
+	+	+	+
11	001011	-11	110100
<hr/>			
24	011000	-24	100110
			+
			1
			<hr/>
			100111

Trong kết quả nếu bit dấu là 0 thì dãy bit sau bit dấu là giá trị kết quả, còn nếu bit dấu là âm thì dãy bit sau bit dấu mới là kết quả ở dạng bù một. Muốn biết giá trị thực của kết quả ta phải lấy bù 1 của kết quả một lần nữa. Như trong ví dụ trên kết quả của phép cộng thứ nhất là 011000 có bit dấu là 0, vậy giá trị thực của kết quả là +11000 = +24. Còn phép cộng thứ hai có kết quả là 100111 có bit dấu là 1, vậy giá trị thực của kết quả là -(bù 1 của 00111) = -11000 = -24.

### 3.4.7. Phép cộng trừ nhị phân dùng bù 2

Số có dấu được biểu diễn bằng bù 2 theo qui tắc sau:

- Bit lớn nhất (MSB) là bit dấu, trong đó 0 là số dương và 1 là số âm
- Các bit còn lại biểu diễn trị thực của số dương hoặc trị bù 2 của số âm.

Thực hiện phép cộng giống như cộng các số nhị phân không dấu, cộng cả bit dấu. Cần lưu ý loại bỏ bit nhớ cuối cùng trong kết quả.

Ví dụ:

$$\begin{array}{r} -12 \quad 110100 \\ + -9 \quad 110111 \\ \hline \end{array}$$

$$\begin{array}{r} -21 \quad 1101011 \end{array}$$

Kết quả có bit dấu bằng 1, vậy kết quả là số âm và dãy bit mới chỉ là bù 2 của kết quả. Muốn có kết quả thật ta lấy Bù 2 một lần nữa. Trong ví dụ trên kết quả không tính đến bit nhớ (bỏ bit nhớ) là 101011 có bit dấu bằng 1 là một số âm, tức là kết quả mới biểu diễn ở dạng bù 2. Muốn biết giá trị thật của kết quả ta phải lấy bù 2 một lần nữa. Tức là  $101011 = -(\text{bù 2 của } 01011) = -(10101) = -21$ .

Đối với phép trừ ta thực hiện thông qua phép cộng.

$$\begin{array}{l} -A = \text{bù 2 của } A \\ A - B = A + (-B) = A + (\text{bù 2 của } B) \end{array}$$

Ví dụ 1:  $13 - 6 = 13 + (-6)$

$$\begin{array}{rcl} 6 & = & 00000110 \\ -6 & = & 11111010 \\ 13 & = & 00001101 \\ & = & 1\ 00000111 \text{ (7)} \end{array}$$

Ví dụ 2: Thực hiện phép tính:  $0111 - 0101$

Ta thực  $0111$  chuyển  $0111$

$$\begin{array}{rcl} \text{hiện:} & -0101 & \text{thành} \\ & & +1011 \text{ (Số bù 2 của 0101)} \\ & & 10010 \text{ Suy ra kết quả là 0010} \end{array}$$

Ví dụ 3: Thực hiện phép tính:  $0101 - 0111$

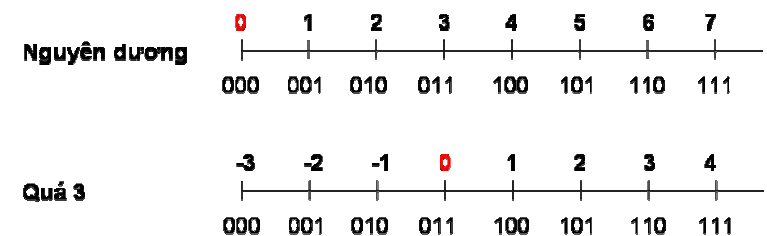
$$\begin{array}{rcl} \text{Ta thực hiện:} & 0101(5) & \text{Chuyển thành} \\ & -0111(-7) & +1001 \text{ (Số bù 2 của 0111)} \\ & & 1110 \end{array}$$

### 3.5. Số quá n (excess-n)

Số quá n hay còn gọi là số thừa n của một số N có được bằng cách “cộng thêm” số N với số quá n, số n được chọn sao cho tổng của n và một số âm bất kỳ luôn luôn dương.

**Quy tắc chung:**

Biểu diễn quá n của N = biểu diễn nguyên dương của  $(N + n)$



**Ví dụ:**

Biểu diễn (quá 127) của 7 là:

$$127 + 7 = 134 = 10000110_2$$

Cách biểu diễn số nguyên có dấu bằng số bù 2 được dùng rộng rãi cho các phép tính số nguyên. Nó có lợi là không cần thuật toán đặc biệt nào cho các phép tính cộng và tính trừ, và giúp phát hiện dễ dàng các trường hợp bị tràn.

Các cách biểu diễn bằng "dấu, trị tuyệt đối" hoặc bằng "số bù 1" dẫn đến việc dùng các thuật toán phức tạp và bất lợi vì luôn có hai cách biểu diễn của số không.

Cách biểu diễn bằng "dấu, trị tuyệt đối" được dùng cho phép nhân của số có dấu chấm động.

Cách biểu diễn bằng số quá n được dùng cho số mũ của các số có dấu chấm động. Cách này làm cho việc so sánh các số mũ có dấu khác nhau trở thành việc so sánh các số nguyên dương.

### 3.6. Cách biểu diễn số với dấu chấm động

Để biểu diễn các con số rất lớn hoặc rất bé, người ta người ta dùng một cách biểu diễn số gọi là số chấm động (*floating point number*). Trước khi đi vào cách biểu diễn số với dấu chấm động, chúng ta xét đến cách biểu diễn một số dưới dạng dấu chấm xác định.

Ví dụ:

- Trong hệ thập phân, số  $254_{10}$  có thể biểu diễn dưới các dạng sau:

$$254 * 10^0; 25.4 * 10^1; 2.54 * 10^2; 0.254 * 10^3; 0.0254 * 10^4; \dots$$

- Trong hệ nhị phân, số  $(0.00011)_2$  (tương đương với số  $0.09375_{10}$ ) có thể biểu diễn dưới các dạng:

$$0.00011 * 2^0; 0.0011 * 2^{-1}; 0.011 * 2^{-2}; 0.11 * 2^{-3}; 1.1 * 2^{-4}; \dots$$

Các cách biểu diễn này gây khó khăn trong một số phép so sánh các số. Để dễ dàng trong các phép tính, các số được chuẩn hoá về một dạng biểu diễn:

$$\pm 1. \text{fff} \dots \text{f} \times 2^{\pm E}$$

đối với hệ nhị phân, trong đó: f là phần lẻ; E là phần mũ.

➤ Đối với các hệ khác thì biểu diễn chấm động được gọi là **chuẩn hóa** khi phần định trị chỉ có duy nhất **một** chữ số bên trái dấu chấm thập phân và chữ số đó khác không  $\rightarrow$  một số chỉ có duy nhất một biểu diễn chấm động được chuẩn hóa.

Ví dụ:

$$2.006 * 10^3 \text{ (chuẩn)}$$

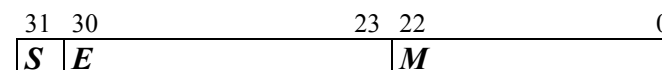
$$20.06 * 10^2 \text{ (không)}$$

$$0.2006 * 10^4 \text{ (không)}$$

Các thành phần của số chấm động bao gồm: phần dấu, phần mũ và phần định trị. Như vậy, cách này cho phép biểu diễn gần đúng các số thực, tất cả các số đều có cùng cách biểu diễn.

Có nhiều cách biểu diễn dấu chấm động, trong đó cách biểu diễn theo chuẩn IEEE 754 được dùng rộng rãi trong khoa học máy tính hiện nay. Trong cách này một số được biểu diễn dưới dạng:

$$F = (-1)^S * M * R^E$$



Hình 3.2. Biểu diễn số có dấu chấm động chính xác đơn với 32 bit

Trong đó: **S**: dấu (Sign bit), **M**: định trị, **R**: cơ số, **E**: mũ (Exponent)

- Dấu: 1 bit (0 – dương, 1 – âm)
- Mũ: 8 bit (từ bit 23 đến bit 30) là một số quá 127 (sẽ có trị từ -127 đến 128, tức là từ 00000000 đến 11111111)
- Không biểu diễn cơ số (R) vì luôn bằng 2
- Phần định trị M 23 bit (từ bit 0 đến bit 22) **chỉ biểu diễn phần lẻ** (bên phải dấu chấm số nhị phân) vì chữ số bên trái dấu chấm luôn là 1.

Ví dụ:

$$\text{a) } 2006_{10} = (-1)^0 * 2.006 * 10^3$$

$$\text{b) } 209.8125_{10} = 11010001.1101_2$$

$$= 1.10100011101 * 2^7$$

Biểu diễn (quá 127) của 7 là:

$$127+7 = 134 = 10000110_2$$

Kết quả: 0 10000110 1010001110100000000000

0	10000110	1010001110100000000000
---	----------	------------------------

Các phép tính với số chấm động phức tạp hơn nhiều là với số chấm tĩnh, thực hiện lâu hơn và phần cứng cho nó cũng phức tạp hơn. Máy tính không có phần cứng tính toán số chấm động, nhưng có các tập trình con giúp giải các bài toán với số chấm động.

### 3.7. Biểu diễn số BCD

Con người thường quen với hệ thập phân, trong khi máy tính lại chỉ thích hợp với hệ nhị phân. Do đó khi nhập xuất dữ liệu thường là nhập xuất theo dạng thập phân. Nếu việc nhập xuất số thập phân không nhiều thì có thể chuyển số hệ 10 khi nhập sang hệ 2, tính toán xong theo hệ 2 rồi lại chuyển ngược lại sang hệ 10 trước khi xuất ra ngoài. Nếu nhập xuất nhiều thì việc chuyển đổi sẽ làm mất nhiều thời gian xử lý. Mặt khác một vài ứng dụng, đặc biệt ứng dụng quản lý, bắt buộc các phép tính thập phân phải chính xác, không làm tròn số. Với một số bit cố định, ta không thể đổi một cách chính xác số nhị phân thành số thập phân và ngược lại.

Vì vậy, khi cần phải dùng số thập phân, ta có thể dùng một cách khác, đó là cách biểu diễn số thập phân mã bằng nhị phân (**BCD: Binary Coded Decimal**). Theo đó mỗi số thập phân nhập vào máy sẽ được mã hóa theo dạng **BCD** bằng cách chuyển mỗi ký số hệ 10 thành 4 bit số nhị phân như trong bảng 3.6. Sau đó việc tính toán sẽ thực hiện trực tiếp trên mã **BCD**. Tính toán xong thì lại chuyển ra ngoài theo dạng thập phân. Khi đó, nên việc tính toán là không nhiều, hoặc việc tính toán là đơn giản thì số **BCD** sẽ giúp cải thiện đáng kể tốc độ xử lý.

Số hệ 10	Số BCD
0	0000

1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Bảng 3.6. Số thập phân mã BCD

Biểu diễn số dạng BCD sẽ tốn kém hơn nhiều biểu diễn dạng nhị phân vì mỗi số BCD cần tới 4 bit. Ví dụ 3257 có dạng BCD là 0011 0010 0101 0111, tức là phải dùng 16 bit, trong khi ở hệ nhị phân chỉ cần 12 bit (110010111001). Con số càng lớn thì sự chênh lệch của nó càng nhiều, trong khi bộ nhớ thì có hạn, cho nên đây là một nhược điểm rất lớn của dạng số BCD.

Để thiết kế mạch tính toán thập phân cũng đòi hỏi độ phức tạp nhiều hơn, tuy nhiên nó có thuận lợi là việc tính toán đều bằng thập phân và cho kết quả chính xác hơn.

Một số ứng dụng như xử lý dữ liệu thương mại - kinh tế thường tính toán ít hơn so với khối dữ liệu nhập xuất. Vì vậy mà một số máy và các máy tính tay đều tính toán trực tiếp trên số thập phân. Một số máy khác lại có khả năng tính toán trên cả thập phân và nhị phân.

Điểm khác biệt rõ nhất với các hệ khác khi tính toán là khi kết quả cộng nếu các ký số vượt quá kết quả cho phép trong khoảng từ **0000** đến **1001** hoặc có nhớ khi cộng thì phải sửa sai bằng cách cộng thêm **0110** vào ký số bị sai.

Hai ví dụ sau đây sẽ cho thấy điều này.

Ví dụ 1:



$$\begin{array}{r}
 27 \quad 0010 \ 0111 \\
 + 36 \quad 0011 \ 0110 \\
 \hline
 63 \quad 0101 \ \boxed{1101} \longrightarrow \text{Ký số vượt quá} \Rightarrow \text{kết quả sai} \\
 \\
 \quad 0000 \ 0110 \longrightarrow \text{Sửa sai kết quả} \\
 \quad 0110 \ 0011 \quad \text{Kết quả} = 63
 \end{array}$$

Trong ví dụ này ta thấy khi cộng hai số 6 với 7 đã cho ta kết quả là 13 (1101). Kết quả này đã vượt qua con số lớn nhất trang hệ BCD là 1001 (9), do đó để sửa lỗi ta phải cộng thêm một giá trị 0110 vào đúng vị trí số cộng sai đó và nếu có số nhớ thì số nhớ đó sẽ được cộng sang số bên cạnh trái.

Ví dụ 2:

$$\begin{array}{r}
 28 \quad 0010 \ 1000 \\
 + 59 \quad 0101 \ 1001 \\
 \hline
 87 \quad 1000 \ 0001 \longrightarrow \text{Có nhớ 1} \Rightarrow \text{kết quả sai} \\
 \\
 \quad 0000 \ 0110 \longrightarrow \text{Sửa sai kết quả} \\
 \quad 1000 \ 0111 \quad \text{Kết quả} = 87
 \end{array}$$

Tương tự khi trừ số BCD, nếu có mượn khi trừ thì cũng phải sửa sai bằng cách trừ bớt **0110** vào ký số bị sai như trong ví dụ sau:

$$\begin{array}{r}
 61 \quad 0110 \ 0001 \\
 - 38 \quad 0011 \ 1000 \\
 \hline
 23 \quad 0010 \ 1001 \longrightarrow \text{Ký số bên phải mượn 1 khi trừ} \\
 \\
 \quad 0000 \ 0110 \longrightarrow \text{Sửa sai kết quả} \\
 \quad 0010 \ 0011 \quad \text{Kết quả} = 23
 \end{array}$$

Từ các ví dụ trên ta thấy nhiều khi các phép tính cứ phải sửa sai như vậy thì sẽ dẫn đến tốc độ tính toán cũng bị giảm bớt

### 3.8. Biểu diễn các ký tự

Ngoài việc biểu diễn số, chúng ta cũng cần đến biểu diễn chữ và một số ký tự khác. Tùy theo các hệ thống khác nhau, có thể sử dụng các bảng mã khác nhau:

- **ASCII** (7 bit) (American Standard Codes for Information Interchange) để biểu diễn 128 ký tự gọi là mã ASCII-7
- **ASCII** mở rộng (8 bit) để biểu diễn 256 ký tự
  - 00 – 1F: ký tự điều khiển
  - 20 – 7F: ký tự in được
  - 80 – FF: ký tự mở rộng (ký hiệu tiền tệ, vẽ khung, ...)
- **Unicode**: Ngày nay do việc sử dụng rộng rãi mạng toàn cầu Internet với rất nhiều ngôn ngữ khác nhau, rất nhiều ký tự khác nhau nên người ta đã chuyển sang dùng bộ mã Unicode (16 bit) (UTF-8) có thể biểu diễn được tới 65.536 ký tự và như vậy cho phép biểu diễn hầu hết các ngôn ngữ trên thế giới.

### CÂU HỎI VÀ BÀI TẬP CHƯƠNG III

1. Khái niệm thông tin trong máy tính được hiểu như thế nào?  
Lượng thông tin là gì?
2. Sự hiểu biết về một trạng thái trong 4096 trạng thái có thể có ứng với lượng thông tin là bao nhiêu?
3. Số nhị phân 8 bit  $(11001100)_2$ , số này tương ứng với số nguyên thập phân có dấu là bao nhiêu nếu số đang được biểu diễn trong cách biểu diễn:
  - a. Dấu và trị tuyệt đối.
  - b. Số bù 1.
  - c. Số bù 2.
4. Đổi các số sau đây:
  - a.  $(011011)_2$  ra số thập phân.
  - b.  $(-2005)_{10}$  ra số nhị phân 16 bits.
  - c.  $(55.875)_{10}$  ra số nhị phân.
5. Đổi các số sau sang hệ thập phân :  $12321_4$ ,  $23245_7$ ,  $194_{11}$
6. Đổi các số thập phân sau
  - a. 56354 sang nhị phân
  - b. 89353 sang bát phân
  - c. 56253 sang thập lục phân
7. Đổi số thập lục E4B3A5 sang nhị phân và bát phân
8. Đổi các số :
  - a.  $241_7$  sang hệ 4, hệ 8 và hệ 12
  - b.  $2BC4_{16}$  sang hệ 8, hệ 13
9. Cơ số của các số là bao nhiêu nếu nghiệm phương trình bậc 2:

$$x^2 - 10x + 31 = 0 \text{ là } x = 5 \text{ và } x = 8?$$

10. Tìm bù 9 các số thập phân sau : 3425890, 4195618
11. Tìm bù 10 các số thập phân sau : 14394500, 24519004, 34040080

12. Tìm bù 1 và bù 2 các số nhị phân sau : 1100110101100, 10110010111011
13. Đổi các số sau sang BCD
  - a.  $478_{10}$
  - b.  $372_8$
14. Biểu diễn số thực  $-206,34_{10}$  dưới dạng số có dấu chấm động chính xác đơn 32 bit.
15. Biểu diễn số thực  $(32.75)_{10}$  dưới dạng số có dấu chấm động chính xác đơn 32 bit.
16. Tìm biểu diễn chấm động (1 bit dấu, 8 bit mũ quá 127, 23 bit định trị) của các số sau.
  - a) **1025.296875<sub>10</sub>**
  - b) **0.06640625<sub>10</sub>**
17. Thực hiện các phép toán sau trong hệ bù 1. Dùng 8 bit (gồm cả bit dấu) cho mỗi số.
  - a) Lấy +47 cộng -19
  - b) Lấy -15 trừ đi +36
18. Thực hiện các phép toán sau trong hệ bù 2. Dùng 8 bit (gồm cả bit dấu) cho mỗi số.
  - a) Cộng +19 vào -24
  - b) Cộng -48 vào -80
19. Thực hiện hai phép toán sau trên hệ nhị phân bằng cách lấy bù 2 các số âm, các số được biểu diễn bằng **6 bit**.
  - a) **25-11**
  - b) **23-30**

## Chương IV: Mạch Logic số

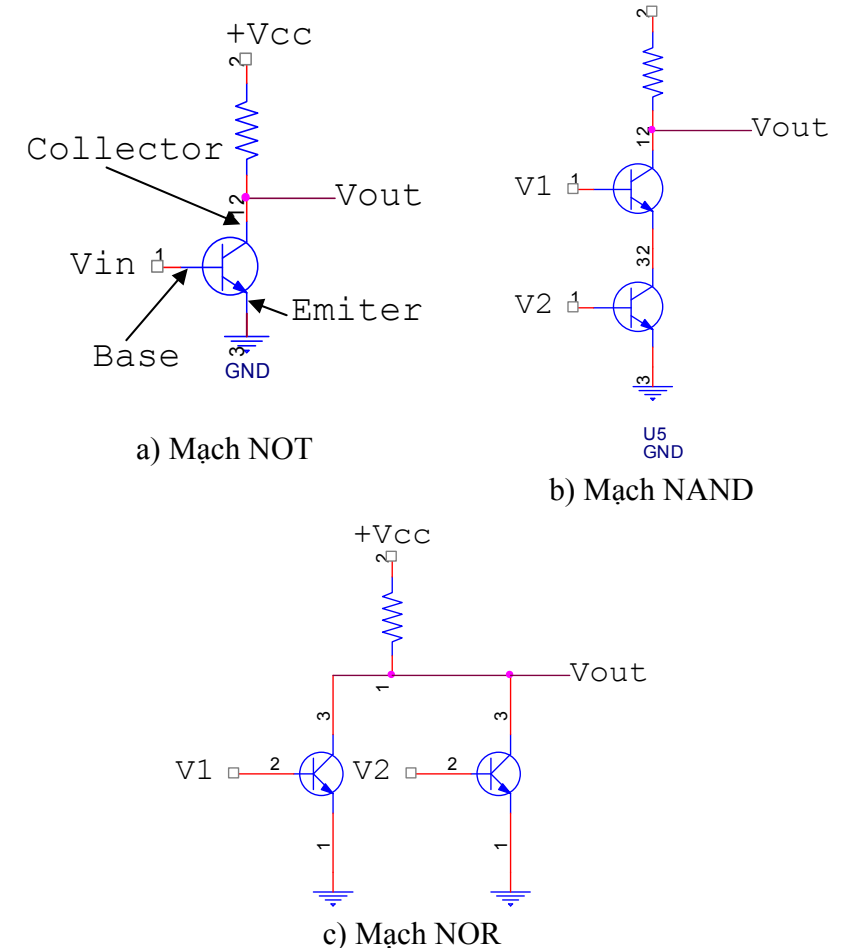
### 4.1. Cổng và đại số Boolean

#### 4.1.1. Cổng (Gate)

**Cổng (hay cổng luận lý) – cơ sở phần cứng, từ đó chế tạo ra mọi máy tính số.** Cổng có một hoặc nhiều lối vào, nhưng chỉ có 1 lối ra. Các giá trị vào hoặc ra chỉ có thể nhận 1 trong 2 giá trị là 1 hoặc 0. Gọi là cổng luận lý vì nó cho kết quả lý luận của đại số logic như nếu A đúng và B đúng thì C đúng (cổng AND:  $C = A \text{ AND } B$ )

Chúng ta sẽ xem xét những ý tưởng cơ bản chế tạo các cổng này để hiểu rõ bản chất của mạch số. Mọi logic số hiện đại rút cuộc cũng dựa trên việc chế tạo transistor vận hành như một công tắc nhị phân cực nhanh. Hình 4.1(a) minh họa (mạch) transistor lưỡng cực đặt vào mạch đơn giản. Transistor này có 3 nối kết với thế giới bên ngoài: cực góp (collector), cực nền (base) và cực phát (emitter). Khi điện áp vào,  $V_{in}$  thấp hơn giá trị tới hạn nào đó (0.8V), transistor sẽ tắt và đóng vai trò như điện trở vô hạn, khiến đầu ra của mạch,  $V_{out}$  nhận giá trị gần với  $V_{cc}$  (điện áp ngoài thường là +3 V). Lúc  $V_{in}$  vượt quá giá trị tới hạn, transistor bật và đóng vai trò như dây dẫn, kéo  $V_{out}$  xuống tới đất (theo qui ước là 0 V).

Chúng ta thấy rằng khi  $V_{in}$  thấp thì  $V_{out}$  cao, và ngược lại. Do đó, mạch này là bộ nghịch đảo (converter hoặc NOT), chuyển logic 0 sang logic 1, và logic 1 sang logic 0, hay tương ứng với một cổng gọi là cổng **NOT**. Cần điện trở để giới hạn dòng điện do transistor kéo qua. Thời gian cần thiết để chuyển từ trạng thái này sang trạng thái khác thường mất vài nano giây tùy theo loại transistor.



Hình 4.1. Cấu tạo cổng **NOT**, **NAND** và **NOR**

Trong Hình 4.1 (b), hai transistor được mắc nối tiếp. Nếu  $V_1$  và  $V_2$  đều cao, cả hai transistor sẽ dẫn điện và  $V_{out}$  sẽ bị kéo xuống thấp. Giả sử một trong hai đầu vào thấp, transistor tương ứng sẽ tắt, và đầu ra sẽ cao. Nói tóm lại  $V_{out}$  sẽ thấp khi và chỉ khi  $V_1$  và  $V_2$  đều cao. Mạch này là một cổng **NAND**. Trong Hình 4.1 (c) hai transistor được mắc song song, thay vì nối tiếp. Trong

trường hợp này, nếu một trong hai đầu vào cao, transistor tương ứng sẽ kéo đầu ra xuống tới đất. Còn như cả hai đầu vào đều thấp, đầu ra sẽ vẫn cao. Mạch này tương ứng với cổng gọi là **NOR**.

- *Cổng là một mạch số gồm một hoặc nhiều tín hiệu nhập và một tín hiệu xuất.*

Một mạch số sẽ được tạo ra từ tập hợp các cổng cơ bản. Mỗi cổng cơ bản có ký hiệu riêng và hoạt động của nó được mô tả qua một bảng gọi là bảng chân trị (truth table).

Tên, ký hiệu, hàm logic biểu diễn và bảng chân trị của các cổng cơ bản liệt kê trong bảng 4.1.


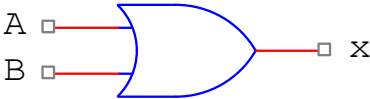
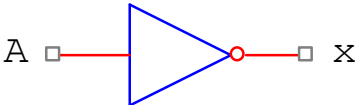
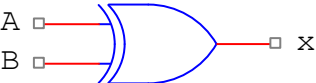
Cổng **AND** có ít nhất 2 đầu vào và 1 đầu ra. Đầu ra chỉ bằng 1 khi và chỉ khi tất cả các đầu vào bằng 1, các trường hợp khác đầu ra sẽ có giá trị bằng 0.

Cổng **OR** có ít nhất 2 đầu vào và 1 đầu ra. Đầu ra bằng 1 khi có một trong các đầu vào bằng 1, các trường hợp khác đầu ra sẽ có giá trị bằng 0.

Cổng **NOT** có một đầu vào và 1 đầu ra. Đầu ra luôn có giá trị nghịch đảo với đầu vào. Đầu vào bằng 1 thì đầu ra bằng 0 và ngược lại.

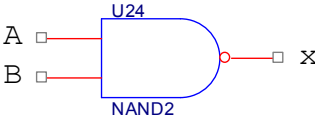
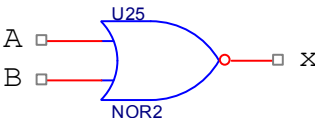
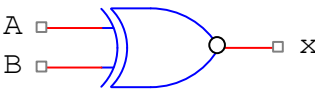
Cổng **XOR** có ký hiệu như cổng OR nhưng có thêm một vòng cung ở đầu vào. Đầu ra là 1 nếu số đầu vào có trị bằng 1 là một số lẻ, các trường hợp khác bằng 0. Trong trường hợp có 2 đầu vào thì đầu ra bằng 1 khi một trong 2 đầu vào bằng 1, các trường hợp khác bằng 0.

Các cổng **NAND**, **NOR**, **NXOR** là bù của các cổng tương ứng AND, OR, XOR và được biểu diễn *thêm một vòng tròn nhỏ ở đầu ra*.

Tên cổng	Ký hiệu	Hàm logic	Bảng chân trị															
AND		$x = A.B$ hoặc $x = AB$ hoặc $x = A \text{ AND } B$	<table><tr><th>A</th><th>B</th><th>x</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	x	0	0	0	0	1	0	1	0	0	1	1	1
A	B	x																
0	0	0																
0	1	0																
1	0	0																
1	1	1																
OR		$x = A + B$ hoặc $x = A \text{ OR } B$	<table><tr><th>A</th><th>B</th><th>x</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	x	0	0	0	0	1	1	1	0	1	1	1	1
A	B	x																
0	0	0																
0	1	1																
1	0	1																
1	1	1																
NOT		$x = \bar{A}$ hoặc $x = \text{NOT } A$	<table><tr><th>A</th><th>x</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table>	A	x	0	1	1	0									
A	x																	
0	1																	
1	0																	
XOR		$x = A \oplus B$	<table><tr><th>A</th><th>B</th><th>x</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	x	0	0	0	0	1	1	1	0	1	1	1	0
A	B	x																
0	0	0																
0	1	1																
1	0	1																
1	1	0																

Bảng 4.1. Các cổng cơ bản

Đối với các cổng có đầu ra nghịch đảo (invert) ta có bảng 4.2.

Tên cổng	Ký hiệu	Hàm logic	Bảng chân trị															
NAND		$x = \overline{A \cdot B}$ <p>hoặc</p> $x = \text{NOT (A AND B)}$	<table><tr><th>A</th><th>B</th><th>x</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	x	0	0	1	0	1	1	1	0	1	1	1	0
A	B	x																
0	0	1																
0	1	1																
1	0	1																
1	1	0																
NOR		$x = \overline{A + B}$	<table><tr><th>A</th><th>B</th><th>x</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	x	0	0	1	0	1	0	1	0	0	1	1	0
A	B	x																
0	0	1																
0	1	0																
1	0	0																
1	1	0																
NXOR		$x = \overline{A \oplus B}$	<table><tr><th>A</th><th>B</th><th>x</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	x	0	0	1	0	1	0	1	0	0	1	1	1
A	B	x																
0	0	1																
0	1	0																
1	0	0																
1	1	1																

Bảng 4.2. Các cổng cơ bản có đầu ra nghịch đảo

#### 4.1.2. Đại số Boolean (Boolean algebra)

Đại số Boolean (hay đại số Logic) là môn toán học nghiên cứu các mệnh đề luận lý và là công cụ toán học để phân tích và tổng hợp các thiết bị mạch số. Các biến số trong đại số Boolean là biến Boolean và là một đại lượng mà tại các thời điểm khác nhau nó chỉ có thể bằng 0 hoặc 1. Biến Boolean thường được sử dụng để biểu diễn mức điện thế có trên dây hay tại các đầu vào/ra (input/output - IO) của một mạch số.

Như vậy, biến Boolean là các biến biểu thị trạng thái của một giá trị điện thế và ta gọi là mức logic. Trong logic số thì nhiều thuật ngữ khác nhau được dùng để biểu thị hai trạng thái nhị phân 0, 1 như trong bảng 4.3.

Logic 0	Logic 1
Sai	Đúng
Tắt	Mở
Thấp	Cao
Không	Có
Công tắc mở	Công tắc đóng

Bảng 4.3. Các thuật ngữ biểu diễn logic “0” và “1”

Như đã nói ở trên, đại số Boolean là một công cụ để phân tích và tổng hợp các thiết bị mạch số hay nói cách khác là để biểu diễn các mối quan hệ giữa đầu vào và ra của mạch số. Các giá trị của biến logic đầu vào sẽ quyết định giá trị của đầu ra tại một thời điểm nhất định. Chúng ta sẽ dùng các ký hiệu bằng chữ để biểu thị các giá trị logic. Ví dụ,  $x$  là các giá trị đầu vào hoặc ra của mạch số, và tại thời điểm bất kỳ có thể  $x = 0$  hoặc  $x = 1$ .

Ba phép tính cơ bản của đại số Boolean (gọi là các phép toán logic) là:

- Phép Phủ định Logic: *NOT*

Ví dụ:

+ phủ định của  $x$ : *NOT*  $x$  hoặc  $\bar{x}$

+ y bằng phủ định của A:  $y = NOT A$  hoặc  $y = \overline{A}$

- Phép cộng logic: *OR*

Ví dụ: x cộng y ta ký hiệu là x *OR* y hoặc  $x + y$

- Phép nhân logic: *AND*

Ví dụ: A nhân B ta ký hiệu A *AND* B hoặc  $A.B$  hoặc  $AB$ .

### ➤ Các quy tắc Logic:

- Quy tắc về phép cộng:

$$X + 0 = X \quad X + X = X$$

$$X + 1 = 1 \quad X + \overline{X} = 1$$

- Quy tắc về phép nhân logic:

$$X . 0 = 0 \quad X . X = X$$

$$X . 1 = X \quad X . \overline{X} = 0$$

- Quy tắc về phủ định:

$$\overline{\overline{X}} = X$$

Các mạch số được thiết kế từ những nguyên tố nhỏ nhất gọi là cổng logic (gate). Các cổng này được cấu thành từ diod, transistor và điện trở, để rồi đầu ra của nó sẽ có giá trị như các phép toán logic cơ bản (NOT, OR, AND). Chúng ta sẽ dùng đại số Boolean để mô tả và phân tích các cổng logic cơ bản này, sau đó sẽ mở rộng ra phân tích và thiết kế cách nối các cổng lại với nhau để tạo thành các mạch số cần thiết.

### Hàm Logic:

Cũng giống như đại số thường, đại số boolean cũng có hàm. Hàm Boolean là hàm của các biến Logic và bản thân cũng chỉ nhận các giá trị 0 hoặc 1. Thường chúng ta dùng hàm boolean để biểu diễn đầu ra của mạch số và các biến logic của hàm đó để biểu diễn các đầu vào của mạch.

➤ **Bảng chân trị** (truth table) là phương tiện mô tả đầu ra của mạch logic phụ thuộc vào các mức đầu vào của mạch. Hay nói cách khác bảng chân trị dùng để biểu diễn mối quan hệ giữa hàm Boolean và các biến logic của hàm đó.

Ví dụ: bảng chân trị của hàm  $y = A OR B = A + B$

A	B	y
0	0	0
0	1	1
1	0	1
1	1	1

Bảng liệt kê mọi tổ hợp có thể có của các biến logic tương trưng cho đầu vào mạch số A và B với các giá trị tương ứng ở đầu ra y.

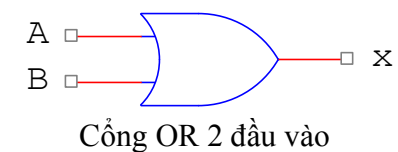
Mỗi cổng cơ bản sẽ có một phép toán Boolean cơ bản tương ứng

### a) Phép toán OR

Hàm x được tổ hợp từ 2 biến logic A và B bằng phép toán OR là:  $x = A + B$  hoặc  $x = A OR B$ . Ở đây dấu “+” không biểu thị cho phép cộng thông thường, mà nó thay cho phép toán OR. Biểu thức  $x = A + B$  được đọc là “x bằng A OR B”, nhưng để đơn giản chúng ta hay dùng là “x bằng A cộng B”. Điều quan trọng ở đây là chúng ta phải nhớ đây là phép toán OR chứ không phải phép toán cộng thông thường. (phép toán cộng thông thường  $1+1=2$ , trong khi phép toán OR là  $1 + 1 = 1$ ).

Tương tự với cổng OR, giá trị của hàm x được xác định qua bảng chân trị sau:

A	B	$x=A+B$
0	0	0
0	1	1
1	0	1
1	1	1

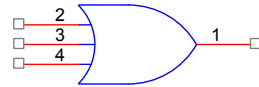




Nhìn vào bảng ta thấy  $x = 1$  khi có một đầu vào trở lên bằng 1. Trường hợp duy nhất có  $x = 0$  là khi tất cả các đầu vào đều bằng 0.

Ví dụ phép toán OR cho 3 biến đầu vào A, B và C:  
 $x = A + B + C$

A	B	C	$x=A+B+C$
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

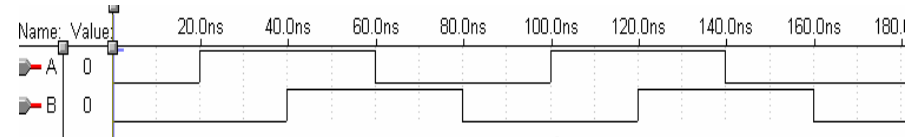
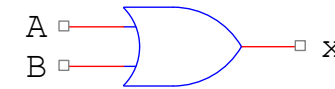


Cổng OR cho 3 đầu vào

#### ➤ Tóm lại:

- Phép toán OR cho kết quả là 1 khi một trong các đầu vào là 1
- Phép toán OR cho kết quả là 0 khi tất cả các đầu vào đều là 0
- Với phép toán OR:  $1+1=1$ ,  $1+1+1=1, \dots$
- Cổng OR là mạch logic thực hiện phép toán OR trên các đầu vào logic của mạch.

**Ví dụ:** xác định đầu ra  $x = A + B$  của cổng OR trong hình 4.2. Tín hiệu các đầu vào A và B của cổng OR thay đổi theo sơ đồ thời gian minh họa:

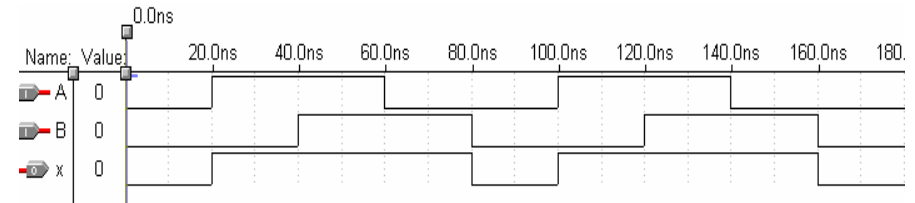


Hình 4.2. Cổng OR và sơ đồ thời gian

Giải:

Chúng ta đã biết là đầu ra của cổng OR chỉ bằng 1 (hay ở mức cao) khi có một trong các đầu vào bằng 1, các trường hợp khác đều bằng 0 (hay ở mức thấp). Từ sơ đồ thời gian của các đầu vào ta thấy:

- cho đến thời điểm  $t=20\text{ns}$  cả A và B đều bằng 0  $\Rightarrow$  tín hiệu đầu ra  $x=0$  trong đoạn này.
- tại thời điểm  $t=20\text{ns}$ , A chuyển từ 0 lên 1  $\Rightarrow$  đầu ra x cũng chuyển lên 1  $\Rightarrow$  đoạn từ  $t=20\text{ns}$  đến  $t=40\text{ns}$  đầu ra x sẽ bằng 1.
- tiếp từ  $t=40\text{ns}$  đến  $t=80\text{ns}$  đầu ra x cũng bằng 1 vì 1 trong 2 đầu vào có trị bằng 1.
- lập luận tương tự như vậy ta có được biểu đồ thời gian cho tín hiệu đầu ra phụ thuộc vào các tín hiệu đầu vào như hình 4.3:



Hình 4.3. Kết quả đầu ra

#### b) Phép toán AND

Hàm AND được tổ hợp từ 2 biến logic A và B bằng phép toán AND là:  $x = A.B$  hoặc  $x=AB$  hoặc  $x = A \text{ AND } B$ . Tương tự với cổng AND, giá trị của hàm x được xác định qua bảng chân trị sau:

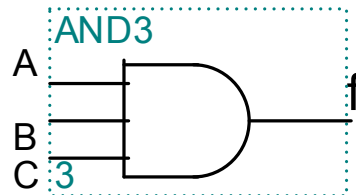
A	B	$x=A.B$
0	0	0
0	1	0
1	0	0
1	1	1



Chú ý rằng trong biểu thức  $x=A.B$  thì dấu nhân ở đây không phải phép toán nhân thông thường, mà là phép toán AND. Nhưng trong trường hợp chỉ áp dụng cho các biến có giá trị là 0 hoặc 1 thì phép toán AND lại tương tự như phép nhân bình thường ( $0.0=0$ ,  $0.1=0$ ,  $1.1=1$ ).

Tương tự, ta cũng có phép toán AND cho 3 biến và cổng AND cho 3 đầu vào như sau:

A	B	C	$f=A.B.C$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1



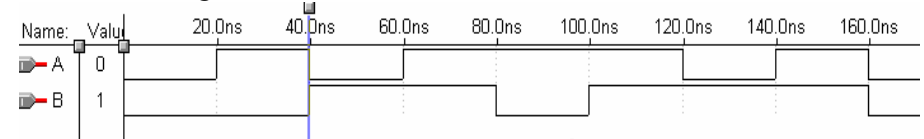
#### ➤ Tóm lại:

- Phép toán AND được thực hiện như phép nhân thông thường giữa các số 0 và 1
- Đầu ra của cổng AND hay giá trị hàm AND chỉ bằng 1 khi tất cả các đầu vào đều bằng 1
- Đầu ra của cổng AND hay giá trị hàm AND bằng 0 khi có một trong các đầu vào bằng 0

- Cổng AND là mạch logic thực hiện phép toán AND trên đầu vào là các biến của hàm AND

Ví dụ:

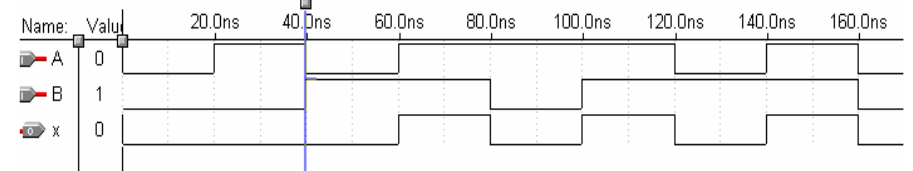
Xác định đầu ra x từ cổng AND, nếu các tín hiệu đầu vào có dạng hình 4.4:



Hình 4.4. Dạng tín hiệu đầu vào

Giải:

Đây là một dạng khác của cách biểu diễn một hàm logic hay một cổng. Thay vì dùng bảng chân trị thì ở đây ta dùng đồ thị tín hiệu các đầu vào và ra. Chúng ta biết rằng đầu ra của cổng AND chỉ bằng 1 khi tất cả các đầu vào đều bằng 1. Như vậy đoạn 0→60ns, đầu ra x sẽ bằng 0 vì trong đoạn này luôn có một trong 2 tín hiệu đầu vào A hoặc B bằng 0. Đoạn 60ns→80ns, tín hiệu đầu ra x sẽ bằng 1 do cả 2 đầu vào A và B đều bằng 1. Tương tự ta có được tín hiệu ở đầu ra như hình 4.5:



Hình 4.5. Tín hiệu đầu ra

#### c) Hàm NOT

$f = \bar{A}$  hay  $f = NOT A$ : Hàm f tương tự với cổng Inverter, có giá trị ngược với biến logic A và được biểu diễn qua bảng chân trị sau:

A	f
0	1
1	0

#### d) Hàm XOR

Hàm XOR cho 2 biến logic  $A$  và  $B$  là:  $f = A \oplus B$  hoặc  $f = A \text{ XOR } B$ . Tương tự với cổng XOR, giá trị của hàm  $f$  được xác định qua bảng chân trị sau:

$A$	$B$	$f$
0	0	0
0	1	1
1	0	1
1	1	0

Ứng dụng của mạch XOR là để so sánh hai số. Nếu hai số bằng nhau thì giá trị đầu ra của mạch là 1, ngược lại là 0.

Như vậy giữa mạch số và đại số Boolean có một mối quan hệ tương ứng qua lại với nhau giữa cổng và hàm Boolean. Nói cách khác ta có thể biểu diễn mạch số bằng hàm Boolean và ngược lại.

#### Nhận xét 1:

- Hàm của  $n$  biến logic sẽ có  $2^n$  tổ hợp biến, với mỗi tổ hợp biến hàm có thể lấy một trong 2 giá trị là 0 hoặc là 1. Như vậy để biểu diễn 1 hàm có 2 biến, ta cần 1 bảng chân trị có 4 dòng hay 4 tổ hợp biến (không tính dòng tiêu đề), hàm cho 3 biến sẽ cần bảng chân trị có  $2^3=8$  dòng,...
- Mục đích của đại số Boolean là cung cấp một công cụ để biểu diễn mạch số, giúp cho việc thiết kế mạch số được dễ dàng hơn. Đặc biệt trong việc đơn giản hàm cũng như tìm ra mạch số tương đương nhưng có kích thước nhỏ gọn và dùng ít cổng hơn.

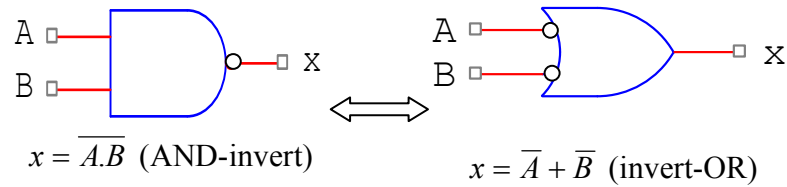
Để tận dụng các phương pháp đơn giản, chúng ta cần một số đồng nhất thức trong đại số Boolean. Bảng 4.3 liệt kê một số đồng nhất thức chính. Điểm đáng lưu ý là mỗi định luật có hai dạng

đối ngẫu của nhau. Bằng cách hoán đổi AND và OR, cũng như 0 và 1, có thể tạo dạng này bằng dạng kia. Thật dễ chứng minh tất cả định luật thông qua xây dựng bảng chân trị.

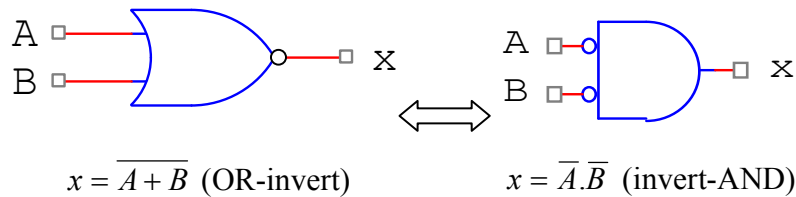
Tên	Dạng AND	Dạng OR
Định luật thống nhất	$1A = A$	$0 + A = A$
Định luật không	$0A = 0$	$1 + A = 1$
Định luật Idempotent	$AA = A$	$A + A = A$
Định luật nghịch đảo	$A.\bar{A} = 0$	$A + \bar{A} = 1$
Định luật giao hoán	$AB = BA$	$A + B = B + A$
Định luật kết hợp	$(AB)C = A(BC)$	$(A+B)+C = A + (B+C)$
Định luật phân bố	$A + BC = (A + B)(A + C)$	$A(B+C) = AB + AC$
Định luật hấp thụ	$A(A + B) = A$	$A + AB = A$
Định luật De Morgan	$\overline{AB} = \bar{A} + \bar{B}$	$\overline{A + B} = \bar{A}\bar{B}$

Bảng 4.4. Các định luật đại số Boolean cơ bản

**Định luật De Morgan** đưa ra ký pháp thay thế và rất quan trọng đối với các cổng NOR và NAND. Nó cho phép ta thay thế từ cổng OR sang AND và ngược lại. Hình 4.6(a) cho ta thấy cách chuyển từ cổng AND sang cổng OR nhờ định lý DeMorgan  $\overline{AB} = \bar{A} + \bar{B}$ . Ta có cổng NAND biểu diễn hàm  $x = \overline{A.B}$  sẽ tương đương với cổng OR với đầu vào được nghịch đảo biểu diễn cho hàm  $x = \bar{A} + \bar{B}$ , hay nói cách khác cổng NAND được biểu diễn bằng 2 cách. Tương tự như vậy ta cũng có cổng NOR tương đương như trong hình 4.6(b) nhờ vào định lý DeMorgan cho dạng OR  $\overline{A + B} = \bar{A}\bar{B}$ .



a)  $\overline{A.B} = \overline{A + B}$



b)  $\overline{A + B} = \overline{A.B}$

Hình 4.6. Các cổng tương đương

Dạng tổng quát của định lý DeMorgan có dạng sau:

$$\overline{x_1 + x_2 + \dots x_n} = \overline{x_1} . \overline{x_2} . \dots \overline{x_n}$$

$$\overline{x_1 x_2 \dots x_n} = \overline{x_1} + \overline{x_2} + \dots + \overline{x_n}$$

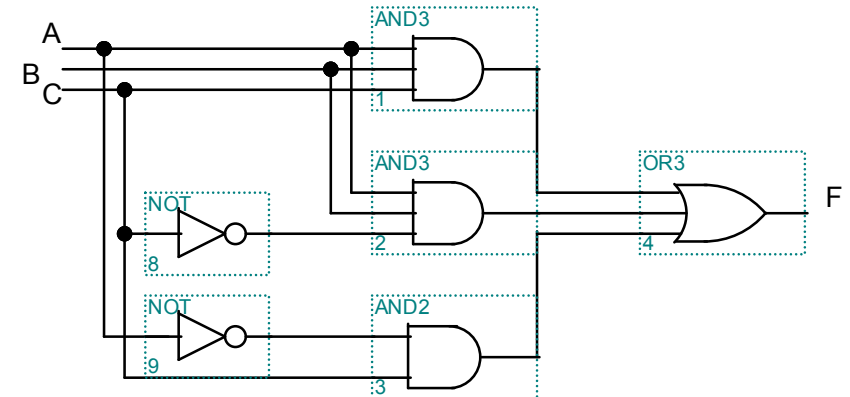
Từ định lý DeMorgan ta rút ra qui tắc lấy bù của một biểu thức đại số. Qui tắc này cho phép ta thay đổi các cổng OR thành các cổng AND và ngược lại. Ví dụ, hàm  $F=AB+BC$  là dạng tổng các tích, hay ta phải dùng 2 cổng AND cho AB và BC, nhưng ta có thể thay thế bằng cổng OR với các đầu vào nghịch đảo bằng cách sau:

$$F = AB + BC \Rightarrow \overline{F} = \overline{AB + BC} = \overline{AB} . \overline{BC} = (\overline{A} + \overline{B}).(\overline{B} + \overline{C})$$

hoặc

$$F = AB + BC \Rightarrow F = \overline{\overline{AB + BC}} = \overline{\overline{AB} . \overline{BC}} = \overline{(\overline{A} + \overline{B}).(\overline{B} + \overline{C})}$$

Để thấy được việc dùng đại số Boolean để đơn giản các mạch số thế nào, chúng ta xem xét ví dụ mạch số như hình 4.7(a)

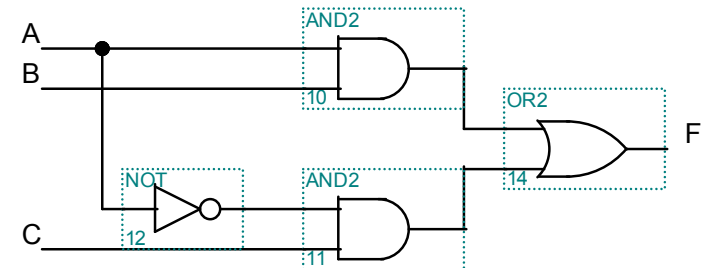


Hình 4.7(a)  $F = ABC + ABC + AC$

Đây là một mạch số biểu diễn hàm  $F = ABC + ABC + AC$ . Tuy nhiên hàm này lại có thể đơn giản dùng đại số Boolean như sau:

$$F = ABC + ABC + AC = AB(C + \overline{C}) + \overline{A}C = AB + \overline{A}C$$

Sơ đồ mạch của hàm F đã được đơn giản như ở hình 4.7(b).



Hình 4.7(b)  $F = AB + AC$

Ta thấy mạch đã đơn giản chỉ cần dùng 4 cổng (2 cổng AND, 1 cổng OR, 1 cổng NOT), trong khi mạch ban đầu phải cần tới 6 cổng và một số cổng lại có nhiều đầu vào hơn. Như vậy rõ ràng đại số Boolean đã giúp ta đơn giản mạch số lại gọn hơn, hiệu quả hơn.

**Một số ví dụ thiết kế và đơn giản mạch:**

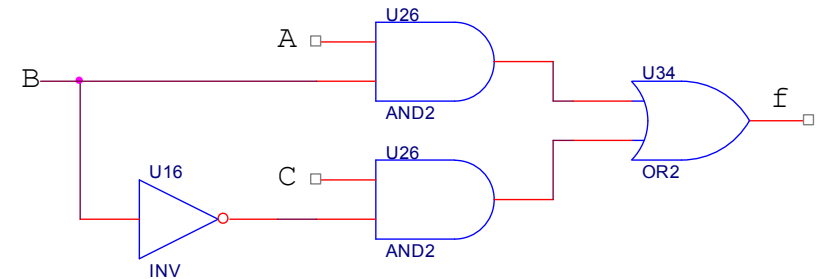
**Ví dụ 1:**

Dùng bảng chân trị để biểu diễn hàm  $f = (A \text{ AND } B) \text{ OR } (C \text{ AND NOT } B)$ , vẽ sơ đồ mạch cho hàm  $f$ .

**Giải:** ta thấy hàm  $f$  có các biến đầu vào là A, B và C. Do đó ta cần bảng chân trị có  $2^3=8$  dòng cho 8 tổ hợp biến mà chúng ta nên sắp xếp theo thứ tự từ nhỏ đến lớn (từ 000 đến 111). Vì hàm  $f$  có nhiều thành phần, nên để đơn giản ta tách chúng ra thành từng phần, rồi sau đó mới tổ hợp lại như bảng sau:

A	B	C	A AND B	NOT B	C AND NOT B	f
0	0	0	0	1	0	0
0	0	1	0	1	1	1
0	1	0	0	0	0	0
0	1	1	0	0	0	0
1	0	0	0	1	0	0
1	0	1	0	1	1	1
1	1	0	1	0	0	1
1	1	1	1	0	0	1

Sơ đồ mạch:



**Ví dụ 2:**

Dùng Boolean Algebra đơn giản các biểu thức sau:

a)  $y = A + AB$

b)  $y = A\bar{B}D + A\bar{B}\bar{D}$

c)  $x = (\bar{A} + B)(A + B)$

d)  $z = (B\bar{C} + \bar{A}D)(\bar{A}\bar{B} + C\bar{D})$

Giải:

a)  $y = A + AB = A(1+B) = A.1 = A$

b)  $y = A\bar{B}D + A\bar{B}\bar{D} = \bar{A}\bar{B}(D + \bar{D}) = \bar{A}\bar{B}.1 = \bar{A}\bar{B}$

c)

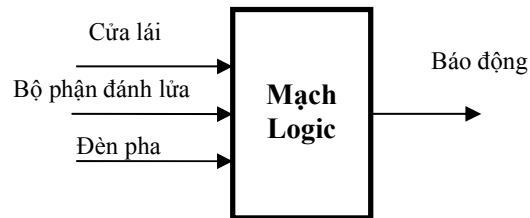
$$x = (\bar{A} + B)(A + B) = \bar{A}.A + \bar{A}.B + B.A + B.B \\ = 0 + \bar{A}.B + B.A + B = B(\bar{A} + A + 1) = B$$

d)

$$z = (B\bar{C} + \bar{A}D)(\bar{A}\bar{B} + C\bar{D}) \\ = B\bar{C}\bar{A}\bar{B} + B\bar{C}C\bar{D} + \bar{A}D\bar{A}\bar{B} + \bar{A}DC\bar{D} \\ = 0 + 0 + 0 + 0 = 0$$

**Ví dụ 3:**

Để làm một bộ báo hiệu cho lái xe biết một số điều kiện, người ta thiết kế 1 mạch báo động như sau:



Tín hiệu từ :  
Cửa lái: 1- cửa mở,  
0 – cửa đóng;  
Bộ phận đánh lửa:  
1 – bật, 0 – tắt;  
Đèn pha: 1 – bật, 0  
– tắt.

Hãy thiết kế mạch logic với 3 đầu vào (cửa, bộ phận đánh lửa, đèn pha), 1 đầu ra (báo động), sao cho bộ phận báo động sẽ hoạt động (báo động = 1) khi tồn tại một trong 2 trạng thái sau:

- Đèn pha sáng trong lúc bộ phận đánh lửa tắt
- Cửa mở trong lúc bộ phận đánh lửa hoạt động

Lập bảng chân trị của hàm ra.

**Giải:**

Đặt các ký hiệu tương ứng:

Cửa lái - A;

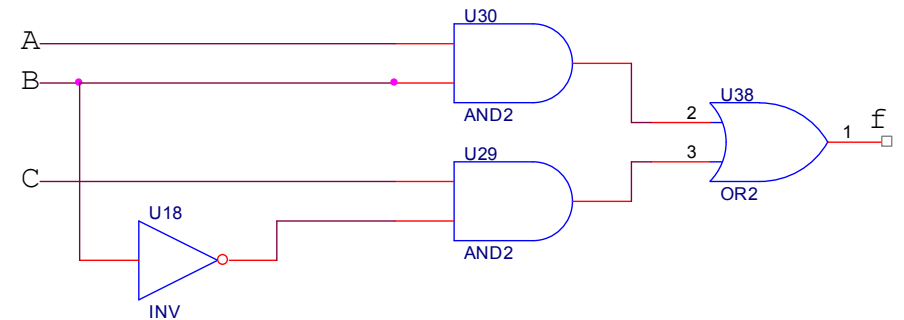
Bộ phận đánh lửa - B

Đèn pha – C

Báo động – f

Theo đề bài  $\Rightarrow f = C\bar{B} + AB$

Sơ đồ mạch:



Bảng chân trị

A	B	C	AB	$\bar{B}$	$C\bar{B}$	f
0	0	0	0	1	0	0
0	0	1	0	1	1	1
0	1	0	0	0	0	0
0	1	1	0	0	0	0
1	0	0	0	1	0	0
1	0	1	0	1	1	1
1	1	0	1	0	0	1
1	1	1	1	0	0	1

## 4.2. Bản đồ Karnaugh

Một mạch số phức tạp sẽ tạo ra một biểu thức đại số rất phức tạp. Bảng chân trị biểu thị của một hàm là duy nhất, nhưng hàm có thể có nhiều dạng khác nhau hay có thể có nhiều mạch số khác nhau cho cùng một chức năng. Ta đã biết rằng biểu thức có thể đơn giản hóa dựa vào đại số Boolean. Tuy nhiên qui trình này thường chỉ áp dụng được đối với các bài toán đơn giản, còn đối với các bài toán phức tạp thì nó không có những qui tắc cho phép ta tiên đoán trước bước đi tiếp theo trong quá trình đơn giản. Phương pháp dùng bản đồ Karnaugh giúp ta đơn giản các biểu thức một cách nhanh chóng, dễ hiểu và hiệu quả hơn.

Để tối thiểu hóa hàm Boole bằng phương pháp bảng Karnaugh phải tuân thủ theo qui tắc về ô kề cận. Hai ô được gọi là



kế cận nhau là hai ô mà khi ta chuyển từ ô này sang ô kia chỉ làm thay đổi giá trị của 1 biến. Quy tắc chung của phương pháp rút gọn bằng bảng Karnaugh là gom (kết hợp) các ô kế cận lại với nhau. Khi gom 2 ô kế cận nhau sẽ loại được 1 biến, gom 4 ô kế cận sẽ loại được 2 biến, gom 8 ô kế cận sẽ loại được 3 biến.

**Tổng quát,** khi gom  $2^n$  ô kế cận sẽ loại được  $n$  biến. Những biến bị loại là những biến khi ta đi vòng qua các ô kế cận mà giá trị của chúng thay đổi.

### ➤ Những điều cần lưu ý:

- Vòng gom được gọi là hợp lệ khi trong vòng gom đó có ít nhất 1 ô chưa thuộc vòng gom nào.
- Những ô nào có giá trị tùy ý ta biểu diễn trong ô đó bằng ký hiệu  $x$ , và những ô đó được gọi là tùy định
- Việc kết hợp những ô kế cận với nhau còn tùy thuộc vào phương pháp biểu diễn hàm Boolean theo dạng tổng các tích (dạng 1) hay theo dạng tích các tổng (dạng 2). Điều này có nghĩa là: nếu ta biểu diễn hàm Boolean theo dạng 1 thì ta chỉ quan tâm những ô kế cận nào có giá trị bằng 1 và tùy định, ngược lại nếu ta biểu diễn hàm Boolean dưới dạng 2 thì ta chỉ quan tâm những ô kế cận nào có giá trị bằng 0 và tùy định. Ta quan tâm những ô tùy định sao cho những ô này kết hợp với những ô có giá trị bằng 1 (nếu biểu diễn theo dạng 1) hoặc bằng 0 (nếu biểu diễn theo dạng 2) sẽ làm cho số lượng ô kế cận là  $2^n$  lớn nhất.
- Các ô kế cận muốn gom được phải là kế cận vòng tròn nghĩa là ô kế cận cuối cũng là ô kế cận đầu tiên.
- Các vòng phải được gom sao cho số ô có thể vào trong vòng là lớn nhất và nhớ là để đạt được điều đó, thường ta phải gom cả những ô đã gom vào trong các vòng khác.

### □ Mục đích cần đạt:

- Biểu thức có chứa ít nhất các thừa số và mỗi thừa số chứa ít nhất các biến.
- Mạch logic thực hiện có chứa ít nhất các vi mạch số

**- Phương pháp dùng bản đồ Karnaugh sẽ được dùng hầu như trong thiết kế mọi mạch số vì vậy có một tầm quan trọng đặc biệt mà các sinh viên phải nắm thật chắc chắn.**

### ▪ Dạng chính tắc và dạng chuẩn của hàm Boole

- **Tích chuẩn (minterm):**  $m_i$  ( $0 \leq i < 2^n - 1$ ) là các số hạng tích (AND) của  $n$  biến mà hàm Boole phụ thuộc với quy ước biến đó có bù nếu nó là 0 và không bù nếu là 1.
- **Tổng chuẩn (Maxterm):**  $M_i$  ( $0 \leq i < 2^n - 1$ ) là các số hạng tổng (OR) của  $n$  biến mà hàm Boole phụ thuộc với quy ước biến đó có bù nếu nó là 1 và không bù nếu là 0

x	y	z	Minterms	Maxterms
0	0	0	$m_0 = \bar{x} \bar{y} \bar{z}$	$M_0 = x + y + z$
0	0	1	$m_1 = \bar{x} \bar{y} z$	$M_1 = x + y + \bar{z}$
0	1	0	$m_2 = \bar{x} y \bar{z}$	$M_2 = x + \bar{y} + z$
0	1	1	$m_3 = \bar{x} y z$	$M_3 = x + \bar{y} + \bar{z}$
1	0	0	$m_4 = x \bar{y} \bar{z}$	$M_4 = \bar{x} + y + z$
1	0	1	$m_5 = x \bar{y} z$	$M_5 = \bar{x} + y + \bar{z}$
1	1	0	$m_6 = x y \bar{z}$	$M_6 = \bar{x} + \bar{y} + z$
1	1	1	$m_7 = x y z$	$M_7 = \bar{x} + \bar{y} + \bar{z}$

Bảng 4.5. Các tích chuẩn và tổng chuẩn của tổ hợp 3 biến

- **Dạng chính tắc 1 (dạng chuẩn 1):** là dạng tổng của các tích chuẩn\_1 (minterm\_1 là minterm mà tại tổ hợp đó hàm Boole có giá trị 1).

Ví dụ ta có hàm  $F$  với các giá trị tương ứng trong bảng 4.6. Theo đó hàm có giá trị bằng 1 tại các tổ hợp biến là 1,3 và 4.

x	y	z	Minterms	Maxterms	F	$\bar{F}$
0	0	0	$m_0 = \bar{x} \bar{y} \bar{z}$	$M_0 = x + y + z$	0	1
0	0	1	$m_1 = \bar{x} \bar{y} z$	$M_1 = x + y + \bar{z}$	1	0
0	1	0	$m_2 = \bar{x} y \bar{z}$	$M_2 = x + \bar{y} + z$	0	1
0	1	1	$m_3 = \bar{x} y z$	$M_3 = x + \bar{y} + \bar{z}$	1	0
1	0	0	$m_4 = x \bar{y} \bar{z}$	$M_4 = \bar{x} + y + z$	1	0
1	0	1	$m_5 = x \bar{y} z$	$M_5 = \bar{x} + y + \bar{z}$	0	1
1	1	0	$m_6 = x y \bar{z}$	$M_6 = \bar{x} + \bar{y} + z$	0	1
1	1	1	$m_7 = x y z$	$M_7 = \bar{x} + \bar{y} + \bar{z}$	0	1

Bảng 4.6. Hàm F theo dạng chính tắc 1

Theo bảng 4.6. ta có:

$$F(x, y, z) = \bar{x} \bar{y} z + \bar{x} y z + x \bar{y} \bar{z} = m_1 + m_3 + m_4$$

Và để đơn giản người ta thường dùng ký hiệu cho dạng chuẩn 1 như sau :

$$\begin{aligned} F(x, y, z) &= \bar{x} \bar{y} z + \bar{x} y z + x \bar{y} \bar{z} = m_1 + m_3 + m_4 \\ &= \sum(1, 3, 4) \end{aligned}$$

- **Dạng chính tắc 2 (dạng chuẩn 2):** là dạng tích của các tổng chuẩn\_0 (Maxterm\_0 là Maxterm mà tại tổ hợp đó hàm Boole có giá trị 0).

Theo bảng 4.6 hàm F sẽ có dạng:

$$\begin{aligned} F(x, y, z) &= (x + y + z)(x + \bar{y} + z)(\bar{x} + y + \bar{z})(\bar{x} + \bar{y} + z)(\bar{x} + \bar{y} + \bar{z}) \\ &= M_0 M_2 M_5 M_6 M_7 \end{aligned}$$

Và để đơn giản người ta thường dùng ký hiệu cho dạng chuẩn 2 như sau :

$$\begin{aligned} F(x, y, z) &= M_0 M_2 M_5 M_6 M_7 \\ &= \prod(0, 2, 5, 6, 7) \end{aligned}$$

### ▪ Trường hợp tùy định (don't care)

Là trường hợp mà tại tổ hợp biến đó giá trị của hàm không xác định. Trong trường hợp này ở dạng chuẩn 1 ta dùng ký hiệu là chữ d nhỏ, còn ở dạng chuẩn 2 là chữ D lớn.

Ví dụ: Hàm F được cho dưới dạng bảng chân trị như bảng 4.7.

A B C	F
0 0 0	X
0 0 1	0
0 1 0	1
0 1 1	1
1 0 0	0
1 0 1	1
1 1 0	0
1 1 1	X

Bảng 4.7. Hàm F có các ô tùy định

Hàm F trong bảng 4.7 sẽ được biểu diễn theo 2 dạng chính tắc:

$$\begin{aligned} F(A, B, C) &= \sum(2, 3, 5) + d(0, 7) \\ &= \Pi(1, 4, 6) \cdot D(0, 7) \end{aligned}$$

- **Chú ý:**
- Ở dạng chuẩn 1 các giá trị trong dấu ngoặc là giá trị của tổ hợp biến mà tại đó hàm có giá trị bằng 1, trong khi ở dạng chuẩn 2 các giá trị trong dấu ngoặc là giá trị của tổ hợp biến mà tại đó hàm có giá trị bằng 0.
- Các giá trị tùy định trong hai dạng chuẩn sẽ giống nhau, tuy nhiên ký hiệu một dạng là chữ thường và dấu cộng, còn một dạng là chữ in hoa và dấu nhân.

### Các dạng bản đồ Karnaugh đơn giản:

Bản đồ Karnaugh (gọi tắt là bản đồ K) giống như bảng chân trị, là phương tiện biểu diễn mối quan hệ giữa các đầu vào logic và đầu ra tương ứng. Dưới đây ta sẽ liệt kê các loại bản đồ K đơn giản, biểu diễn tương ứng với bảng chân trị của chúng.

### - Bản đồ Karnaugh 2 biến

Bản đồ K 2 biến là một bản đồ có 4 ô, vị trí trong mỗi ô tương ứng với tổ hợp biến đầu vào. Ở ngoài các cột và dòng ta ghi các giá trị tương ứng của các biến. Ở đây có 2 hàng biểu thị cho 2 giá trị của biến  $A$  (0 và 1) và 2 cột biểu thị cho 2 giá trị của biến  $B$ . Tọa độ của mỗi ô tương ứng với tổ hợp nhị phân của các biến đầu vào. Ví dụ như trong hình 4.8, tại tọa độ  $A=0$  và  $B=0$  tương ứng với tổ hợp  $AB=00$  hay ta ghi là  $\overline{A}\overline{B}$  là giá trị đầu ra  $x=1$ . Tại tọa độ  $AB=01$  giá trị đầu ra tương ứng  $x=0$  nên ta ghi vào ô tương ứng giá trị 0. Tương tự như vậy cho các ô còn lại ta sẽ có được bản đồ K như trong hình 4.8(c).

A	B	x
0	0	1
0	1	0
1	0	0
1	1	1

$$x = \overline{A}\overline{B} + AB$$

b) Hàm biểu diễn đầu ra

B A	0	1
	0	1
0	1	0
1	0	1

c) Bản đồ K

a) Ví dụ bảng chân trị

Hình 4.8. Ví dụ bản đồ K 2 biến

### - Bản đồ Karnaugh 3 biến

Cách điền vào bản đồ K 3 biến cũng như trong trường hợp trên và cụ thể thế nào ta sẽ xem trong các ví dụ ngay sau đây. Bản đồ K 3 biến sẽ có dạng như sau:

BC A	00	01	11	10
	0	1	3	2
0	0	1	3	2
1	4	5	7	6

### - Bản đồ Karnaugh 4 biến

Cách điền vào trong bản đồ K 4 biến cũng sẽ xem trong ví dụ, còn dạng của như sau

CD AB	00	01	11	10
	0	1	3	2
00	0	1	3	2
01	4	5	7	6
11	12	13	15	14
10	8	9	11	10

Trong thực tế chúng ta chỉ dùng các bản đồ cho từ 3 biến trở lên, và cũng chỉ dùng cho đến 5 biến vì nhiều hơn nữa thì sẽ rất phức tạp vì ở đây chúng ta dùng phương pháp trực quan. Để hiểu rõ cách dùng bản đồ Karnaugh chúng ta sẽ xem xét các ví dụ cụ thể sau.

#### Ví dụ 1:

Dùng bản đồ Karnaugh đơn giản hàm  $f(A,B,C) =$

$$\sum(0,2,4,5,6).$$

Giải:

Trước hết đề bài ghi như vậy có nghĩa là tại các giá trị mà tổ hợp đầu vào bằng 0 ( $ABC=000$ ), 2 ( $ABC=010$ ), 4 ( $ABC=100$ ), 5 ( $ABC=101$ ), hoặc 6 ( $ABC=110$ ) thì giá trị của hàm  $f$  sẽ bằng 1 hay nói cách khác giá trị đầu ra bằng 1

**Bước 1:** vẽ bản đồ Karnaugh.

Hàm  $f$  biểu diễn các ô cho giá trị hàm bằng 1, mỗi ô tương ứng với một tổ hợp các biến đầu vào. Như vậy ở các ô có giá trị đầu vào là 0 ( $ABC=000$ ), 2 ( $ABC=010$ ), 4 ( $ABC=100$ ), 5 ( $ABC=101$ ) và 6 ( $ABC=110$ ) sẽ có giá trị là 1 (Hàm  $f=1$ ).

		BC			
A		00	01	11	10
	0	1			1
	1	1	1		1

Bước 2: nhóm các phần tử gần nhau theo từng nhóm, từ bản đồ chính ta có 2 nhóm

		BC			
A		00	01	11	10
	0	1			1
	1	1	1		1

Vòng 1 (nhóm các ô 1 ở hàng A=0 và A=1)

Vòng 2 (nhóm các ô 1 ở cột BC=00 và BC=01)

Từ bản đồ ta sẽ nhóm được 2 vòng. Vòng 1 ta nhóm tối đa được 4 ô, và 4 ô này cho ta biểu thức  $\overline{C}$  (trong 4 ô này thì chỉ có biến C là không đổi và  $C=0$  nên ta biểu diễn dưới dạng  $\overline{C}$ ). Vòng 2 nhóm ô còn lại với 1 ô đã nhóm ở vòng 1 và cho ta biểu thức  $A\overline{B}$ .

Bước 3: Viết lại hàm theo các nhóm ở bản đồ Karnaugh, ta sẽ có:

$$f(A, B, C) = A\overline{B} + \overline{C}$$

**Ví dụ 2:**

Dùng bản đồ Karnaugh rút gọn hàm

$f(A, B, C, D) = \sum(0, 2, 3, 4, 6, 7, 9, 12, 13)$  và vẽ sơ đồ mạch của hàm f dùng các cổng AND, OR và NOT.

Giải:

Từ đề bài ta thấy hàm cho có 4 biến đầu vào do đó ta cần bản đồ K cho 4 biến và sau khi điền các ô tương ứng với giá trị hàm bằng 1 ta có Bản đồ Karnaugh:

		CD			
AB		00	01	11	10
	00	1		1	1
	01			1	1
	11	1	1		
	10		1		

Sau khi nhóm ta có được 4 vòng như sau:

		CD			
AB		00	01	11	10
	00	1		1	1
	01			1	1
	11	1	1		
	10		1		

Vòng 1 (nhóm các ô 1 ở hàng AB=00 và AB=01)

Vòng 2 (nhóm các ô 1 ở hàng AB=11 và AB=10)

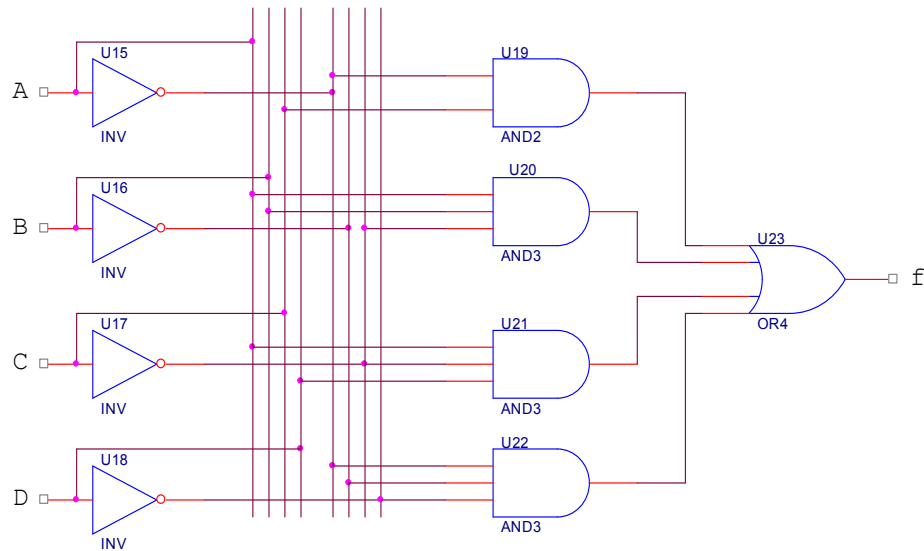
Vòng 3 (nhóm các ô 1 ở cột CD=11 và CD=10)

Vòng 4 (nhóm các ô 1 ở cột CD=00 và CD=01)

Kết quả hàm rút gọn:

$$f(A, B, C, D) = \overline{A}C + ABC\overline{C} + A\overline{C}D + \overline{A}\overline{B}\overline{D}$$

Sơ đồ mạch:



**Ví dụ 3:**

Dùng bản đồ Karnaugh rút gọn hàm  $f(A, B, C, D) = \sum (0, 1, 2, 3, 4, 6, 7, 8, 9, 10, 11, 13)$  và vẽ sơ đồ mạch của hàm f.

Lời giải:

Tương tự như các ví dụ trên, trong bài này chúng ta cũng cần có 4 biến cho đầu vào. Các giá trị hàm bằng 1 được xác định trong dấu ngoặc của hàm.

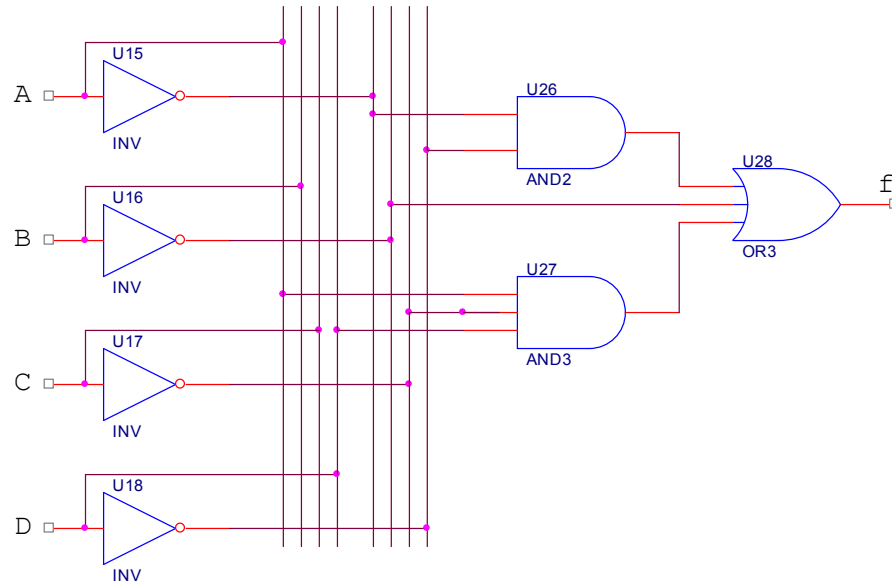
- Bản đồ Karnagh

AB \ CD				
	00	01	11	10
00	1	1	1	1
01	1			1
11		1		
10	1	1	1	1

- Sau khi nhóm:

		CD				
		00	01	11	10	
AB	00	1	1	1	1	1
	01	1			1	2
	11		1			
	10	1	1	1	1	1

Kết quả hàm rút gọn:  $f(A, B, C, D) = \bar{B} + \bar{A}\bar{D} + A\bar{C}D$   
Sơ đồ mạch:



Ví dụ 4: Dùng bản đồ Karnaugh để đơn giản hàm sau:

$$f(A, B, C, D) = \prod(3, 4, 5, 7, 10, 12, 13) + D(8, 9, 11)$$

Lời giải:

Tương tự như các ví dụ trên, trong bài này hàm cho dưới dạng chuẩn 2, là một hàm có 4 biến. Các giá trị hàm bằng 0 được xác định trong dấu ngoặc của  $\prod$  và các vị trí mà giá trị hàm không xác định trong dấu ngoặc của D.

- Bản đồ Karnaugh

CD \ AB	CD			
	00	01	11	10
00			0	
01	0	0	0	
11	0	0		
10	X	X	X	0

- Sau khi nhóm theo các ô 0 ta có:

CD \ AB	CD			
	00	01	11	10
00			0	
01	0	0	0	
11	0	0		
10	X	X	X	0

Group 1 (dashed box): (01, 00), (01, 01), (11, 00), (11, 01)  
Group 2 (dashed box): (10, 00), (10, 01), (10, 11), (10, 10)  
Group 3 (dashed box): (00, 01), (01, 01), (10, 01), (11, 01)

Kết quả hàm rút gọn:  $f(A, B, C, D) = (\bar{B} + C)(\bar{A} + B)(A + \bar{C} + \bar{D})$

#### ⊕ Lưu ý:

- Khi dùng bản đồ K là ở các vị trí không xác định (có thể 1 hoặc 0) thì ta biểu diễn bằng chữ “x” và các ô này ta có thể coi là “1” hoặc “0” tùy thuộc vào trường hợp của bản đồ K để có thể gom số ô lại được nhiều nhất.

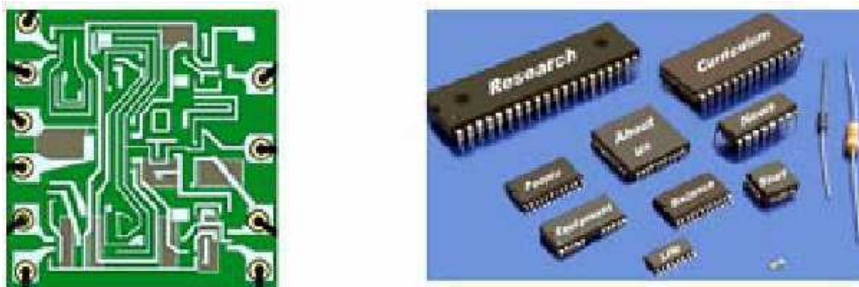


- Nếu ta xét gom theo giá trị hàm bằng 0, thì ta chỉ xét các ô có giá trị 0, những ô có giá trị là “x” thì không cần xét, nhưng có thể được gom chung vào các ô có giá trị 0 để được hàm tối giản

### 4.3. Những mạch logic số cơ bản

#### 4.3.1. Mạch tích hợp IC (Integrated Circuit)

Các cổng logic không được chế tạo hoặc bán riêng lẻ, mà theo đơn vị mạch tích hợp (integrated circuit), thường gọi là IC hay vi mạch (chíp). IC là mảnh silicon hình vuông khoảng 5x5 mm, trên đó đã lắng đọng một số cổng. IC thường được gắn trong vỏ bọc nhựa hoặc ceramic rộng 5-15 mm và dài 20-50 mm. Dọc theo cạnh dài là hai hàng chân song song dài khoảng 5 mm có thể cắm vào ổ cắm hoặc hàn vào bảng mạch in. Mỗi chân nối với đầu vào hay đầu ra của cổng nào đó trên vi mạch, hoặc nối nguồn hoặc nối đất. Về mặt kỹ thuật vỏ bọc có hai hàng chân bên ngoài và IC bên trong được gọi tên là lớp vỏ có hai hàng chân (DIP), tuy nhiên mọi người gọi chúng là vi mạch, do đó làm mờ nhạt sự khác biệt giữa mảnh silicon và vỏ bọc. Đối với vi mạch lớn, người ta thường dùng vỏ bọc hình vuông với các chân trên cả 4 cạnh. Hình 4.9 cho ta thấy một số IC được đóng gói.



Hình 4.9. Một số IC

Các IC có những ưu điểm hơn hẳn các loại linh kiện trước đó.

- Kích thước nhỏ gọn, trọng lượng nhỏ.
- Tiêu thụ năng lượng thấp.
- Tốc độ hoạt động cao.
- Chịu được nhiệt cao, ít chịu tác động của môi trường.
- Giá thành hạ.

Vì vậy IC đã tạo cơ sở để hàng loạt thiết bị điện tử ra đời với những tính năng hơn hẳn các thế hệ trước.

Có thể chia vi mạch thành các lớp **tùy thuộc vào khả năng chứa và sắp xếp các cổng trên cùng một chip gọi là mức tích hợp:**

Mạch SSI (tích hợp cỡ nhỏ): 1 - 10 cổng

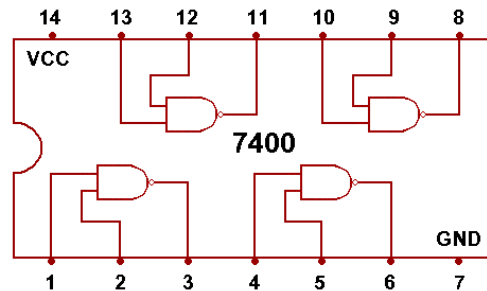
- Mạch MSI (tích hợp cỡ trung bình): 10 - 100 cổng
- Mạch LSI (tích hợp cỡ lớn): 100 - 100.000 cổng
- Mạch VLSI (tích hợp cỡ rất lớn): > 100.000 cổng

Những lớp trên có thuộc tính khác nhau và ứng dụng theo cách khác nhau. Thường khi sản xuất các IC sẽ đi kèm theo bộ hướng dẫn chức năng và các chân tương ứng của IC đó. Ví dụ IC hình 4.10 là loại IC logic đơn giản có 4 cổng NAND - 2 đầu vào, các cổng NAND giống nhau và độc lập với nhau.

IC có 14 chân, chân số 7 là chân nối đất, chân 14 nối với nguồn Vcc:

Vcc: +5V

GND: nối đất.



Hình 4.10. Sơ đồ một IC

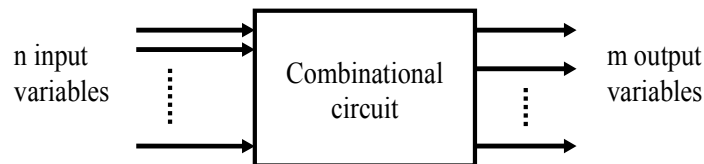
#### 4.3.2. Mạch kết hợp (Combinational circuit)

Nhiều ứng dụng logic số đòi hỏi mạch phải có nhiều đầu vào và đầu ra trong đó đầu ra được xác định qua đầu vào hiện tại. Mạch như thế được gọi là mạch kết hợp (combinational circuit). Không phải mạch nào cũng có thuộc tính này. Ví dụ, mạch chứa phần tử nhớ có thể tạo đầu ra tùy vào giá trị lưu và cả biến nhập.

**Mạch kết hợp** là tổ hợp các cổng luận lý kết nối với nhau tạo thành một bản mạch có chung một tập các ngõ vào và ra.

Tại một thời điểm, trị nhị phân ở ngõ ra là hàm của tổ hợp nhị phân các ngõ vào. Sơ đồ khối mạch tổ hợp như hình vẽ 4.11. n biến nhập nhị phân xuất phát từ một nguồn nào đó đi vào sơ đồ mạch và xuất ra ngoài m biến nhị phân.

Mạch tổ hợp được xác định qua bảng chân trị với n biến nhập và m biến xuất hoặc được xác định qua m hàm Boolean.



Lược đồ khối mạch kết hợp

Hình 4.11.

#### ➤ Thiết kế mạch tổ hợp

Để thiết kế một mạch tổ hợp, nhằm tránh những sai sót không cần thiết, chúng ta cần tuân thủ theo các bước sau:

1. Xác định bài toán để đi đến kết luận có những đầu nhập, xuất nào
2. Lập bảng chân trị xác định mối quan hệ giữa nhập và xuất
3. Dựa vào bảng chân trị, xác định hàm cho từng ngõ ra
4. Dùng đại số boolean hoặc bản đồ Karnaugh để đơn giản các hàm ngõ ra
5. Vẽ sơ đồ mạch theo các hàm đã đơn giản.

Sau đây chúng ta sẽ xem xét một số mạch tổ hợp thông dụng nhất, mà thường từ các mạch này người ta tạo ra các mạch khác phức tạp hơn.

#### 4.3.3. Bộ dồn kênh (Multiplexer) – Bộ phân kênh (Demultiplexer)

Bộ dồn kênh hay còn gọi là mạch chọn kênh là mạch có chức năng chọn lần lượt 1 trong N kênh vào để đưa đến ngõ ra duy nhất (ngõ ra duy nhất đó gọi là đường truyền chung). Do đó, mạch chọn kênh còn gọi là mạch chuyển dữ liệu song song ở ngõ vào thành dữ liệu nối tiếp ở ngõ ra, được gọi là Multiplexer (viết tắt là MUX).

Bộ phân kênh hay mạch phân đường còn gọi là mạch tách kênh (phân kênh, giải đa hợp), mạch này có nhiệm vụ tách 1 nguồn dữ liệu ở đầu vào để rẽ ra N ngõ ra khác nhau. Do đó, mạch phân đường còn gọi là mạch chuyển dữ liệu nối tiếp ở ngõ vào thành dữ liệu song song ở ngõ ra, được gọi là Demultiplexer (viết tắt là DEMUX).

##### a) Bộ dồn kênh

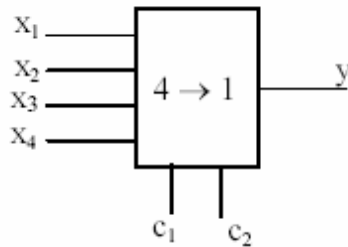
Ở cấp độ logic số, bộ dồn kênh (multiplexer) là mạch có  $2^n$  đầu vào dữ liệu, một đầu ra dữ liệu và n đầu vào điều khiển chọn

một trong các đầu vào dữ liệu. Đầu vào được chọn sẽ định tuyến tới đầu ra.

Xét mạch chọn kênh đơn giản có 4 ngõ vào và 1 ngõ ra như hình 4.12.

Trong đó :

- +  $x_1, x_2, x_3, x_4$  : các kênh dữ liệu vào
- + Ngõ ra  $y$  : Đường truyền chung.
- +  $c_1, c_2$  : các ngõ vào điều khiển



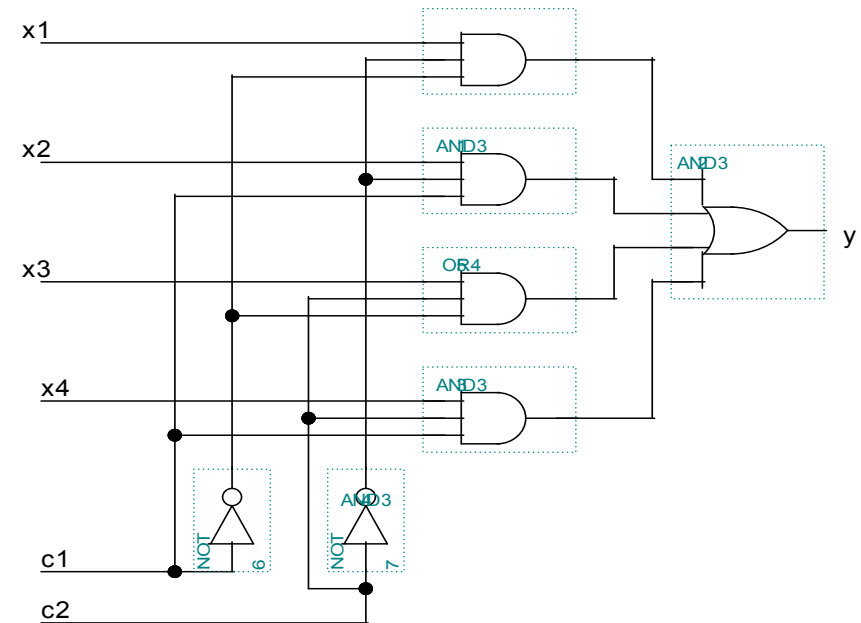
Hình 4.12. Sơ đồ khối MUX 4 đầu vào

Để thay đổi lần lượt từ  $x_1 \rightarrow x_4$  phải có điều khiển do đó đối với mạch chọn kênh để chọn lần lượt từ 1 trong 4 kênh vào cần có các ngõ vào điều khiển  $c_1, c_2$ . Nếu có  $N$  kênh vào thì cần có  $n$  ngõ vào điều khiển thỏa mãn quan hệ:  $N=2^n$ . Nói cách khác: Số tổ hợp ngõ vào điều khiển bằng số lượng các kênh vào.

Việc chọn dữ liệu từ 1 trong 4 ngõ vào để đưa đến đường truyền chung là tùy thuộc vào tổ hợp tín hiệu điều khiển. Trong bảng 4.5 cho ta thấy tùy thuộc vào tín hiệu điều khiển  $c_1, c_2$  mà ngõ ra sẽ nhận tín hiệu từ ngõ vào nào.

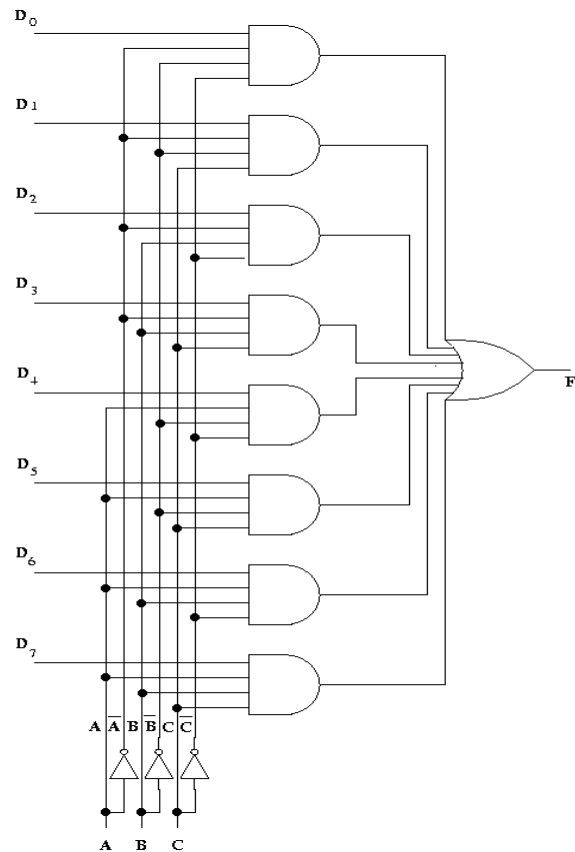
$c_1$	$c_2$	$y$
0	0	$x_1$
0	1	$x_2$
1	0	$x_3$
1	1	$x_4$

Bảng 4.4. Tín hiệu đầu ra phụ thuộc vào tín hiệu điều khiển  
Sơ đồ mạch đôn 4-1 như hình 4.13.



Hình 4.13. Bộ đôn kênh 4-1

Một ví dụ khác ở hình 4.14 là bộ đôn kênh 8 đầu vào.



Hình 4.14. Bộ dồn kênh (Multiplexer) 8 đầu vào

Ba đường điều khiển, A, B, và C mã hóa con số 3 bit qui định 1 trong 8 đường vào nào sẽ định tuyến tới cổng OR rồi ra. Bất luận giá trị nào nằm trên đường điều khiển, 7 cổng AND sẽ luôn xuất 0, cổng còn lại có thể xuất 0 hay 1, tùy theo giá trị đường vào được chọn. Mỗi cổng AND được kích hoạt bằng kết hợp đầu vào điều khiển khác nhau.

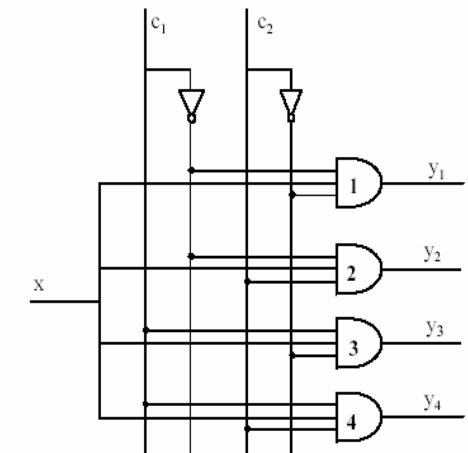
Như vậy khi thiết kế các mạch mà chỉ có 1 đầu vào duy nhất, nhưng tín hiệu vào lại được lựa chọn từ nhiều nguồn khác nhau thì chúng ta có thể dùng bộ dồn kênh để làm việc đó. Trong

các thiết bị số bộ dồn kênh được dùng rất thường xuyên cho nên để hiểu tốt các phần sau các sinh viên cần hiểu thật rõ mạch này.

### b) Bộ phân kênh (Demultiplexer)

Ngược lại với bộ dồn kênh là bộ phân kênh. Nó cho phép từ một kênh vào cho ra nhiều kênh khác nhau tùy thuộc vào đường điều khiển. Bảng trạng thái mô tả hoạt động của mạch và sơ đồ mạch bộ phân kênh như trong hình 4.15.

$c_1$	$c_2$	$y_1$	$y_2$	$y_3$	$y_4$
0	0	x	0	0	0
0	1	0	x	0	0
1	0	0	0	x	0
1	1	0	0	0	x



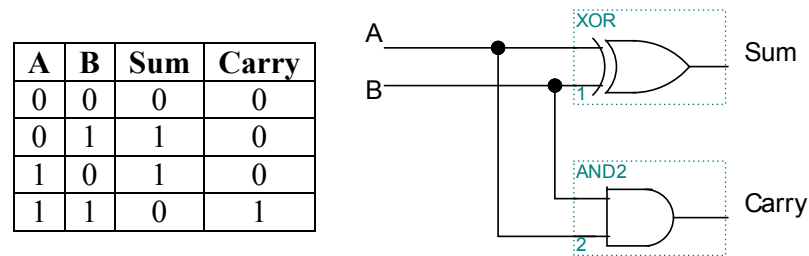
Hình 4.15. Bộ phân kênh 1-4

### 4.3.4. Mạch Cộng

#### a) Mạch nửa cộng (Half Adder)

Đối với tất cả các máy tính thì việc thực hiện các phép tính số học là quan trọng nhất. Vì vậy mạch thực hiện phép cộng là thành phần thiết yếu trong mỗi CPU. Hình 4.16 minh họa bảng chân trị cho phép cộng 1 bit. Mạch nửa cộng là một mạch gồm 2 cổng XOR và AND. Hai đầu ra của mạch:

- Tín hiệu tổng đầu vào A và B: **Sum**
- Tín hiệu số mang sang vị trí kế tiếp (bên trái): **Carry**



Hình 4.16. Bộ nửa cộng

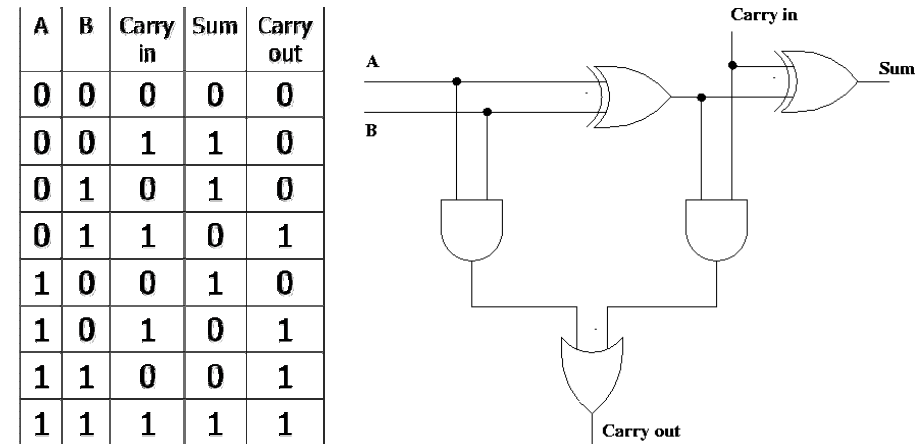
Bộ nửa cộng này chỉ cho phép ta tính tổng bit cực phải của hai từ đầu vào nhiều bit, nhưng không thực hiện được cho vị trí bit ở giữa từ vì nó không xử lý số mang từ bên phải sang vị trí này, hay nói cách khác không cộng với số nhớ trong phép cộng thông thường. Như vậy bộ nửa cộng này không thể áp dụng để thiết kế một bộ cộng cho 2 số có nhiều bit, thay vào đó, phải cần tới bộ cộng đầy đủ (full adder).

#### b) Bộ cộng đầy đủ(Full Adder)

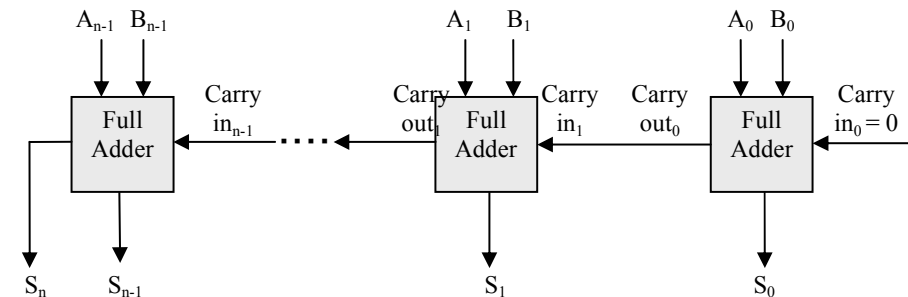
Bảng chân trị và mạch cho bộ cộng 1 bit đầy đủ trong hình 4.17. Bộ cộng đầy đủ được cấu thành từ hai bộ nửa cộng. Đầu ra **Sum** là 1 nếu số lẻ A, B, và **Carry in** bằng 1. **Carry out** bằng 1 khi cả A và B đều bằng 1(đầu vào trái của cổng OR) hoặc đúng một trong số chúng bằng 1 và bit Carry in cũng bằng 1.

Giả sử để tạo bộ cộng cho hai từ A và B, mỗi từ 16 bit, chỉ việc sao chép mạch trong hình 4.12 đúng 16 lần. Số nhớ từ bit được dùng làm số nhớ vào bit bên trái. Số nhớ vào bit cực trái được nối vào 0. Loại bộ cộng này được gọi là bộ cộng số nhớ **ripple (ripple carry adder)**. Vì trong trường hợp xấu nhất, cộng 1 vào 1 1 1 ... 1 1 1 (nhị phân), số nhớ ripple từ bit cực phải sang bit cực trái thì mới cộng xong được. Do đó trong các trường hợp như vậy thì bộ cộng này sẽ rất chậm và không hiệu quả. Cũng có bộ cộng không có sự

trễ này, và do đó nhanh hơn. Sơ đồ bộ cộng đầy đủ cho n bit như hình 4.18.



Hình 4.17. Bộ cộng đầy đủ



Hình 4.18. Bộ cộng n bit

#### 4.3.5. Mạch giải mã và mã hóa

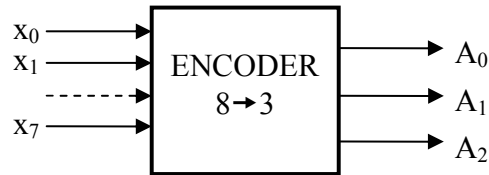
##### ➤ Khái niệm :

Mạch mã hoá (**ENCODER**) là mạch có nhiệm vụ biến đổi những ký hiệu quen thuộc với con người sang những ký hiệu không quen thuộc con người. Mạch giải mã (**DECODER**) là mạch làm

nhiệm vụ ngược lại mạch mã hóa, biến đổi những ký hiệu không quen thuộc với con người sang những ký hiệu quen thuộc với con người.

#### a) Mạch mã hoá (Encoder)

Xét mạch mã hóa nhị phân từ 8 sang 3 (8 ngõ vào và 3 ngõ ra). Sơ đồ khối của mạch được cho trên hình 4.19.



Hình 4.19. Sơ đồ khối Encoder 8→3

Trong đó :

- $x_0, x_1, \dots, x_7$  là các ngõ vào tín hiệu.
- $A_0, A_1, A_2$  là các ngõ ra.

Mạch mã hóa nhị phân 8→3 thực hiện biến đổi tín hiệu ngõ vào thành một từ mã nhị phân tương ứng ở ngõ ra, cụ thể như sau:

$0 \rightarrow 000$        $2 \rightarrow 100$        $4 \rightarrow 100$        $6 \rightarrow 110$   
 $1 \rightarrow 001$        $3 \rightarrow 011$        $5 \rightarrow 101$        $7 \rightarrow 111$

Chọn mức tác động (tích cực) ở ngõ vào là mức logic 1, ta có bảng trạng thái mô tả hoạt động của mạch như sau:

$x_7$	$x_6$	$x_5$	$x_4$	$x_3$	$x_2$	$x_1$	$x_0$	$A_2$	$A_1$	$A_0$
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

Giải thích bảng trạng thái: Khi một ngõ vào ở trạng thái cao (mức logic 1) và các ngõ vào còn lại thấp (mức logic 0) thì ngõ ra xuất hiện từ mã tương ứng. Ngõ vào nào ở trạng thái cao thì tương ứng với con số đó ở hệ thập phân, ví dụ ngõ vào 4 ở trạng thái cao sẽ tương ứng với số 4 được đưa vào ngõ nhập. Cụ thể là: khi ngõ vào  $x_0=1$  và các ngõ vào còn lại bằng 0 thì từ mã ở ngõ ra là 000, khi ngõ vào  $x_1=1$  và các ngõ vào còn lại bằng 0 thì từ mã nhị phân ở ngõ ra là 001,...

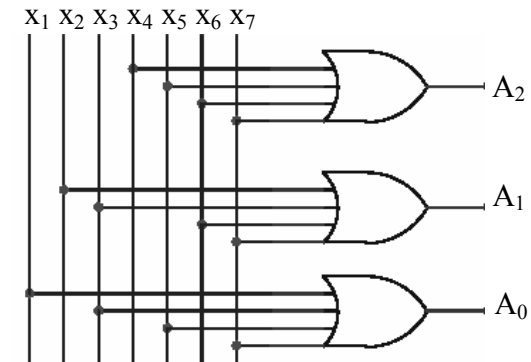
Phương trình logic tối giản:

$$A_0 = x_1 + x_3 + x_5 + x_7$$

$$A_1 = x_2 + x_3 + x_6 + x_7$$

$$A_2 = x_4 + x_5 + x_6 + x_7$$

Sơ đồ mạch của ENCODER 8→3 như hình 4.20.



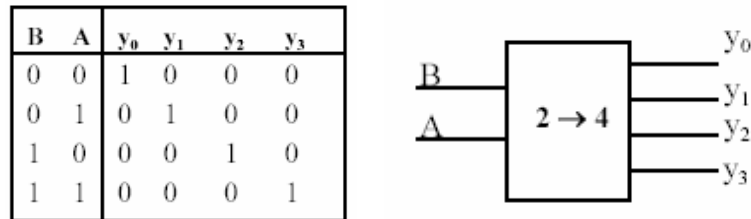
Hình 4.20. ENCODER 8→3

Tương tự ta dễ dàng có thể thiết kế mạch mã hóa thập phân, dùng mã hóa các số từ 0 đến 9 sang hệ nhị phân. Trong trường hợp này ta cần có 4 đầu ra để mã hóa được số 8(1000) và 9(1001).

**b) Mạch giải mã (Decoder)**

Ngược với mạch mã hóa, mạch giải mã là mạch tổ hợp đổi thông tin nhị phân với  $n$  ngõ nhập thành  $2^n$  ngõ xuất. Nếu ngõ nhập có một số tổ hợp không dùng thì số ngõ ra có thể ít hơn  $2^n$ . Khi đó mạch giải mã gọi là mạch giải mã  $n$ - $m$ , với  $m \leq 2^n$ .

Để đơn giản ta xét mạch giải mã 2-4 với sơ đồ khối và bảng chân trị mạch giải mã nhị phân 2→4 như hình 4.21.



Hình 4.21. Decoder 2→4

Từ bảng chân trị ta có phương trình logic tối giản cho mạch:

$$y_0 = \overline{A}\overline{B}$$

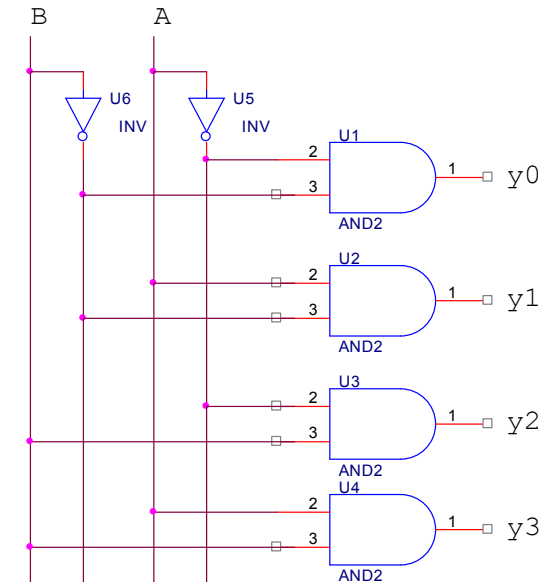
$$y_1 = A\overline{B}$$

$$y_2 = \overline{A}B$$

$$y_3 = AB$$

Sơ đồ mạch của DECODER 2→4 như hình 4.22.

Mạch giải mã được đóng gói thành các vi mạch và được bán ra trên thị trường thường có dạng 4-16,3-8 và 2-4 kép (tức hai bộ giải mã được đóng chung vào trong một vi mạch đơn). Ngoài ra còn phổ biến bộ giải mã 4-10 dùng giải mã số nhị phân sang hệ thập phân. Ngoài các ngõ nhập và xuất dữ liệu thường còn có một ngõ điều khiển hoạt động của mạch. Ngõ này thường ký hiệu là E, khi  $E = 1$ , cho phép mạch hoạt động và khi  $E = 0$  thì không cho phép mạch hoạt động.



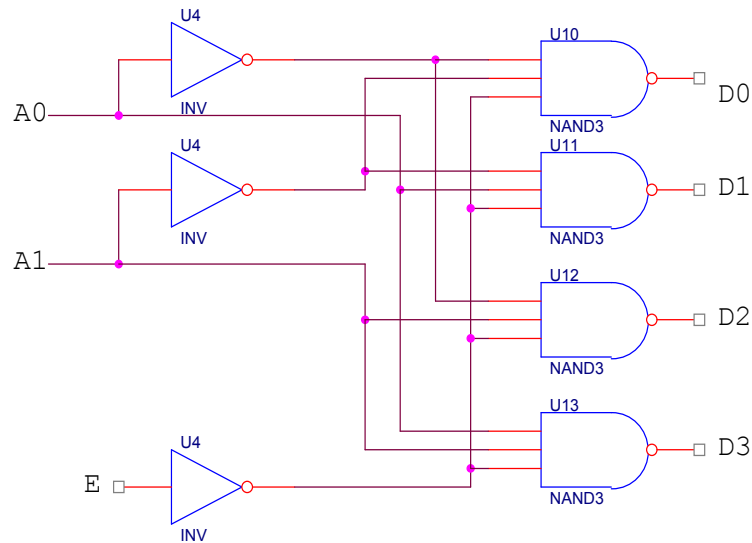
Hình 4.22. Sơ đồ mạch Decoder 2→4

**Mạch giải mã với cổng NAND**

Một số mạch giải mã được tạo từ cổng NAND thay vì cổng AND. Nó tạo ra ngõ xuất theo dạng đảo lại. Hình 4.23 là mạch giải mã 2→4 với cổng NAND với một đường vào điều khiển E. Tương ứng với nó là Bảng chân trị sau:

E	A1	A0	D0	D1	D2	D3
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0
1	x	x	1	1	1	1

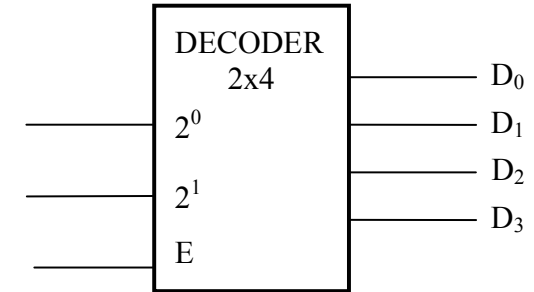




Hình 4.23. Mạch giải mã với cổng NAND

Mạch này hoạt động khi tín hiệu điều khiển  $E = 0$  và ngõ ra sẽ có giá trị 0 tương ứng với số nhị phân ở các ngõ vào. Khi  $E = 1$  thì không cho phép mạch hoạt động tức là không phụ thuộc vào các giá trị đầu vào, đầu ra luôn bằng 1.

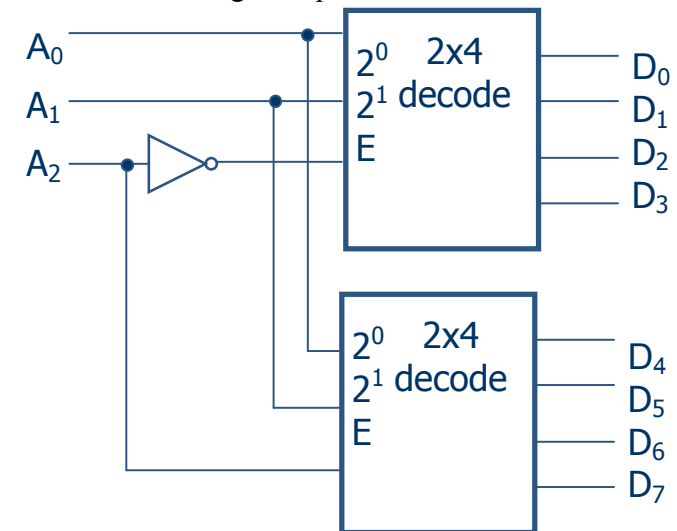
Các mạch giải mã ngoài thị trường thường được đóng gói và có ký hiệu như hình 4.24. Đó là một mạch giả mã 2→4 dùng cổng AND và với một đường điều khiển E cho phép mạch hoạt động khi  $E = 1$  và không hoạt động khi  $E = 0$ .



Hình 4.24. Ký hiệu Decoder 2→4

### Mở rộng mạch giải mã

Trong một số trường hợp cần mạch giải mã với một kính cỡ lớn mà ta lại chỉ có mạch với kích thước nhỏ hơn thì ta có thể ghép hai hoặc nhiều hơn các mạch đang có để tạo một mạch mã hóa lớn hơn. Ví dụ ta có thể tạo mạch giải mã 3→8 từ hai mạch giải mã 2→4 (hình 4.25). Trong trường hợp này ta đã tận dụng ngõ vào điều khiển E để làm ngõ nhập thứ 3.



Hình 4.25. Mạch giải mã 3→8

## CÂU HỎI VÀ BÀI TẬP CHƯƠNG IV

1. Lập bảng chân trị và vẽ sơ đồ mạch cho hàm 4 biến sau:

- a)  $x = AB + A(C + D)$
- b)  $y = (A + BC)(D + AB)$
- c)  $z = \overline{A}B + \overline{C}(A + D)$

2. Rút gọn các hàm sau dùng các định lý của Boolean algebra

- a)  $x = ACD + \overline{A}BCD$
- b)  $y = AB + A(CD + \overline{C}\overline{D})$
- c)  $z = (\overline{B}\overline{C} + \overline{A}D)(\overline{A}\overline{B} + \overline{C}\overline{D})$

3. Dùng định lý De Morgan, rút gọn biểu thức sau cho đến khi chỉ còn biến đơn đảo (một gạch trên)

$$z = \overline{(\overline{A} + C)}.(B + \overline{D})$$

4. Một nhà luận lý học lái xe vào một tiệm bán đồ ăn, ngồi trong xe ông nói: “Làm ơn cho tôi một bánh Hamburger **hoặc** xúc xích **và** khoai tây chiên”. Tiếc rằng người bán hàng còn chưa học hết lớp 6 và không biết (và không muốn biết) trong hai từ logic “hoặc” và “và” thì từ nào được ưu tiên. Anh ta cho rằng trong trường hợp này diễn giải thế nào cũng được. Trong trường hợp nào dưới đây là diễn đạt đúng đơn đặt hàng:

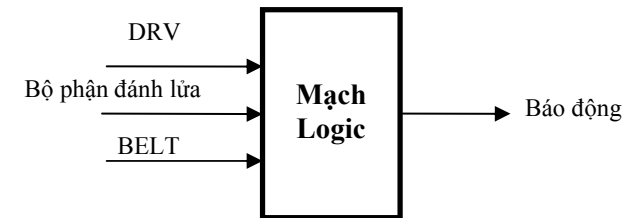
- a) Chỉ Hamburger
- b) Chỉ xúc xích
- c) Chỉ khoai tây chiên
- d) Xúc xích và khoai tây chiên
- e) Hamburger và khoai tây chiên
- f) Xúc xích và hamburger

g) Tất cả 3 thứ

h) Không có gì – nhà luận lý bị đối bụng vì quá thông minh

5. Một nhà truyền giáo lạc đường tại ngã rẽ ba ở chặng dừng Nam California. Ông ta biết hai toán đi xe máy ở khu vực này, một toán luôn nói thật và một toán luôn nói dối. Ông ta muốn biết đường nào đi tới Disneyland thì ông ta phải đặt câu hỏi như thế nào?

6. Để làm một thiết bị điều khiển báo động trong xe hơi, người ta thiết kế 1 mạch báo động như sau:



Tín hiệu:

- DRV (driver) ở mức cao khi tài xế ngồi vào ghế lái và ở mức thấp khi không ngồi vào;
- Bộ phận đánh lửa: 1 – bật, 0 – tắt;
- BELT ở mức cao khi tài xế cài dây an toàn và ở mức thấp khi không cài dây an toàn.

Hãy thiết kế mạch logic với 3 đầu vào (DRV, bộ phận đánh lửa, BELT), 1 đầu ra (báo động), sao cho bộ phận báo động sẽ hoạt động (báo động = 1) khi tồn tại một trong 2 trạng thái sau:

- Tài xế chưa ngồi vào xe trong lúc bộ phận đánh lửa bật,
- Tài xế đã ngồi vào xe nhưng chưa cài dây an toàn trong lúc bộ phận đánh lửa bật

Lập bảng chân trị của hàm ra.

7. Đơn giản các hàm sau dùng bản đồ Karnaugh

a)  $f(A, B, C) = \sum (0, 2, 3, 4, 6)$

b)  $f(A, B, C, D) = \sum (0, 1, 2, 4, 5, 7, 11, 15)$

c)  $f(X_1, X_2, X_3, X_4) = \sum (3, 7, 11, 13, 14, 15)$

d) Cực tiểu các hàm trên ở dạng tích các tổng

8. Dùng bản đồ Karnaugh rút gọn hàm

a)  $f(A, B, C, D) = \sum (0, 2, 6, 8, 9, 10, 11, 13)$ .

b)  $f(A, B, C, D) = \sum (0, 1, 2, 3, 4, 6, 7, 8, 9, 10, 11, 13)$

c)  $f(A, B, C, D) = \prod (0, 2, 3, 4, 6, 7, 9, 12, 13)$

d)  $f(A, B, C, D) = \prod (0, 2, 8, 9, 10, 11, 13, 14)$

9. Cho hàm bool

$f(A, B, C, D) = \sum (0, 1, 2, 6, 8, 9, 11, 14, 15) + d(3, 10)$ , Dùng bản đồ Karnaugh để :

- Xác định dạng chuẩn tổng các tích của hàm f (gọi là hàm g).
- Xác định dạng chuẩn tích các tổng của hàm f (gọi là hàm h).
- So sánh hai hàm g và h.
- Vẽ sơ đồ mạch hàm g mà chỉ sử dụng cổng NAND.

10. Cho hàm bool

$f(A, B, C, D) = \sum (3, 4, 5, 7, 10, 12, 13) + d(8, 9, 11)$ , Dùng bản đồ Karnaugh để :

- Xác định dạng chuẩn tổng các tích của hàm f (gọi là hàm g).
- Xác định dạng chuẩn tích các tổng của hàm f (gọi là hàm h).
- So sánh hai hàm g và h.
- Vẽ sơ đồ mạch hàm g mà chỉ sử dụng cổng NOR.

11. Cho hàm bool

$f(A, B, C, D) = \prod (0, 1, 2, 6, 8, 9, 11, 14, 15) + D(3, 10)$ , Dùng bản đồ Karnaugh để :

- Xác định dạng chuẩn tổng các tích của hàm f (gọi là hàm g).

f. Xác định dạng chuẩn tích các tổng của hàm f (gọi là hàm h).

g. So sánh hai hàm g và h.

h. Vẽ sơ đồ mạch hàm g mà chỉ sử dụng cổng NAND.

12. Đơn giản hàm Logic 4 biến

a)  $f(A, B, C, D) = ABC\bar{D} + \bar{A}BCD + \bar{A}\bar{B}\bar{C} + A\bar{C} + \bar{A}\bar{B}C + \bar{B}$

b)

$$f(A, B, C, D) = (A + B + \bar{C} + \bar{D}).(\bar{A} + C + \bar{D}).(\bar{A} + B + \bar{C} + \bar{D}).(\bar{B} + C).(\bar{B} + \bar{C}).(A + \bar{B}).(\bar{B} + \bar{D})$$

13. Mạch so sánh hai số 2 bit là mạch gồm có 4 đầu vào  $x_0, x_1, y_0, y_1$  và 2 đầu ra  $R_x, R_y$ . Trong đó,  $(x_0, x_1)$  là 2 bit của số thứ nhất và  $(y_0, y_1)$  là hai bit của số thứ 2. Đầu ra  $R_x$  có trị 1 khi  $x_1x_0 > y_1y_0$  (ngược lại có trị 0) và đầu ra  $R_y$  có trị 1 khi  $y_1y_0 > x_1x_0$  (ngược lại có trị 0)

- Lập bảng chân trị cho mạch so sánh nói trên, từ đó suy ra biểu thức chưa đơn giản của  $R_x$  và  $R_y$
- Dùng bảng đồ Karnaugh để đơn giản biểu thức của  $R_x$  và  $R_y$
- Vẽ mạch

14. Vẽ sơ đồ mạch giải mã 2-4 chỉ dùng các cổng NOR bao gồm ngõ cho phép/không cho phép hoạt động E.

15. Xây dựng lược đồ khối mạch dồn kênh 16-1 bằng lược đồ khối của hai mạch dồn 8-1 và một mạch dồn 2-1.

16. Thiết kế mạch dồn kênh 16-1 bằng 5 mạch dồn kênh 4-1. Các mạch dồn kênh dùng dưới dạng sơ đồ khối.

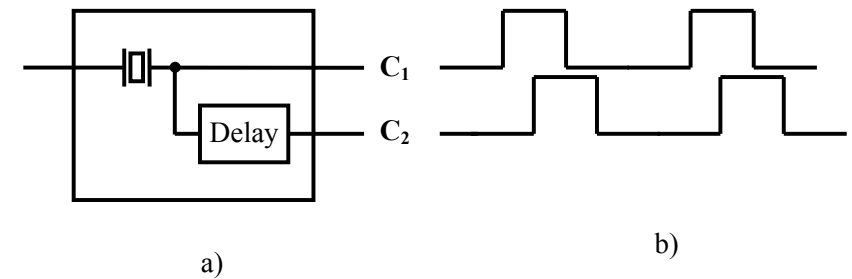
## Chương V: Mạch tuần tự

Trong chương trước chúng ta đã xem xét các mạch tổ hợp mà các ngõ ra tại một thời điểm chỉ phụ thuộc vào duy nhất các giá trị đầu vào tại thời điểm đó. Tuy nhiên phần lớn các mạch số đều hoạt động mà ở một thời điểm nhất định các ngõ ra sẽ phụ thuộc không những vào các ngõ vào ở thời điểm đó mà còn phụ thuộc vào ngõ ra ở thời điểm trước đó, hay nói cách khác một số ngõ ra của một mạch lại là chính ngõ vào của mạch đó. Những mạch như vậy chủ yếu là các thành phần lưu trữ mà ta gọi là mạch tuần tự. Chúng ta cũng biết rằng hầu hết các thiết bị số ngày nay đều có các thành phần lưu trữ, do đó trước khi tìm hiểu về bộ nhớ máy tính ta cần tìm hiểu về mạch tuần tự. Kiểu mạch tuần tự thông dụng thuộc loại đồng bộ. Mạch tuần tự đồng bộ sử dụng các tín hiệu ảnh hưởng đến các thành phần lưu trữ chỉ tại các khoảng thời gian rời rạc.

### 5.1. Xung đồng hồ

Trong nhiều mạch số, thứ tự diễn ra biến cố là vấn đề rất quan trọng. Đôi khi biến cố này phải đi trước biến cố kia, thỉnh thoảng hai biến cố phải diễn ra đồng thời. Nhằm cho phép nhà thiết kế đạt được quan hệ định thời gian cần thiết, nhiều mạch số sử dụng một ngõ vào cho xung đồng hồ. Khi đó, đồng hồ (*clock*) là mạch phát xung với độ rộng xung và thời khoảng chính xác giữa các xung liên tiếp. Thời khoảng giữa các biến tương ứng của hai xung liên tiếp là thời gian chu kỳ đồng hồ (*clock cycle time*).

Trong máy tính, nhiều biến cố xảy ra trong suốt chu kỳ đồng hồ. Giả sử biến cố phải diễn ra theo thứ tự cụ thể, thì cần chia chu kỳ đồng hồ thành những chu kỳ con. Cách đơn giản nhất để tạo ra các chu kỳ đồng hồ khác nhau là từ đồng hồ chính gắn thêm vào một bộ làm trễ (*Delay*) tín hiệu như trong hình 5.1.



Hình 5.1. Đồng hồ và các xung nó tạo ra

Trong hình 5.1 a) là Đồng hồ (clock) hay bộ phát tần (*impulse generator*), nhờ có bộ làm trễ Delay mà ta có 2 tín hiệu xung  $C_1$  và  $C_2$  khác nhau, từ đó tạo ra 4 thời điểm khác nhau là:

1. Biên lên của  $C_1$
2. Biên xuống của  $C_1$
3. Biên lên của  $C_2$
4. Biên xuống của  $C_2$

Ta đã biết rằng các mạch số hoạt động ở các mức cao và thấp, do đó các thời điểm khác nhau có thể được gắn với các biên của xung đồng hồ. Từ đó ta có thể điều khiển được tại thời điểm nào thì cho phép hay kích thích mạch nào đó hoạt động, và tại thời điểm nào thì không.

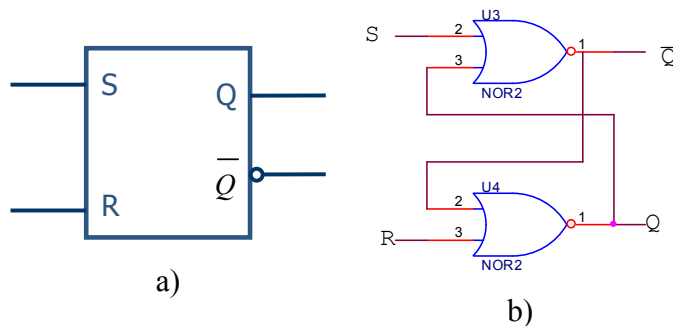
### 5.2. Mạch Lật (chốt – latch)

Mạch lật hay một số sách gọi là chốt, là dạng mạch tuần tự đơn giản nhất có chức năng lưu trữ một bit nhị phân. Nó có hai ngõ ra, một cho trị bình thường và một cho trị bù. Mạch lật đồng bộ duy trì trạng thái nhị phân cho đến khi có một xung đồng hồ điều khiển làm đổi trạng thái. Sự khác nhau giữa các loại mạch lật ở chỗ số ngõ vào chúng có và cách thức các ngõ vào tác động đến trạng thái nhị phân. Các loại mạch lật thông dụng nhất như trình bày dưới đây.

### 5.2.1. Mạch lật SR (SR-latch)

Để tạo ra bộ nhớ 1 bit cần có một mạch điện có khả năng lưu trữ một giá trị nào đó được nhập vào. Một mạch như vậy có thể được xây dựng từ cổng NAND hoặc NOR mà ta gọi là mạch lật. Mạch lật đầu tiên đưa ra xem xét là mạch lật SR. Đầu tiên ta xét mạch lật SR không đồng bộ hay không dùng xung đồng hồ điều khiển. Ký hiệu mạch lật SR không đồng bộ (không dùng xung đồng hồ) dùng cổng NOR như ở hình 5.2 (a) và hình 5.2 (b) là sơ đồ mạch tương ứng của nó.

Nó có 2 ngõ vào, S (Setting- đặt) và R (Resetting - Khởi động). Nó có một ngõ ra Q và đôi khi có ngõ ra bù, ký hiệu bằng một vòng tròn nhỏ. Đầu ra Q từ cổng NOR thứ nhất (ký hiệu U3) sẽ lại được cho vào ngõ nhập của cổng NOR thứ hai (ký hiệu U4) và ngược lại đầu ra  $\bar{Q}$  từ U4 sẽ lại được cho vào ngõ nhập của cổng NOR (cổng U3).



Hình 5.2. Mạch lật SR không đồng bộ

Ta thử xem hoạt động của mạch như sau:

- Giải sử Q đang ở trạng thái 0 ( $Q=0$ ,  $\bar{Q}=1$ ), cho tín hiệu vào  $S=R=0$ , như vậy đầu ra của U3 sẽ là:

$$\bar{Q} = \overline{Q + S} = \overline{0 + 0} = 1 ,$$

và đầu ra của U4 sẽ là :

$$Q = \overline{\bar{Q} + R} = \overline{1 + 0} = 0 \Rightarrow Q \text{ không đổi}$$

- Giải sử Q đang ở trạng thái 1 ( $Q=1$ ,  $\bar{Q}=0$ ), cho tín hiệu vào  $S=R=0$ , như vậy đầu ra của U3 sẽ là:

$$\bar{Q} = \overline{Q + S} = \overline{1 + 0} = 0 ,$$

và đầu ra của U4 sẽ là :

$$Q = \overline{\bar{Q} + R} = \overline{0 + 0} = 1 \Rightarrow Q \text{ không đổi}$$

**Như vậy trong trường hợp  $S=R=0$  thì giá trị đầu ra của mạch là không thay đổi và mạch đóng vai trò như một bộ nhớ một bit.**

Lập luận tương tự như trường hợp trên, ta có các trường hợp sau :

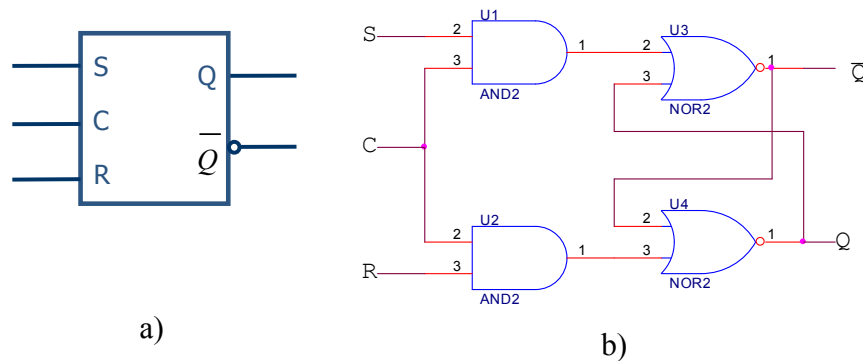
- Cho  $S=0$ ,  $R=1$ . Nếu  $Q=0$  thì dẫn tới trạng thái kế tiếp  $Q=0$ , còn nếu  $Q=1$  thì dẫn tới trạng thái kế  $Q=0 \Rightarrow Q$  **luôn bằng 0** mà không phụ thuộc vào trạng thái Q trước đó. Trạng thái này dùng để nhập giá trị 0 vào ô nhớ.
- Cho  $S=1$ ,  $R=0$ . Nếu  $Q=0$  thì dẫn tới trạng thái kế tiếp  $Q=1$ , còn nếu  $Q=1$  thì dẫn tới trạng thái kế  $Q=1 \Rightarrow Q$  **luôn bằng 1** mà không phụ thuộc vào trạng thái Q trước đó. Trạng thái này dùng để nhập giá trị 1 vào ô nhớ
- Trong trường hợp  $S=R=1$  thì trạng thái của mạch không xác định, do đó tổ hợp này bị cấm sử dụng trong các mạch SR.

$\Rightarrow$  Hoạt động của mạch lật SR được thể hiện qua bảng trạng thái 5.1.

S	R	Q(t+1)	
0	0	Q(t)	No change
0	1	0	Clear to 0
1	0	1	Set to 1
1	1	X	Indeterminate

Bảng 5.1. Bảng trạng thái của mạch lật SR

Đối với mỗi mạch lật thì bao giờ cũng có 2 loại, không đồng bộ và mạch lật đồng bộ, nhưng trên thực tế thì người ta chủ yếu dùng mạch đồng bộ, do đó ở đây chúng ta cũng sẽ tìm hiểu kỹ hơn về loại này. Mạch lật SR đồng bộ (dùng xung đồng hồ) như ở hình 5.3 (a), hình 5.3 (b) là sơ đồ mạch của mạch lật này. Nó có ba ngõ vào, S (Setting- đặt), R (Resetting - Khởi động) và C (Clock- đồng hồ). Nó có một ngõ ra Q và đôi khi có ngõ ra bù, ký hiệu bằng một vòng tròn nhỏ.



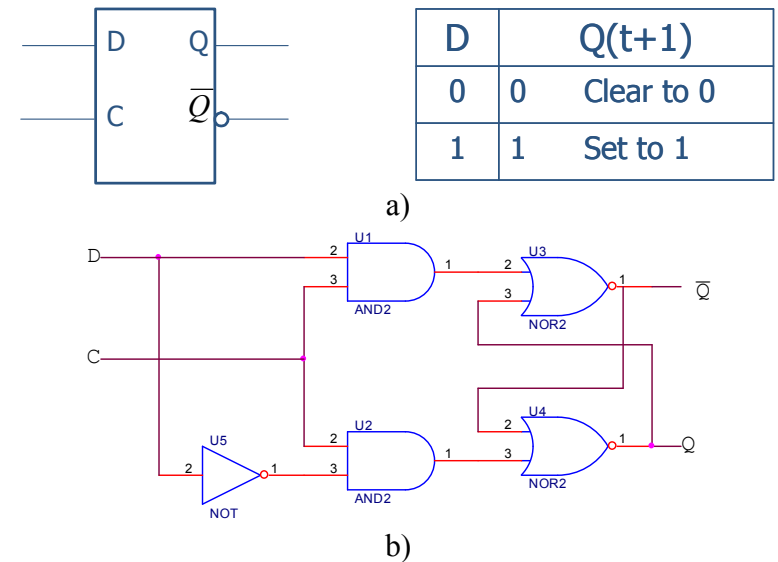
Hình 5.3. Mạch lật SR

Hoạt động mạch lật SR như sau: Nếu không có tín hiệu nhập đồng hồ C ( $C=0$ ), ngõ ra của mạch không thể thay đổi bất chấp trị của R và S vì đầu ra của 2 cổng U1 và U2 luôn bằng 0 (0 AND số bất kỳ = 0). Chỉ khi tín hiệu đồng hồ  $C=1$ , ngõ ra mới bị ảnh hưởng theo trị của ngõ vào S và R. Nếu  $S=1$ ,  $R=0$ , Q chuyển sang 1. Nếu  $S=0$ ,  $R=1$  Q chuyển sang 0. Nếu S và R là 0 khi đồng hồ chuyển, ngõ ra không đổi. Khi cả S và R là 1, ngõ ra không xác định, có thể là 0 hoặc 1 tùy thuộc vào khoảng thời gian trì hoãn trong mạch. Hay nói cách khác khi C luôn bằng 1 thì mạch lật SR đồng bộ (hình 5.3) hoạt động như mạch lật SR không đồng bộ (hình 5.2) ở trên.

### 5.2.2. Mạch lật D

Mạch lật D (Data) là loại mạch lật đơn giản nhất, nó chỉ hơi khác mạch lật SR. Mạch lật SR được đổi sang mạch lật D bằng cách đưa vào một cổng đảo giữa S và R và dùng ký hiệu D cho ngõ vào duy nhất (xem hình 5.4 b). Khi  $D=1$ , ngõ ra là 1, khi  $D=0$ , ngõ ra là 0.

Hình 5.4(a) cho ta thấy qui ước ký hiệu và bảng đặc tính của mạch lật D. Hình 5.4(b) là sơ đồ của mạch lật này. Chú ý là trạng thái kế  $Q(t+1)$  được xác định từ ngõ vào D. Mỗi quan hệ có thể biểu diễn bằng phương trình đặc tính:  $Q(t+1) = D$ . Điều này có nghĩa ngõ ra Q của mạch lật nhận trị từ ngõ vào D khi tín hiệu đồng hồ bằng 1.



Hình 5.4. Mạch lật D

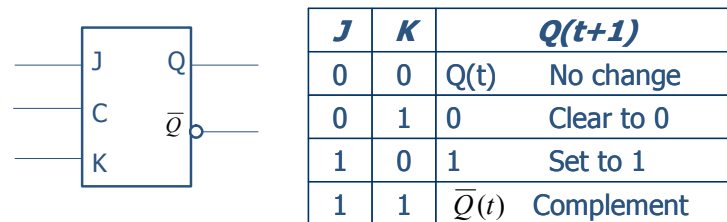
Lưu ý là không có điều kiện nhập để giữ trạng thái của mạch lật D. Tuy mạch lật D thuận tiện là chỉ có một ngõ vào nhưng bất tiện là không có điều kiện không đổi  $Q(t+1) = Q(t)$ . Điều kiện không đổi có thể lấy bằng cách vô hiệu tín hiệu đồng hồ hoặc cho

ngõ ra trở lại ngõ vào, lúc đó xung đồng hồ sẽ giữ trạng thái mạch lật không đổi.

### 5.2.3. Mạch lật JK

Một mạch lật khác thường hay được sử dụng là mạch lật JK, là một cải tiến của mạch lật SR trong đó điều kiện không xác định của SR được định nghĩa trong JK. Ngõ vào J, K hoạt động giống như S, R để đặt và xóa mạch lật. Khi J và K đều bằng 1, khi đồng hồ C = 1 sẽ chuyển ngõ ra mạch lật sang trạng thái bù.

Ký hiệu và bảng đặc tính mạch lật JK ở hình 5.5. J tương đương với S trong SR và K tương đương với R.

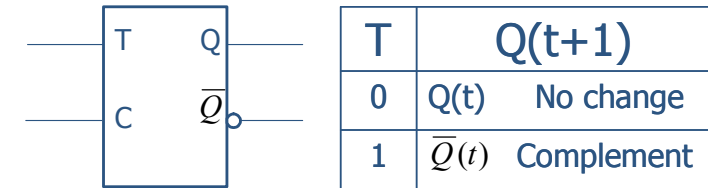


Hình 5.5. Mạch lật JK

Điểm khác biệt lớn nhất ở đây là thay vì không xác định, mạch lật JK có điều kiện bù Q(t+1) khi J=K=1. Trong một mạch số thì tồn tại một trạng thái không xác định là điều không mong muốn, chính do điều đó mà mạch lật JK được sử dụng nhiều hơn.

### 5.2.4. Mạch lật T

Mạch lật cuối cùng là kế thừa của mạch lật JK bằng cách kết nối hai ngõ vào với nhau thành một ngõ vào T. Hình 5.6, là ký hiệu và bảng trạng thái mạch. Xuất phát từ mạch lật JK với hai ngõ vào được kết nối thành một ngõ vào T. Vì vậy mạch lật T chỉ có hai điều kiện. Khi T=0 (J=K=0), với mọi giá trị của C không thay đổi trạng thái của mạch lật. Khi T=1 (J=K=1), và khi C = 1 sẽ làm bù trạng thái mạch lật. Các điều kiện này có thể biểu diễn bằng phương trình thuộc tính:  $Q(t+1) = Q(t) \oplus T$ .



Hình 5.6. Mạch lật T

## 5.3. Mạch lật lề (Flip-flop)

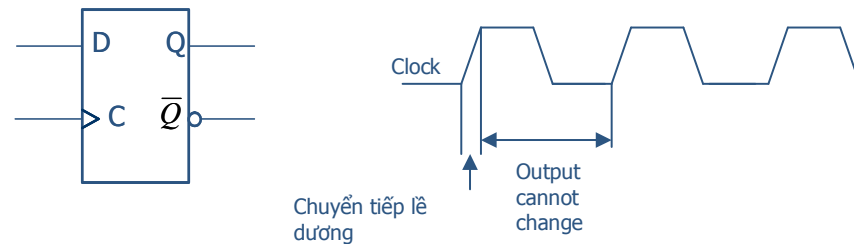
Các loại mạch lật nói trên thực sự chỉ là một trong hai loại mạch lật lề. Đây là loại mạch lật thông dụng nhất để đồng bộ việc thay đổi trạng thái trong một chuyển tiếp xung đồng hồ. Trong loại mạch lật này, các chuyển tiếp xuất xảy ra tại một mức xung đồng hồ xác định. Khi mức nhập xung vượt quá ngưỡng này, các ngõ nhập bị khóa lại sao cho đến khi xung đồng hồ trở về 0 và một xung khác đến. Một số mạch lật lề tạo chuyển tiếp ở lề lên của tín hiệu đồng hồ (chuyển tiếp lề dương – positive-edge transition) và một số khác tạo chuyển tiếp ở lề xuống của tín hiệu đồng hồ (chuyển tiếp lề âm – negative-edge transition)

**Điểm khác biệt giữa các mạch lật và mạch lật lề là ở chỗ mạch lật kích thích bằng mức (level triggered), còn mạch lật lề kích thích bằng biên (edge triggered).** Ngoài ra ở mạch lật lề còn có một ký hiệu mũi tên trước chữ C biểu thị một ngõ nhập động (xem hình 5.7. Ký hiệu chỉ báo động cho biết mạch lật lề thay đổi trạng thái với một chuyển tiếp dương (từ 0 sang 1) của tín hiệu đồng hồ ở ngõ nhập.

Hình 5.7 cho thấy tín hiệu xung đồng hồ trong mạch lật D lề dương. Trị ở ngõ nhập D chuyển sang ngõ xuất Q khi đồng hồ tạo chuyển tiếp dương. Ngõ xuất không thể thay đổi khi đồng hồ ở mức 1, mức 0 hoặc trong chuyển tiếp từ mức 1 xuống 0. Chuyển tiếp đồng hồ dương có hiệu lực bao gồm một thời gian tối thiểu gọi là thời định (setup time) trong đó ngõ nhập D phải duy trì một hằng trị trước khi chuyển tiếp và một thời gian hữu hạn gọi là thời lưu (hold time) trong đó ngõ nhập D không được thay đổi sau chuyển

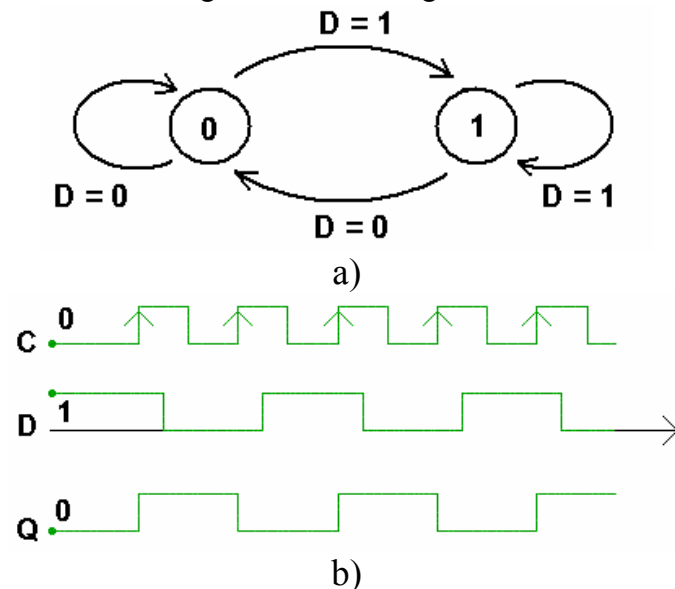


tiếp dương. Chuyển tiếp dương có hiệu lực thường là một phần rất nhỏ trong tổng chu kỳ xung đồng hồ.



Hình 5.7. Flip-flop D với chuyển tiếp dương

Thường đối với các flip-flop ngoài cách dùng bảng trạng thái người ta còn hay dùng biểu đồ trạng thái như trong hình 5.8 (a) và đồ thị miêu tả hoạt động của mạch ở dạng tín hiệu ở hình 5.8(b).

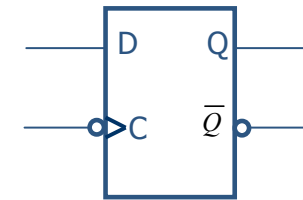


Hình 5.8. Biểu đồ trạng thái và đồ thị của flip-flop D

Biểu đồ trạng thái (hình 5.8a) cho ta cái nhìn khái quát và dễ hiểu hơn của việc chuyển đổi các trạng thái mạch. Khi đang ở trạng thái 0 (vòng tròn có số 0), nếu  $D=0$  thì trạng thái không đổi (mũi tên quay trở lại chính nó); khi  $D=1$  thì trạng thái chuyển qua

trạng thái mới 1 (vòng tròn có số 1). Tương tự như vậy khi đang ở trạng thái 1, nếu  $D=1$  thì không đổi, nếu  $D=0$  thì chuyển trạng thái. Đồ thị biểu diễn ở hình 5.8b cũng cho ta thấy các thay đổi này.

Trong trường hợp mạch lật lẽ D được kích hoạt ở chuyển tiếp âm ta có ký hiệu như hình 5.8. Ký hiệu chỉ khác chỗ ngõ vào của clock có thêm một ô tròn. Trong trường hợp này đầu ra của mạch chỉ thay đổi ở chuyển tiếp 1 xuống 0.



Hình 5.8. Flip-flop D với chuyển tiếp âm

### Bảng kích thích

Để thiết kế mạch tuần tự chúng ta thường biết việc chuyển tiếp từ trạng thái này sang một trạng thái khác và muốn tìm các điều kiện nhập của mạch lật để tạo ra chuyển tiếp đó. Như vậy, để mô tả hoạt động của các mạch lật lẽ chúng ta cần một bảng liệt kê các tổ hợp nhập cần có để tạo ra một thay đổi trạng thái yêu cầu. Bảng này ta gọi là bảng kích thích mạch lật lẽ.

Trong bảng 5.2. cho ta thấy hoạt động của bốn loại mạch lật lẽ D, SR, JK và T. Mỗi bảng gồm các cột :

- $Q(t)$  – cho giá trị mạch ở thời điểm  $t$
- $Q(t+1)$  – cho giá trị mạch ở thời điểm sau đó  $t+1$
- Các cột cho mỗi ngõ vào.

Bảng này cho thấy ứng với các ngõ vào trạng thái của mạch sẽ được chuyển tiếp ra sao. Có bốn khả năng chuyển tiếp từ trạng thái hiện hành  $Q(t)$  sang trạng thái kế  $Q(t+1)$ . Các điều kiện nhập cho mỗi một chuyển tiếp này xuất phát từ thông tin trong bảng đặc tính. Ký hiệu x trong bảng biểu diễn một điều kiện không cần

(don't care condition) hoặc tùy chọn; tức là 0 hoặc 1 đều không ảnh hưởng.

Mạch lật SR			
Q(t)	Q(t+1)	S	R
0	0	0	x
0	1	1	0
1	0	0	1
1	1	x	0

Mạch lật D		
Q(t)	Q(t+1)	D
0	0	0
0	1	1
1	0	0
1	1	1

Mạch lật JK			
Q(t)	Q(t+1)	J	K
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

Mạch lật T		
Q(t)	Q(t+1)	T
0	0	0
0	1	1
1	0	1
1	1	0

Bảng 5.2 Bảng kích thích của bốn mạch lật lẹ

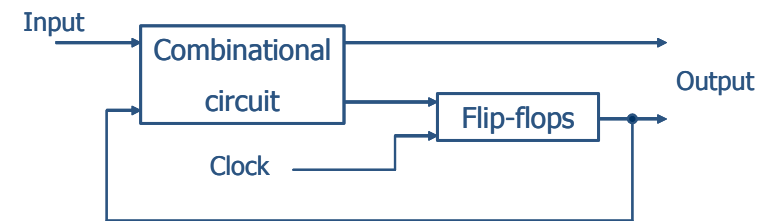
Lý do có điều kiện không cần trong bảng kích thích vì có hai cách lấy chuyển tiếp. Ví dụ, trong mạch lật JK, một chuyển tiếp từ trạng thái hiện hành 0 sang trạng thái kế 0 có thể nhận được bằng cách cho  $J=K=0$  (không đổi) hoặc  $J=0, K=1$  để xóa mạch lật (dù đã xóa rồi). Trong cả hai trường hợp J phải là 0, nhưng K là 0 ở trường hợp một và 1 ở trường hợp hai. Vì chuyển tiếp yêu cầu xuất hiện trong cả hai trường hợp, chúng ta ghi K là x.

#### 5.4. Mạch tuần tự.

Sơ đồ khối mạch tuần tự được minh họa có đồng hồ được minh họa như trong hình 5.9. Từ sơ đồ ta thấy mạch tuần tự là một kết nối các mạch lật với một mạch tổ hợp khác, mà mạch tổ hợp này lại được tạo ra từ các cổng cơ bản. Bản thân các cổng tạo thành mạch tổ hợp, nhưng khi gộp vào các mạch lật toàn bộ mạch được sắp vào loại mạch tuần tự. Nó gồm một mạch tổ hợp và một số mạch lật có đồng hồ. Như trong lược đồ, khối mạch tổ hợp nhận

các tín hiệu nhị phân từ các ngõ nhập ngoài và từ các ngõ ra của mạch lật. Ngõ ra mạch tổ hợp đi ra ngoài (gọi là xuất ngoài) và đi vào mạch lật.

Các cổng trong mạch tổ hợp xác định trị nhị phân lưu vào mạch lật sau mỗi chuyển tiếp đồng hồ. Đến phiên các ngõ ra của mạch lật được đưa vào mạch tổ hợp và xác định hành vi của mạch. Hơn nữa, trạng thái kế của mạch lật cũng là hàm của trạng thái hiện tại và các ngõ nhập ngoài. Như vậy mạch tuần tự được xác định bởi các ngõ nhập ngoài, các ngõ xuất ngoài và trạng thái nhị phân của mạch lật.



Hình 5.9. Sơ đồ khối mạch tuần tự

**Qui trình thiết kế mạch tuần tự** được thể hiện qua các bước sau:

- Bước 1: Chuyển đặc tả mạch sang lược đồ trạng thái
- Bước 2: lược đồ trạng thái => bảng trạng thái
- Bước 3: Từ bảng trạng thái viết hàm cho các ngõ nhập của Flip-flops
- Bước 4: vẽ sơ đồ mạch

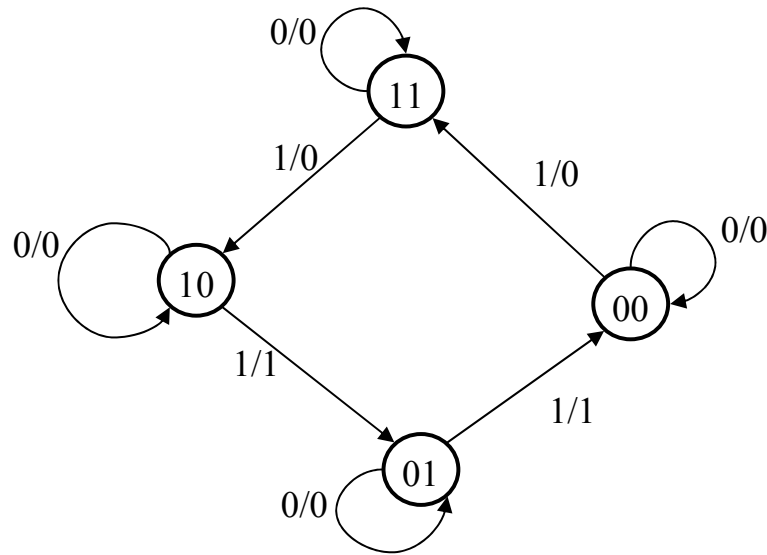
Để hiểu rõ mạch tuần tự và cách thiết kế nó, ta sẽ bắt đầu bằng việc xem xét một ví dụ đơn giản sau

**Ví dụ:** Thiết kế mạch tuần tự dùng mạch lật SR. Khi ngõ nhập  $x=0$ , trạng thái mạch lật lẹ không thay đổi, ngõ xuất  $y=0$ . Khi  $x=1$ , dãy trạng thái là 11,10,01,00 và lặp lại còn ngõ xuất  $y$  sẽ có giá trị là 1 khi số bit trạng thái mạch lật lẹ bằng 1 là lẻ, các trường hợp còn lại thì bằng 0.

**Giải:**

**Bước 1:**

Từ yêu cầu của ví dụ ta xây dựng lược đồ trạng thái của mạch. Theo đề bài ta có 4 trạng thái là  $11 \rightarrow 10 \rightarrow 01 \rightarrow 00$ , như vậy sẽ có 4 vòng tròn để biểu diễn 4 trạng thái này. Lược đồ trạng thái của mạch như hình 5.10.



Hình 5.10. Lược đồ trạng thái mạch

Trong lược đồ này mỗi trạng thái biểu thị bằng một vòng tròn nhỏ với trạng thái được chỉ bên trong vòng tròn và các đường dẫn trực tiếp nối các trạng thái chỉ cho biết hướng chuyển tiếp giữa các trạng thái. Trên mỗi đường có mũi tên định hướng và trên đó ghi giá trị của biến đầu vào mạch  $x$  và biến đầu ra  $y$ . Giá trị của hai biến này được cách nhau bởi dấu “/”. Ví dụ như nếu ta đang ở trạng thái 11, và nếu tín hiệu đầu vào là 0 ( $x=0$ ) thì trạng thái sẽ không thay đổi (đường có mũi tên xuất phát từ nó và quay về chính nó) và đầu ra  $y=0$ . Như vậy trên đường mũi tên ta ghi 0/0 tức là  $x/y$  hay  $x=0$  và  $y=0$ . Nếu tín hiệu vào  $x=1$  thì trạng thái sẽ chuyển sang một trạng thái mới là 10 (mũi tên chỉ đến trạng thái 10) và vì số bit của các flip-flop có giá trị bằng 1 là 2, tức là chẵn cho nên đầu ra  $y=0$

$\Rightarrow$  trên đường chuyển trạng thái ta ghi 1/0. Ở đây ta lưu ý là trạng thái mạch lật lể chỉ thay đổi ở chuyển tiếp của xung đồng hồ, nhưng trong khi lập luận để cho đơn giản và đỡ nhầm lẫn ta không đề cập đến nó.

### Bước 2:

Từ lược đồ trạng thái này ta sẽ xây dựng một **bảng trạng thái**. Theo yêu cầu dùng flip-flop SR, mà ta đã biết mỗi flip-flop cho phép ta nhớ 1 bit. Vậy ở đây ta có 4 trạng thái, do đó cần có hai flip-flop để mã hóa chúng. Ta gọi 2 flip-flop đó là A và B, và các đầu vào của chúng tương ứng sẽ là  $S_A, R_A, S_B, R_B$ . Từ sơ đồ khối mạch tuần tự hình 5.9, chúng ta cũng biết được đầu ra của các flip-flop cũng chính là đầu vào của mạch. Như vậy bảng trạng thái sẽ có 3 đầu vào là A, B và  $x \Rightarrow$  có  $2^3=8$  tổ hợp. Ngoài ra trong bảng trạng thái ta còn có một đầu ra  $y$ . Các trạng thái của mạch được biểu hiện trong bảng 5.3. Dựa vào bảng trạng thái của mạch lật lể ở bảng 5.2 ta sẽ tìm ra các giá trị trong bảng 5.3. Ba cột đầu tiên là giá trị nhập vào, do đó ta chỉ cần điền các giá trị sao cho thể hiện tất cả các tổ hợp có thể có. Để tránh nhầm lẫn ta điền theo thứ tự từ  $000 \rightarrow 001 \rightarrow 010 \rightarrow \dots$

### Xác định giá trị trong cột “trạng thái kế”

Tiếp theo đến cột trạng thái kế, từ lược đồ trạng thái ta có nếu đang ở trạng thái 00 ( $AB=00$ ) thì khi  $x=0$  trạng thái sẽ không thay đổi, do đó ở hàng đầu tiên  $A=0, B=0$ ; ở hàng tiếp theo khi  $x=1$  thì trạng thái sẽ thay đổi từ  $00 \rightarrow 11 \Rightarrow A=1, B=1$ . Tương tự, ta sẽ có được giá trị cho tất cả các hàng ở cột trạng thái kế.

### Xác định các ngõ nhập vào các flip-flop

Cũng dựa vào bảng trạng thái của flip-flop SR ta sẽ xác định được các giá trị của các cột còn lại. Chẳng hạn như đối với flip-flop A, khi A chuyển từ 0 sang 0 thì  $S_A=0, R_A=x$ ; khi A chuyển từ 0 sang 1 thì  $S_A=1, R_A=0; \dots$

Trạng thái hiện tại		Nhập	Trạng thái kế		Ngõ nhập vào các flip-flop				Đầu ra
A	B	x	A	B	S <sub>A</sub>	R <sub>A</sub>	S <sub>B</sub>	R <sub>B</sub>	y
0	0	0	0	0	0	x	0	x	0
0	0	1	1	1	1	0	1	0	0
0	1	0	0	1	0	x	x	0	0
0	1	1	0	0	0	x	0	1	1
1	0	0	1	0	x	0	0	x	0
1	0	1	0	1	0	1	1	0	1
1	1	0	1	1	x	0	x	0	0
1	1	1	1	0	x	0	0	1	0

Bảng 5.3. Bảng trạng thái mạch tuần tự

**Bước 3:**

Từ bảng trạng thái 5.3 ta tìm hàm cho các ngõ nhập vào các flip-flop. Chú ý là các hàm tìm được phải là ngắn gọn nhất, để làm được việc đó ta phải dùng bản đồ Karnaugh để rút gọn chúng. Trước hết từ bảng 5.3  $\Rightarrow$  bản đồ Karnaugh cho đầu vào S<sub>A</sub> như sau :

		Bx			
A		00	01	11	10
	0		1		
1		x		x	x

Từ bản đồ này ta suy ra  $\Rightarrow S_A = \bar{A}\bar{B}x$

Tương tự như vậy cho các cổng còn lại, ta có bản đồ Karnaugh cho R<sub>A</sub> :

		Bx			
A		00	01	11	10
	0	x		x	x
1			1		

$\Rightarrow R_A = A\bar{B}x$

Đối với flip-flop B ta có :

		Bx			
A		00	01	11	10
	0		1		x
1			1		x

$\Rightarrow S_B = \bar{B}x$

		Bx			
A		00	01	11	10
	0	x		1	
1		x		1	

$\Rightarrow R_B = Bx$

Bản đồ cho đầu ra y :

		Bx			
A		00	01	11	10
	0			1	
1			1		

Từ bản đồ này ta suy ra  $\Rightarrow y = \bar{A}Bx + A\bar{B}x$

**Phương trình nhập mạch lật**

Như vậy trong bước 3 này ta đã tìm ra được hàm hay phương trình nhập cho các mạch flip-flop như sau:



trạng thái hiện hành. Phần trạng thái kế cho thấy trạng thái của mạch lật tại một chu kỳ sau đó là thời điểm  $t+1$ . Phần xuất cho trị của  $y$  với mỗi trạng thái hiện hành và điều kiện nhập.

- Lược đồ trạng thái và bảng trạng thái để mô tả hoạt động của mạch tuần tự. Có lược đồ trạng thái thì ta có thể suy ra bảng trạng thái và ngược lại.

## Bài tập chương V

1. Hãy chứng minh rằng JK flip-flop có thể chuyển sang D flip-flop với một cổng đảo đặt giữa các ngõ nhập J và K

2. Thiết kế mạch tuần tự dùng mạch lật JK. Khi ngõ nhập  $x=0$ , trạng thái mạch lật không thay đổi. Khi  $x=1$ , dãy trạng thái là 11,01,10,00 và lặp lại.

3. Một mạch tuần tự gồm 2 D flip-flop A và B, 2 ngõ nhập  $x, y$  một ngõ xuất  $z$ . Phương trình các ngõ nhập vào các flip-flop và ngõ xuất mạch như sau:

$$D_A = \bar{x}y + xA$$

$$D_B = \bar{x}B + xA$$

$$Z = B$$

- Vẽ lược đồ luận lý của mạch
- Lập bảng trạng thái.

4. Thiết kế mạch đếm nhị phân 2-bit là một mạch tuần tự có đồng hồ đi qua một dãy trạng thái nhị phân 00, 01, 10, 11 và lặp lại khi ngõ nhập ngoài  $x$  có trị 1. Trạng thái mạch không đổi khi  $x = 0$ .

5. Thiết kế mạch đếm giảm 2 bit. Đây là mạch tuần tự có 2 flip-flop và 1 ngõ nhập  $x$ . Khi  $x=0$ , trạng thái mạch lật không đổi. Khi  $x=1$ , dãy trạng thái là 11, 10, 01, 00 và lặp lại.

6. Thiết kế mạch tuần tự có 2 mạch lật JK, A và B và 2 ngõ vào E và  $x$ . Nếu  $E=0$  mạch giữ nguyên trạng thái bất chấp  $x$ . Khi  $E=1$  và  $x=1$  mạch chuyển trạng thái từ 00 sang 01 sang 10 sang 11 về 00 và lặp lại. (ở đây E-Enable giống như cổng điều khiển cho phép mạch hoạt động hay không)

7. Thiết kế mạch tuần tự dùng mạch lật T. Khi ngõ nhập  $x=0$ , trạng thái mạch lật không thay đổi. Khi  $x=1$ , dãy trạng thái là 00,10,01,11 và lặp lại.

## Chương VI: Kiến trúc bộ lệnh

Trong chương này chúng ta sẽ tập trung vào kiến trúc bộ lệnh của máy tính, giới thiệu các trường hợp khác nhau của các kiểu kiến trúc bộ lệnh. Đặc biệt trong chương này sẽ tập trung vào bốn chủ đề chính. Đầu tiên là phân loại các kiểu kiến trúc bộ lệnh và đánh giá những ưu khuyết điểm của chúng

### 6.1. Phân loại kiến trúc bộ lệnh

Có ba loại kiến trúc bộ lệnh cơ bản: kiến trúc ngăn xếp (stack), kiến trúc thanh ghi tích lũy (Accumulator) và kiến trúc thanh ghi đa dụng GPRA (general-purpose register architecture). Trong đó kiến trúc GPRA lại chia làm hai loại thông dụng là thanh ghi – bộ nhớ (register-memory) và nạp-lưu (load-store). Ví dụ phép tính  $C = A + B$  được dùng trong các kiểu kiến trúc trong hình 6.1.

Stack	Accumulator	Register (register-memory)	Register (load-store)
Push A	Load A	Load R1,A	Load R1,A
Push B	Add B	Add R1,B	Load R2,B
Add	Store C	Store C,R1	Add R3,R1,R2
Pop C			Store C,R3

Hình 6.1. Thực hiện lệnh  $C = A + B$  cho 4 kiểu kiến trúc bộ lệnh

Trong một câu lệnh thì chúng ta có các toán hạng, mà các toán hạng lại được chia thành hai loại: ẩn (implicitly) và hiện (explicitly): Toán hạng trong kiến trúc ngăn xếp là loại ẩn ở trên đầu của ngăn xếp, kiến trúc thanh ghi tích lũy có một toán hạng ẩn ở trong accumulator, còn kiến trúc GPRA thì chỉ dùng toán hạng hiện, hoặc là thanh ghi hoặc là trên bộ nhớ. Toán hạng dạng hiện có thể truy cập trực tiếp từ bộ nhớ hoặc đầu tiên được nạp vào thanh ghi tạm thời nào đó phụ thuộc vào kiểu kiến trúc bộ lệnh đặc trưng của nó.

Kiến trúc GPRA có một loại có thể truy cập trực tiếp vào bộ nhớ (register-memory); một loại có thể truy cập vào bộ nhớ nhưng phải nạp thông qua một register gọi là kiến trúc nạp-lưu (load-store) hay kiến trúc trên các thanh ghi (register-register); ngoài ra còn một kiểu kiến trúc GPRA nữa là kiến trúc bộ nhớ-bộ nhớ (memory-memory), nhưng kiểu này không còn thấy ngoài thị trường ngày nay nữa.

Những máy tính ra đời trước kia thường dùng kiểu kiến trúc ngăn xếp hoặc kiến trúc thanh ghi tích lũy, nhưng từ sau năm 1980 thì đều dùng kiến trúc GPR. Hiện tại các nhà sản xuất máy tính có khuynh hướng dùng kiến trúc phần mềm thanh ghi đa dụng vì việc thâm nhập các thanh ghi đa dụng nhanh hơn thâm nhập bộ nhớ trong, và vì các chương trình dịch dùng các thanh ghi đa dụng có hiệu quả hơn. Bảng 6.1 cho ta thấy các ưu và nhược điểm của ba loại kiến trúc cơ bản này.

Như vậy kiểu kiến trúc GPR được dùng ngày nay bởi các yếu tố sau:

- Dùng thanh ghi, một dạng lưu trữ trong của CPU có tốc độ nhanh hơn bộ nhớ ngoài
- Dùng thanh ghi thì dễ dàng cho trình biên dịch và có thể dùng hiệu quả hơn là bộ nhớ ngoài. Ví dụ biểu thức  $(A*B) + (C*D) - (E*F)$  có thể tính bằng cách nhân các phần trong ngoặc ở mọi thứ tự, và khi ứng dụng kỹ thuật pipeline, một kỹ thuật làm cho các giai đoạn khác nhau của nhiều lệnh được thi hành cùng một lúc => sẽ hiệu quả hơn. Trong khi đó nếu là kiến trúc stack thì biểu thức đó chỉ có thể được tính tuần tự từ trái sang phải
- Có thể dùng thanh ghi để lưu các biến và như vậy sẽ giảm thâm nhập đến bộ nhớ => chương trình sẽ nhanh hơn

Tuy nhiên do các lệnh đều dùng thanh ghi vậy bao nhiêu thanh ghi là đủ? câu trả lời phụ thuộc vào việc các thanh ghi được sử dụng bởi trình biên dịch như thế nào. Phần lớn trình biên dịch



dành riêng một vài thanh ghi cho tính toán một biểu thức, dùng vài thanh ghi để chuyển các thông số cần thiết và cho phép các thanh ghi còn lại lưu trữ các biến cần thiết.

<i>Loại kiến trúc</i>	<i>Ưu điểm</i>	<i>Nhược điểm</i>
Ngăn xếp (Stack)	<ul style="list-style-type: none"> <li>- Lệnh ngắn</li> <li>- Ít mã máy</li> <li>- Làm tối thiểu trạng thái bên trong của một máy tính</li> <li>- Dễ dàng tạo ra một bộ biên dịch đơn giản cho kiến trúc ngăn xếp</li> </ul>	<ul style="list-style-type: none"> <li>- Thâm nhập ngăn xếp không ngẫu nhiên.</li> <li>- Mã không hiệu quả</li> <li>- Khó dùng trong xử lý song song và ống dẫn</li> <li>- Khó tạo ra một bộ biên dịch tối ưu</li> </ul>
Thanh ghi tích lũy (Accumulator Register)	<ul style="list-style-type: none"> <li>- Lệnh ngắn</li> <li>- Làm tối thiểu trạng thái bên trong của máy tính (yêu cầu ít mạch chức năng)</li> <li>- Thiết kế dễ dàng</li> </ul>	<ul style="list-style-type: none"> <li>- Lưu giữ ở thanh ghi tích lũy là tạm thời.</li> <li>- Nghẽn ở thanh ghi tích lũy</li> <li>- Khó dùng trong xử lý song song và ống dẫn</li> <li>- Trao đổi nhiều với bộ nhớ.</li> </ul>
Thanh ghi đa dụng (General Register)	<ul style="list-style-type: none"> <li>- Tốc độ xử lý nhanh, định vị đơn giản.</li> <li>- Ít thâm nhập bộ nhớ.</li> <li>- Kiểu rất tổng quát để tạo các mã hữu hiệu</li> </ul>	<ul style="list-style-type: none"> <li>- Lệnh dài</li> <li>- Số lượng thanh ghi bị giới hạn</li> </ul>

Bảng 6.1. So sánh các kiểu kiến trúc bộ lệnh

### ➤ **Kiểu kiến trúc thanh ghi đa dụng**

Do hiện nay kiểu kiến trúc thanh ghi đa dụng chiếm vị trí hàng đầu nên trong các phần sau, ta chỉ đề cập đến kiểu kiến trúc này.

Đối với một lệnh tính toán hoặc logic điển hình (lệnh ALU), có 2 điểm cần nêu:

- Trước tiên, một lệnh ALU phải có 2 hoặc 3 toán hạng. Nếu trong lệnh có 3 toán hạng thì một trong các toán hạng chứa kết quả phép tính trên hai toán hạng kia (Ví dụ: ADD A, B, C). Nếu trong lệnh có 2 toán hạng thì một trong hai toán hạng phải vừa là toán hạng nguồn, vừa là toán hạng đích (Ví dụ: ADD A, B).
- Thứ hai, số lượng toán hạng bộ nhớ có trong lệnh. Số toán hạng bộ nhớ có thể thay đổi từ 0 tới 3.

Như đã nêu ở trên, trong nhiều cách tổ hợp có thể có các loại toán hạng của một lệnh ALU, các máy tính hiện nay chọn một trong 3 kiểu sau :

- thanh ghi-thanh ghi (kiểu này còn được gọi nạp - lưu trữ),
- thanh ghi - bộ nhớ
- bộ nhớ - bộ nhớ.

Kiểu thanh ghi - thanh ghi được nhiều nhà chế tạo máy tính lưu ý với các lý do: việc tạo các mã máy đơn giản, chiều dài mã máy cố định và số chu kỳ xung nhịp cần thiết cho việc thực hiện lệnh là cố định, ít thâm nhập bộ nhớ. Tuy nhiên, kiểu kiến trúc này cũng có một vài hạn chế của nó như: số lượng thanh ghi bị giới hạn, việc các thanh ghi có cùng độ dài dẫn đến không hiệu quả trong các lệnh xử lý chuỗi cũng như các lệnh có cấu trúc. Việc lưu và phục hồi các trạng thái khi có các lời gọi thủ tục hay chuyển đổi ngữ cảnh.

## 6.2. Địa chỉ bộ nhớ

Trong kiến trúc bộ lệnh bao giờ chúng ta cũng phải đề cập đến các toán hạng, mà một số toán hạng này được lưu trong bộ nhớ. Vậy cách tổ chức địa chỉ bộ nhớ ra sao là điều cần biết trước khi đi vào nghiên cứu các bộ lệnh.

Bộ nhớ (**memory**) là thành phần lưu trữ chương trình và dữ liệu trong máy tính mà trong chương 5 chúng ta đã biết **Bit** là Đơn vị cơ bản của bộ nhớ. Ngoài ra chúng ta cũng đã biết 1 bit có thể được tạo ra bằng 1 flip-flop. Nhưng cách bố trí các ô nhớ trong một bộ nhớ chung như thế nào? thứ tự sắp xếp của chúng ra sao? là điều chúng ta cần biết trong phần này.

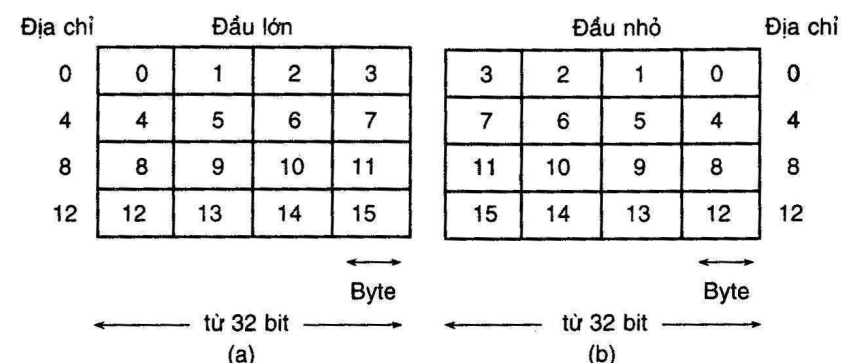
**Địa chỉ bộ nhớ** - Bộ nhớ gồm một số ô (hoặc vị trí), mỗi ô (**cell**) có thể chứa một mẫu thông tin. Mỗi ô gắn một con số gọi là địa chỉ (**address**), qua đó chương trình có thể tham chiếu nó. Giả sử bộ nhớ có  $n$  ô, chúng sẽ có địa chỉ từ 0 đến  $n-1$ . Tất cả ô trong bộ nhớ đều chứa cùng số bit. Trong trường hợp ô có  $k$  bit nó có khả năng chứa một trong số  $2^k$  tổ hợp bit khác nhau. Trong một bộ nhớ thì các ô kế cận nhau sẽ có địa chỉ liên tiếp nhau.

Ô là đơn vị có thể lập địa chỉ nhỏ nhất và các hãng khác nhau dùng qui định số bit trong một ô cho từng loại máy tính của mình là khác nhau như IBM PC 8 bit/ô, DEC PDP-8 12bit/ô, IBM 1130 16 bit/ô,... Tuy nhiên trong những năm gần đây, đa số các nhà sản xuất máy tính đều dùng chuẩn hóa ô 8 bit, gọi là **byte**. Byte nhóm lại thành từ (**word**) và máy tính với từ 16 bit sẽ có 2 byte/từ, còn máy tính với 32 bit sẽ có 4 byte/từ. Hầu hết các lệnh được thực hiện trên toàn bộ từ. Vì vậy máy tính 16 bit sẽ có thanh ghi 16 bit và lệnh thao tác trên 1 từ 16 bit, còn máy 32 bit sẽ có thanh ghi 32 bit và các lệnh thao tác trên 1 từ 32 bit.

## Sắp xếp thứ tự byte

Có hai cách sắp xếp thứ tự byte trong một từ, đánh số byte trong một từ từ trái sang phải và đánh số byte trong một từ từ phải sang trái.

Hình 6.2(a) cho thấy thứ tự byte trong bộ nhớ trên máy tính 32 bit có số byte được đánh số từ trái sang phải, như họ Motorola chẳng hạn. Hình 6.2(b) là một minh họa tương tự về máy tính 32 bit, đánh số từ phải sang trái, ví dụ như họ Intel. Hệ thống trước kia bắt đầu đánh số từ đầu lớn được gọi là máy tính đầu lớn (Big endian), trái ngược với đầu nhỏ (little endian).



Hình 6.2. (a) Bộ nhớ đầu lớn (họ Motorola), (b) Bộ nhớ đầu nhỏ (họ Intel)

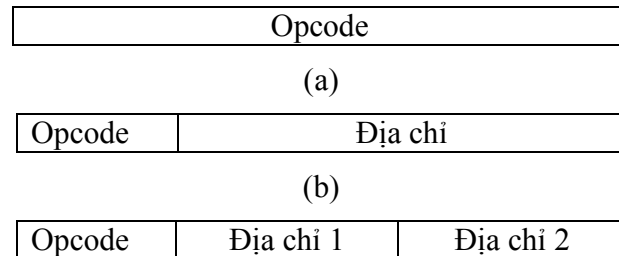
Cần biết rằng trong hệ thống đầu lớn lẫn đầu nhỏ, số nguyên 32 bit với trị số là 6 sẽ được biểu diễn bằng bit 110 ở 3 bit bên góc phải của từ và 0 ở 29 bit bên góc trái. Trong lược đồ đầu lớn, những bit này nằm trong byte đầu tiên bên phải, byte 3 (hoặc 7,11,...), trong khi đó ở lược đồ đầu nhỏ, chúng ở trong byte 0 (hoặc 4,8,...). Trong cả hai trường hợp, từ chứa số nguyên có địa chỉ 0. Điểm khác biệt này bên trong một máy tính là không có vấn đề gì, nhưng khi kết nối chúng vào trong cùng một mạng và khi trao đổi thông tin với nhau thì sẽ gặp nhiều vấn đề trực tiếp.

### 6.3. Mã hóa tập lệnh

Một chương trình bao gồm một dãy các lệnh (hay còn gọi là chỉ thị), mỗi lệnh chỉ rõ một việc làm cụ thể nào đó của máy tính như thực hiện phép cộng, thực hiện nhập dữ liệu, thực hiện đọc dữ liệu từ bộ nhớ,...

Như chúng ta đã thấy ở trên, trong một câu lệnh gồm có nhiều phần. Trong đó tác vụ thực hiện mỗi lệnh được chỉ rõ trong một trường gọi là **mã phép toán (operation code)** hay **mã tác vụ** và được gọi tắt là opcode, cho biết hành động nào sẽ được thi hành (từ đây trở đi ta dùng cụm từ mã tác vụ). Trong một lệnh còn chỉ ra được thực hiện trên các thanh ghi hay địa chỉ (*address*) ô nhớ, nơi chứa đựng dữ liệu cần xử lý.

Trong hình 6.3 cho ta 3 trường hợp mã hóa lệnh đơn giản là một số khuôn dạng điển hình của các bộ lệnh. Trường hợp thứ nhất chỉ có mã tác vụ mà không có phần địa chỉ. Trường hợp thứ hai mã một lệnh có 2 phần, phần tác vụ lệnh và địa chỉ và trường hợp thứ 3 (c) thì phần địa chỉ chiếm 2 vùng của mã lệnh.



Hình 6.3. Một vài dạng mã lệnh

Tùy thuộc vào kiến trúc của máy tính, trong một loại máy tính mã lệnh có thể có cùng chiều dài hoặc khác nhau. Trên một số máy tính tất cả các lệnh đều có cùng độ dài (Power PC, SPARC, MIPS), một số máy khác lại có thể có hai hoặc ba độ dài khác nhau (IBM 360/70, Intel 80x86), thậm chí độ dài mã lệnh có thể

thay đổi tùy ý (VAX). Ngoài ra một lệnh có thể ngắn hơn, dài hơn hoặc bằng với độ dài một từ.

#### 6.3.1. Các tiêu chuẩn thiết kế dạng thức lệnh

Vì có thể có nhiều dạng khác nhau của các lệnh, cho nên khi thiết kế máy tính cần có các tiêu chí rõ ràng để lựa chọn dạng thức lệnh cho máy cần thiết kế. Một số các tiêu chí chính được các nhà thiết kế đưa ra như sau:

##### ➤ Tiêu chuẩn thiết kế 1: Mã lệnh ngắn ưu việt hơn mã lệnh dài

Đây là tiêu chuẩn đầu tiên và cũng là quan trọng nhất. Một chương trình gồm  $n$  lệnh 16 bit chỉ chiếm chừng một nửa không gian bộ nhớ so với  $n$  lệnh 32 bit. Suy cho cùng, bộ nhớ không miễn phí, bởi vậy nhà thiết kế không thích lãng phí nó.

Ngoài ra thì mỗi bộ nhớ có một tốc độ truyền cụ thể, được quyết định qua công nghệ và thiết kế kỹ thuật. Nếu tốc độ truyền của bộ nhớ là  $T$  bit/giây (bps - bit per second) và chiều dài lệnh trung bình là  $L$  thì nó có thể truyền đi nhiều nhất là  $T/L$  lệnh trên một giây. Vì vậy, tốc độ thi hành lệnh (tức tốc độ bộ xử lý) tùy thuộc vào độ dài lệnh. Lệnh ngắn hơn đồng nghĩa với bộ xử lý nhanh hơn.

Nếu thời gian thi hành lệnh quá lâu so với thời gian tìm nạp nó từ bộ nhớ, thời gian tìm nạp lệnh sẽ không quan trọng. Nhưng với CPU nhanh, bộ nhớ thường là nút cổ chai. Bởi vậy, tăng số lệnh tìm nạp trên mỗi giây là tiêu chuẩn thiết kế quan trọng.

##### ➤ Tiêu chuẩn thiết kế thứ 2: Độ dài mã lệnh đủ để biểu diễn tất cả phép toán mong muốn

Nếu chúng ta cần thiết kế một máy tính với  $2^n$  các thao tác hay các vi tác vụ, thì ta không thể dùng mã hóa lệnh với độ dài nhỏ hơn  $n$ . Đơn giản không đủ chỗ trong opcode để cho biết là lệnh nào hay mã hóa tất cả các lệnh đó.

##### ➤ Tiêu chuẩn thiết kế thứ 3: độ dài word của máy bằng bội số nguyên của độ dài ký tự.

Trong trường hợp mà ký tự có  $k$  bit, độ dài từ phải là  $k, 2k, 3k, 4k, \dots$  còn không sẽ lãng phí không gian khi lưu trữ các từ.

Tất nhiên có thể lưu trữ 3.5 ký tự trong một word, song việc đó sẽ làm việc sẽ gây ra tình trạng kém hiệu quả nghiêm trọng trong khi truy cập các ký tự. Hạn chế do mã ký tự áp đặt lên chiều dài từ cũng ảnh hưởng đến chiều dài mã lệnh, bởi vì một lệnh tốt nhất là chiếm một số nguyên các byte hoặc số lệnh nguyên phải nằm gọn trong một từ.

➤ **Tiêu chuẩn thiết kế thứ 4: số BIT trong trường địa chỉ càng ngắn càng tốt**

Tiêu chuẩn này liên quan tới việc chọn kích thước ô nhớ. Cứ xem thiết kế máy với ký tự 8 bit (có thể 7 bit cộng tính chẵn lẻ) và bộ nhớ chính chứa  $2^{16}$  ký tự sẽ thấy. Nhà thiết kế có thể chọn gán địa chỉ liên tiếp cho đơn vị 8, 16, 24, hoặc 32 bit, cùng những khả năng khác.

Hãy hình dung chuyện gì sẽ xảy ra nếu đội ngũ thiết kế phân hóa thành hai phe gây chiến, một phe dốc sức tạo byte 8 bit, đơn vị cơ bản của bộ nhớ, còn phe kia ra sức tạo từ 32 bit như là đơn vị cơ bản của bộ nhớ. Phe đầu đề nghị bộ nhớ  $2^{16}$  byte, được đánh số 0, 1, 2, 3, ..., 65535. Phe sau đề xuất bộ nhớ  $2^{14}$  từ, được đánh số 0, 1, 2, 3, ..., 16383.

Nhóm thứ nhất chỉ ra rằng để so sánh hai ký tự trong tổ chức từ 32 bit, chương trình chẳng những tìm nạp từ chứa ký tự mà còn phải trích từng ký tự trong từ mới so sánh được. Làm vậy sẽ tốn thêm lệnh và lãng phí không gian. Trái lại, tổ chức 8 bit cung cấp địa chỉ cho từng lệnh, giúp so sánh dễ dàng hơn nhiều.

Phe đề xuất 32 bit sẽ phản bác bằng lập luận rằng đề xuất của họ chỉ cần  $2^{14}$  địa chỉ riêng biệt, cho độ dài địa chỉ 14 bit mà thôi, còn đề nghị byte 8 bit đòi hỏi 16 bit để lập địa chỉ cùng một bộ nhớ. Địa chỉ ngắn hơn đồng nghĩa với lệnh ngắn hơn, không những chiếm ít không gian mà còn đòi hỏi ít thời gian tìm nạp hơn.

Hoặc có thể giữ nguyên địa chỉ 16 bit để tham chiếu bộ nhớ lớn gấp 4 lần mức cho phép của tổ chức 8 bit.

Ví dụ này cho thấy rằng để đạt sự phân giải bộ nhớ tốt hơn, người ta phải trả cái giá bằng địa chỉ dài hơn, nói chung, có nghĩa là lệnh dài hơn. Mục tiêu tối thượng trong sự phân giải là tổ chức bộ nhớ có thể lập địa chỉ trực tiếp từng bit.

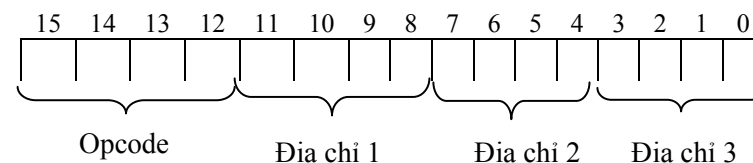
Thực tế có những máy tính mà chiều dài word chỉ có 1 bit (ví dụ máy Burroughs B1700), lại có những máy mà word rất dài, tới 60 bit (như máy CDC Cyber)

### 6.3.2. Opcode mở rộng

Trong phần này chúng ta xem xét những cân nhắc và thỏa hiệp liên quan đến mã tác vụ - opcode và địa chỉ.

Giả sử ta có lệnh  $(n+k)$  bit với opcode chiếm  $k$  bit và địa chỉ chiếm  $n$  bit. Lệnh này cung cấp  $2^k$  phép toán khác nhau và  $2^n$  ô nhớ lập địa chỉ được. Hoặc, cùng  $n + k$  bit đó có thể chia thành opcode  $(k - 1)$  bit và địa chỉ  $(n+1)$  bit, tức chỉ một nửa số lệnh nhưng gấp đôi bộ nhớ lập địa chỉ được, hoặc cũng dung lượng bộ nhớ đó nhưng Opcode  $(k+1)$  bit và địa chỉ  $(n-1)$  bit cho nhiều phép toán hơn, song phải trả giá bằng số ô lập địa chỉ được ít hơn. Giữa bit opcode và bit địa chỉ có những quân bình rất tinh tế cũng như đơn giản hơn như vừa trình bày.

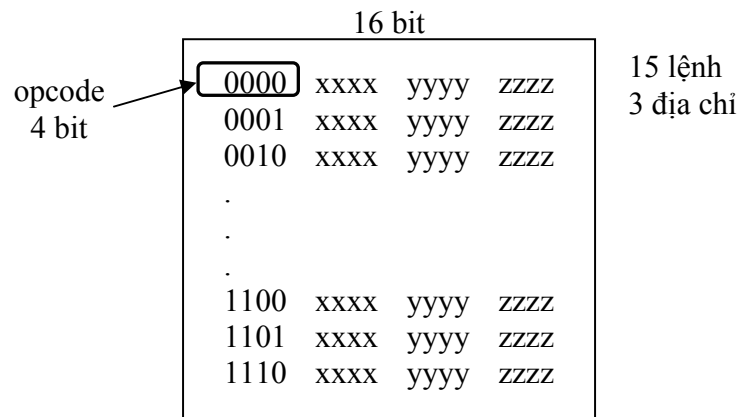
Để hiểu rõ vấn đề chúng ta xem ví dụ một máy tính có lệnh dài 16 bit, trong đó mã vi tác vụ opcode dài 4 bit và 3 trường địa chỉ, mỗi trường dài 4 bit như hình 6.4.



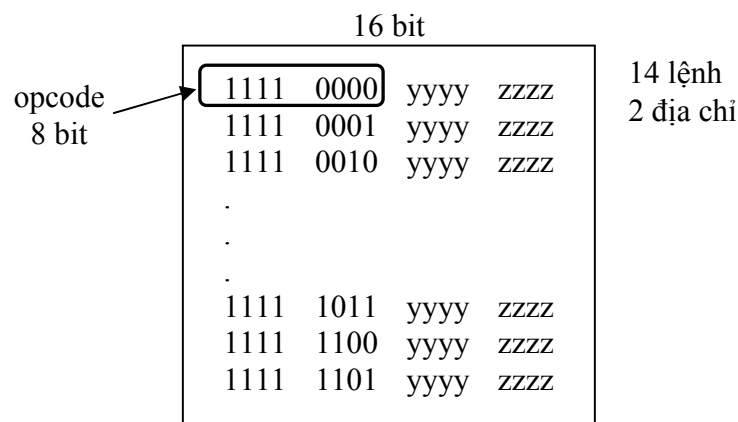
Hình 6.4. Lệnh có opcode 4 bit và 3 địa chỉ 4 bit

Như vậy, ứng với mã vi tác vụ 4 bit sẽ cung cấp cho ta  $2^4=16$  lệnh khác nhau với 3 địa chỉ. Nhưng nếu nhà thiết kế cần 15 lệnh ba địa chỉ, hoặc 14 lệnh hai địa chỉ, hoặc 31 lệnh một địa chỉ thì họ sẽ phải làm thế nào?

Đối với trường hợp thứ nhất 15 lệnh ba địa chỉ thì nhà thiết kế có thể lấy nguyên cấu trúc trên hình 6.4, nhưng bỏ đi một trường hợp của opcode như trong hình 6.5(a).

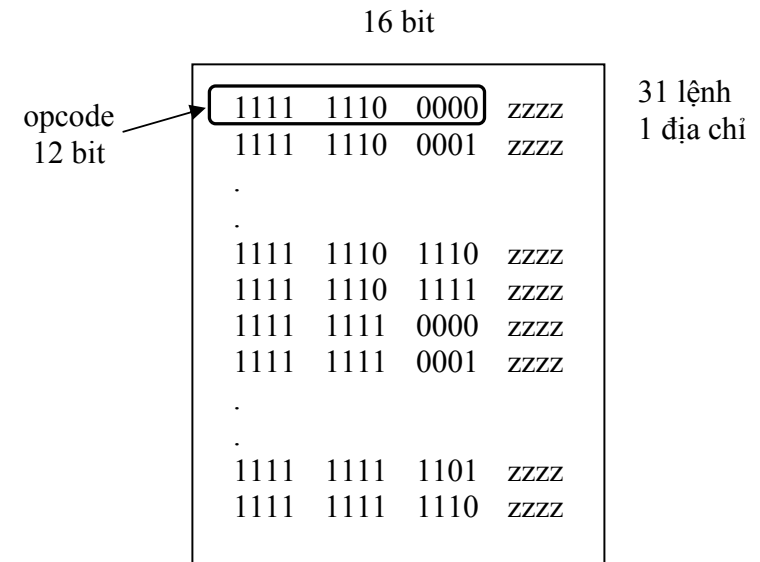


a)



b)

Hình 6.5. Một số dạng thức lệnh cho 16 bit



c)

Hình 6.5 (tiếp theo). Một số dạng thức lệnh cho 16 bit

Tương tự để có 14 lệnh hai địa chỉ ta làm như hình 6.5(b) và 31 lệnh một địa chỉ như hình 6.7(c).

Các bit cao nhất (bit 12 đến 15) trong trường hợp b) được gán mặc định trị nhị phân “1”, bốn bit kế đó (bit 8 đến 11) sẽ mã hóa các tác vụ cần thiết. Vì 4 bit thì mã hóa được 16 tác vụ, nhưng trong trường hợp này ta chỉ cần 14 tác vụ, do đó còn 2 vị trí không dùng đến. Trong các trường hợp có tổ hợp code còn lại không dùng, như trong hình 6.5 (b) thì tổ hợp opcode 1111 1110 và 1111 1111 sẽ được xử lý đặc biệt.

Như vậy độ dài các lệnh là như nhau, đều 16 bit, nhưng trong trường hợp a) thì độ dài opcode là 4, trường hợp b) là 8 trong khi trường hợp c) là 12 bit.

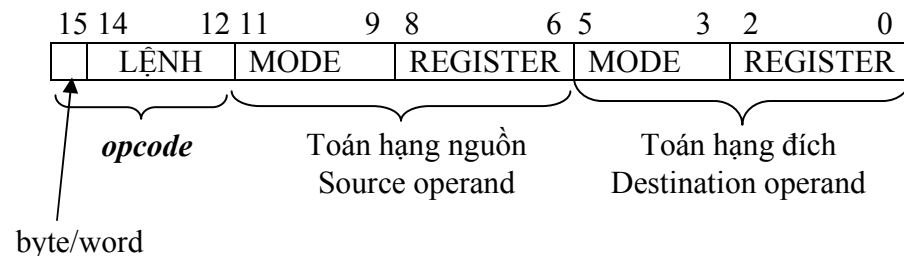
### 6.3.3. Ví dụ về dạng thức lệnh

Để hiểu rõ hơn các vấn đề trong mã hóa lệnh, trong phần này chúng ta sẽ khảo sát dạng thức lệnh trong các máy PDP-11 và Intel. Ở đây chúng ta đưa ra máy PDP-11 vì tập lệnh của nó được xem là một điển hình về tính đơn giản và hợp lý.

#### ➤ PDP-11

Đa số lệnh hai toán hạng của PDP-11 được mã hóa như trong hình 6.6. Mỗi lệnh chứa opcode 4 bit và hai trường địa chỉ 6 bit. Một bit opcode ở cực trái cho biết lệnh vận hành trên byte hay word. Trường địa chỉ được chia nhỏ thành hai phần, phần chế độ (mode) 3 bit và thanh ghi (Register) 3 bit (Máy PDP-11 có 8 thanh ghi cho nên chỉ cần 3 bit là có thể mã hóa được toàn bộ các thanh ghi). Trường chế độ cho biết toán hạng nằm trong thanh ghi, trong bộ nhớ hay là một hằng số, v.v. Có tám mode giống nhau cho toán hạng nguồn cũng như cho toán hạng đích. Mọi opcode đều có thể dùng bất kỳ toán hạng nguồn và toán hạng đích nào.

Tập lệnh trong PDP-11 sử dụng là trực giao (orthogonal), là tập lệnh trong đó phương thức qui định địa chỉ các toán hạng độc lập với opcode được các nhà viết trình biên dịch rất thích vì tập lệnh trực giao làm cho công việc của họ đơn giản đi rất nhiều.



Hình 6.6. Mã hóa lệnh trên máy PDP-11

Đối với một số lệnh khác, kể cả lệnh một toán hạng, PDP-11 áp dụng lược đồ opcode mở rộng, theo cách này thì các opcode có dạng x111 được dùng để tránh khỏi phải dùng các mã lệnh dài hơn. Với hầu hết các lệnh một toán hạng lấy opcode 10 bit và trường mode/register 6 bit, như vậy độ dài lệnh vẫn là 16 bit giống như các lệnh 2 toán hạng. Khi đó trường mã tác vụ sẽ có 10 bit bao gồm 4 bit của trường opcode và 6 bit của trường toán hạng nguồn (xem hình vẽ 6.6).

Các lệnh có lập địa chỉ bộ nhớ trong PDP-11 sẽ có thêm một hay hai word 16 bit đi theo sau lệnh địa chỉ để chỉ ra các địa chỉ này.

#### ➤ Họ Intel 8088/80286/80386/Pentium

Với CPU Intel, tình hình phức tạp hơn nhiều và kém đều đặn hơn nhiều. Đặc biệt đối với Pentium, mô hình chung như hình 6.7. Cấu tạo phức tạp của Pentium là do nó được kế thừa từ nhiều thế hệ trong một khoảng thời gian dài và do ngay từ đầu việc lựa chọn các tính chất đã không được thành công lắm. Điều này chính là do đòi hỏi phải kế thừa các đặc tính ra trước mà nó không thể thay đổi cấu trúc của mình. Nói chung, đối với lệnh hai toán hạng, nếu toán hạng này nằm trong bộ nhớ thì toán hạng kia có thể không nằm trong bộ nhớ. Do đó tồn tại các lệnh cộng hai thanh ghi, lệnh cộng thêm giá trị thanh ghi vào bộ nhớ và cộng thêm bộ nhớ vào thanh ghi, nhưng không tồn tại lệnh cộng thêm một từ vào một từ nhớ khác, một điều mà PDP-11 cho phép như là kết quả trực giao.

Trên 8088, mỗi opcode là 1 byte, nhưng đến khi 80386 ra đời thì opcode 1 byte dùng không đủ để mã hóa hết tập lệnh, do đó bit 15 của opcode được dùng để tránh phải dùng opcode 2 byte. Cấu trúc duy nhất trong trường opcode là sử dụng bit thứ tự thấp trong một số lệnh để cho biết byte/word, và bit bên cạnh để chỉ địa chỉ bộ nhớ (nếu có) là nguồn hay là đích.

Tiếp sau byte opcode trong nhiều lệnh là byte thứ hai cho biết vị trí toán hạng, tương tự như hai trường mode/register trong

hình 6.6. Do chỉ có sẵn 8 bit, nên tách ra thành trường chế độ 2 bit và hai trường thanh ghi 3 bit. Vậy chỉ có bốn cách lập địa chỉ toán hạng (so với tám cách trên PDP-11), và một trong số toán hạng luôn phải là thanh ghi. Về logic, AX, BX, CX, DX, SI, DI, BP, và SP phải cụ thể như thanh ghi, song nguyên tắc mã hóa ngăn cấm một số kết hợp và dùng chúng vào trường hợp đặc biệt.

Bên cạnh đó, một số lệnh có 1, 2, hoặc 4 byte trở lên qui định địa chỉ bộ nhớ và có thể 1, 2, hoặc 4 byte nữa dùng làm toán hạng hằng (chẳng hạn như di chuyển con số 100 vào thanh ghi).

Bảng 6.2 cung cấp tập dạng thức lệnh 8088, 80286, 80386 và Pentium. Mỗi lệnh tiềm chứa tối đa sáu trường, mỗi trường từ 0 đến 4 byte. Trên 8088 và 80286, lệnh ngắn nhất là 1 byte và dài nhất là 9 byte. Trên 80386 cũng như Pentium, lệnh ngắn nhất vẫn là 1 byte, nhưng thêm tiền tố kích thước toán hạng và tiền tố kích thước địa chỉ, lệnh có thể tối đa 16 byte.

CPU	PREFIX	OPCODE	MODE	SIB	DISPLACEMENT	IMMEDIATE
8088	0-3	1	0-1	0	0-2	0-2
80286	0-3	1	0-1	0	0-2	0-2
80386	0-4	1-2	0-1	0-1	0-4	0-4
Pentium	0-4	1-2	0-1	0-1	0-4	0-4

Bảng 6.2. Dạng thức lệnh của các máy tính Intel

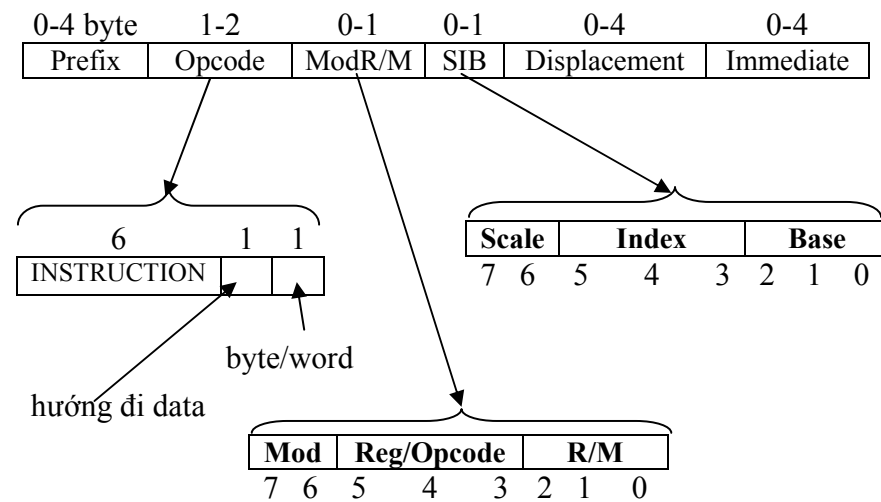
Hình 6.7 cho ta thấy dạng thức lệnh của máy Pentium với các trường được định nghĩa như sau :

- **PREFIX byte** : đó là phần mã thêm của mã lệnh được đặt trước opcode. Nếu nó tồn tại thì nó bao gồm LOCK tiếp đầu tố (prefix) và tiếp đầu tố lặp lại (repeat prefix). LOCK prefix được dùng để đảm bảo việc dành riêng vùng nhớ chia sẻ trong môi trường đa bộ xử lý. Trong khi repeat prefix đặc trưng cho một chuỗi phép toán được lập đi lập lại, điều này

cho phép CPU Pentium thực hiện nhanh hơn là một vòng lặp được lập trình.

- **Opcode** : chiếm 1 hoặc 2 byte. Opcode có thể bao gồm những bit chỉ ra dữ liệu là không đầy đủ hay là đầy đủ (16 hoặc 32 bit phụ thuộc vào từng trường hợp cụ thể), bit chỉ ra hướng của dữ liệu đến bộ nhớ hoặc từ bộ nhớ đi
- **ModR/M** : byte này chỉ ra các thông tin về toán hạng. Byte ModR/M chỉ ra toán hạng là một thanh ghi hay là trong bộ nhớ. Nếu nó là trong bộ nhớ thì bên trong trường này sẽ chỉ ra mode địa chỉ nào được dùng. Trường ModR/M bao gồm 3 phần : phần mode 2 bit và hai phần Reg/Opcode và R/M mỗi phần 3 bit nữa. Đôi khi 3 bit của trường Reg/Opcode được sử dụng với tính cách là phần mở rộng của Opcode tạo nên trường Opcode với 11 bit. Trường Mod chỉ có 2 bit, điều đó có nghĩa là chỉ có 4 cách tiếp cận với toán hạng và một trong các toán hạng luôn luôn phải là thanh ghi.
- **SIB** : Cho phép chỉ ra một số chi tiết kỹ thuật thêm nhằm mục đích thêm vào một số tính năng mới nhưng vẫn thích hợp (support) với các kiểu cũ
- **Displacement**: Địa chỉ dịch chuyển (sẽ tìm hiểu kỹ hơn ở phần tiếp theo sau)
- **Immediate**: địa chỉ tức thời





Hình 6.7. Format lệnh Pentium

#### 6.3.4. Các chế độ lập địa chỉ

Có thể phân lệnh theo số lượng địa chỉ sử dụng. Lệnh qui định một, hai hay ba địa chỉ đều phổ biến. Trên nhiều máy tính phép tính số học được thực hiện với một địa chỉ duy nhất. Có một thanh ghi đặc biệt gọi là thanh bộ tích lũy (accumulator) sẽ cung cấp một trong các toán hạng, toán hạng còn lại sẽ nằm ở bộ nhớ. Trên máy này, địa chỉ thường là địa chỉ của từ nhớ m, trong đó đặt toán hạng.

Trong kiến trúc GPR chế độ lập địa chỉ cần phải chỉ rõ đó là một hằng số, một thanh ghi hay một vị trí trong bộ nhớ. Khi là một vị trí trong bộ nhớ được dùng thì địa chỉ ô nhớ thực được chỉ ra bởi cách dùng chế độ lập địa chỉ và được gọi là **địa chỉ hiệu dụng (effective address)**.

Để hiểu cách thực hiện một lệnh ta phải biết phương cách thông dịch bit trong trường địa chỉ để tìm toán hạng. Khả năng là chúng chứa địa chỉ bộ nhớ của toán hạng. Thế nhưng cũng có những khả năng khác và trong phần này chúng ta sẽ khám phá những kỹ thuật đánh địa chỉ cơ bản nhất sau:

- Địa chỉ tức thời – Immediate
- Địa chỉ trực tiếp – Direct
- Địa chỉ gián tiếp – Indirect
- Địa chỉ thanh ghi – Register
- Địa chỉ gián tiếp thanh ghi – Register indirect
- Địa chỉ dịch chuyển – Displacement
- Địa chỉ ngăn xếp - Stack

Các cách lập địa chỉ thông dụng trong hình 6.8.

Trong bảng bảng 6.3. chỉ ra cách tính địa chỉ thực cho mỗi chế độ lập địa chỉ và các ưu khuyết điểm của mỗi loại.

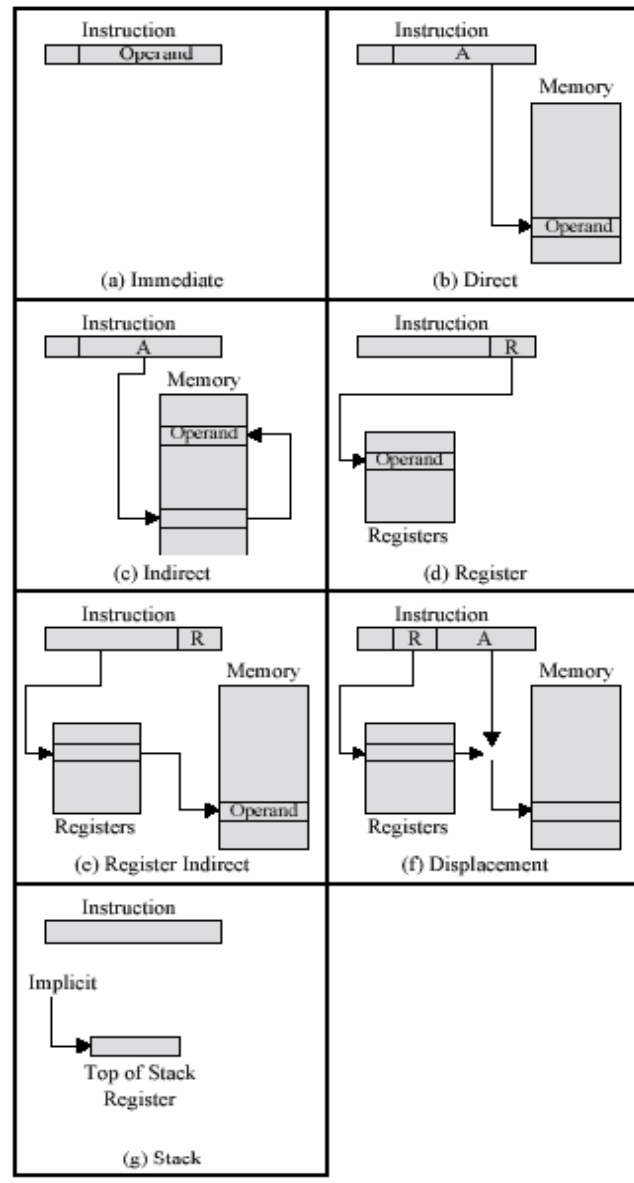
Các ký hiệu trong hình 6.8 và bảng 6.3:

A – Nội dung trong trường địa chỉ trong một lệnh

R – Nội dung trong trường địa chỉ mà chỉ ra một thanh ghi nào đó

EA – Địa chỉ thực của nơi chứa (memory hoặc register) toán hạng

(X) – Nội dung của vị trí bộ nhớ X hoặc là thanh ghi X



Hình 6.8. Các chế độ lập địa chỉ

Chế độ	Cách tính	Ưu điểm	Khuyết điểm
Immediate	Operand = A	Không có tham chiếu bộ nhớ	Độ lớn toán hạng giới hạn
Direct	EA = A	Đơn giản	không gian địa chỉ giới hạn
Indirect	EA = (A)	không gian địa chỉ lớn	Tham chiếu bộ nhớ phức tạp
Register	EA = R	Không có tham chiếu bộ nhớ	không gian địa chỉ giới hạn
Register indirect	EA = (R)	không gian địa chỉ lớn	Tham chiếu bộ nhớ phụ
Displacement	EA = A + (R)	Linh động	Phức tạp
Stack	EA = đầu của ngăn xếp	Không có tham chiếu bộ nhớ	Ứng dụng giới hạn

Bảng 6.3. Cách tính địa chỉ thực

Trước khi đi vào cụ thể từng chế độ lập địa chỉ có hai điểm cần lưu ý, thứ nhất đó là trên thực tế tất cả các kiến trúc máy tính ngày nay cung cấp cho ta nhiều hơn một trong những chế độ lập địa chỉ trên. Vấn đề đặt ra là làm thế nào để bộ điều khiển xác định được chế độ địa chỉ nào được dùng trong lệnh. Có nhiều phương pháp khác nhau. Thường là những opcode khác nhau sẽ dùng các chế độ địa chỉ khác nhau, cũng có thể là một trong các bit của lệnh được dùng làm trường chế độ, mà giá trị chứa trường này chỉ ra chế độ địa chỉ được dùng.

Điểm thứ hai ở đây liên quan đến địa chỉ thực EA. Trong một hệ thống không dùng bộ nhớ ảo thì địa chỉ hiệu dụng sẽ hoặc là một địa chỉ của bộ nhớ chính, hoặc là một thanh ghi. Trong một hệ thống nhớ ảo thì địa chỉ hiệu dụng là một địa chỉ ảo hoặc là một thanh ghi. Việc sắp xếp

thực của địa chỉ vật lý là một chức năng của cơ chế phân trang và người lập trình không thấy được.

#### a) Lập địa chỉ tức thời (Immediate Addressing)

Cách đơn giản nhất cho lệnh qui định toán hạng là để phần địa chỉ trong lệnh chứa chính toán hạng thay vì địa chỉ hoặc thông tin khác mô tả vị trí toán hạng:

$$\text{OPERAND} = A$$

Toán hạng như vậy được gọi là toán hạng tức thời (immediate operand) vì tự động được tìm nạp từ bộ nhớ cùng lúc với tìm nạp bản thân lệnh, nhờ đó khả dụng tức thời.

Lập địa chỉ tức thời có ưu điểm là khỏi cần thêm tham chiếu bộ nhớ bộ nhớ để tìm nạp toán hạng. Nhược điểm là hạn chế toán hạng ở con số vừa vào trường địa chỉ. Ở lệnh có địa chỉ 3 bit (ví dụ như trường thanh ghi), toán hạng sẽ giới hạn ở 3 bit, và làm hạn hẹp tính hữu ích của chúng.

Chế độ này được dùng để định nghĩa các hằng số hoặc là khởi tạo một giá trị nào đó cho một biến

Một ví dụ trong các trường hợp lập địa chỉ tức thời là đưa giá trị “4” vào thanh ghi R1 như sau:

MOV R1, #4

#### b) Lập địa chỉ trực tiếp (Direct Addressing)

Một phương thức đơn giản khác để qui định toán hạng là cung cấp địa chỉ của từ nhớ có chứa toán hạng và đặt nó vào trường địa chỉ của lệnh. Hình thức này gọi là lập địa chỉ trực tiếp (*direct addressing*):

$$\text{EA} = A$$

Tất nhiên phải có cách để máy tính biết được địa chỉ nào là tức thời và địa chỉ nào là trực tiếp. Nói chung, có hai phương pháp: sử dụng opcode khác nhau hoặc sử dụng một mode đánh địa chỉ đặc biệt cho mỗi loại toán hạng.

Cũng như cách lập địa chỉ tức thì, địa chỉ trực tiếp có một số giới hạn: Lệnh luôn truy cập đến chỉ một địa chỉ ô nhớ. Tức là giá trị tại địa chỉ đó có thể thay đổi nhưng địa chỉ thì không. Như

vậy địa chỉ trực tiếp có thể được sử dụng với các biến toàn cục, mà địa chỉ của nó là biết trước trong thời gian biên dịch.

#### c) Lập địa chỉ gián tiếp (Indirect Addressing)

Đánh địa chỉ trực tiếp là cách trong đó trường địa chỉ chỉ ra một từ nhớ nào hay thanh ghi nào **chứa toán hạng**. Tuy nhiên trong trường hợp lập địa chỉ trực tiếp thì chiều dài của trường địa chỉ thường là ngắn hơn chiều dài một word, do đó số địa chỉ có thể mã hóa được bị hạn chế. Cách đánh địa chỉ gián tiếp là trường địa chỉ chỉ ra từ nhớ nào hoặc thanh ghi nào **chứa địa chỉ của toán hạng** ( Xem hình 6.8):

$$\text{EA} = (A)$$

Nói cách khác là từ nhớ hay thanh ghi trong trường địa chỉ giống như là một con trỏ (trong C++), trỏ tới một toán hạng

Việc đánh địa chỉ gián tiếp cần hai lần truy cập bộ nhớ, lần thứ nhất để lấy con trỏ về và lần thứ hai để lấy toán hạng về.

#### d) Lập địa chỉ thanh ghi (Register Addressing)

Về khái niệm, lập địa chỉ thanh ghi tương tự như lập địa chỉ trực tiếp. Điểm khác biệt duy nhất ở đây là thay vì trường địa chỉ trỏ tới một địa chỉ trong bộ nhớ thì ở đây là trỏ tới một thanh ghi (thanh ghi số mấy) trong đó lưu trữ toán hạng:

Máy với 16 thanh ghi và bộ nhớ 65.536 từ thật sự sẽ có hai không gian địa chỉ. Ta có thể xem một địa chỉ trên máy như thế là có hai phần:

- Một bit cho biết là ta muốn dùng thanh ghi hay từ nhớ
- Một trường địa chỉ cho biết là sẽ cần thanh ghi hay từ nhớ nào.

Vì số thanh ghi ít hơn số từ nhớ, do đó người ta thường dùng các dạng thức lệnh khác nhau cho toán hạng thanh ghi và toán hạng nhớ.

Các máy ngày nay được thiết kế có các thanh ghi vì 2 lý do chính sau:

- Các thanh ghi hoạt động nhanh hơn bộ nhớ chính

- Số lượng thanh ghi là rất ít do đó để mã hóa chúng cũng chỉ cần một số ít bit.

### e) Địa chỉ gián tiếp thanh ghi (Register Indirect)

Địa chỉ thanh ghi thì giống địa chỉ trực tiếp, còn địa chỉ gián tiếp thanh ghi thì lại giống với địa chỉ gián tiếp. Trong cả hai trường hợp sự khác nhau chỉ là một cách là địa chỉ bộ nhớ chính, còn một cách là thanh ghi:

$$EA = (R)$$

Trường địa chỉ chứa số thanh ghi, mà trong thanh ghi đó chứa địa chỉ của toán hạng cần thiết.

### f) Địa chỉ dịch chuyển – Displacement

Một chế độ được tổng hợp từ hai chế độ: địa chỉ trực tiếp và địa chỉ gián tiếp thanh ghi. Cách tính địa chỉ thực như sau:

$$EA = A + (R)$$

Chế độ này đòi hỏi trong mã lệnh phải có hai trường địa chỉ, một trường cho địa chỉ thanh ghi (R) và một trường cho địa chỉ trực tiếp bộ nhớ (A). Địa chỉ thực là tổng của địa chỉ A với giá trị địa chỉ chứa trong thanh ghi R.

Một cách đánh địa chỉ dịch chuyển hay dùng là dạng đánh địa chỉ “**chỉ số**” (**Indexing**). Có nhiều thuật toán đòi hỏi một số thao tác trên dãy các cấu trúc dữ liệu được chứa trong các vị trí nhớ liên tiếp. Ví dụ chúng ta xem một khối gồm N từ chiếm các vị trí nhớ:

$$A, A+1, A+2, \dots, A+N-1$$

Giả sử chúng cần được chuyển tới vị trí nhớ:

$$B, B+1, B+2, \dots, B+N-1$$

Và giả sử ta dùng lệnh máy

**MOVE B,A**

để chuyển nội dung vị trí nhớ A đến vị trí nhớ B. Máy thi hành lệnh này và sau đó sửa lại thành

**MOVE B+1,A+1**

rồi thi hành lệnh này, rồi lại sửa lại, lại thi hành.... cứ lặp đi lặp lại chu kỳ này cho đến khi tất cả N word được copy xong.

Bài toán như thế được giải quyết tốt nhất bằng cách sử dụng một thanh ghi gọi là thanh ghi chỉ số (Index register) và chúng làm việc như sau:

Trường địa chỉ sẽ có hai phần: một con số của thanh ghi chỉ số và một hằng số. Trong ví dụ trên nếu cả hai địa chỉ đều được đánh chỉ số sử dụng một index register (IR) có chứa số nguyên k, thì lệnh **MOVE B,A** sẽ chuyển nội dung của vị trí nhớ A+k tới B+k. Bằng cách khởi tạo cho IR giá trị ban đầu 0 và tăng nó lên bằng kích thước một word mỗi khi copy xong một word, thì chúng ta chỉ cần sử dụng một thanh ghi cho vòng lặp copy.

Việc đánh địa chỉ Index được sử dụng rộng rãi để đánh địa chỉ một trường tại một khoảng cách đã biết tính từ điểm đầu của cấu trúc dữ liệu đã cho.

### g) Địa chỉ ngăn xếp – Stack

Ở trên ta đã đưa ra các tiêu chí thiết kế dạng thức lệnh, mà một trong các tiêu chí là lệnh càng ngắn càng tốt để tiết kiệm thời gian của CPU và tiết kiệm bộ nhớ. Giới hạn cuối cùng của việc giảm chiều dài địa chỉ là làm cho lệnh không còn trường địa chỉ nữa, chỉ có opcode thôi. Thật đáng ngạc nhiên là điều này có thể thực hiện được bằng cách sử dụng một cấu trúc dữ liệu có tên là Stack.

Stack chứa các phần tử dữ liệu (words, characters,...) theo trật tự liên tiếp trong bộ nhớ. Phần tử đầu tiên được đẩy lên Stack được gọi là nằm ở đáy của Stack, phần tử sau cùng vừa mới đẩy lên Stack được gọi là nằm ở đỉnh của Stack. Dữ liệu được đưa vào và lấy ra theo phương thức vào đầu tiên thì ra sau cùng **FILO (first in last out)**. Mỗi Stack được gắn với một thanh ghi hay word bộ nhớ chứa địa chỉ đỉnh Stack và được gọi là con trỏ Stack.

Hình 6.9 cho ta thấy các chế độ lập địa chỉ của dữ liệu trong các máy tính mới nhất và các ví dụ tương ứng với nó. Trong hình này và trong cuốn sách này chúng ta dùng phần mở rộng của ngôn ngữ lập trình C để biểu diễn các câu lệnh. Ở đây chúng ta dùng mảng **Mem** như là tên của bộ nhớ chính và **Regs** để chỉ Registers. Ví dụ chúng ta ghi **Mem[Regs[R1]]** để chỉ một ô nhớ có địa chỉ được ghi trong thanh ghi tên là R1. Trong bảng này mỗi chế độ lập địa chỉ sẽ có một ví dụ minh họa được đưa ra, đồng thời giải thích ý nghĩa của nó và cho biết nó được dùng khi nào.

Chế độ	Ví dụ	Diễn giải
Register	Add R4, R3	$Regs[R4] \leftarrow Regs[R4] + Regs[R3]$
Immediate	Add R4, #3	$Regs[R4] \leftarrow Regs[R4] + 3$
Displacement	Add R4, 100 (R1)	$Regs[R4] \leftarrow Regs[R4] + Mem[100 + Regs[R1]]$
Register deferred or indirect	Add R4, (R1)	$Regs[R4] \leftarrow Regs[R4] + Mem[Regs[R1]]$
Indexed	Add R3, (R1 + R2)	$Regs[R3] \leftarrow Regs[R3] + Mem[Regs[R1] + Regs[R2]]$
Direct or absolute	Add R1, (1001)	$Regs[R1] \leftarrow Regs[R1] + Mem[1001]$
Memory indirect or memory deferred	Add R1, @(R3)	$Regs[R1] \leftarrow Regs[R1] + Mem[Mem[Regs[R3]]]$
Autoincrement	Add R1, (R2) +	$Regs[R1] \leftarrow Regs[R1] + Mem[Regs[R2]]$ $Regs[R2] \leftarrow Regs[R2] + d$
Autodecrement	Add R1, -(R2)	$Regs[R2] \leftarrow Regs[R2] - d$ $Regs[R1] \leftarrow Regs[R1] + Mem[Regs[R2]]$
Scaled	Add R1, 100 (R2) [R3]	$Regs[R1] \leftarrow Regs[R1] + Mem[100 + Regs[R2] + Regs[R3] * d]$

Hình 6.9. Các chế độ lập địa chỉ thông dụng

Ví dụ ở chế độ lập địa chỉ “thanh ghi” (Register) để làm phép toán cộng hai số ta dùng câu lệnh:

**Add R4, R3.**

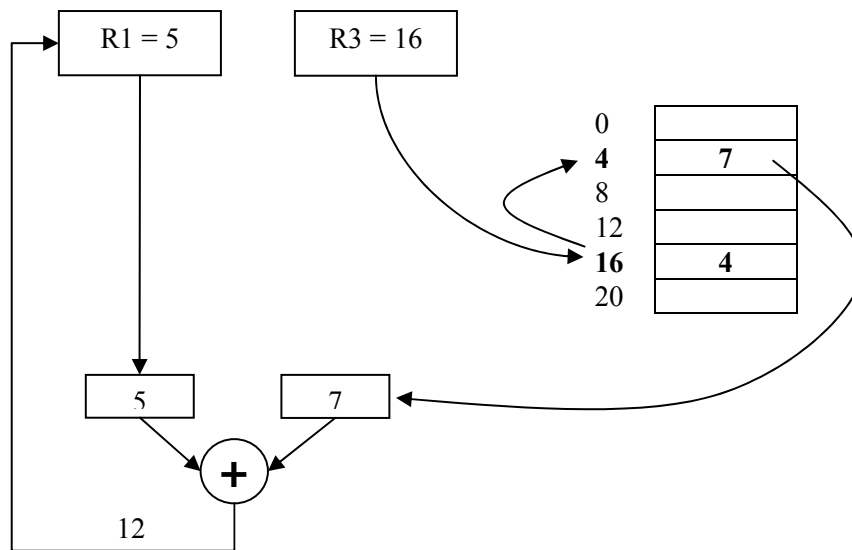
Câu lệnh này cho thấy ở chế độ thanh ghi các toán hạng đều là các thanh ghi. Ở đây R3 và R4 là hai toán hạng của phép toán cộng. Ý nghĩa của câu lệnh này có nghĩa là:

$Regs[R4] \leftarrow Regs[R4] + Regs[R3]$

Tức là toán hạng thứ nhất nằm trên thanh ghi có tên R4 và toán hạng thứ hai nằm trên thanh ghi có tên là R3 sẽ được cộng lại và kết quả cuối cùng sẽ lưu vào thanh ghi R4. Mũi tên  $\leftarrow$  chỉ ra kết quả sẽ được lưu vào đâu. Đặc biệt trong câu lệnh với tham chiếu bộ nhớ (Memory indirect) hay trong C ta gọi là với con trỏ ta thấy ở toán hạng thứ hai có thêm dấu “@” để chỉ toán hạng đó là một địa chỉ và ở địa chỉ đó chứa một địa chỉ khác, mà ở địa chỉ cuối này mới chứa giá trị thật của toán hạng này. Giả sử trong câu lệnh

**Add R1, @(R3)**

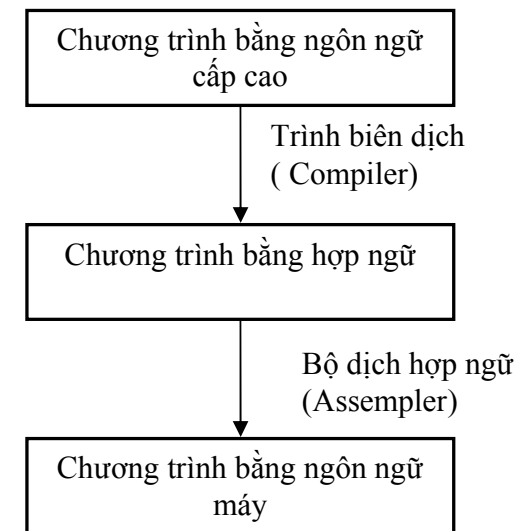
R1 chứa giá trị 5, R3 chứa giá trị 16 thì lệnh này có thể được diễn giải rõ hơn như hình 6.9. Trong R3 chứa giá trị 16 là một địa chỉ, mà trong địa chỉ này chứa giá trị 4 là một địa chỉ khác, mà trong địa chỉ 4 này mới chứa giá trị thật 7 của toán hạng thứ hai. Giá trị toán hạng thứ nhất là 5 cộng với giá trị toán hạng thứ hai là 7, được 12 và giá trị này lại ghi trở lại vào R1.



Hình 6.9. Lệnh Add với tham chiếu bộ nhớ

## 6.4. Bộ lệnh

Trong phần này chúng ta sẽ xem xét các loại lệnh cơ bản của các kiến trúc phần mềm được dùng nhiều nhất, để cho thấy các kỹ thuật ở mức ngôn ngữ máy dùng để thi hành các cấu trúc trong các ngôn ngữ cấp cao. Đa số ngày nay, để viết các chương trình cho máy tính, người ta dùng các ngôn ngữ cấp cao dễ hiểu và tiện lợi hơn như C, Pascal, C#,... Quá trình biên dịch từ một ngôn ngữ cấp cao sang ngôn ngữ máy tiến hành như trong hình 6.10.



Hình 6.10. Quá trình biên dịch ra ngôn ngữ máy

Do mỗi lệnh trong máy tính là tổ hợp các con số nhị phân 0, 1 nên rất khó nhớ. Để khắc phục điểm yếu này người ta dùng hợp ngữ **Assembly** để mô tả các mã lệnh nhị phân bằng các từ ngắn gọn mà ta gọi là **từ gọi nhớ mã lệnh**. Ngoài ra trong quá trình diễn giải chúng ta còn dùng các khái niệm thanh ghi đích, thanh ghi nguồn 1, thanh ghi nguồn 2.

*Từ gọi nhớ mã lệnh mô tả ngắn gọn tác vụ phải thi hành trên các thanh ghi nguồn, kết quả được lưu giữ trong thanh ghi đích.*

Mỗi lệnh của ngôn ngữ cấp cao được xây dựng bằng một lệnh mã máy hoặc một chuỗi nhiều lệnh mã máy. Lệnh nhảy (**GOTO**) được thực hiện bằng các lệnh hợp ngữ về nhảy (**JUMP**) hoặc lệnh hợp ngữ về vòng. Chúng ta phân biệt lệnh nhảy làm cho bộ đếm chương trình được nạp vào địa chỉ tuyệt đối nơi phải nhảy đến ( $PC \leftarrow$  địa chỉ tuyệt đối nơi phải nhảy tới), với lệnh vòng theo

đó ta chỉ cần cộng thêm một độ dời vào bộ đếm chương trình (PC  $\leftarrow$  PC + độ dời).

Ngoài ra do mỗi kiểu kiến trúc máy tính có cách mã hóa lệnh và tên các câu lệnh khác nhau, do đó trong phần này chúng ta chỉ đề ý đến kiểu cấu trúc RISC.

#### 6.4.1. Nhóm lệnh truyền dữ liệu

Nhóm lệnh này nhằm truyền dữ liệu (a word or a block) từ một nguồn có thể là thanh ghi hoặc bộ nhớ đến một đích. Phần lớn trong nhóm này là những lệnh truyền dữ liệu giữa các thanh ghi khác nhau trong CPU. Việc truyền dữ liệu từ một thanh ghi đến một thanh ghi khác có thể thông qua lệnh sau:

**MOVE  $R_i, R_j$**  // truyền dữ liệu từ thanh ghi  $R_j$  đến thanh ghi  $R_i$

Ở đây dữ liệu từ thanh ghi nguồn  $R_j$  được ghi đè lên thanh ghi đích  $R_i$ , còn dữ liệu của  $R_j$  thì không thay đổi.

Một số ví dụ của lệnh MOVE như trong bảng 6.4.

Để nạp một giá trị từ bộ nhớ vào thanh ghi hoặc lưu một giá trị từ thanh ghi vào bộ nhớ ta dùng các lệnh sau:

**LOAD *đích, nguồn***

ví dụ: **LOAD  $R_i, M$**  (địa chỉ) //  $R_i \leftarrow M[\text{địa chỉ}]$

**STORE *đích, nguồn***

ví dụ: **STORE  $M(\text{địa chỉ}), R_i$**  //  $M[\text{địa chỉ}] \leftarrow R_i$

Trong bảng 6.5 cho ta thấy một số các lệnh và ý nghĩa của nó

Đích	Nguồn	Ví dụ	Giải thích
Bộ nhớ	Thanh ghi	<b>MOVE 100H, AX</b>	Chuyển nội dung trong AX vào vị trí nhớ 100H
Thanh ghi	Bộ nhớ	<b>MOVE AX, MEM1</b>	Chuyển nội dung trong vị trí nhớ MEM1 chỉ ra vào thanh ghi AX
Thanh ghi	Thanh ghi	<b>MOVE AX, BX</b>	Chuyển nội dung trong thanh ghi BX vào thanh ghi AX
Thanh ghi	Hằng số	<b>MOVE AX, 0FFFFH</b>	Chuyển giá trị hằng số ở hệ 16: FFFF vào thanh ghi AX, số 0 ở đầu để chỉ rõ FFFFH là một giá trị hằng chứ không phải là một nhãn

Bảng 6.4. Một số ví dụ lệnh MOVE

Lệnh truyền dữ liệu	Ý nghĩa
MOVE	Di chuyển (một từ hay một khối) từ địa chỉ nguồn (thanh ghi hay bộ nhớ trong) đến địa chỉ đích.
LOAD	Đọc dữ liệu từ bộ nhớ vào thanh ghi
STORE	Lưu dữ liệu từ thanh ghi vào bộ nhớ
PUSH	Lưu dữ liệu từ thanh ghi vào ngăn xếp
POP	Nhận dữ liệu từ ngăn xếp vào thanh ghi

Bảng 6.5. Một số lệnh truyền dữ liệu



**6.4.2. Nhóm lệnh tính toán số học:**

Các lệnh số học bao gồm bốn phép tính số học cơ bản là cộng, trừ, nhân, chia và đảo dấu toán hạng.

**ADD/SUB**

Dạng tổng quát của các lệnh cộng (add) và trừ (subtract) như sau:

ADD đích, nguồn // đích  $\leftarrow$  đích + nguồn

SUB đích, nguồn // đích  $\leftarrow$  đích – nguồn

trong đó các toán hạng đích và nguồn có thể tìm được theo các địa chỉ khác nhau, nhưng phải chứa dữ liệu có cùng độ dài và không được phép đồng thời là hai ô nhớ và cũng không được là thanh ghi đoạn

Ví dụ 1:

ADD AX, BX // AX  $\leftarrow$  AX + BX

ADD AL, 74H // AL  $\leftarrow$  AL + [74H]

SUB CL, AL // CL  $\leftarrow$  CL – AL

SUB AX, 0405H // AX  $\leftarrow$  AX – 0405H

Ví dụ 2: Viết đoạn chương trình bằng ngôn ngữ Assembly để cộng 5H với 3H, dùng các thanh ghi AL và BL. Kết quả của phép cộng lưu vào bộ nhớ tại địa chỉ 100H.

MOV AL, 05H // AL  $\leftarrow$  05H

MOV BL, 03H // BL  $\leftarrow$  03H

ADD AL, BL // AL  $\leftarrow$  AL + BL

MOV 100H, AL // MEM[100H]  $\leftarrow$  AL: di  
// chuyển kết quả từ AL vào  
// vị trí nhớ DS:100H

Bảng 6.6 cho ta tóm tắt các lệnh tính toán số học và ý nghĩa tương ứng của nó

Tên lệnh	Ý nghĩa
<b>ADD</b>	Cộng
<b>ADDD</b>	Cộng số có dấu chấm động, chính xác kép
<b>SUB</b>	Trừ
<b>SUBD</b>	Trừ số có dấu chấm động, chính xác kép
<b>MUL</b>	Nhân
<b>DIV</b>	Chia
<b>INC</b>	Tăng lên 1
<b>DEC</b>	Giảm đi 1
<b>NEG</b>	Đảo dấu toán hạng

Bảng 6.6. Các lệnh tính toán số học cơ bản

**6.4.3. Nhóm lệnh logic:**

Thực hiện phép tính logic NOT, AND và OR cho từng bit một. Lệnh NOT đảo tất cả các bit trong toán hạng, các lệnh AND/OR thực hiện các phép tính AND/OR đối với một đôi bit trong toán hạng nguồn và toán hạng đích.

**AND/OR**

Dạng tổng quát của lệnh AND/OR như sau:

AND đích, nguồn

OR đích, nguồn

AND/OR thực hiện phép toán Boolean đối với các toán hạng nguồn và đích. Phép AND thường dùng để che đi hoặc giữ lại một vài bit nào đó của một toán hạng bằng cách nhân logic toán hạng đó với toán hạng tức thời có các bit 0/1 tại các vị trí cần che/ giữ lại tương ứng. Phép OR thường dùng để lập một vài bit nào đó của toán hạng bằng cách cộng logic toán hạng đó với toán hạng tức thời có các bit 1 tại các vị trí tương ứng cần thiết lập (toán hạng tức thời trong những trường hợp này còn được gọi là mặt nạ)

Ví dụ:

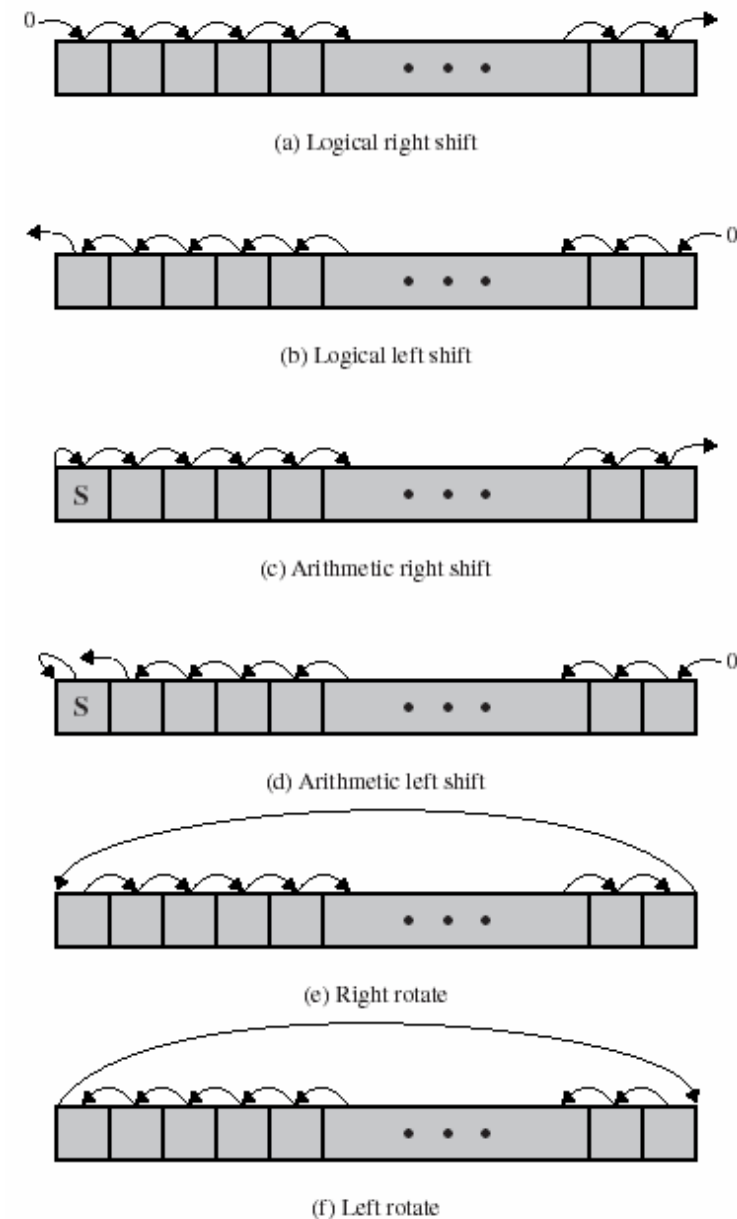
AND AL, BL // Nội dung thanh ghi BL được giao  
// với nội dung trong thanh ghi AL

```
// và kết quả được lưu lại vào trong
// thanh ghi AL. Nếu con số trong
// AL là 00001101B và trong BL là
// 00110011B thì kết quả trong thanh
// ghi AL sau phép AND là 00000001B
```

**6.4.4 Nhóm các lệnh dịch chuyển số học hoặc logic (SHIFT), quay vòng (ROTATE)** có hoặc không có số giữ ở ngõ vào, sang phải hoặc sang trái. Các lệnh này được thực hiện trên một thanh ghi và kết quả lưu giữ trong thanh ghi khác. Số lần dịch chuyển (mỗi lần dịch sang phải hoặc sang trái một bit) thường được xác định trong thanh ghi thứ ba. Hình 6.11 minh họa cho các lệnh cơ bản nhất của nhóm này.

- SRL (Shift Right Logical - dịch phải logic): Các bit của word được dịch chuyển sang phải, bit thấp nhất (bit 0) mất đi còn trị nhị phân “0” sẽ dịch chuyển vào bit cao nhất
- SLL (Shift Left Logical - dịch trái logic): Các bit của word được dịch chuyển sang trái, trị nhị phân “0” sẽ dịch chuyển vào bit thấp nhất (bit 0), còn bit cao nhất sẽ mất đi
- SRA (Shift Right Arithmetic - dịch phải số học): Bit cao nhất là bit dấu sẽ được giữ lại, các bit còn lại sẽ dịch chuyển sang phải còn bit thấp nhất sẽ mất đi.
- SLA (Shift Left Arithmetic – dịch trái số học): Bit cao nhất là bit dấu sẽ được giữ nguyên, các bit dịch sang trái, bit kế bit dấu mất đi, trị nhị phân “0” dịch chuyển vào bit thấp nhất.

Tương tự cho các lệnh quay vòng như trong hình 6.11.



Hình 6.11. Các lệnh dịch chuyển và quay vòng

#### 6.4.5. Nhóm các lệnh có điều kiện và lệnh nhảy (không điều kiện)

##### Lệnh có điều kiện có dạng :

Nếu <điều kiện> thì <chuỗi lệnh 1> nếu không <chuỗi lệnh 2>  
(IF <condition> THEN <instructions1> ELSE <instructions2>)

Lệnh này buộc phải ghi nhớ điều kiện và nhảy vòng nếu điều kiện được thoả.

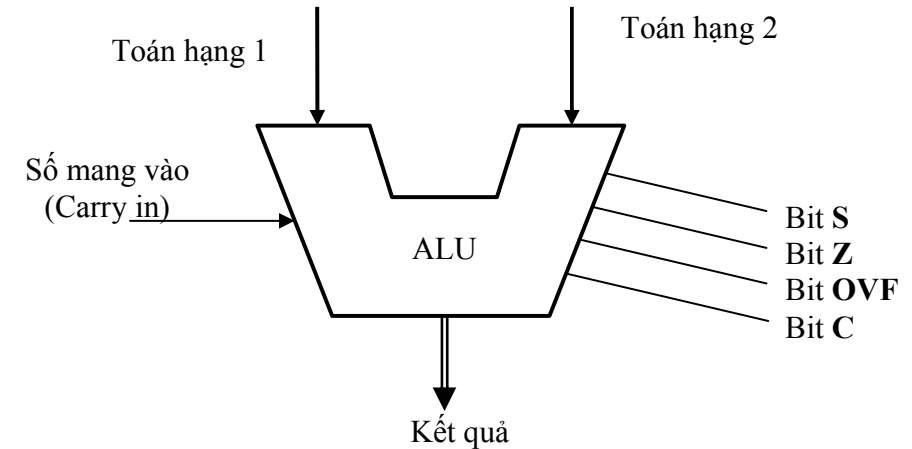
##### *Ghi nhớ điều kiện .*

Bộ tính toán logic số học ALU cung cấp kết quả ở ngã ra tùy theo các ngã vào và phép tính cần làm. Nó cũng cho một số thông tin khác về kết quả dưới dạng các bit trạng thái (Hay các bit cờ - flag). Các bit này là những đại lượng logic **ĐÚNG** hoặc **SAI** (hình 6.12).

Trong các bit trạng thái ta có:

- + bit dấu **S** (Sign - Đúng nếu kết quả âm),
- + bit trắc nghiệm zero **Z** (Zero - Đúng nếu kết quả bằng không),
- + bit tràn **OVF** (Overflow) ĐÚNG nếu phép tính số học làm thanh ghi không đủ khả năng lưu trữ kết quả,
- + bit số giữ C (carry) ĐÚNG nếu số giữ ở ngã ra là 1 ....

Các bit trên thường được gọi là bit mã điều kiện.



Hình 6.12. Các bit trạng thái mà ALU tạo ra

Có hai kỹ thuật cơ bản để ghi nhớ các bit trạng thái

**Cách thứ nhất**, ghi các trạng thái trong một thanh ghi đa dụng.

Ví dụ lệnh CMP Rk, Ri, Rj

Lệnh trên sẽ làm phép tính trừ Ri - Rj mà không ghi kết quả phép trừ, mà lại ghi các bit trạng thái vào thanh ghi Rk. Thanh ghi này được dùng cho một lệnh nhảy có điều kiện. Điểm lợi của kỹ thuật này là giúp lưu trữ nhiều trạng thái sau nhiều phép tính để dùng về sau. Điểm bất lợi là phải dùng một thanh ghi đa dụng để ghi lại trạng thái sau mỗi phép tính mà số thanh ghi này lại bị giới hạn ở 32 trong các bộ xử lý hiện đại.

**Cách thứ hai**, là để các bit trạng thái vào một thanh ghi đặc biệt gọi là thanh ghi trạng thái. Vấn đề lưu giữ nội dung thanh ghi này được giải quyết bằng nhiều cách. Trong kiến trúc SPARC, chỉ có một số giới hạn lệnh được phép thay đổi thanh ghi trạng thái ví dụ như lệnh ADDCC, SUBCC (các lệnh này thực hiện các phép tính cộng ADD và phép tính trừ SUB và còn làm thay đổi thanh ghi trạng thái). Trong kiến trúc PowerPC, thanh ghi trạng thái được

phân thành 8 trường, mỗi trường 4 bit, vậy là thanh ghi đã phân thành 8 thanh ghi trạng thái con.

Đặc tính chung của các lệnh này là thứ tự thực thi chương trình thay đổi nhờ tác động vào giá trị lưu trong thanh ghi đếm chương trình (Program Counter - PC). Sự thay đổi trong thanh ghi PC có thể là **không có điều kiện**, ví dụ như khi chương trình thực hiện đến một chỗ nào đó rồi cần phải nhảy đến một vị trí khác, khi đó ta sử dụng lệnh nhảy (Jump instruction). Trường hợp này trong ngôn ngữ cấp cao ta hay gặp đó là lệnh GOTO, khi đó giá trị được nạp vào trước trong thanh ghi PC sẽ bị xóa đi và một lệnh mới trong bộ nhớ sẽ được nạp vào.

Thanh ghi PC có thể thay đổi có điều kiện, mà những điều kiện này chủ yếu dựa vào các cờ như bit dấu (S), bit Zero (Z), Overflow (O) và bit Carry (C). Những bit cờ này được lưu trữ trên một thanh ghi đặc biệt gọi là thanh ghi mã điều kiện (Condition Code register - CC). Giá trị của các bit cờ này thay đổi dựa vào kết quả thực thi của các lệnh khác nhau.

Chúng ta xem một ví dụ cụ thể sau:

<i>LOAD</i>	<i>R1, #100</i>
<i>Loop: ADD</i>	<i>R0, (R2)+</i>
<i>DECREMENT</i>	<i>R1</i>
<i>BEQZ</i>	<i>R1, Loop</i>

Câu lệnh thứ 4 là một lệnh có điều kiện. Khi kết quả tăng giá trị trong thanh ghi *R1* là bằng 0, thì flag *Z* sẽ chuyển thành 1 và lệnh tiếp theo được thực thi sẽ là lệnh ở vị trí nhãn *Loop*.

## 6.5. Cấu trúc lệnh CISC và RISC

Trong máy tính mọi thứ đều được đưa về các con số nhị phân “0” và “1” bởi vì máy tính chỉ hiểu các mức điện thế tương ứng với 0/1 trên mỗi transistor cụ thể, người sử dụng muốn thực hiện một chương trình nào đấy, phải nạp các mã lệnh chỉ gồm các con số 0-1 vào bộ nhớ cho máy tính. Có 3 cách cơ bản để làm việc ấy:

- Viết ngay dạng mã máy với các con số 0, 1 và nạp vào bộ nhớ. Cách này rất khó thực thi bởi vì thứ nhất rất dễ bị nhầm lẫn giữa các con số 0 và 1; thứ hai rất khó nhớ các lệnh được mã hóa như thế nào và thứ ba là rất mất thời gian để làm việc đó.
- Viết dạng tên gọi nhớ bằng hợp ngữ (Assembler), sau đó biên dịch ra mã máy, cấp này cũng rất gần với ngôn ngữ máy và cũng khó thực hiện với các chương trình phức tạp. Tuy nhiên, cấu trúc gọn nhẹ, các lệnh có tên tương ứng dễ nhớ.
- Viết bằng một ngôn ngữ cấp cao như C++, Pascal, Java,..., sau đó dùng một trình biên dịch (compiler) để dịch ra mã máy. Cách này tuy dễ với người viết chương trình nhưng cũng sẽ làm chương trình có dung lượng lớn hơn nếu viết bằng ASM. Và thách thức là làm sao các nhà sản xuất phần mềm, phần cứng bắt tay nhau để chương trình biên dịch này thật chuẩn tắc, nhỏ gọn, không tạo nhiều code trung gian.

Trong suốt thập niên 1980, các nhà thiết kế cố gắng thu hẹp khoảng cách giữa ngôn ngữ cấp cao của con người và ngôn ngữ máy, họ đã đưa ra cấu trúc với các chỉ lệnh phức tạp gọi là CISC (Complex Instruction Set Computer), có các chế độ định địa chỉ khác nhau, mỗi lệnh thực thi cần nhiều lần định địa chỉ để lấy dữ liệu, và do đó, tốn nhiều chu kì xung nhịp cho mỗi chỉ lệnh.

Nếu việc giảm thiểu ranh giới giữa tập lệnh của vi điều khiển và ngôn ngữ cấp cao không phải là một cách hay để máy tính hoạt động hiệu quả, các nhà thiết kế phải làm sao để tối ưu tốc độ xử lý?

Nếu muốn biết cách làm để vi xử lý hoạt động nhanh hơn, ta phải biết vi xử lý dùng hầu hết thời gian của chúng vào việc gì? Chúng ta dễ nghĩ rằng: Vi xử lý tất nhiên dùng hầu hết thời gian của nó để tính toán; nghĩa là thời gian hầu hết ở bộ ALU. Thật ra, theo thống kê (xem bảng 6.7) thì suy đoán này hoàn toàn sai lầm:

Loại lệnh	% sử dụng thời gian
Chuyển dữ liệu	43%
Điều khiển dòng chảy	23%
Tính toán số học	15%
So sánh	13%
Phép toán Logic	5%
Các lệnh khác	1%

Bảng 6.7. Thống kê thời gian thực hiện các loại lệnh

#### ➤ So sánh CISC và RISC

Sự khác biệt cơ bản giữa các chip dòng máy tính với tập lệnh rút gọn RISC (reduced instruction set computer) và máy tính với tập lệnh phức tạp CISC (complex instruction set computer, chẳng hạn như dòng chip x86 của Intel) có thể được xem như cuộc ganh đua giữa nhà lập trình và nhà thiết kế chip. Chip CISC được thiết kế nhằm tạo thuận lợi cho các nhà lập trình ứng dụng bằng cách rút gọn nhiều câu lệnh đơn giản, thông dụng thành một câu lệnh thực thi dài. Điều này làm cho CISC xử lý chậm hơn nhưng lại đạt yếu tố thân thiện. Ở mặt khác, RISC nhanh nhưng kém thân thiện hơn, mỗi câu lệnh đơn giản trong RISC phục vụ cho một mục đích hẹp rất cụ thể, thực hiện rất nhanh và các lệnh này được tiến hành song song. RISC đòi hỏi nhà lập trình phải kiên nhẫn, giỏi và một trình biên dịch được tối ưu kỹ lưỡng.

#### ➤ Điểm mạnh của bộ xử lý dùng tập lệnh RISC:

- Kích thước miếng bán dẫn nhỏ hơn: bộ xử lý đơn giản đòi hỏi ít transistor hơn, do đó, kích thước cần dùng nhỏ lại, dành vùng diện tích trống để tăng các chức năng như bộ nhớ cache, chức năng quản lý bộ nhớ, ..vv...

- Thời gian phát triển một sản phẩm ngắn hơn do kỹ thuật đơn giản hơn
- Tốc độ xử lý tăng lên đáng kể. Khi ta đặt ra các chỉ lệnh phức tạp, tuy nó gần gũi với ngôn ngữ cấp cao, nhưng như thế, vô tình cũng làm các chỉ lệnh khác phức tạp lên, và để thực thi một chỉ lệnh như vậy cần tốn nhiều chu kỳ xung nhịp. Trong khi đó, nếu dùng RISC chỉ mất một chu kỳ xung nhịp cho mỗi lệnh, khi ta phân nhỏ vấn đề phức tạp thành các vấn đề đơn giản thì cách giải quyết sẽ tốt hơn.

#### ➤ Các điểm bất lợi của RISC:

Không phải RISC chỉ có điều thuận lợi, nó cũng có một vài bất cập, mà cụ thể là:

- Tập lệnh của RISC không phong phú bằng CISC, như vậy khi cần thiết kế một chương trình nào đó mà không có lệnh cần thiết thì phải thông qua một loạt các lệnh khác làm tăng lên số chu kỳ xung nhịp cần thiết, tức là đã làm chậm hệ thống đi.
- Cắm thâm nhập bộ nhớ đối với tất cả các lệnh ngoại trừ các lệnh đọc và ghi vào bộ nhớ. Do đó ta buộc phải dùng nhiều lệnh để làm một công việc nhất định.
- Cần thiết phải tính các địa chỉ hiệu dụng vì không có nhiều cách định vị.
- Không thể thực thi các mã lệnh của x86, một kiểu kiến trúc máy tính đã quá thông dụng và quen thuộc với mọi người. Như vậy để tích hợp được phải dùng các phần mềm hỗ trợ nền cơ sở cho RISC, tuy nhiên, với máy tính của IBM, có thể bị từ chối.

Tóm lại các điểm khác biệt cơ bản giữa hai kiểu kiến trúc RISC và CISC có thể liệt kê như trong bảng 6.8.

RISC	CISC
<ul style="list-style-type: none"> <li>– Kích thước các lệnh (độ dài lệnh) là cố định ( 32 bit) với chỉ một vài định dạng.</li> <li>– Sử dụng kiến trúc load-store các lệnh xử lý dữ liệu hoạt động chỉ trong thanh ghi và cách ly với các lệnh truy cập bộ nhớ</li> <li>– Một số lớn các thanh ghi đa dụng 32 bit, cho phép cấu trúc load-store hoạt động hiệu quả.</li> <li>– Có một số ít lệnh (thường dưới 100 lệnh)</li> <li>– Có một số ít các kiểu định vị ( thường là định vị tức thì và định vị gián tiếp qua một thanh ghi).</li> <li>– Có một số ít dạng lệnh (một hoặc hai)</li> <li>– Chỉ có các lệnh ghi hoặc đọc ô nhớ mới thâm nhập vào bộ nhớ.</li> </ul>	<ul style="list-style-type: none"> <li>– Kích thước tập lệnh thay đổi với rất nhiều định dạng khác nhau</li> <li>– Cho phép giá trị trong bộ nhớ được dùng như như toán hạng trong các chỉ lệnh xử lý dữ liệu</li> <li>– Có rất nhiều thanh ghi, nhưng hầu hết chỉ để sử dụng cho một mục đích riêng biệt nào đấy</li> <li>– Có rất nhiều lệnh (khoảng 500)</li> <li>– Có nhiều kiểu định vị (xem phần 6.3.4)</li> <li>– Có nhiều dạng lệnh</li> <li>– Có nhiều lệnh khác cũng thâm nhập vào bộ nhớ được</li> </ul>
<ul style="list-style-type: none"> <li>– Giải mã các lệnh logic bằng kết nối phần cứng</li> <li>– Thực thi chỉ lệnh theo cấu trúc dòng chảy (xem hình 7.9 trong chương sau)</li> <li>– Một lệnh thực thi trong 1 chu kì xung nhịp</li> </ul>	<ul style="list-style-type: none"> <li>– Sử dụng rất nhiều code trong ROM giải mã các chỉ lệnh</li> <li>– Các máy cũ ít khi cho phép các dòng lệnh thực thi kiểu này, chúng phải tuần tự hết dòng lệnh này mới đến dòng lệnh khác</li> <li>– Cần nhiều chu kì xung nhịp để hoàn thành một lệnh</li> </ul>

Bảng 6.8. Các điểm khác nhau cơ bản giữa RISC và CISC

## CÂU HỎI VÀ BÀI TẬP CHƯƠNG VI

1. Có mấy loại kiến trúc bộ lệnh cơ bản? Nêu các ưu và nhược điểm của từng loại
2. Cho biết các đặc tính cơ bản của kiểu kiến trúc thanh ghi đa dụng.
3. Địa chỉ bộ nhớ được sắp xếp như thế nào? Giữa cách của Intel và Motorola khác biệt nhau gì? Vương mắc gì có thể xảy ra khi máy tính của hai hãng này kết nối với nhau và đưa ra ví dụ cho sự rắc rối này?
4. Cho biết cách mã hóa tập lệnh và đưa ra một vài dạng mã hóa lệnh cơ bản.
5. Hãy cho biết và giải thích các tiêu chuẩn thiết kế dạng thức lệnh.
6. Giả sử cần thiết kế máy với ký tự 8 bit và bộ nhớ chính chứa  $2^{24}$  ký tự. Hãy cho biết trường địa chỉ cần bao nhiêu bit trong trường hợp:
  - a) Ô nhớ kích thước 8 bit
  - b) Ô nhớ kích thước 16 bit
  - c) Ô nhớ kích thước 32 bit
7. Thiết kế opcode mở rộng nhằm cho phép mã hóa nội dung sau trong lệnh 36 bit
  - a) 7 lệnh có hai địa chỉ 15 bit và một số hiệu thanh ghi 3 bit
  - b) 500 lệnh có một địa chỉ 15 bit và một số hiệu thanh ghi 3 bit
  - c) 50 lệnh không có địa chỉ hoặc thanh ghi
8. Có thể thiết kế opcode mở rộng để cho phép mã hóa nội dung sau trong lệnh 12 bit được không? Trường thanh ghi rộng 3 bit.

- a) 4 lệnh có ba thanh ghi
  - b) 255 lệnh có hai thanh ghi
  - c) 2048 lệnh không có thanh ghi
9. Cho biết các chế độ lập địa chỉ và các ưu, nhược điểm của từng loại. Mô tả bằng hình các cách lập địa chỉ đó.
10. Mô tả các kiểu thi hành lệnh của một máy tính. Tại sao kiểu thi hành lệnh thanh ghi – thanh ghi được dùng nhiều hiện tại?
11. Hãy diễn giải quá trình biên dịch ra ngôn ngữ máy từ các ngôn ngữ cấp cao
12. Các lệnh máy tính được phân ra những nhóm lệnh nào? đưa ra một ví dụ cho từng nhóm lệnh.
13. Mô tả bằng hình vẽ các lệnh dịch chuyển và quay vòng và giải thích tác dụng của các lệnh.
14. Hãy cho biết một số bit trạng thái mà ALU tạo ra và cách dùng các bit này trong các lệnh nhảy
15. Hãy phân biệt sự khác nhau giữa hai kiểu kiến trúc máy tính RISC và CISC.

## Chương VII: TỔ CHỨC BỘ XỬ LÝ

Trong các chương trước chúng ta đã các khái niệm cơ bản liên quan đến các phần khác nhau của máy tính. Trong chương này chúng ta tập trung vào bộ phận chính yếu của mọi máy tính – bộ xử lý trung tâm (central processing unit - CPU). Chức năng chính yếu của CPU là thực thi các lệnh được lưu trong bộ nhớ chính. Như đã đề cập trong phần giới thiệu, do giáo trình này được thiết kế dành cho sinh viên học ngay học kỳ đầu tiên cho nên chúng ta chỉ đề cập đến các vấn đề ở mức độ đơn giản.

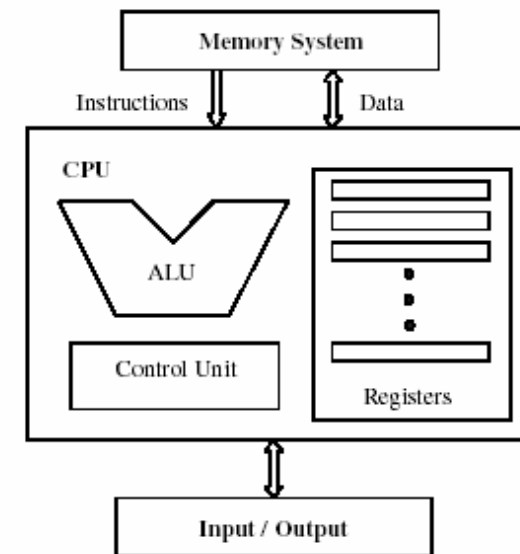
### 7.1. Tổ chức bộ xử lý trung tâm

Để hiểu được cách tổ chức trong CPU như thế nào chúng ta hãy xem những đòi hỏi ở bên trong nó và những gì nó phải thực hiện. Nói chung những công việc nó phải làm là:

- Tìm nạp lệnh (Fetch Instruction): CPU đọc một lệnh từ bộ nhớ
- Diễn giải lệnh (Interpret Instruction): Lệnh được giải mã và xác định xem hành động gì đòi hỏi thực hiện
- Tìm nạp dữ liệu (Fetch data): Để thực hiện câu lệnh có thể đòi hỏi đọc dữ liệu từ bộ nhớ hoặc từ các thiết bị nhập xuất
- Xử lý dữ liệu (Process data): Việc thi hành lệnh có thể đòi hỏi thực hiện một vài phép tính số học hay logic trên các dữ liệu tìm nạp vào
- Ghi dữ liệu (Write data): Kết quả thực thi lệnh cũng có thể đòi hỏi phải ghi dữ liệu vào bộ nhớ hoặc ghi ra các thiết bị nhập xuất.

Để làm được các việc như vậy đòi hỏi CPU cần lưu một số dữ liệu tạm thời. Nó phải nhớ vị trí của lệnh vừa thực hiện cũng như nó cần phải biết được ở đâu có thể nhận được lệnh tiếp theo để thực hiện. Nó cần lưu tập lệnh và dữ liệu tạm thời trong khi thực thi một lệnh nào đó. Nói cách khác CPU cần một bộ nhớ trong nhỏ để tiện lợi làm việc.

Nói chung để thực hiện các việc vừa liệt kê ở trên, một bộ xử lý trung tâm đơn giản cần bao gồm ba nguyên tố quan trọng nhất: tập các thanh ghi, một bộ xử lý logic số học (Arithmetic Logic Unit - ALU) và một bộ điều khiển (Control Unit - CU). Tổ chức một máy tính đơn giản gồm CPU và tương tác của nó với bộ nhớ chính và các thiết bị nhập xuất có thể biểu diễn như hình 7.1.



Hình 7.1. Tổ chức một máy tính đơn giản

Bộ các thanh ghi của các loại máy tính là khác nhau và thường chia làm hai loại, các thanh ghi được dùng với mục đích chung (*general-purpose registers*), và các thanh ghi với mục đích đặc biệt (*special-purpose registers*). Các thanh ghi mục đích chung



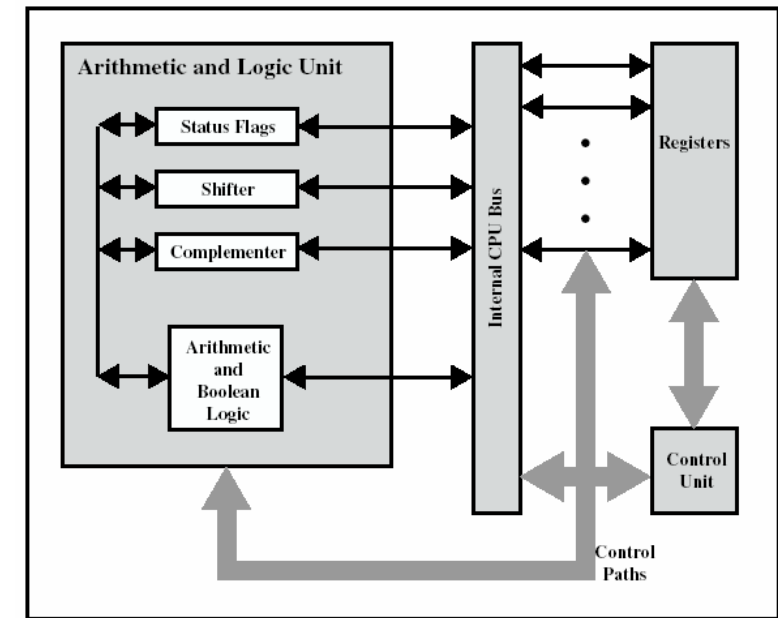
được sử dụng cho bất kỳ mục đích nào, ví dụ như có thể dùng nó làm nơi lưu trữ các loại dữ liệu khác nhau, dùng làm lưu các toán hạng của một lệnh. Trong khi các thanh ghi với mục đích đặc biệt chỉ có một số chức năng bên trong CPU, ví dụ như bộ đếm chương trình (program counter - PC) là một thanh ghi với mục đích đặc biệt đó là chỉ dùng để lưu địa chỉ của lệnh cần thực hiện tiếp theo. Một ví dụ khác của thanh ghi với mục đích đặc biệt là thanh ghi mã lệnh (Instruction Registers – IR), thanh ghi này chỉ được dùng để lưu mã lệnh đang thực hiện tại thời điểm hiện tại.

Bộ logic số học ALU cung cấp một mạch số cần thiết để thực hiện các phép tính số học (như cộng, trừ, nhân, chia), logic (như NOT, AND, OR) và các phép toán dịch chuyển trong bộ lệnh.

Bộ điều khiển (**control block**) chịu trách nhiệm điều khiển mọi hoạt động của CPU như tìm nạp lệnh từ bộ nhớ chính, giải mã nó, định loại xem nó thuộc loại nào và cuối cùng là ra lệnh cho các bộ phận trong CPU làm việc gì.

CPU tìm nạp lệnh từ bộ nhớ chính, đọc dữ liệu từ bộ nhớ chính, ghi dữ liệu ngược lại đó khi cần và truyền các dữ liệu từ các thiết bị ngoại vi vào máy tính cũng như đưa các dữ liệu trong máy tính ra các thiết bị ngoại vi. Các thiết bị ngoại vi ở đây có thể là ổ đĩa cứng, máy in, flash memory,...

Trong hình 7.2 cho ta sơ đồ tổng quát của CPU một cách tỷ mỉ hơn. Trong đó chỉ ra việc truyền dữ liệu và đường điều khiển logic cũng được chỉ ra bao gồm cả một nguyên tố gọi là trục CPU nội (*Internal CPU bus*). Nguyên tố này được cần để truyền dữ liệu giữa các thanh ghi khác nhau và ALU, bởi vì trên thực tế thì ALU thực thi chỉ trên các dữ liệu nằm ở bộ nhớ bên trong CPU.



Hình 7.2. Cấu trúc bên trong của CPU

## 7.2. Bộ điều khiển

Giống như trong một trung đội thì mọi hoạt động được điều khiển bởi trung đội trưởng, bộ điều khiển chịu trách nhiệm điều khiển mọi hoạt động của CPU. Bộ điều khiển tạo các tín hiệu điều khiển di chuyển số liệu (tín hiệu di chuyển số liệu từ các thanh ghi đến bus hoặc tín hiệu viết vào các thanh ghi), điều khiển các tác vụ mà các bộ phận chức năng phải làm (điều khiển ALU, điều khiển đọc và viết vào bộ nhớ trong...). Bộ điều khiển cũng tạo các tín hiệu giúp các lệnh được thực hiện một cách tuần tự.

Để thiết kế một bộ điều khiển, ta có thể dùng một trong hai cách là dùng mạch điện tử hoặc dùng vi chương trình (microprogram).

## *Bộ điều khiển mạch điện tử*

Trong chương 5 chúng ta đã học về mạch tuần tự. Một bộ điều khiển mạch điện tử có nguyên lý hoạt động như một mạch tuần tự. Tức là sẽ có các trạng thái và các đường tín hiệu mà sự thay đổi của các tín hiệu tại một thời điểm nhất định sẽ làm trạng thái giữ nguyên hoặc chuyển sang một trạng thái mới.

Để hiểu được vận hành của bộ điều khiển mạch điện tử, chúng ta xét đến mô tả về Automate (mạch tự động hóa) trạng thái hữu hạn như một mạch tuần tự có nhiều hệ thống hay nhiều thành phần mà ở mỗi thời điểm xem xét đều có một trạng thái (state).

Mục đích của trạng thái là ghi nhớ những gì có liên quan trong quá trình hoạt động của hệ thống. Vì chỉ có một số trạng thái nhất định nên nói chung không thể ghi nhớ hết toàn bộ lịch sử của hệ thống, do vậy nó phải được thiết kế cẩn thận để ghi nhớ những gì quan trọng.

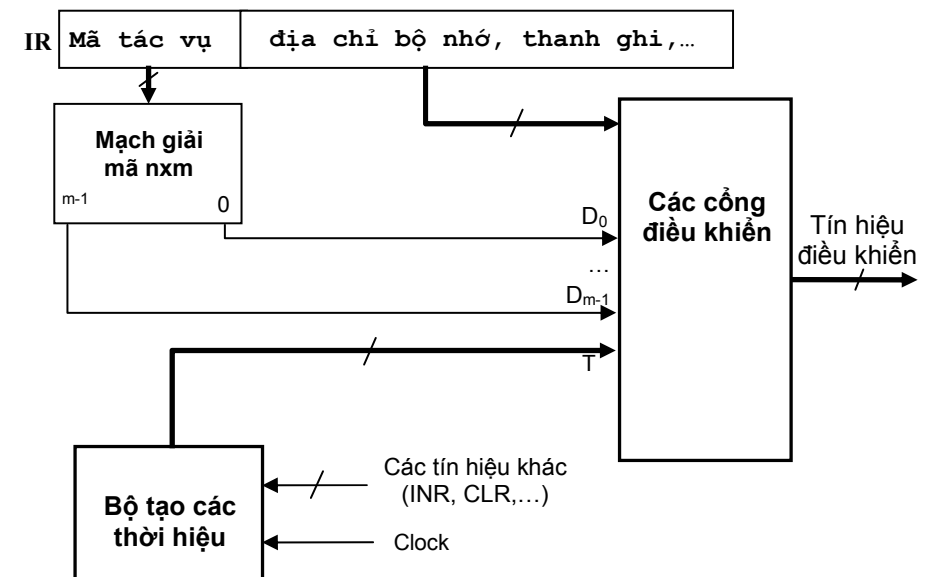
Ưu điểm của hệ thống (chỉ có một số hữu hạn các trạng thái) đó là có thể cài đặt hệ thống với một lượng tài nguyên cố định. Chẳng hạn, chúng ta có thể cài đặt Automate trạng thái hữu hạn trong phần cứng máy tính ở dạng mạch điện hay một dạng chương trình đơn giản, trong đó, nó có khả năng quyết định khi chỉ biết một lượng giới hạn dữ liệu hoặc bằng cách dùng vị trí trong đoạn mã lệnh để đưa ra quyết định.

Theo tổ chức điều khiển cứng, luận lý điều khiển được cài đặt qua các cổng , mạch lật, mạch giải mã và các mạch số khác. Thuận lợi là có thể tối ưu để tạo ra chế độ nhanh cho tác vụ

Theo tổ chức điều khiển vi trình, thông tin điều khiển được lưu trong bộ nhớ điều khiển, bộ nhớ điều khiển được lập trình để khởi động dãy vi tác vụ theo yêu cầu. Khi thay đổi thiết kế, cần thay đổi cách kết nối các thành phần nếu tổ chức theo kiểu điều khiển cứng. Nếu tổ chức theo điều khiển vi trình chỉ cần cập nhật vi trình trong bộ nhớ điều khiển.

Hình 7.3. cho ta sơ đồ khối một bộ điều khiển cơ bản. Lệnh được đọc từ bộ nhớ và đưa vào thanh ghi lệnh IR. Thanh ghi lệnh thì như trong chương 6 chúng ta đã biết, trong đó có phần mã hóa vi tác vụ hay opcode sẽ được qua một mạch giải mã để biết được mã tác vụ phải làm là gì. Sau khi giải mã, các tín hiệu này được đưa vào mạch điều khiển gồm các cổng điều khiển.

Để điều khiển các hoạt động được đồng bộ, ta dùng một bộ tạo các thời hiệu, mỗi thời hiệu ứng với một thời điểm nhất định. Các thời hiệu này cùng với các tín hiệu từ bộ giải mã và từ các tín hiệu khác trong phần còn lại của thanh ghi IR sau khi qua các cổng điều khiển sẽ cho ta các tín hiệu điều khiển cần thiết. Các tín hiệu điều khiển có thể là đưa dữ liệu một thanh ghi nào đó lên BUS, đưa dữ liệu từ trên BUS vào thanh ghi, mở tín hiệu ghi vào một thanh ghi, đưa ra tín hiệu đọc/ghi vào bộ nhớ chính, khởi động ngắt để truyền dữ liệu ra HDD, máy in,...



Hình 7.3. Sơ đồ khối bộ điều khiển máy tính cơ bản

### Bộ điều khiển vi chương trình

Thay vì bộ điều khiển cứng dùng mạch tổ hợp các cổng để cho ra các tín hiệu điều khiển thì điều khiển vi trình dùng một vi chương trình lập sẵn nằm trong bộ nhớ điều khiển để khởi động dãy vi tác vụ theo yêu cầu.

Bộ điều khiển bằng vi chương trình được dùng rộng rãi trong các bộ xử lý CISC. Bộ xử lý này có tập lệnh phức tạp với các lệnh có chiều dài khác nhau và có dạng thức phức tạp. Trong các bộ xử lý CISC, người ta cài đặt một lệnh mã máy bằng cách viết một vi chương trình. Như vậy công việc khá đơn giản và rất hữu hiệu. Các sai sót trong thiết kế automat điều khiển cũng dễ sửa đổi.

### 7.3. Bộ thanh ghi

Các thanh ghi là một loại bộ nhớ đặc biệt nhanh nằm bên trong CPU và được dùng để tạo ra và lưu trữ các kết quả của các lệnh trong CPU và các phép toán khác. Các loại máy tính khác nhau có các bộ thanh ghi khác nhau. Chúng khác nhau về số lượng các thanh ghi, các loại thanh ghi và cả chiều dài của mỗi thanh ghi. Chúng cũng khác nhau trong cách dùng cho mỗi thanh ghi.

Thanh ghi mục đích chung có thể được dùng cho nhiều mục đích và được nhà lập trình dùng để gán cho những trạng thái khác nhau của các hàm, các biến.

Thanh ghi có mục đích đặc biệt chỉ được dùng cho một số chức năng đặc biệt. Trong một số trường hợp, một vài thanh ghi chỉ được dùng để lưu trữ dữ liệu và không thể dùng trong tính toán địa chỉ của các toán hạng.

Chiều dài của thanh ghi phải đủ để lưu trữ hầu hết các loại dữ liệu, đặc biệt là thanh ghi địa chỉ phải đủ dài để có thể lưu được địa chỉ lớn nhất.

Số lượng thanh ghi trong một kiến trúc máy tính ảnh hưởng trực tiếp đến việc thiết kế bộ lệnh. Nếu số lượng thanh ghi ít sẽ dẫn

đến việc phải sử dụng tham chiếu hay con trỏ tới bộ nhớ nhiều hơn làm giảm đáng kể đến tốc độ thực thi của chương trình.

Một dạng khác của các thanh ghi được dùng để lưu các bit trạng thái của bộ xử lý hay là các cờ trạng thái. Những bit này thay đổi tùy theo kết quả thực hiện một lệnh nào đó trong CPU

### Thanh ghi truy cập bộ nhớ

Có hai thanh ghi được dùng đặc biệt trong các lệnh đọc/ghi với bộ nhớ: thanh ghi dữ liệu bộ nhớ (*memory data register - MDR*) và thanh ghi địa chỉ bộ nhớ (*memory address register - MAR*). Hai thanh ghi này được CPU sử dụng đặc biệt và các nhà lập trình không thể truy cập trực tiếp vào chúng.

Trong thứ tự thực hiện một lệnh ghi vào một vùng nhớ đặc biệt, MDR và MAR được dùng như sau:

1. Một từ cần lưu vào một vị trí nào đó trong bộ nhớ, đầu tiên sẽ được CPU tải vào thanh ghi MDR
2. Địa chỉ của vùng nhớ đó (vùng nhớ sẽ ghi dữ liệu vào) được CPU tải vào trong MAR
3. Sau cùng bộ điều khiển sẽ phát ra tín hiệu điều khiển “ghi” (*write*) để ghi dữ liệu nằm trên MDR vào địa chỉ nằm trên MAR.

Tương tự khi thực hiện một lệnh đọc (*read*) từ bộ nhớ, thanh ghi MDR và MAR sẽ được dùng như sau:

1. Địa chỉ của từ nhớ nằm trong bộ nhớ cần đọc được đưa vào MAR
2. Bộ điều khiển sẽ phát ra tín hiệu điều khiển “đọc” (*read*) để đọc dữ liệu nằm trong bộ nhớ.
3. Dữ liệu nằm trong bộ nhớ có địa chỉ chứa trong MAR sẽ được tải vào trong MDR

### Thanh ghi chuyển tải lệnh

Có hai thanh ghi rất quan trọng được dùng để lưu các câu lệnh cho việc thực thi của chương trình là: Bộ đếm chương trình (**program counter – PC**) và thanh ghi lệnh (**instruction register – IR**).

Thanh ghi **PC** dùng để lưu địa chỉ của lệnh tiếp theo qua đó sẽ điều khiển thứ tự thực hiện các lệnh trong một chương trình. Các lệnh thực hiện lần lượt được tìm thấy và lưu vào trong thanh ghi **IR**. Tại mỗi thời điểm chỉ có một lệnh được lưu trong **IR**. Sau khi đã tìm và đem một lệnh về cho **IR**, thanh ghi **PC** sẽ được nạp một giá trị mới là địa chỉ của lệnh tiếp theo.

### Thanh ghi trạng thái

Các thanh ghi trạng thái hay là các cờ trạng thái được sử dụng để giữ các thông tin về trạng thái của quá trình thực thi lệnh. Một số kiến trúc máy tính chứa một thanh ghi đặc biệt gọi là thanh ghi từ trạng thái của chương trình (program status word – **PSW**). Trong **PSW** chứa các bit trạng thái cho biết kết quả của các phép tính số học (có bằng 0 không? có số thừa không?...), trạng thái ngắt, thông tin bảo vệ bộ nhớ, tình trạng của bộ xử lý,...

### Các thanh ghi họ 80x86

Trong các thế hệ máy tính 32 bit của Intel như 386, 486 và Pentium có ba nhóm thanh ghi, đó là:

- Thanh ghi mục đích chung
- Thanh ghi segment
- Thanh ghi đếm chương trình **PC** và thanh ghi cờ trạng thái

Trong hình 7.4 cho ta thấy ba nhóm này. Nhóm thứ nhất là các thanh ghi mục đích chung được đặt tên bằng các ký tự **A, B, C, D, SI** (source index), **DI** (destination index), **SP** (stack pointer) và **BP** (base pointer). Nhóm thứ hai bao gồm các thanh ghi **CS** (code segment), **SS** (stack segment) và 4 thanh ghi đoạn dữ liệu

**DS, ES, FS** và **GS**. Nhóm thanh ghi thứ ba bao gồm thanh ghi lệnh **IR** và thanh ghi trạng thái. Trong số những bit trạng thái, 5 bit đầu tiên đồng nhất với các bit của bộ vi xử lý 8 bit trước đây 8085. Các bit tiếp theo 6-11 giống như được giới thiệu trong 8086. Các bit 12-14 được giới thiệu trong 80286 trong khi các bit 16-17 trong 80386. Bit flag 18 trong 80486. Trong đó còn nhiều bit chưa được dùng tới và để làm dự trữ cho những mục đích có thể xuất hiện sau này.

31	16 15	8 7	0
EAX	AH	AL	
EBX	BH	BL	
ECX	CH	CL	
EDX	DH	DL	
ESI	SI		
EDI	DI		
ESP	SP		
EBP	BP		

Các thanh ghi mục đích chung

15	0
CS (Code segment pointer)	
SS (Stack segment pointer – top)	
DS (Data segment pointer 0)	
ES (Data segment pointer 1)	
FS (Data segment pointer 2)	
GS (Data segment pointer 3)	

Các thanh ghi đoạn

Thanh ghi lệnh

IR
----

E Flags	Flags H	Flags L
---------	---------	---------

Thanh ghi cờ

Hình 7.4. Các thanh ghi họ 80x86

## 7.4. Đường đi dữ liệu (Datapath)

Phần đường đi dữ liệu gồm có bộ logic-số học (ALU: Arithmetic and Logic Unit), các mạch dịch, các thanh ghi và các đường nối kết các bộ phận trên. Phần này chứa hầu hết các trạng thái của bộ xử lý. Ngoài các thanh ghi tổng quát, phần đường đi dữ liệu còn chứa thanh ghi đếm chương trình (PC: Program Counter), thanh ghi trạng thái (SR: Status Register), các thanh ghi địa chỉ bộ nhớ MAR, thanh ghi số liệu bộ nhớ MDR, bộ dồn kênh (MUX: Multiplexor), đây là điểm cuối của các kênh dữ liệu - CPU và bộ nhớ, với nhiệm vụ lập thời biểu truy cập bộ nhớ từ CPU và các kênh dữ liệu, các hệ thống bus

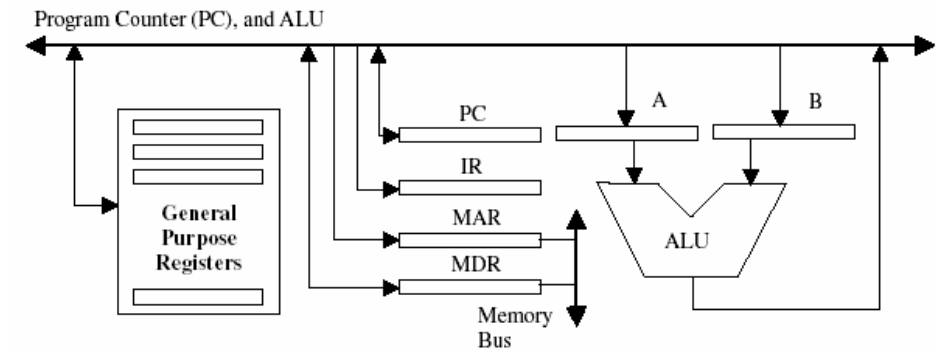
Nhiệm vụ chính của phần đường đi dữ liệu là đọc các toán hạng từ các thanh ghi tổng quát, thực hiện các phép tính trên toán hạng này trong ALU và lưu trữ kết quả trong các thanh ghi tổng quát. Ở ngõ vào và ngõ ra các thanh ghi tổng quát có các mạch chốt A, B, C. Như đã đề cập ở trên, thông thường, số lượng các thanh ghi tổng quát là 32.

Phần đường đi của dữ liệu chiếm phân nửa diện tích của bộ xử lý nhưng là phần dễ thiết kế và cài đặt trong bộ xử lý.

### 7.4.1. Tổ chức One-Bus

Các thanh ghi CPU và ALU dùng một *BUS* đơn để di chuyển dữ liệu ra vào giữa chúng với nhau. Vì một *BUS* chỉ có thể sử dụng một dữ liệu di chuyển trong một chu kỳ đồng hồ, cho nên một phép toán có hai toán hạng sẽ phải cần hai chu kỳ đồng hồ để tìm và nạp các toán hạng cho ALU. Dạng tổ chức *BUS* này là đơn giản nhất nhưng nó bị nhiều hạn chế về số lượng dữ liệu được chuyển tải trong một chu kỳ đồng hồ nói chung đã làm chậm cả hệ thống lại rất nhiều.

Hình 7.5 cho ta sơ đồ tổ chức đường đi dữ liệu one-bus. Nó bao gồm bộ các thanh ghi đa mục đích, một thanh ghi địa chỉ bộ nhớ MAR, một thanh ghi dữ liệu bộ nhớ MDR, một thanh ghi lệnh IR, một thanh ghi đếm chương trình PC và bộ logic số học ALU.



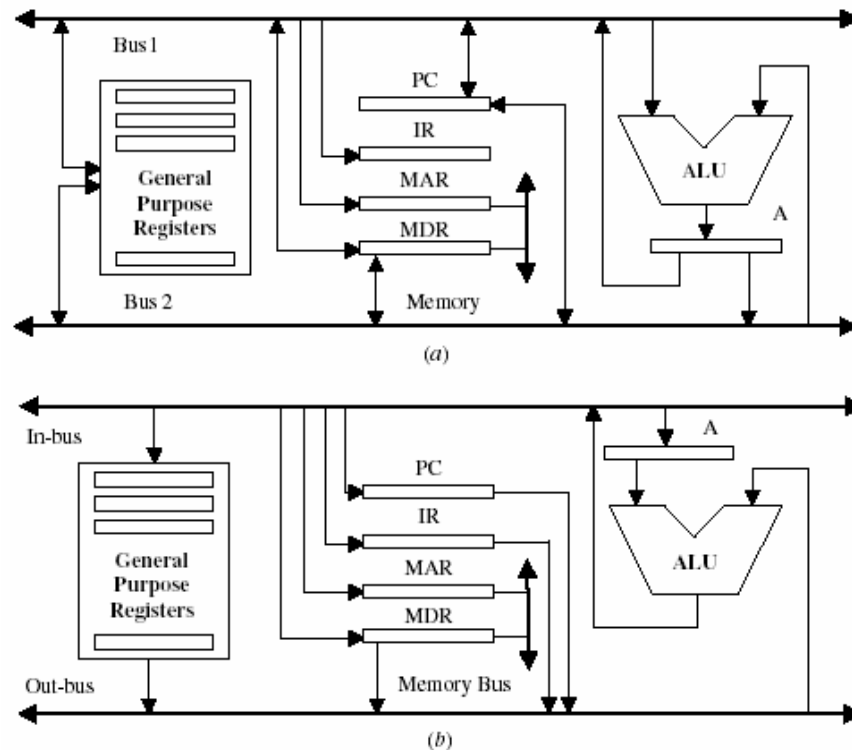
Hình 7.5. Tổ chức One-Bus

### 7.4.2. Tổ chức Two-Bus, Three-Bus

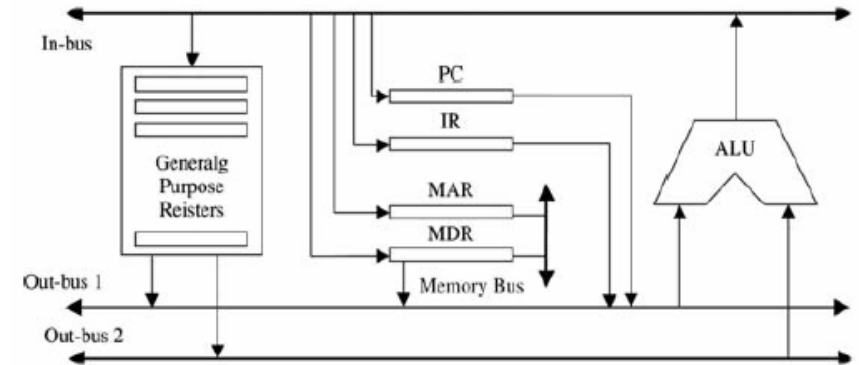
Sử dụng hai *BUS* để tổ chức đường đi dữ liệu là một giải pháp giúp cải thiện tốc độ rất nhiều so với công nghệ dùng one-bus. Trong trường hợp này các thanh ghi đa mục đích được nối với cả hai bus. Dữ liệu có thể truyền từ hai thanh ghi khác nhau đến đầu vào của ALU tại cùng một thời điểm. Do vậy, một lệnh có hai toán hạng có thể tìm và nạp trong cùng một chu kỳ đồng hồ và như vậy rõ ràng tốc độ xử lý lệnh lúc này đã tăng gấp đôi. Hơn nữa, trong cấu trúc này (hình 7.6) còn có một thanh ghi đệm (*buffer register*) có thể cần thiết để lưu trữ dữ liệu đầu ra của ALU khi hai bus đang bận chuyển tải dữ liệu.

Trong hình 7.6(a) cho ta thấy cách tổ chức kiến trúc kiểu two-bus này. Trong một số trường hợp, một bus có thể được dùng để di chuyển dữ liệu vào thanh ghi (*in-bus*). Ở trường hợp này một thanh ghi đệm thêm có thể được dùng như là một bộ nhớ tạm thời trước khi đưa vào ALU. Đầu ra của ALU được nối trực tiếp với *in-bus* và cho phép chuyển kết quả tính toán tới một trong các thanh ghi. Hình 7.6(b) cho ta tổ chức *two-bus* với *in-bus* và *out-bus*.

Trong tổ chức three-bus thì hai trong số đó được dùng như là các bus nguồn, bus còn lại như là bus đích. Các bus nguồn chuyển dữ liệu ra khỏi thanh ghi (*out-bus*), và bus đích có thể chuyển dữ liệu vào trong các thanh ghi (*in-bus*). Mỗi đầu của *out-bus* được nối với đầu vào của ALU. Đầu ra của ALU thì nối trực tiếp với *in-bus*. Như vậy chúng ta lại có thể truyền dữ liệu nhiều hơn trong cùng một chu kỳ đồng hồ. Tuy nhiên việc tăng các bus lên sẽ làm tăng độ phức tạp của phần cứng lên nhiều hơn, vì vậy tùy theo mức độ cần thiết mà các loại máy tính khác nhau sẽ có số lượng bus khác nhau. Tổ chức máy tính đơn giản loại three-bus có thể minh họa như hình 7.7.



Hình 7.6. Tổ chức đường truyền dữ liệu dạng *two-bus*



Hình 7.7. Tổ chức đường truyền dữ liệu dạng *three-bus*

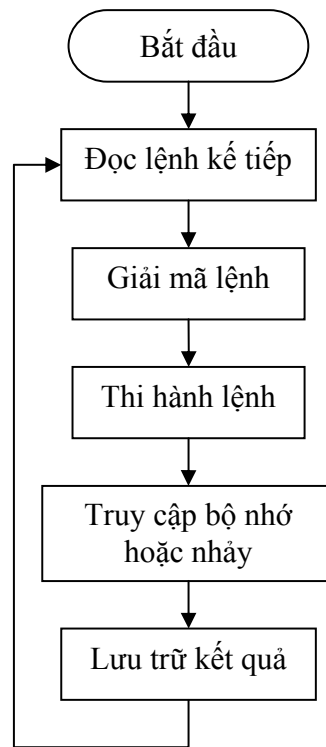
## 7.5. Diễn tiến thi hành lệnh mã máy(*CPU instruction cycle*)

Công việc của CPU diễn ra liên tục và việc thực hiện các lệnh diễn ra như trong hình 7.8. Các lệnh được lấy ra từ bộ nhớ chính và khi CPU tìm thấy một lệnh, nó sẽ phân tích lệnh đó để thực thi, sau đó lại sẽ tìm nạp lệnh tiếp theo và cứ như vậy diễn ra liên tục.

Việc thi hành một lệnh mã máy có thể chia thành 5 giai đoạn sau:

- Đọc lệnh (IF: Instruction Fetch)
- Giải mã lệnh (ID: Instruction Decode)
- Thi hành lệnh (EX: Execute)
- Thâm nhập bộ nhớ trong hoặc nhảy (MEM: Memory access)
- Lưu trữ kết quả (RS: Result Storing).

Mỗi giai đoạn được thi hành trong một hoặc nhiều chu kỳ xung nhịp.



Hình 7.8. Diễn biến thi hành lệnh của CPU

### 1. Đọc lệnh (fetch instruction):

Chuỗi các sự kiện trong một lần tìm đọc lệnh có thể tổng quát như sau:

1. Dữ liệu trong PC được load vào MAR:  $MAR \leftarrow PC$
2. Giá trị trong thanh ghi PC tăng lên 1:  $PC \leftarrow PC + 1$
3. Kết quả của lệnh đọc từ bộ nhớ, dữ liệu được load vào MDR:  $MDR \leftarrow M[MAR]$

4. Dữ liệu trong MDR được load vào IR:  $IR \leftarrow M[MDR]$

Bộ đếm chương trình PC được đưa vào MAR. Lệnh được đọc từ bộ nhớ trong, tại các ô nhớ có địa chỉ nằm trong MAR và được đưa vào thanh ghi lệnh IR. Thứ tự thực hiện lệnh theo thời gian đối với loại *one-bus* như sau:

Step	Micro-operation
$t_0$	$MAR \leftarrow (PC); A \leftarrow (PC)$
$t_1$	$MDR \leftarrow Mem[MAR]; PC \leftarrow (A) + 4$
$t_2$	$IR \leftarrow (MDR)$

Nếu dùng *three-bus* như hình 7.7 thì ta có thứ tự các bước như sau:

Step	Micro-operation
$t_0$	$MAR \leftarrow (PC); PC \leftarrow (PC) + 4$
$t_1$	$MDR \leftarrow Mem[MAR]$
$t_2$	$IR \leftarrow (MDR)$

### 2. Giải mã lệnh và đọc các thanh ghi nguồn:

Để hiểu rõ giai đoạn này, ta lấy dạng thức của một lệnh làm tính tiêu biểu sau đây:

Mã lệnh	Thanh ghi RS1	Thanh ghi RS1	Thanh ghi RD
8	4	4	4

Các thanh ghi nguồn RS1 và RS2 được sử dụng tùy theo tác vụ, kết quả được đặt trong thanh ghi đích RD.

Lệnh được giải mã. Kế đó các thanh ghi RS1 và RS2 được đưa vào A và B để chuẩn bị thực thi lệnh (ví dụ như trong tổ chức one-bus). Thanh ghi PC được tăng lên để chỉ tới lệnh kế đó.

$$A \leftarrow RS1$$

$$B \leftarrow RS2$$

$$PC \leftarrow PC + 4$$

Trong giai đoạn này việc giải mã sẽ được thực hiện cùng lúc với việc đọc dữ liệu các thanh ghi nguồn RS1 và RS2. Như vậy kết thúc giai đoạn này thì lệnh sẽ được giải mã và máy tính sẽ biết chính xác lệnh phải thực thi là lệnh cộng, trừ, AND,... và các thanh ghi nguồn cũng được nạp vào vị trí cần thiết cho việc thực thi lệnh.

### 3. Thi hành lệnh:

Tuỳ theo loại lệnh mà một trong ba nhiệm vụ sau đây được thực hiện:

- Liên hệ tới bộ nhớ

$$MAR \leftarrow \text{Địa chỉ do ALU tính tuỳ theo kiểu định vị (RS2).}$$

$$MDR \leftarrow RS1$$

Địa chỉ hiệu dụng do ALU tính được đưa vào MAR và thanh ghi nguồn RS1 được đưa vào MDR để được lưu vào bộ nhớ trong.

- Một lệnh của ALU

$$\text{Ngã ra ALU} \leftarrow \text{Kết quả của phép tính}$$

ALU thực hiện phép tính xác định trong mã lệnh, đưa kết quả ra ngã ra.

- Một phép nhảy

$$\text{Ngã ra ALU} \leftarrow \text{Địa chỉ lệnh tiếp theo do ALU tính.}$$

ALU cộng địa chỉ của PC với độ dời để làm thành địa chỉ đích và đưa địa chỉ này ra ngã ra. Nếu là một phép nhảy có điều kiện thì thanh ghi trạng thái được đọc quyết định có cộng độ dời vào PC hay không.

### 4. Thâm nhập bộ nhớ trong hoặc nhảy lần cuối

Giai đoạn này thường chỉ được dùng cho các lệnh nạp dữ liệu, lưu giữ dữ liệu và lệnh nhảy.

- Tham khảo đến bộ nhớ:

$$MDR \leftarrow M[MAR] \text{ hoặc } M[MAR] \leftarrow MDR$$

Số liệu được nạp vào MDR hoặc lưu vào địa chỉ mà MAR trỏ đến.

- Nhảy:

$$\text{If (điều kiện), } PC \leftarrow \text{ngã ra ALU}$$

Nếu điều kiện đúng, ngã ra ALU được nạp vào PC. Đối với lệnh nhảy không điều kiện, ngã ra ALU luôn được nạp vào thanh ghi PC.

### 5. Lưu trữ kết quả

$$RD \leftarrow \text{Ngã ra ALU hoặc } RD \leftarrow MDR$$



Lưu trữ kết quả trong thanh ghi đích.

## 7.6. Xử lý ngắt (Interrupt Handling)

Trong máy tính, đôi khi một số công việc nào đó cần được xử lý ngay tức thời và như vậy cần phải làm đình trệ một công việc nào đó đang xử lý, việc làm đó của CPU ta gọi là ngắt. Vậy ngắt là một sự kiện xảy ra một cách ngẫu nhiên trong máy tính và làm ngưng tính tuần tự của chương trình (nghĩa là tạo ra một lệnh nhảy). Phần lớn các nhà sản xuất máy tính (ví dụ như IBM, INTEL) dùng từ ngắt quãng để ám chỉ sự kiện này, tuy nhiên một số nhà sản xuất khác dùng từ “ngoại lệ”, “lỗi”, hay “bẫy” để chỉ định hiện tượng này.

Khi thiết kế máy tính thì bộ điều khiển của CPU là bộ phận khó thực hiện nhất và ngắt quãng lại là phần khó thực hiện nhất trong bộ điều khiển. Để nhận biết được một ngắt quãng lúc đang thi hành một lệnh, ta phải biết điều chỉnh chu kỳ xung nhịp và điều này có thể ảnh hưởng đến hiệu quả của máy tính.

Ngắt quãng được ứng dụng trong nhiều trường hợp, ví dụ như để nhận biết các sai sót trong tính toán số học như chia cho 0, để ứng dụng cho những hiện tượng thời gian thực cần được phục vụ tức thì. Có thể liệt kê những công việc sau đây đòi hỏi phải có ngắt quãng:

- Ngoại vi đòi hỏi nhập hoặc xuất số liệu.
- Người lập trình muốn dùng dịch vụ của hệ điều hành.
- Cho một chương trình chạy từng lệnh.
- Làm điểm dừng của một chương trình.

- Báo tràn số liệu trong tính toán số học.
- Trang bộ nhớ thực sự không có trong bộ nhớ.
- Báo vi phạm vùng cấm của bộ nhớ.
- Báo dùng một lệnh không có trong tập lệnh.
- Báo phần cứng máy tính bị hư.
- Báo điện bị cắt.

Thông thường trong máy tính ngắt quãng không xảy ra thường xuyên nhưng bộ xử lý phải được thiết kế sao cho có thể lưu giữ trạng thái của nó trước khi bị ngắt quãng đi phục vụ một công việc gì đó đòi hỏi tức thì. Sau khi thực hiện xong công việc đó thì chương trình xử lý ngắt phải khôi phục lại nguyên trạng thái của nó trước khi ngắt quãng xảy ra để có thể tiếp tục công việc.

Khi thiết kế máy tính, để đơn giản người ta có thể thiết kế bộ xử lý chỉ chấp nhận ngắt sau khi thực hiện xong lệnh đang chạy.

Trong trường hợp đơn giản này, khi một ngắt xảy ra, bộ xử lý thì hành các bước sau đây:

1. Thực hiện xong lệnh đang làm.
2. Lưu trữ trạng thái hiện tại.
3. Nhảy đến chương trình phục vụ ngắt
4. Khi chương trình phục vụ chấm dứt, bộ xử lý khôi phục lại trạng thái cũ của nó và tiếp tục thực hiện chương trình mà nó đang thực hiện khi bị ngắt.

Bảng 7.1 cho ta thấy thứ tự thực hiện các sự kiện khi xảy ra ngắt, ở đây  $t_1 < t_2 < t_3$ .

Step	Micro-operation
$t_1$	MDR $\leftarrow$ (PC)
$t_2$	MAR $\leftarrow$ address1 (where to save old PC); PC $\leftarrow$ address2 (interrupt handling routine)
$t_3$	Mem[MAR] $\leftarrow$ (MDR)

Bảng 7.1. Thực hiện các vi tác vụ khi ngắt

## 7.7. Kỹ thuật ống dẫn (PIPELINE)

Đối với máy tính dạng bó hoạt động theo nguyên lý Von neumann thì các lệnh thực hiện một cách tuần tự, hết lệnh này rồi mới đến lệnh khác. Trong các máy tính ngày nay có một kỹ thuật làm cho tốc độ của máy tính tăng lên nhiều lần đó là việc chia các lệnh ra thành các giai đoạn khác nhau và các giai đoạn này có thể được thi hành cùng một lúc.

Ví dụ: Chúng ta có những lệnh đều đặn, mỗi lệnh được thực hiện trong cùng một khoảng thời gian. Giả sử, mỗi lệnh được thực hiện trong 5 giai đoạn và mỗi giai đoạn được thực hiện trong 1 chu kỳ xung nhịp. Các giai đoạn thực hiện một lệnh là:

- **Lấy lệnh (IF: Instruction Fetch)**,
- **Giải mã (ID: Instruction Decode)**,
- **Thi hành (EX: Execute)**,
- **Thâm nhập bộ nhớ (MEM: Memory Access)**,
- **Lưu trữ kết quả (RS: Result Storing)**.

Hình 7.9 cho thấy để xử lý 5 lệnh chúng ta cần 9 chu kỳ xung nhịp. Trong chu kỳ xung nhịp thứ nhất, lệnh thứ  $i$  nhất được

tìm nạp vào. Trong chu kỳ xung nhịp thứ hai, trong khi lệnh thứ  $i$  đang thực hiện giải mã thì lại tiếp tục nạp lệnh tiếp theo  $i+1$ . Sang chu kỳ xung nhịp thứ ba, trong khi lệnh thứ  $i$  đang thực hiện, lệnh thứ  $i+1$  đang giải mã thì lại tiếp tục tìm nạp lệnh thứ  $i+2$ .

Chuỗi lệnh	Chu kỳ xung nhịp								
	1	2	3	4	5	6	7	8	9
Lệnh thứ $i$	IF	ID	EX	MEM	RS				
Lệnh thứ $i+1$		IF	ID	EX	MEM	RS			
Lệnh thứ $i+2$			IF	ID	EX	MEM	RS		
Lệnh thứ $i+3$				IF	ID	EX	MEM	RS	
Lệnh thứ $i+4$					IF	ID	EX	MEM	RS

Hình 7.9. Thực hiện lệnh trong kỹ thuật pipeline

So sánh với kiểu xử lý tuần tự thông thường, 5 lệnh được thực hiện trong 25 chu kỳ xung nhịp, thì xử lý lệnh theo kỹ thuật ống dẫn thực hiện 5 lệnh chỉ trong 9 chu kỳ xung nhịp.

Cứ như thế, nếu số lệnh tăng lên nữa thì tiến tới chỉ trong một chu kỳ xung nhịp, bộ xử lý có thể thực hiện một lệnh (bình thường lệnh này được thực hiện trong 5 chu kỳ).

Như vậy kỹ thuật ống dẫn làm tăng tốc độ thực hiện các lệnh. Tuy nhiên kỹ thuật ống dẫn có một số ràng buộc:

- Cần phải có một mạch điện tử để thi hành mỗi giai đoạn của lệnh vì tất cả các giai đoạn của lệnh được thi hành cùng lúc. Trong một bộ xử lý không dùng kỹ thuật ống dẫn, ta có thể dùng bộ logic số học ALU để cập nhật thanh ghi PC, cập nhật địa chỉ của toán hạng bộ nhớ, địa chỉ ô nhớ mà chương trình cần nhảy tới, làm các phép tính trên các toán hạng vì các phép tính này có thể xảy ra ở nhiều giai đoạn khác nhau.

- Phải có nhiều thanh ghi khác nhau dùng cho các tác vụ đọc và viết. Trên hình 7.9, tại một chu kỳ xung nhịp (chu kỳ xung

nhịp 5), ta thấy cùng một lúc có 2 tác vụ đọc (IF, MEM) và 1 tác vụ viết (RS).

- Trong một máy có kỹ thuật ống dẫn, có khi kết quả của một tác vụ trước đó, là toán hạng nguồn của một tác vụ khác. Giả sử kết quả của lệnh thứ  $i$  là toán hạng của lệnh thứ  $i+1$  thì tại chu kỳ xung nhịp thứ 4 lệnh thứ  $i+1$  không thể thực hiện (EX) được vì còn chưa có giá trị toán hạng. Như vậy sẽ có thêm những khó khăn đòi hỏi phải có cơ chế xử lý riêng.

- Cần phải giải mã các lệnh một cách đơn giản để có thể giải mã và đọc các toán hạng trong một chu kỳ duy nhất của xung nhịp.

- Cần phải có các bộ làm tính ALU hữu hiệu để có thể thi hành lệnh số học dài nhất, có số giữ, trong một khoảng thời gian ít hơn một chu kỳ của xung nhịp.

- Cần phải có nhiều thanh ghi lệnh để lưu giữ lệnh mà chúng ta phải xem xét cho mỗi giai đoạn thi hành lệnh.

- Cuối cùng phải có nhiều thanh ghi bộ đếm chương trình PC để có thể tái tục các lệnh trong trường hợp có ngắt quãng.

#### ➤ Những khó khăn trong kỹ thuật ống dẫn

Khi thi hành lệnh trong một máy tính dùng kỹ thuật ống dẫn, có nhiều trường hợp làm cho việc thực hiện kỹ thuật ống dẫn không thực hiện được như là: thiếu các mạch chức năng xử lý một việc cụ thể nào đó, một lệnh dùng kết quả của lệnh trước thì mới thi hành được, một lệnh nhảy.

Ta có thể phân biệt 3 loại khó khăn: khó khăn do cấu trúc, khó khăn do số liệu và khó khăn do điều khiển.

#### a. Khó khăn do cấu trúc:

Khó khăn đầu tiên là khó khăn do thiếu bộ phận chức năng, ví dụ trong một máy tính dùng kỹ thuật ống dẫn phải có nhiều ALU, nhiều PC, nhiều thanh ghi lệnh IR ... Như vậy những khó khăn này là do bộ phận phần cứng thiếu và cách giải quyết duy nhất là thêm các bộ phận chức năng cần thiết và hữu hiệu vào.

#### b. Khó khăn do số liệu:

Lấy ví dụ trường hợp các lệnh liên tiếp sau:

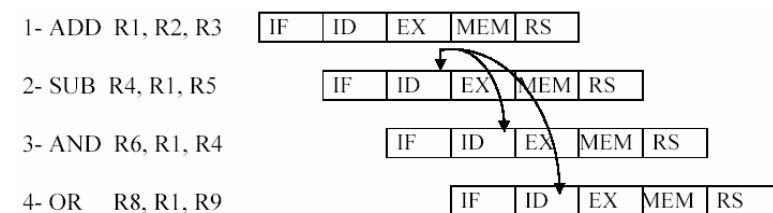
Lệnh 1: ADD R1, R2, R3

Lệnh 2: SUB R4, R1, R5

Lệnh 3: AND R6, R1, R7

Lệnh 4: OR R8, R1, R9

Trong hình 7.10 cho ta thấy thanh ghi R1 lưu kết quả của lệnh 1 chỉ có thể được dùng cho lệnh 2 sau giai đoạn MEM của lệnh 1, nhưng R1 được dùng cho lệnh 2 vào giai đoạn EX của lệnh 1. Chúng ta cũng thấy R1 được dùng cho các lệnh 3 và 4.



Hình 7.10. Những khó khăn do số liệu

Để khắc phục khó khăn này, một bộ phận phần cứng được dùng để đưa kết quả từ ngã ra ALU trực tiếp vào một trong các thanh ghi ngã vào như trong hình 7.6.

Khi bộ phận phần cứng nêu trên phát hiện có dùng kết quả của ALU làm toán hạng cho liệt kê, nó tác động vào mạch đa hợp để đưa ngõ ra của ALU vào ngõ vào của ALU hoặc vào ngõ vào của một đơn vị chức năng khác nếu cần.

### c. Khó khăn do điều khiển:

Trong một số trường hợp, các lệnh làm thay đổi tính thi hành các lệnh một cách tuần tự (nghĩa là thay vì PC tăng đều đặn sau mỗi lệnh thì đột xuất nó tăng hoặc giảm mạnh hơn), gây khó khăn về điều khiển. Các lệnh này là lệnh nhảy đến một địa chỉ tuyệt đối chứa trong một thanh ghi, hay lệnh nhảy đến một địa chỉ xác định một cách tương đối so với địa chỉ hiện tại của bộ đếm chương trình PC. Các lệnh nhảy trên có thể có hoặc không điều kiện.

Trong trường hợp đơn giản nhất, tác vụ nhảy không thể biết trước giai đoạn giải mã. Như vậy, nếu lệnh nhảy bắt đầu ở chu kỳ C thì lệnh mà chương trình nhảy tới chỉ được bắt đầu ở chu kỳ C+2. Ngoài ra, phải biết địa chỉ cần nhảy đến mà ta có ở cuối giai đoạn giải mã ID. Trong lệnh nhảy tương đối, ta phải cộng độ dời chứa trong thanh ghi lệnh IR vào thanh ghi PC. Việc tính địa chỉ này chỉ được thực hiện vào giai đoạn ID với điều kiện phải có một mạch công việc riêng biệt.

Vậy trong trường hợp lệnh nhảy không điều kiện, lệnh mà chương trình nhảy đến bắt đầu thực hiện ở chu kỳ C+2 nếu lệnh nhảy bắt đầu ở chu kỳ C.

Cho các lệnh nhảy có điều kiện thì phải tính toán điều kiện. Thông thường các kiến trúc RISC đặt kết quả việc so sánh vào trong thanh ghi trạng thái, hoặc vào trong thanh ghi tổng quát. Trong cả 2 trường hợp, đọc điều kiện tương đương với đọc thanh ghi. Đọc thanh ghi có thể được thực hiện trong phân nửa chu kỳ cuối giai đoạn ID.

Một trường hợp khó hơn có thể xảy ra trong những lệnh nhảy có điều kiện. Đó là điều kiện được có khi so sánh 2 thanh ghi và chỉ thực hiện lệnh nhảy khi kết quả so sánh là đúng. Việc tính toán trên các đại lượng logic không thể thực hiện được trong phân nửa chu kỳ và như thế phải kéo dài thời gian thực hiện lệnh nhảy có điều kiện. Người ta thường tránh các trường hợp này để không làm giảm mức hữu hiệu của máy tính.

Vậy trường hợp đơn giản, người ta có thể được địa chỉ cần nhảy đến và điều kiện nhảy cuối giai đoạn ID. Vậy có chậm đi một chu kỳ mà người ta có thể giải quyết bằng nhiều cách.

Cách thứ nhất là đóng băng kỹ thuật ống dẫn trong một chu kỳ, nghĩa là ngưng thi hành lệnh thứ  $i+1$  đang làm nếu lệnh thứ  $i$  là lệnh nhảy. Ta mất trắng một chu kỳ cho mỗi lệnh nhảy.

Cách thứ hai là thi hành lệnh sau lệnh nhảy nhưng lưu ý rằng hiệu quả của một lệnh nhảy bị chậm mất một lệnh. Vậy lệnh theo sau lệnh nhảy được thực hiện trước khi lệnh mà chương trình phải nhảy tới được thực hiện. Chương trình dịch hay người lập trình có nhiệm vụ xen vào một lệnh hữu ích sau lệnh nhảy.

Trong lập trình thường xảy ra trường hợp chúng ta so sánh một điều kiện nào đó và việc nhảy có thể được thực hiện hay không thực hiện phụ thuộc vào điều kiện đó. Người ta gọi lệnh hữu ích đặt sau lệnh nhảy không làm sai lệch chương trình dù điều kiện nhảy đúng hay sai. Hầu hết các bộ xử lý RISC có những lệnh nhảy với khả năng huỷ bỏ. Các lệnh này cho phép thi hành lệnh sau lệnh nhảy nếu điều kiện nhảy đúng và huỷ bỏ thực hiện lệnh đó nếu điều kiện nhảy sai giống như trong câu lệnh if (điều kiện đúng) GOTO địa chỉ else thực hiện lệnh ngay sau dòng if.

## CÂU HỎI VÀ BÀI TẬP CHƯƠNG VII

1. Liệt kê các thành phần và nhiệm vụ của CPU?
2. Mô tả tổ chức một máy tính đơn giản và hoạt động của các bộ phận chính trong CPU
3. Nhiệm vụ của bộ điều khiển và các loại bộ điều khiển?
4. Bộ thanh ghi gồm các loại nào? Trong họ 80x86, hãy nêu các thanh ghi và các đặc tính tương ứng của mỗi loại.
5. Các loại đường đi của dữ liệu và các ưu, nhược điểm của mỗi loại?
6. Việc thi hành lệnh mã máy được thực thi ra sao? giải thích việc thực thi đoạn chương trình sau:  
MOV R4, #24  
ADD R4, (R1)
7. Thế nào là ngắt quãng? Các giai đoạn thực hiện ngắt quãng của CPU.
8. Vẽ hình để mô tả kỹ thuật ống dẫn. Kỹ thuật ống dẫn làm tăng tốc độ CPU lên bao nhiêu lần (theo lý thuyết)? Tại sao trên thực tế sự gia tăng này lại ít hơn?
9. Các điều kiện mà một CPU cần phải có để tối ưu hoá kỹ thuật ống dẫn. Giải thích từng điều kiện.
10. Các khó khăn trong kỹ thuật ống dẫn và cách giải quyết khó khăn này.

## CHƯƠNG VIII: HỆ THỐNG BỘ NHỚ

Trong chương này sẽ giới thiệu về các chức năng và nguyên lý hoạt động của hệ thống bộ nhớ máy tính như bộ nhớ cache, bộ nhớ trong, bộ nhớ thứ cấp và bộ nhớ ảo. Bộ nhớ máy tính được tổ chức thành một hệ thống gọi là các cấp bộ nhớ, trong đó các bộ nhớ lớn hơn, chậm hơn và rẻ tiền hơn sẽ được kết hợp với các bộ nhớ nhỏ hơn, nhưng nhanh hơn và đắt tiền hơn để tạo thành một hệ thống bộ nhớ có tốc độ và giá cả phù hợp nhất.

### 8.1. CÁC CẤP BỘ NHỚ (Memory Hierarchy)

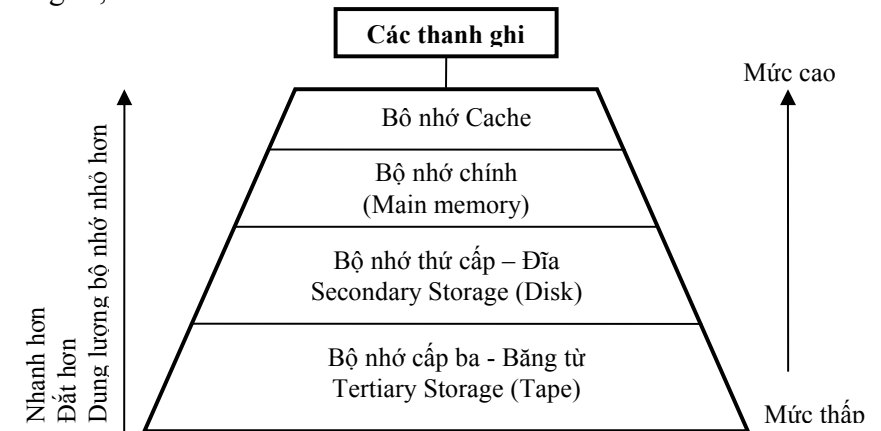
Người ta chia bộ nhớ thành các cấp dựa vào các đặc tính như lượng thông tin lưu trữ, thời gian thâm nhập bộ nhớ, chu kỳ bộ nhớ, giá tiền mỗi bit nhớ. Các cấp bộ nhớ bắt đầu từ bộ nhớ nhanh với dung lượng ít đến các bộ nhớ chậm với dung lượng lớn hơn nhiều.

Các cấp bộ nhớ giúp cho người lập trình có được một bộ nhớ thật nhanh với chi phí đầu tư thấp hơn nhiều. Vì các bộ nhớ nhanh, có dung lượng ít thì đắt tiền hơn các bộ nhớ có dung lượng cao hơn, nhưng chậm hơn. Do đó mục tiêu của việc thiết lập các cấp bộ nhớ là người dùng có được một hệ thống bộ nhớ rẻ tiền như cấp bộ nhớ thấp nhất và gần nhanh như cấp bộ nhớ cao nhất. Các cấp bộ nhớ thường được lồng vào nhau. Mọi dữ liệu trong một cấp thì được gặp lại trong cấp thấp hơn và có thể tiếp tục gặp lại trong cấp thấp nhất.

Chúng ta có nhận xét rằng, mỗi cấp bộ nhớ có dung lượng lớn hơn cấp trên mình, ánh xạ một phần địa chỉ các ô nhớ của mình vào địa chỉ ô nhớ của cấp trên trực tiếp có tốc độ nhanh hơn, và các cấp bộ nhớ phải có cơ chế quản lý và kiểm tra các địa chỉ ánh xạ.

Các cấp bộ nhớ được phân loại như trên hình 8.1 bắt đầu với bộ nhớ nhỏ, đắt tiền và nhanh gọi là cache. Tiếp sau đó là bộ nhớ lớn hơn, rẻ hơn và chậm hơn gọi là bộ nhớ chính hay bộ nhớ

trong (main memory). Bộ nhớ cache và bộ nhớ chính được chế tạo sử dụng chất bán dẫn rắn (điển hình là CMOS transistors). Tiếp theo sau đó là các bộ nhớ có dung lượng lớn hơn, chậm hơn và rẻ tiền hơn mà tiêu biểu là đĩa cứng, đĩa mềm, các loại CD, DVD, băng từ,...



Hình 8.1. Các cấp bộ nhớ cơ bản

Hiệu quả của việc phân thành các cấp bộ nhớ phụ thuộc vào nguyên lý chuyển đổi thông tin trong các bộ nhớ nhanh. Hiệu quả càng cao khi thông tin trong đó càng ít khi bị thay đổi và việc truy cập vào thông tin trong đó càng nhiều lần càng tốt trước khi nó bị thay thế bởi thông tin mới. Để đánh giá hiệu quả của hệ thống các cấp bộ nhớ, chúng ta sẽ xem xét trường hợp dưới đây.

Khi bộ vi xử lý (VXL) gửi một yêu cầu truy cập đến một từ nhớ hay một câu lệnh, máy tính sẽ thực thi tìm kiếm từ nhớ theo một thứ tự như sau:

- Tìm từ nhớ trong bộ nhớ mức cao nhất (thông thường gọi là cache) của các cấp bộ nhớ. Xác suất tìm thấy từ nhớ trong đó gọi là tỷ số thành công (*hit ratio*)  $h_1$ , không tìm thấy là tỷ số thất bại (*miss ratio*)  $(1-h_1)$ ;
- Khi không tìm thấy từ nhớ trong bước một thì tìm ở cấp bộ nhớ thấp hơn (như cache L2, main memory). Tương tự như trên ta sẽ có các tỷ số  $h_2$  và  $(1-h_2)$ ;

- Quá trình này sẽ tiếp diễn cho đến khi tìm thấy từ nhớ cần thiết hoặc hết cấp bộ nhớ.
- Khi tìm thấy từ nhớ sẽ được chuyển cho Bộ xử lý và cập nhật lại dữ liệu trong các cấp bộ nhớ nếu cần thiết.

Giả sử các cấp bộ nhớ có 3 cấp. Thời gian truy cập vào các cấp bộ nhớ trung bình, hay thời gian trung bình để tìm thấy một từ nhớ được tính theo công thức sau:

$$t_{av} = h_1 * t_1 + (1-h_1) * [t_1 + h_2 * t_2 + (1-h_2) * (t_2 + t_3)] \\ = t_1 + (1-h_1) * [t_2 + (1-h_2) * t_3]$$

Trong đó:

- $h_i$  là tỷ số thành công khi tìm thấy từ nhớ cần thiết trong mức bộ nhớ cấp  $i$ .
- $t_i$  là thời gian truy cập cần thiết vào bộ nhớ cấp  $i$ .

Theo [6] thì một chương trình tiêu tốn 90% thời gian thi hành của nó chỉ để thực hiện 10% số lệnh của chương trình (thường là những lệnh nằm trong các vòng lặp, được thực thi nhiều lần). Như vậy 90% số lệnh còn lại chỉ tiêu tốn 10% thời gian thực thi hay nói cách khác chúng hoặc là không được thực thi (Ví dụ các lệnh xử lý lỗi chỉ được thực thi khi phát sinh lỗi) hoặc là chỉ được thực thi một số ít lần.

Nguyên tắc trên cũng được áp dụng cho việc thâm nhập dữ liệu, nhưng ít hiệu nghiệm hơn việc thâm nhập lệnh. Như vậy có hai nguyên tắc: nguyên tắc về không gian và nguyên tắc về thời gian

- **Nguyên tắc về không gian:** Khi bộ xử lý thâm nhập vào ô nhớ nào đó thì có nhiều khả năng sẽ thâm nhập vào những ô nhớ có địa chỉ kế tiếp trong thời gian sau đó do các lệnh được sắp xếp thành chuỗi có thứ tự. Ví dụ danh sách các lệnh nằm liên tiếp nhau trong một chương trình.

- **Nguyên tắc về thời gian:** Các ô nhớ được hệ thống xử lý thâm nhập có khả năng sẽ được thâm nhập lại trong tương lai gần. Thật vậy, các chương trình được cấu tạo với phần chính là phần được thi hành nhiều nhất và các phần phụ dùng để xử lý các trường hợp ngoại lệ. Còn số liệu luôn có cấu trúc và thông thường chỉ có một phần số liệu được thâm nhập nhiều nhất mà thôi. Ví dụ như một lệnh trong một vòng lặp của chương trình.

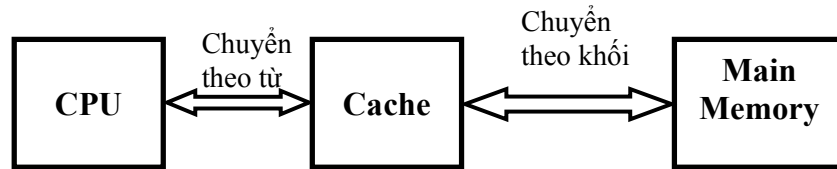
Như vậy, chúng ta cần tổ chức các cấp bộ nhớ sao cho các lệnh và dữ liệu thường dùng được nằm trong bộ nhớ cache và việc ô chức như vậy sẽ làm tăng hiệu quả cũng như tốc độ chung của máy tính lên một cách đáng kể.

## 8.2. BỘ NHỚ CACHE (Cache memory)

Cache là một bộ nhớ tốc độ cao nằm gần CPU. Ý tưởng là dùng bộ nhớ cache ở mức đầu tiên của các cấp bộ nhớ để lưu trữ các thông tin mà CPU thường xuyên sử dụng. Kết quả là trong bất kỳ thời gian nào cũng có một phần đang vận hành của bộ nhớ trong (hay bộ nhớ chính – *main memory*) được nhân bản lại trong bộ nhớ cache. Để cho chương trình vận hành nhanh hơn thì cache phải chứa một phần con của bộ nhớ trong để bộ xử lý có thể thâm nhập vào các lệnh hoặc dữ liệu thường dùng từ bộ nhớ cache. Do dung lượng của bộ nhớ cache nhỏ nên nó chỉ chứa một phần chương trình nằm trong bộ nhớ trong. Khi bộ VXL yêu cầu đọc một bộ nhớ liên quan thì đầu tiên yêu cầu này được tìm trong cache. Nếu như yêu cầu tương ứng với một phần tử nằm trong cache thì ta gọi trường hợp này là thành công cache (**cache hit**). Trường hợp không tìm thấy phần tử yêu cầu trong cache thì ta gọi là thất bại cache (**cache miss**). Tỷ số thành công cache  $h_c$  (**cache hit ratio**) được định nghĩa là xác suất tìm thấy phần tử yêu cầu trong cache. Tỷ số thất bại cache  $(1-h_c)$  (**cache miss ratio**) là xác suất không tìm thấy phần tử yêu cầu trong cache.

Để đảm bảo sự đồng nhất giữa nội dung của cache và bộ nhớ trong thì cache và bộ nhớ trong phải có cùng cấu trúc. Việc

chuyển dữ liệu giữa cache và bộ nhớ trong là việc tải lên hay ghi xuống các *khối dữ liệu*. Mỗi khối chứa nhiều từ bộ nhớ tùy thuộc vào cấu trúc bộ nhớ cache. Sự lựa chọn kích thước của khối rất quan trọng cho vận hành của cache có hiệu quả. Việc trao đổi dữ liệu giữa CPU, cache và bộ nhớ trong được thực thi như hình 8.2.



Hình 8.2. Sơ đồ chuyển dữ liệu giữa CPU-Cache-Main

Để đánh giá sự ảnh hưởng của hai nguyên lý không gian và thời gian vào tốc độ truy cập vào bộ nhớ của máy tính, chúng ta giả sử rằng thời gian truy cập vào bộ nhớ trong là  $t_m$  và thời gian truy cập vào cache là  $t_c$ . Giả sử hệ thống các cấp bộ nhớ có hai mức và ta sẽ khảo sát thời gian truy cập trung bình  $t_{av}$  đến một từ nhớ trong mô hình này trong các nguyên lý không gian và thời gian

#### Ảnh hưởng của nguyên lý lân cận thời gian

Trong trường hợp này chúng ta giả sử những lệnh trong vòng lặp của một chương trình được chạy nhiều lần và giả sử là  $n$  lần. Một lần được tải vào bộ nhớ cache và được sử dụng nhiều lần trước khi bị thay thế bởi các lệnh mới. Như vậy thời gian truy cập trung bình  $t_{av}$  được tính bởi công thức sau:

$$t_{av} = \frac{nt_c + t_m}{n} = t_c + \frac{t_m}{n}$$

Từ công thức trên ta thấy  $n$  càng tăng thì thời gian truy cập trung bình càng giảm và như vậy hiệu quả của việc sử dụng cache càng cao.

#### Ảnh hưởng của nguyên lý lân cận không gian

Trong trường hợp này giả sử kích thước của một khối được chuyển từ bộ nhớ trong vào cache khi thất bại cache là  $m$  phần tử. Ta cũng giả sử rằng trong lân cận không gian này thì tất cả  $m$  phần tử đều được yêu cầu bởi bộ VXL một lần. Trên cơ sở những giả thiết đó ta có thời gian truy cập trung bình được tính bởi công thức:

$$t_{av} = \frac{mt_c + t_m}{m} = t_c + \frac{t_m}{m}$$

Trong công thức rõ ràng là khi số phần tử trong khối được tăng lên thì thời gian trung bình sẽ giảm đi, điều đó cũng tương đương với việc càng nhiều phần tử nằm trong cache thì càng tốt.

#### Ảnh hưởng của tổng hợp hai nguyên lý lân cận không gian và thời gian

Trong trường hợp này giả sử phần tử được yêu cầu bởi bộ VXL không có trong cache hay cache miss và cần phải chuyển một khối có chứa  $m$  phần tử vào trong cache (việc làm này mất một thời gian là  $t_m$ ). Ta cũng giả sử rằng trong lân cận không gian này thì tất cả  $m$  phần tử đều được yêu cầu bởi bộ VXL một lần ( $mt_c$ ). Ngoài ra sau đó bộ VXL còn yêu cầu phần tử này thêm  $(n-1)$  lần nữa (lân cận thời gian) hay tổng số lần truy cập đến phần tử này là  $n$  lần. Trên cơ sở những giả thiết đó ta có thời gian truy cập trung bình được tính bởi công thức:

$$t_{av} = \frac{(\frac{mt_c + t_m}{m}) + (n-1)t_c}{n} = \frac{t_c + \frac{t_m}{m} + (n-1)t_c}{n} = t_c + \frac{t_m}{nm}$$

Trong công thức trên rõ ràng là khi số phần tử trong khối hoặc số lần truy cập đến phần tử được tăng lên thì thời gian trung bình sẽ giảm đi và có thể tiến tới giá trị  $t_c$ . Điều đó cũng tương đương với việc càng nhiều phần tử nằm trong cache thì càng tốt và số lần truy cập đến phần tử nằm trong cache càng nhiều thì càng tốt.

Qua các thảo luận ở trên ta biết rằng càng có nhiều yêu cầu truy cập đến từ nhớ không nằm trong cache thì càng cần chuyển



nhiều khối bộ nhớ từ bộ nhớ trong vào cache. Vấn đề là bộ nhớ cache thì có giới hạn nên việc vận chuyển các khối nhớ phải giải quyết được bốn câu hỏi cơ bản là: Phải để một khối bộ nhớ vào chỗ nào của cache hay cách sắp xếp khối nhớ trong cache như thế nào? Làm sao để tìm một từ nhớ trong khối nhớ khi khối này đang nằm trong cache? Trong trường hợp không tìm thấy khối nhớ cần thiết trong cache và cache đã đầy thì khối nào phải được đẩy ra khỏi cache để thay thế khối mới? và câu hỏi cuối cùng là việc gì sẽ xảy ra khi ghi khối nhớ từ cache ngược lại vào bộ nhớ? Để trả lời cho các câu hỏi này ta phải xem cách tổ chức và vận hành của cache và nội dung này sẽ được trình bày trong phần sau.

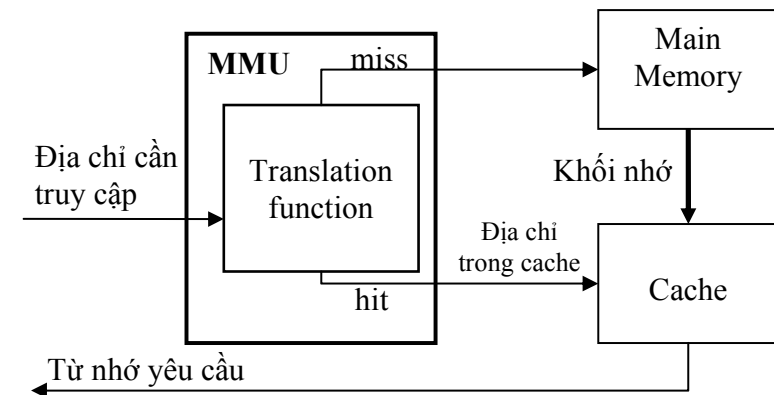
### 8.2.1. TỔ CHỨC BỘ NHỚ CACHE

Để hiểu rõ sự vận hành và tổ chức bộ nhớ cache, trước hết chúng ta sẽ xem xét chức năng ánh xạ hay tương ứng địa chỉ giữa các cấp bộ nhớ. Trong phần này để đơn giản hóa vấn đề ta xét các cấp bộ nhớ có hai mức, mức một là bộ nhớ cache và mức hai là bộ nhớ trong (main memory). Nguyên lý ứng dụng cho giao tiếp giữa các cấp bộ nhớ sẽ giống với trường hợp mà ta xem xét ở đây.

Yêu cầu truy cập tới một phần tử hay một từ nhớ được phát ra bởi bộ VXL và phần tử này có thể hiện tại đang nằm trong bộ nhớ cache trong trường hợp cache hit, nhưng cũng có thể tương ứng với một phần tử mà hiện tại nó không nằm trong cache (cache miss) mà nằm trong bộ nhớ trong. Do đó cần phải xác định địa chỉ của phần tử cần truy cập để xác định nó nằm ở đâu. Việc xác định này được một đơn vị chức năng trong CPU giải quyết và ta gọi là đơn vị quản lý bộ nhớ (memory management unit - MMU). Sơ đồ khối của chức năng ánh xạ địa chỉ như trong hình 8.3.

Như trên hình này, địa chỉ được phát ra bởi bộ VXL sẽ được MMU chuyển đổi thành một dạng địa chỉ tương ứng và nếu địa chỉ này được xác định là nằm trong cache thì từ nhớ cần truy cập sẽ được trả về cho bộ VXL. Trong trường hợp địa chỉ của từ nhớ cần truy cập không nằm trong cache, mà nằm trong bộ nhớ trong thì khối bộ nhớ có chứa từ này sẽ được chuyển vào trong

cache dưới dạng một khối bộ nhớ và sau đó thì từ nhớ này mới sẵn sàng cho bộ VXL.



Hình 8.3. Sơ đồ ánh xạ địa chỉ

Có ba kỹ thuật tổ chức bộ nhớ cache là: *kiểu tương ứng trực tiếp (Direct Mapping)*, *kiểu hoàn toàn phối hợp (Fully Associative Mapping)* và *kiểu phối hợp theo tập hợp (Set – Associative Mapping)*. Các kỹ thuật này dựa trên hai khía cạnh chính:

- Cách đặt vào cache một khối nhớ từ bộ nhớ trong
- Cách thay thế một khối cache (khi cache đầy).

Sau đây ta sẽ xem xét lần lượt từng kỹ thuật này một cách tỉ mỉ.

#### 8.2.1.1. Kiểu tương ứng trực tiếp (Direct mapping)

Đây là kỹ thuật đơn giản nhất trong ba kỹ thuật nêu trên. Theo kỹ thuật này thì mỗi khối bộ nhớ chỉ có một vị trí đặt khối duy nhất trong cache được xác định theo công thức:

$$j = i \bmod N$$

Trong đó:

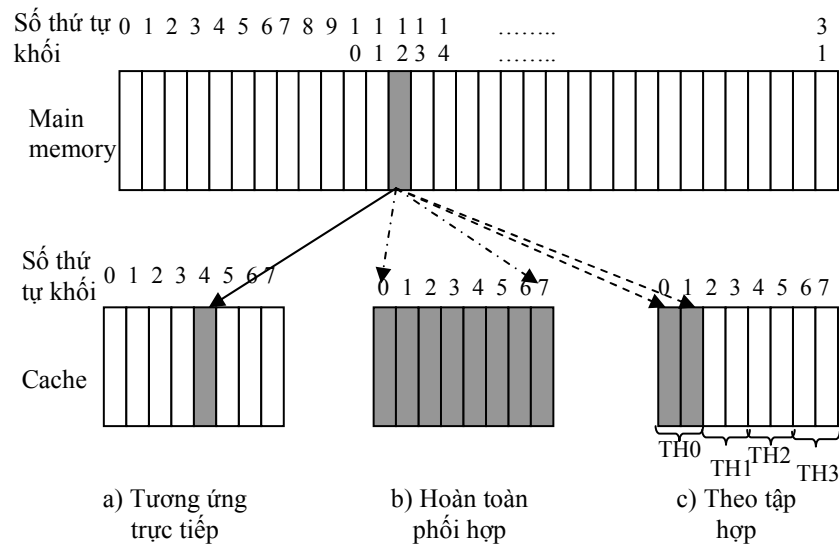
- $j$ : vị trí khối đặt trong cache
- $i$ : số thứ tự của khối trong bộ nhớ trong
- $N$ : số khối của cache

Ví dụ 1: Giả sử máy tính có bộ nhớ trong có 32 khối, cache có 8 khối, mỗi khối gồm 32 byte, khối thứ 12 của bộ nhớ trong được đưa vào cache.

Như vậy theo công thức trên thì khối nhớ thứ 12 sẽ được đưa vào cache ở vị trí:

$$j = 12 \bmod 8 = 4$$

Ví dụ này được minh họa trong hình 8.4 a).



Hình 8.4. Các kỹ thuật xếp đặt khối nhớ trong cache

Như vậy, trong kiểu xếp đặt khối này, mỗi vị trí đặt khối trong cache có thể chứa một trong các khối trong bộ nhớ cách nhau 8 khối.

Để nhận diện một khối có nằm trong cache hay không, mỗi khối của cache đều có một nhãn địa chỉ cho biết số thứ tự của các khối bộ nhớ trong đang hiện diện trong cache. Nhãn của một khối bộ nhớ cache chứa đựng thông tin cần thiết để biết được khối đó có nằm trong cache hay không và có chứa thông tin mà bộ xử lý cần

đọc hay không. Tất cả các nhãn đều được xem xét song song (trong kiểu tương ứng trực tiếp và phối hợp theo tập hợp) do đó tốc độ tìm kiếm sẽ rất cao. Để biết xem một khối của cache có chứa thông tin mà bộ xử lý cần tìm hay không, người ta thêm một bit đánh dấu (valid bit) vào nhãn để nói lên khối đó có chứa thông tin mà bộ xử lý cần tìm hay không.

Như đã đề cập ở trên, với thao tác đọc (ghi) bộ nhớ, bộ xử lý đưa ra một địa chỉ và nhận (viết vào) một dữ liệu từ (vào) bộ nhớ trong. Địa chỉ mà bộ xử lý đưa ra có thể phân tích thành hai thành phần: phần nhận dạng số thứ tự khối và phần xác định vị trí từ cần đọc trong khối.

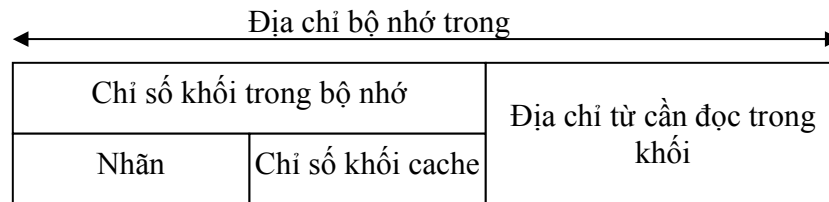
Tương ứng với ba kiểu lắp đặt khối đã xét, ta có:

- Căn cứ vào số từ trong một khối bộ nhớ mà số bit trong trường địa chỉ sẽ xác định vị trí từ cần đọc trong khối. Trường hợp này cũng giống như việc xác định địa chỉ tương đối trong bộ nhớ. Cách tính này đúng với cả ba cách xếp đặt khối đã xét.
- Phần nhận dạng số thứ tự khối sẽ khác nhau tùy thuộc vào cách xếp đặt khối, trường chỉ số khối được so sánh với nhãn của cache để xác định khối trong cache.

Dữ liệu được bộ xử lý đọc cùng lúc với việc đọc nhãn. Phần chỉ số khối của khối trong bộ nhớ trong được so sánh với bảng tương quan để xác định khối có nằm trong cache hay không. Để chắc rằng nhãn chứa thông tin đúng đắn (tức là khối có chứa từ mà bộ xử lý cần đọc-ghi), nếu việc so sánh nhãn của khối cache giống với số thứ tự khối, bit đánh dấu (Valid bit) phải được bật lên. Ngược lại, kết quả so sánh được bỏ qua. Bộ xử lý căn cứ vào phần xác định từ trong khối để đọc (ghi) dữ liệu từ (vào) cache.

Theo cách tương ứng trực tiếp thì MMU sẽ diễn giải địa chỉ phát ra từ CPU bằng cách chia địa chỉ thành 3 phần như là trong hình 8.5. Chiều dài tính bằng bit của mỗi phần trong đó tính như sau:

- Địa chỉ từ cần đọc trong khối (Word field) =  $\log_2 B$ ,  $B$  – kích thước khối theo từ
- Chỉ số khối cache (Block field) =  $\log_2 N$ ,  $N$ -kích thước cache theo block
- Nhãn (Tag field) =  $\log_2(M/N)$ ,  $M$ -kích thước bộ nhớ trong theo khối
- Số bit trong trường địa chỉ bộ nhớ trong =  $\log_2(B.M)$



Hình 8.5. Các trường địa chỉ trong tương ứng trực tiếp

Ví dụ 2: Xét trường hợp bộ nhớ trong chứa 4K khối, bộ nhớ cache chứa 128 khối và mỗi khối có kích thước 16 từ nhớ. Hình 8.6 cho thấy cách tổ chức xếp đặt khối nhớ theo kiểu tương ứng trực tiếp.

Tag	Cache	Main memory
3	0 384	0 128 256 384 3968
1	1 129	1 129 257 385 3969
0	2	2 130 258 386 3970
	126	126
31	127 4095	127 255 383 4095
		0 1 2 3 31

Hình 8.6. Ánh xạ khối bộ nhớ trong đến khối bộ nhớ cache  
Để xác định số bit của các trường trong địa chỉ bộ nhớ trong, ta áp dụng cách tính ở trên thì có:

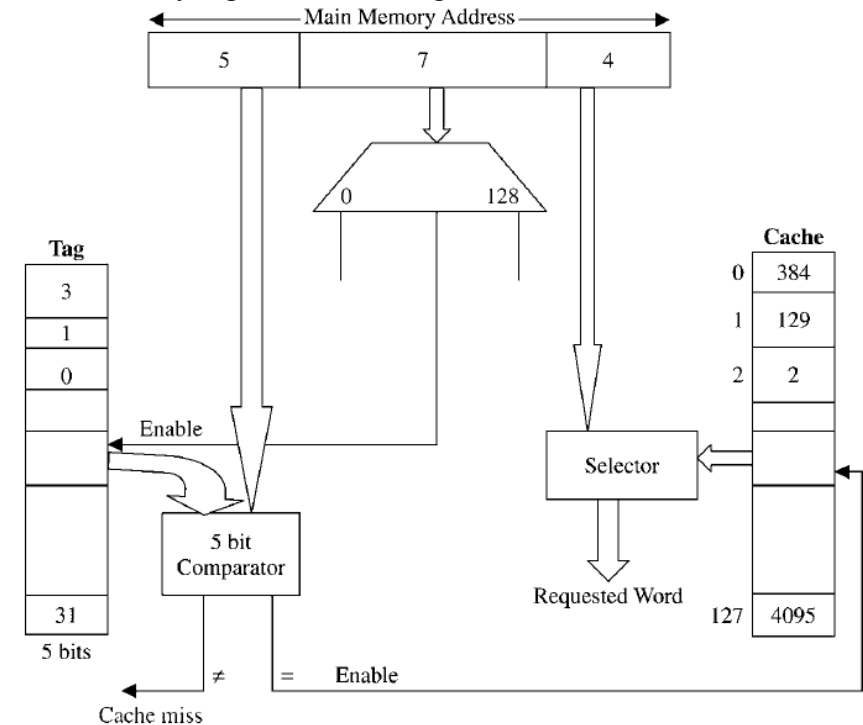
- Word field =  $\log_2 B = \log_2 16 = 4 \text{ bits}$
- Chỉ số khối cache =  $\log_2 N = \log_2 128 = 7 \text{ bits}$
- Nhãn =  $\log_2(M / N) = \log_2(4.2^{10} / 128) = 5 \text{ bits}$

Số bit trong trường địa chỉ bộ nhớ trong:

$$\log_2(B.M) = \log_2(16 \times 4.2^{10}) = 16 \text{ bits}$$

Trong hình 8.7. đưa ra minh họa giải thích quá trình diễn giải địa chỉ bởi MMU theo các bước sau:

1. Sử dụng Block field để xác định khối bộ nhớ cache có chứa từ nhớ mà bộ VXL yêu cầu
2. Kiểm tra sự tương ứng trong trường Tag memory và so sánh nó với trường Tag field để xác định có hay không khối chứa từ nhớ cần thiết
3. Nếu có (cache hit) thì dựa vào trường word field để lấy ra từ nhớ cần thiết cho bộ VXL
4. Nếu không có (cache miss) thì phải đem khối nhớ có chứa từ cần truy cập từ bộ nhớ trong vào cache.



Hình 8.7. Chuyển đổi địa chỉ tương ứng trực tiếp

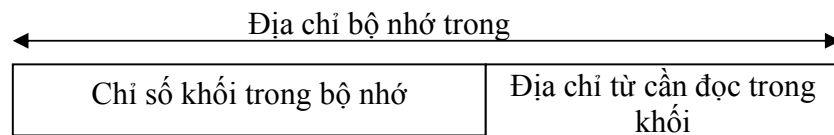
Ưu điểm của kiểu tương ứng trực tiếp là nó rất đơn giản, không cần tốn thời gian tìm kiếm và các cơ chế thay thế khối nhớ cũng đơn giản. Nhược điểm chủ yếu là nó sử dụng cache không hiệu quả do một số khối có thể luôn luôn bị thay đổi trong khi có thể một số khác lại không được dùng đến. Ví dụ như trường hợp ví dụ 2 ở trên, giả sử bộ VXL yêu cầu đến các khối nhớ theo thứ tự 0, 128, 256, 384 và 3968. Như vậy chỉ có khối nhớ cache ở vị trí số 0 được sử dụng, trong khi các khối nhớ khác trống.

### 8.2.1.2. Kiểu hoàn toàn phối hợp (Fully Associative Mapping)

Một khối trong bộ nhớ trong có thể được đặt vào vị trí bất kỳ trong cache. Tất nhiên đầu tiên tìm đặt vào các khối nào còn trống, nếu không còn khối nào trống thì mới phải áp dụng kỹ thuật thay thế khối mà ta sẽ xem xét sau. Ở ví dụ 1 nếu đặt khối nhớ theo cách này thì hình minh họa được chỉ ra trong hình 8.4 b).

Theo cách này thì MMU sẽ diễn giải địa chỉ phát ra từ CPU bằng cách chia địa chỉ thành 2 phần như là trong hình 8.8. Chiều dài tính bằng bit của mỗi phần trong đó tính như sau:

- Địa chỉ từ cần đọc trong khối (Word field) =  $\log_2 B$ ,  $B$  – kích thước khối theo từ
- Chỉ số khối (hay nhãn - Tag field) =  $\log_2 M$ ,  $M$  – kích thước bộ nhớ trong theo khối
- Số bit trong trường địa chỉ bộ nhớ trong =  $\log_2(B.M)$



Hình 8.8. Các trường địa chỉ trong hoàn toàn phối hợp

Như vậy với ví dụ 2 ở trên ta sẽ có các thông số cho trường địa chỉ như sau:

- Word field =  $\log_2 B = \log_2 16 = 4 \text{ bits}$

- Nhãn =  $\log_2 M = \log_2 (4.2^{10}) = 12 \text{ bits}$

Số bit trong trường địa chỉ bộ nhớ trong:

$$\log_2(B.M) = \log_2(16 \times 4.2^{10}) = 16 \text{ bits}$$

Ưu điểm của phương pháp này là sử dụng cache rất hiệu quả. Trong bất kỳ trường hợp nào nếu một khối nhớ trong cache còn trống thì khi có yêu cầu từ bộ VXL đến một khối nhớ khác trong bộ nhớ trong thì nó sẽ được đặt vào khối nhớ còn trống trong cache. Trừ trường hợp không còn khối nào trống thì phải áp dụng nguyên lý thay thế khối nhớ mà ta sẽ xem xét trong các phần sau. Tuy nhiên nhược điểm của phương pháp này là đòi hỏi thiết kế phần cứng phức tạp nhằm đáp ứng việc tìm kiếm khối cần thiết được nhanh hơn.

### 8.2.1.3. Kiểu phối hợp theo tập hợp (Set – Associative Mapping)

Hai cách trình bày ở trên đều có những ưu điểm và nhược điểm nhất định và cách phối hợp theo tập hợp nhằm dung hòa các ưu, nhược điểm của cả hai cách đó. Theo đó, cache bao gồm các tập hợp của các khối cache. Mỗi tập hợp của các khối cache chứa số khối như nhau. Một khối của bộ nhớ trong có thể được đặt vào một số vị trí khối giới hạn trong tập hợp được xác định bởi công thức:

$$j = i \bmod N$$

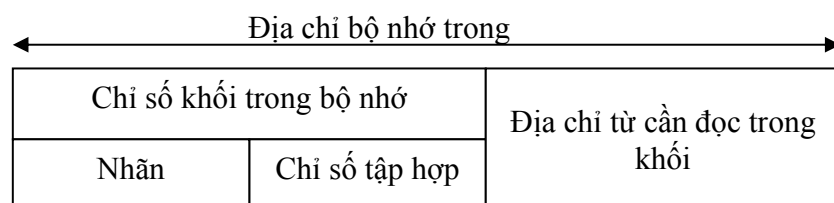
Trong đó:

- $j$ : vị trí khối đặt trong cache
- $i$ : số thứ tự của khối trong bộ nhớ trong
- $N$ : số khối của cache

Ở ví dụ 1 nếu đặt khối nhớ theo cách này và nếu mỗi tập hợp gồm 2 khối nhớ thì hình minh họa được chỉ ra trong hình 8.4 c).

Theo cách này thì MMU sẽ diễn giải địa chỉ phát ra từ CPU bằng cách chia địa chỉ thành 3 phần như là trong hình 8.9. Chiều dài tính bằng bit của mỗi phần trong đó tính như sau:

- Địa chỉ từ cần đọc trong khối (Word field) =  $\log_2 B$
- Chỉ số tập hợp (Set field) =  $\log_2 S$ ,  $S$  – số tập hợp trong cache. Chỉ số này chỉ ra số thứ tự tập hợp trong cache cần xem xét
- Nhãn (Tag field) =  $\log_2 (M/S)$ ,  $S = N/Bs$ ,  $Bs$  số khối trong một tập hợp. Dùng chỉ số này để so sánh tương ứng với nhãn của các khối cache thuộc tập hợp được chỉ ra bởi phần chỉ số tập hợp.
- Số bit trong trường địa chỉ bộ nhớ trong =  $\log_2(B.M)$



Hình 8.9. Các trường địa chỉ trong phối hợp theo tập hợp

Như vậy với ví dụ 2 ở trên ta, nếu mỗi tập hợp chứa 4 khối thì ta sẽ có các thông số cho trường địa chỉ như sau:

- Số tập hợp trong cache  $S = 128/4 = 32$  tập hợp
- Word field =  $\log_2 B = \log_2 16 = 4 \text{ bits}$
- Set field =  $\log_2 S = \log_2 32 = 5 \text{ bits}$
- Nhãn =  $\log_2 M / S = \log_2 (4.2^{10} / 32) = 7 \text{ bits}$

Số bit trong trường địa chỉ bộ nhớ trong:

$$\log_2(B.M) = \log_2(16 \times 4.2^{10}) = 16 \text{ bits}$$

### 8.2.2. KỸ THUẬT THAY THẾ KHỐI NHỚ

Như đã trình bày ở trên, trong trường hợp thất bại cache hay không tìm thấy khối nhớ trong cache thì một vấn đề đặt ra là khối nào phải được thay thế? Khi có thất bại cache, bộ điều khiển cache thâm nhập vào bộ nhớ trong và chuyển khối mà bộ xử lý cần đọc (ghi) vào cache. Như vậy, trong trường hợp các khối nhớ trong cache đều không trống thì khối nào trong cache sẽ bị thay thế bởi khối mới được chuyển lên. Đối với kiểu tương ứng trực tiếp, vị trí

đặt khối không có sự lựa chọn, nó được xác định bởi trường chỉ số khối cache trong địa chỉ của từ cần đọc (ghi). Nếu cache là kiểu hoàn toàn phối hợp hay phối hợp theo tập hợp thì khi thất bại phải chọn lựa thay thế trong nhiều khối. Có bốn chiến thuật chủ yếu dùng để chọn khối thay thế trong cache:

- *Thay thế ngẫu nhiên* (RS: Random Selection): để phân bố đồng đều việc thay thế, các khối cần thay thế trong cache được chọn ngẫu nhiên.

- *Khối xưa nhất* (LRU: Least Recently Used): các khối đã được thâm nhập sẽ được đánh dấu và khối bị thay thế là khối không được dùng từ lâu nhất.

- *Vào trước ra trước* (FIFO: First In First Out): Khối được đưa vào cache đầu tiên, nếu bị thay thế, khối đó sẽ được thay thế trước nhất.

- *Tần số sử dụng ít nhất* (LFU: Least Frequently Used): Khối trong cache được tham chiếu đến ít nhất. Nguyên tắc này sử dụng hệ quả của nguyên tắc sử dụng ô nhớ theo thời gian: nếu các khối mới được dùng có khả năng sẽ được dùng trong tương lai gần, khối bị thay thế là khối không dùng trong thời gian lâu nhất.

### 8.2.3. CHIẾN THUẬT GHI

Chúng ta đã xem xét các vấn đề chính liên quan đến kỹ thuật ánh xạ giữa các cấp bộ nhớ và chiến lược thay thế khối nhớ. Còn một vấn đề liên quan quan trọng nữa mà chúng ta sẽ xem xét trong phần này đó là sự gắn kết giữa các cấp bộ nhớ mà trong trường hợp đơn giản chúng ta xem xét giữa cache và bộ nhớ trong. Câu hỏi đặt ra là việc gì sẽ xảy ra khi cần ghi vào bộ nhớ?

Thông thường bộ VXL thâm nhập cache để đọc thông tin. Chỉ có khoảng 15% các thâm nhập vào cache là để thực hiện thao tác ghi (con số này là 33% với các tính toán vector-vector và 55% đối với các phép dịch chuyển ma trận). Như vậy, để tối ưu hoá các hoạt động của cache, các nhà thiết kế tìm cách tối ưu hoá việc đọc bởi vì các bộ xử lý phải đợi đến khi việc đọc hoàn thành nhưng sẽ

không đợi đến khi việc ghi hoàn tất. Hơn nữa, một khối có thể được đọc, so sánh và như thế việc đọc một khối có thể được bắt đầu khi chỉ số khối được biết. Nếu thao tác đọc thành công, dữ liệu ô nhớ cần đọc sẽ được giao ngay cho bộ xử lý. Cần chú ý rằng, khi một khối được ánh xạ từ bộ nhớ trong vào cache, việc đọc nội dung của khối cache không làm thay đổi nội dung của khối so với khối còn nằm trong bộ nhớ trong.

Đối với việc ghi vào bộ nhớ thì không giống như trên, việc thay đổi nội dung của một khối không thể bắt đầu trước khi nhân được xem xét để biết có thành công hay thất bại. Thao tác ghi vào bộ nhớ sẽ tốn nhiều thời gian hơn thao tác đọc bộ nhớ. Trong việc ghi bộ nhớ còn có một khó khăn khác là bộ xử lý cho biết số byte cần phải ghi, thường là từ 1 đến 8 byte. Để đảm bảo đồng nhất dữ liệu khi lưu trữ, có hai cách chính để ghi vào cache:

- *Ghi đồng thời (Write-through)*: Thông tin được ghi đồng thời vào khối của cache và khối của bộ nhớ trong. Cách ghi này làm chậm tốc độ chung của hệ thống. Các ngoại vi có thể truy cập bộ nhớ trực tiếp

- *Ghi lại (Write-back)*: Để đảm bảo tốc độ xử lý của hệ thống, thông tin cần ghi chỉ được ghi vào khối trong cache. Việc ghi vào bộ nhớ trong sẽ bị hoãn lại cho đến khi cần có thay thế khối. Để quản lý sự khác biệt nội dung giữa khối của cache và khối của bộ nhớ trong, một bit trạng thái (Dirty bit hay Update bit) được dùng để chỉ thị. Khi một thao tác ghi vào trong cache, bit trạng thái (Dirty bit hay Update bit) của khối cache sẽ được thiết lập. Khi một khối bị thay thế, khối này sẽ được ghi lại vào bộ nhớ trong chỉ khi bit trạng thái đã được thiết lập. Với cách ghi này, các ngoại vi liên hệ đến bộ nhớ trong thông qua cache. Cách này làm tăng tốc độ của hệ thống lên rất nhiều tuy nhiên việc đồng nhất dữ liệu giữa cache và bộ nhớ trong chỉ được đảm bảo vào thời gian thay thế khối.

Khi có một thất bại ghi vào cache (*cache miss*) thì phải lựa chọn một trong hai giải pháp sau:

- *Ghi có nạp (write-allocate)*: khối cần ghi từ bộ nhớ trong được nạp vào trong cache như mô tả ở trên. Cách này thường được dùng trong cách ghi lại.

- *Ghi không nạp (write-no-allocate)*: khối được thay đổi ở bộ nhớ trong không được đưa vào cache. Cách này được dùng trong cách ghi đồng thời.

Trong các tổ chức có nhiều hơn một bộ xử lý với các tổ chức cache và bộ nhớ chia sẻ, các vấn đề liên quan đến tính đồng nhất của dữ liệu cần được đảm bảo. Sự thay đổi dữ liệu trên một cache riêng lẻ sẽ làm cho dữ liệu trên các hệ thống cache và bộ nhớ liên quan không đồng nhất. Vấn đề trên có thể được giải quyết bằng một trong các hệ thống cache tổ chức như sau:

- Mỗi bộ điều khiển cache sẽ theo dõi các thao tác ghi vào bộ nhớ từ các bộ phận khác. Nếu thao tác ghi vào phần bộ nhớ chia sẻ được ánh xạ vào cache của nó quản lý, bộ điều khiển cache sẽ vô hiệu hoá sự thâm nhập này. Chiến lược này phụ thuộc vào cách ghi đồng thời trên tất cả các bộ điều khiển cache.
- Một vi mạch được dùng để điều khiển việc cập nhật, một thao tác ghi vào bộ nhớ từ một cache nào đó sẽ được cập nhật trên các cache khác.
- Một vùng nhớ chia sẻ cho một hay nhiều bộ xử lý thì không được ánh xạ lên cache. Như vậy, tất cả các thâm nhập vào vùng nhớ chia sẻ này đều bị thất bại cache.

#### 8.2.4. CÁC LOẠI CACHE

Trong một máy tính có thể có nhiều loại cache. Một số máy dùng một *Cache duy nhất* để chứa đồng thời cả lệnh và dữ liệu, nhưng một số máy lại sử dụng *Cache riêng lẻ* bằng cách sử dụng một cache lệnh riêng và một cache dữ liệu riêng (ví dụ Pentium, Pentium 4, Itanium, PowerPC 620, IBM SP,...). Giải pháp dùng cache riêng lẻ có lợi là tránh các khó khăn do kiến trúc, khi thì

hành các lệnh dùng kỹ thuật ống dẫn như đã trình bày trong chương trước.

Khi sử dụng một cache duy nhất, sẽ có tranh chấp khi một lệnh muốn thâm nhập một số liệu trong cùng một chu kỳ của giai đoạn đọc một lệnh khác. Cache riêng lẻ còn giúp tối ưu hoá mỗi loại cache về mặt kích thước tổng quát, kích thước các khối và độ phối hợp các khối.

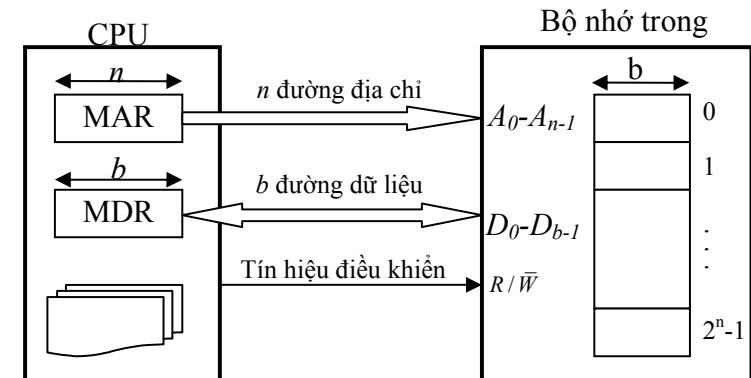
Việc dùng cache trong có thể làm cho sự cách biệt giữa kích thước và thời gian thâm nhập giữa cache trong và bộ nhớ trong càng lớn. Để linh hoạt cho việc thay đổi kích thước cache và làm giảm giá thành chung ngày nay người ta thường dùng nhiều mức cache khác nhau:

- *Cache mức một* (L1 cache): thường là cache trong (on-chip cache; nằm bên trong CPU). Cache này có kích thước nhỏ nhất và vì nằm gần CPU nhất nên dữ liệu nằm trên nó sẽ được xử lý nhanh nhất.
- *Cache mức hai* (L2 cache) thường là cache ngoài (off-chip cache; cache này nằm bên ngoài CPU). Như vậy nếu các CPU được thiết kế trên cùng một lõi có thể được cài đặt cache L2 có kích thước khác nhau.
- Ngoài ra, trong một số hệ thống (PowerPC G4, IBM S/390 G4, Itanium của Intel) còn có tổ chức *cache mức ba* (L3 cache), đây là mức cache trung gian giữa cache L2 và một thẻ bộ nhớ.

### 8.3. BỘ NHỚ TRONG (Main memory)

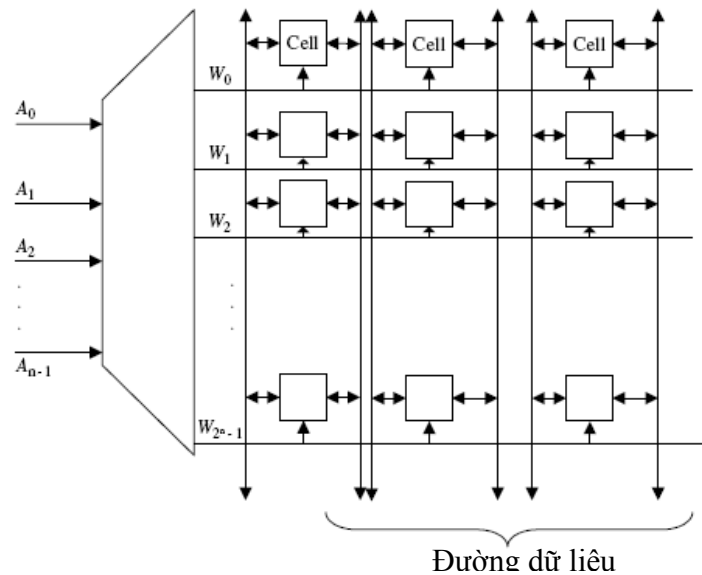
Bộ nhớ trong hay bộ nhớ chính như tên gọi của nó là bộ nhớ cung cấp nơi lưu trữ chính trong máy tính. Hình 8.10 cho ta thấy sơ đồ khối giao tiếp cơ bản giữa bộ nhớ trong và CPU. Có hai thanh ghi đặc biệt của CPU được dùng cho các thao tác giữa CPU và bộ nhớ trong đó là thanh ghi địa chỉ bộ nhớ (*Memory Address Register – MAR*) và thanh ghi dữ liệu bộ nhớ (*Memory Data*

*Register – MDR*). Thanh ghi MDR được dùng để giữ tạm thời các dữ liệu cần ghi vào (hay nhận được từ) bộ nhớ trong, nơi có địa chỉ được xác định trong thanh ghi MAR.



Hình 8.10. Giao tiếp cơ bản giữa CPU và bộ nhớ trong

Bộ nhớ trong (RAM – bộ nhớ truy cập ngẫu nhiên) được chế tạo bằng chất bán dẫn và có cấu trúc tạo thành từ những hàng và cột của các ô nhớ cơ bản (basic cells), mỗi ô nhớ chứa đựng một bit thông tin. Hình 8.11 cho ta thấy một mô hình tổ chức bộ nhớ trong đơn giản được cấu tạo từ các hàng và cột của các ô nhớ (cell). Các đường địa chỉ  $A_{n-1}A_{n-2}...A_1A_0$  được dùng như đầu vào của bộ giải mã địa chỉ theo đó cho ra các đường lựa chọn hàng  $W_{2^{n-1}}...W_1W_0$ . Tại một thời điểm nhất định thì chỉ có một trong các đường  $W_{2^{n-1}}...W_1W_0$  được chọn và các ô nhớ thuộc đường này (hàng này) sẽ được kích hoạt để cho phép truyền thông tin từ các ô nhớ ra các đường dữ liệu hoặc chuyển thông tin từ đường dữ liệu vào trong các ô nhớ



Hình 8.11. Tổ chức bộ nhớ trong trên CHIP đơn giản

Bộ nhớ RAM được gọi là bộ nhớ truy cập ngẫu nhiên do có đặc tính là các ô nhớ có thể được đọc hoặc viết vào trong khoảng thời gian bằng nhau cho dù chúng ở bất kỳ vị trí nào trong bộ nhớ. Mỗi ô nhớ có một địa chỉ, thông thường, mỗi ô nhớ là một byte (8 bit), nhưng hệ thống có thể đọc ra hay viết vào nhiều byte (2, 4, hay 8 byte). Bộ nhớ trong được đặc trưng bằng dung lượng và tổ chức của nó (số ô nhớ và số bit cho mỗi ô nhớ), thời gian thâm nhập (thời gian từ lúc đưa ra địa chỉ ô nhớ đến lúc đọc được nội dung ô nhớ đó) và chu kỳ bộ nhớ (thời gian giữa hai lần liên tiếp thâm nhập bộ nhớ).

Tuỳ theo công nghệ chế tạo, người ta phân biệt RAM tĩnh (SRAM: Static RAM) và RAM động (DRAM: Dynamic RAM).

RAM tĩnh được chế tạo theo công nghệ ECL (CMOS và BiCMOS). Mỗi bit nhớ gồm có các cổng logic với độ 6 transistor MOS, việc nhớ một dữ liệu là tồn tại nếu bộ nhớ được cung cấp

điện. SRAM là bộ nhớ nhanh, việc đọc không làm huỷ nội dung của ô nhớ và thời gian thâm nhập bằng chu kỳ bộ nhớ.

RAM động dùng kỹ thuật MOS. Mỗi bit nhớ gồm có một transistor và một tụ điện. Cũng như SRAM, việc nhớ một dữ liệu là tồn tại nếu bộ nhớ được cung cấp điện. Việc ghi nhớ dựa vào việc duy trì điện tích nạp vào tụ điện và như vậy việc đọc một bit nhớ làm nội dung bit này bị huỷ (do tụ điện phóng điện tích). Vậy sau mỗi lần đọc một ô nhớ, bộ phận điều khiển bộ nhớ phải viết lại ô nhớ đó nội dung vừa đọc và do đó chu kỳ bộ nhớ động ít nhất là gấp đôi thời gian thâm nhập ô nhớ. Việc lưu giữ thông tin trong bit nhớ chỉ là tạm thời vì tụ điện sẽ phóng hết điện tích đã nạp vào và như vậy phải làm tươi bộ nhớ sau mỗi vài mili giây. Làm tươi bộ nhớ là đọc ô nhớ và viết lại nội dung đó vào lại ô nhớ. Việc làm tươi được thực hiện với tất cả các ô nhớ trong bộ nhớ. Việc làm tươi bộ nhớ được thực hiện tự động bởi một vi mạch bộ nhớ. Bộ nhớ DRAM chậm nhưng rẻ tiền hơn SRAM. Hình 8.12. cho ta thấy sơ đồ cấu tạo cơ bản của một bit nhớ của hai loại bộ nhớ này. Ô nhớ SRAM chứa một trong hai trạng thái bền vững. Ví dụ trong hình 8.12 a) nếu  $A = 1$  thì transistor  $N_2$  sẽ được bật lên và nó đóng vai trò như một dây dẫn kéo điện thế tại điểm B xuống 0 ( $B = 0$ ), điều đó lại dẫn đến transistor  $P_1$  được bật lên và kéo điện thế A bằng điện thế nguồn ( $A = 1$ ). Ta gọi trạng thái bền vững này là trạng thái 1, hay ô nhớ đang chứa giá trị tương ứng với bit 1. Lập luận tương tự nếu  $A = 0$  ta sẽ có trạng thái tương ứng với bit 0.

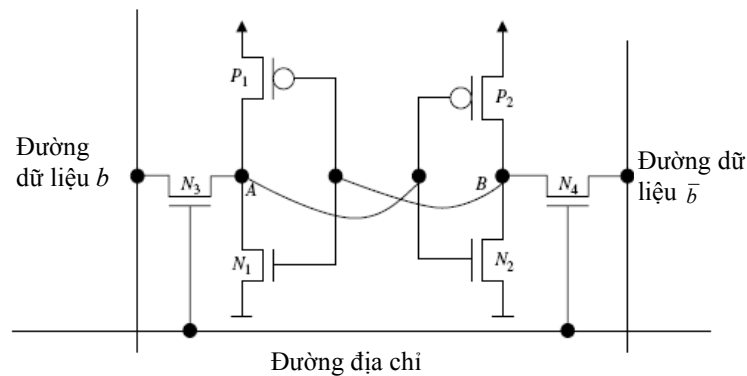
Khi đọc dữ liệu sẽ theo các bước:

- Đường dữ liệu được kích hoạt và được nạp ở mức cao ( $b = 1$ ).
- Đường địa chỉ được kích hoạt ( $=1$ ) làm bật 2 transistor  $N_3$  và  $N_4$ .
- Phụ thuộc vào điện thế tại điểm A mà ta đọc được giá trị của bit nhớ này. Nếu điện thế tại A cao thì điện thế này chuyển sang đường  $b$  làm điện thế  $b$  cũng cao, hay

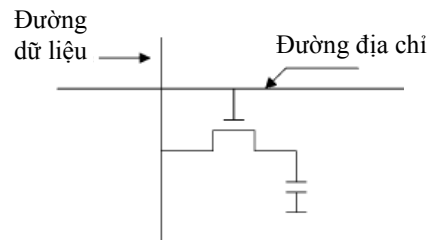


ta nhận được giá trị bit nhớ là 1. Ngược lại, nếu điện thế tại A thấp thì điện thế ở đường  $b$  sẽ truyền qua điểm A do đó làm suy giảm điện thế trên  $b$  và ta xác định được bit nhớ có giá trị là 0.

Lập luận tương tự cho trường hợp ghi một giá trị vào bit nhớ.



a) Ô nhớ SRAM



a) Ô nhớ DRAM

Hình 8.12. Cấu tạo ô nhớ SRAM và DRAM

Trong các bộ nhớ RAM lại chia ra làm nhiều loại cơ bản như sau:

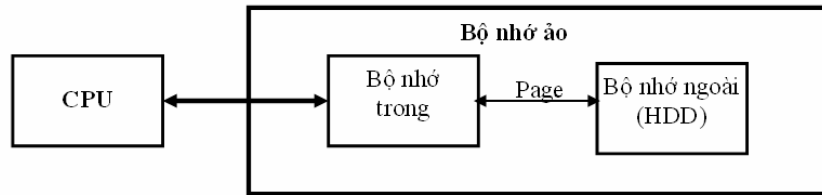
- SDRAM (*Synchronous DRAM* – DRAM đồng bộ), một dạng DRAM đồng bộ bus bộ nhớ. Tốc độ SDRAM đạt từ 66-133MHz (thời gian truy nhập bộ nhớ từ 75ns-150ns).

- DDR SDRAM (*Double Data Rate SDRAM*) là cải tiến của bộ nhớ SDRAM với tốc độ truyền tải gấp đôi SDRAM nhờ vào việc truyền tải hai lần trong một chu kỳ bộ nhớ. Tốc độ DDR SDRAM đạt từ 200-400MHz. Trong thời gian gần đây xuất hiện chuẩn RAM mới dựa trên nền tảng DDR là DDR-II, DDR-III có tốc độ cao hơn nhờ cải tiến thiết kế.
- RDRAM (*Rambus RAM*) là một loại DRAM được thiết kế với kỹ thuật hoàn toàn mới so với kỹ thuật SDRAM. RDRAM hoạt động đồng bộ theo một hệ thống lập và truyền dữ liệu theo một hướng. Một kênh bộ nhớ RDRAM có thể hỗ trợ đến 32 chip DRAM. Mỗi chip được ghép nối tuần tự trên một module gọi là RIMM (*Rambus Inline Memory Module*) nhưng việc truyền dữ liệu giữa các mạch điều khiển và từng chip riêng biệt chứ không truyền giữa các chip với nhau. Bus bộ nhớ RDRAM là đường dẫn liên tục đi qua các chip và module trên bus, mỗi module có các chân vào và ra trên các đầu đối diện. Do đó, nếu các khe cắm không chứa RIMM sẽ phải gắn một module liên tục để đảm bảo đường truyền được nối liền. Tốc độ RDRAM đạt từ 400-800MHz

## 8.4. BỘ NHỚ ẢO

Người ta đã sử dụng cơ chế *bộ nhớ ảo* nhằm giải quyết vấn đề về kích thước bộ nhớ vật lý không đủ chứa cả hệ điều hành cùng với các chương trình của người sử dụng, đồng thời vấn đề các vùng nhớ phải được bảo vệ một cách chắc chắn để khỏi bị chương trình của người sử dụng làm hỏng. Bộ nhớ ảo có được dựa trên sự kết hợp các bộ nhớ với tốc độ rất cao như bộ nhớ trong (RAM) và bộ nhớ có tốc độ chậm như bộ nhớ phụ ( ổ đĩa cứng), hoạt động dưới sự quản lý của MMU, sao cho dưới quan điểm của người lập trình và đối với người sử dụng thì tập hợp các bộ nhớ trên được quan niệm là một bộ nhớ thuần nhất với dung lượng lớn (gần hoặc bằng dung lượng ổ đĩa cứng) nhưng lại làm việc ở tốc độ cao (gần bằng tốc độ bộ nhớ trong).

Cấu trúc phân cấp bộ nhớ được minh họa như sơ đồ hình 8.13.



Hình 8.13. Tổ chức bộ nhớ ảo

Bộ nhớ ảo có thể được quản lý bằng cách chia bộ nhớ thành các mảng nhỏ có độ lớn tính theo *đoạn*, cơ chế này gọi là *phân đoạn* (đối với họ Intel có từ các bộ VXL 80286 trở đi) hoặc *trang*, cơ chế này gọi là *phân trang* (đối với họ Intel có từ các bộ VXL 80386) trở đi. Trong bộ nhớ ảo như vậy, từng mảng mã lệnh và mảng dữ liệu dùng cho chương trình hiện tại được tải từ ổ đĩa vào bộ nhớ trong (RAM) và được truy nhập đến bởi bộ điều khiển của bộ nhớ khi cần thiết. Nếu chương trình đang chạy cần đến một mảng mã lệnh hay một mảng dữ liệu nào đó mà không chứa trong RAM thì nó sẽ được tải vào RAM. Nếu RAM không còn đủ không gian để chứa thì một mảng nào đó của RAM sẽ được xác định theo các tiêu chuẩn nhất định để bị đẩy lại ổ đĩa, nhường chỗ cho mảng đang cần lấy vào.

Cách hoạt động của bộ nhớ ảo giữa bộ nhớ chính và bộ nhớ phụ rất giống nguyên lý hoạt động giữa bộ nhớ cache và bộ nhớ chính mà ta đã trình bày trong phần trên. Bộ nhớ RAM đóng vai trò là bộ nhớ chứa bản sao của một phần bộ nhớ phụ, bộ nhớ cache đóng vai trò là bộ nhớ chứa bản sao của một phần bộ nhớ chính. Tuy có nguyên lý hoạt động giống nhau nhưng sự khác biệt lớn nhất là khi thất bại cache, sự thay thế một khối trong cache được điều khiển bằng phần cứng, trong khi sự thay thế trong bộ nhớ ảo là chủ yếu do hệ điều hành. Ngoài ra khi thất bại cache thì thời gian bị phạt chỉ tương đương với khoảng 5-10 lần trong trường hợp thành

công cache, trong khi nếu có lỗi trang thì có thể phải mất một thời gian gấp 1000 lần khi không có lỗi trang.

Trong các máy tính hiện đại 1 đoạn có thể có độ lớn từ 1 byte đến 4GB còn 1 trang thông thường có độ lớn từ 2KB đến 16 K bytes.

Để truy cập đến bộ nhớ ảo ta dùng địa chỉ ảo (địa chỉ logic), còn truy nhập đến bộ nhớ vật lý ta phải dùng địa chỉ vật lý. Khi cần truy cập đến một đoạn dữ liệu nào đó thì CPU phát ra một yêu cầu truy cập đến một địa chỉ, đó chính là địa chỉ ảo. MMU nhận địa chỉ ảo và dịch (chuyển đổi) nó ra địa chỉ vật lý, địa chỉ này được đưa lên bus địa chỉ để truy nhập bộ nhớ vật lý.

Tương tự như khi làm việc với cache, để hiểu rõ nguyên lý vận hành của bộ nhớ ảo, ta cũng phải trả lời được 4 câu hỏi cơ bản sau:

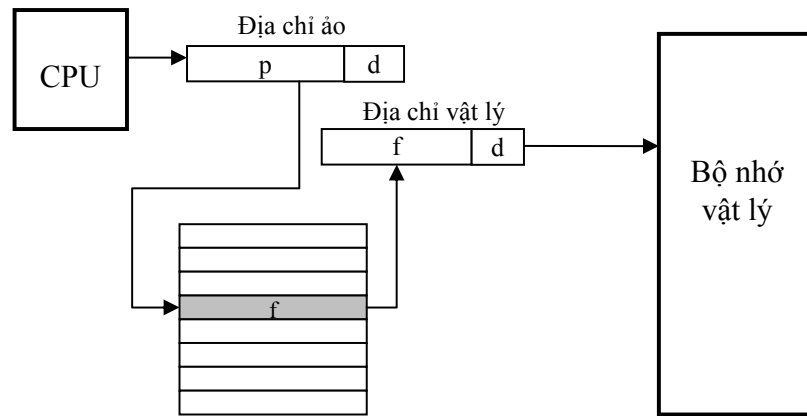
- Một khối bộ nhớ ngoài sẽ được đặt tại đâu trong bộ nhớ trong?
- Làm thế nào để tìm một khối khi nó đang nằm trong bộ nhớ trong?
- Khối nào phải được thay thế khi có thất bại trang?
- Việc gì xảy ra khi cần ghi số liệu?

Để đặt một khối nhớ vào trong bộ nhớ trong cũng có 3 kỹ thuật là tương ứng trực tiếp, hoàn toàn phối hợp và phối hợp theo tập hợp. Việc trừng phạt bộ nhớ ảo khi có thất bại, tương ứng với việc phải thâm nhập vào ổ đĩa. Việc thâm nhập này rất chậm nên người ta chọn phương án hoàn toàn phối hợp trong đó các khối (trang) có thể nằm ở bất kỳ vị trí nào trong bộ nhớ trong. Cách này cho tỉ lệ thất bại thấp.

Để tìm một khối nhớ ta cần phân biệt hai dạng định vị, đó là định vị trang và định vị đoạn. Địa chỉ phát ra bởi CPU là địa chỉ ảo và trong cả hai loại định vị, địa chỉ này đều chứa một trường để lưu số thứ tự trang hoặc số thứ tự đoạn.

Trong trường hợp định vị trang, dựa vào bảng trang, địa chỉ trong bộ nhớ vật lý được xác định bằng cách đặt kế nhau số thứ của trang vật lý với địa chỉ trong trang như trong hình 8.14.

Trong trường hợp định vị đoạn, dựa vào thông tin trên bảng đoạn, việc kiểm tra tính hợp lệ của địa chỉ được tiến hành. Địa chỉ vật lý được xác định bằng cách cộng địa chỉ đoạn và độ dời trong đoạn như trong hình 8.15.



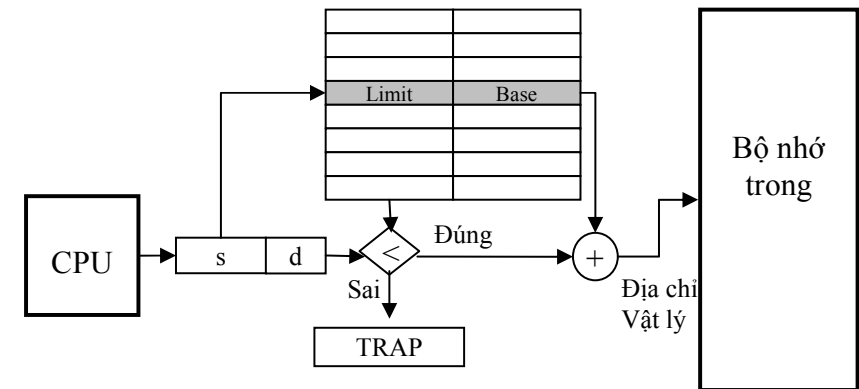
p – chỉ số trang    d – độ dời trang  
f – chỉ số khung trang (frame)

Hình 8.14. Ánh xạ địa chỉ ảo sang địa chỉ vật lý trong định vị trang

Khi có thất bại trang thì khối cần được thay thế sẽ tuân theo các chiến thuật thay thế trang giống như các chiến thuật thay thế khối trong cache gồm có: *Thay thế ngẫu nhiên*, *Khối xưa nhất*, *Vào trước ra trước*, *Tần số sử dụng ít nhất*. tuy nhiên hầu hết các hệ điều hành ngày nay đều cố gắng dùng chiến thuật thay thế khối ít dùng gần đây nhất (LRU: Least Recent Utilized) vì cho rằng đây là khối ít cần nhất.

Đối với chiến thuật trong bộ nhớ ảo thì luôn sử dụng chiến thuật ghi lại (write-back), nghĩa là thông tin chỉ được viết vào trong

khối của bộ nhớ trong. Khối có thay đổi thông tin, được chép vào đĩa từ nếu khối này bị thay thế.



s – chỉ số đoạn    d – độ dời đoạn  
Limit – Giới hạn tối đa của đoạn  
Base – địa chỉ cơ sở của đoạn

Hình 8.15. Ánh xạ địa chỉ ảo sang địa chỉ vật lý trong định vị đoạn

## • TỔNG KẾT

Trong chương này đã trình bày các kiến thức cơ bản về tổ chức bộ nhớ trong máy tính. Qua đó cho thấy sự khác biệt giữa các loại bộ nhớ trong máy tính và các kỹ thuật kết hợp các loại bộ nhớ, từ bộ nhớ lớn chậm, đến các bộ nhớ rất nhỏ nhưng cực nhanh thành một không gian nhớ chung với tốc độ cao và giá cả phải chăng. Ngoài ra còn cung cấp các kiến thức về thiết kế và cấu tạo của các loại bộ nhớ, qua đó cung cấp cho sinh viên các hiểu biết cơ bản về hệ thống bộ nhớ trong máy tính.

## CÂU HỎI VÀ BÀI TẬP CHƯƠNG VIII

- Mục tiêu chính của các cấp bộ nhớ trong máy tính là gì? Vẽ sơ đồ các cấp bộ nhớ cơ bản.
- Tính thời gian truy cập trung bình của một hệ thống bộ nhớ có 3 cấp: cache, bộ nhớ trong và bộ nhớ phụ nếu thời gian truy cập vào từng loại bộ nhớ tương ứng là 20 ns, 100 ns và 1 ms biết rằng tỷ số thành công của cache là 90% và bộ nhớ trong là 95%.
- Hãy xác định số bit của các trường trong địa chỉ bộ nhớ trong ở ví dụ 1 trong 3 trường hợp ánh xạ. Trong trường hợp phối hợp theo tập hợp, giả sử mỗi tập hợp gồm 2 khối nhớ.
- Nêu và giải thích các nguyên tắc vận hành của cache.
- Vẽ sơ đồ chuyển đổi địa chỉ cho ví dụ 2 trong trường hợp phối hợp hoàn toàn như trên hình 8.7 và giải thích cơ chế hoạt động của nó.
- Vẽ sơ đồ chuyển đổi địa chỉ cho ví dụ 2 trong trường hợp phối hợp theo tập hợp nếu mỗi tập hợp gồm 4 khối như trên hình 8.7 và giải thích cơ chế hoạt động của nó.
- Cho một bộ nhớ cache tương ứng trực tiếp có 8 khối, mỗi khối có 16 byte. Bộ nhớ trong có 64 khối. Giả sử lúc khởi động máy, 8 khối đầu tiên của bộ nhớ trong được đưa lên cache.
  - Viết bảng nhãn của các khối hiện đang nằm trong cache
  - CPU lần lượt đưa các địa chỉ sau đây để đọc số liệu: 04AH, 27CH, 3F5H. Nếu thất bại thì cập nhật bảng nhãn.
  - CPU dùng cách ghi lại. Khi thất bại cache, CPU dùng cách ghi có nạp. Mô tả công việc của bộ quản lý cache khi CPU đưa ra các từ sau đây để ghi vào bộ nhớ trong: 0C3H, 05AH, 1C5H.

- Hãy nêu các nguyên nhân chính gây thất bại cache và cách hạn chế nó?
- Xét một ma trận số có kích thước 4x8. Giả sử mỗi số được lưu trong một từ và các phần tử của ma trận được lưu theo thứ tự cột trong bộ nhớ từ địa chỉ 1000 đến địa chỉ 1031. Bộ nhớ cache gồm 8 khối với mỗi khối chứa được 2 từ. Ta cũng giả sử dùng chiến thuật thay thế khối là LRU. Hãy khảo sát sự thay đổi trong cache (sự thay đổi trong các khối nhớ trong cache) trong 3 kỹ thuật tổ chức bộ nhớ nếu CPU yêu cầu truy cập lần lượt đến các phần tử theo thứ tự sau:

$$a_{0,0}, a_{0,1}, a_{0,2}, a_{0,3}, a_{0,4}, a_{0,5}, a_{0,6}, a_{0,7}$$

$$a_{1,0}, a_{1,1}, a_{1,2}, a_{1,3}, a_{1,4}, a_{1,5}, a_{1,6}, a_{1,7}$$

- Hãy cho biết sự khác nhau giữa SRAM và DRAM?
- Tại sao phải dùng bộ nhớ ảo? cho biết sự khác biệt cơ bản giữa cache và bộ nhớ ảo?

## PHỤ LỤC

	Trang
<b>Lời nói đầu</b>	1
<b>Chương I: Giới thiệu</b>	3
1.1. Lịch sử phát triển của máy tính	3
1.1.1. Thế hệ zero-máy tính cơ học (1642-1945)	3
1.1.2. Thế hệ I – bóng đèn điện (1945-1955)	4
1.1.3. Thế hệ II – transistor (1955-1965)	9
1.1.4. Thế hệ III – mạch tích hợp (1965-1980)	13
1.1.5. Thế hệ IV – máy tính cá nhân (1980-đến nay)	16
1.2. Khối các nước XHCN và Việt Nam	18
1.3. Khuynh hướng hiện tại	21
1.4. Phân loại máy tính	23
1.5. Các dòng Intel	24
Câu hỏi và bài tập chương I	31
<b>Chương II: Các bộ phận cơ bản của máy tính</b>	32
2.1. Bộ xử lý (CPU)	32
2.2. Bản mạch chính (Mainboard)	41
2.3. Ổ đĩa mềm (FDD)	48
2.4. Ổ đĩa cứng (HDD)	50
2.5. Ổ CD và DVD	53
2.6. Bộ nhớ RAM và ROM	55
2.7. Bàn phím (Keyboard)	59
2.8. Chuột (Mouse)	60
2.9. Card màn hình (VGA Card)	63
2.10. Màn hình (Monitor)	66
2.11. Card mạng (Network adapter) và Modem	74
Câu hỏi và bài tập chương II	76
<b>Chương III: Biểu diễn dữ liệu</b>	77
3.1. Khái niệm thông tin	77
	275

3.2. Lượng thông tin và sự mã hóa thông tin	78
3.3. Hệ thống số	79
3.4. Các phép tính số học cho hệ nhị phân	85
3.4.1. Phép cộng nhị phân không dấu	85
3.4.2. Phép trừ nhị phân không dấu	86
3.4.3. Phép nhân và chia hai số nhị phân không dấu	87
3.4.4. Biểu diễn số nguyên có dấu	88
3.4.5. Số bù của một số	89
3.4.6. Phép cộng trừ nhị phân dùng bù 1	92
3.4.7. Phép cộng trừ nhị phân dùng bù 2	93
3.5. Số quá n (excess-n)	94
3.6. Cách biểu diễn số với dấu chấm động	95
3.7. Biểu diễn số BCD	98
3.8. Biểu diễn các ký tự	100
Câu hỏi và bài tập chương III	101
<b>Chương IV: Mạch Logic số</b>	103
4.1. Cổng và đại số Boolean	103
4.1.1. Cổng (Gate)	103
4.1.2. Đại số Boolean	108
4.2. Bản đồ Karnaugh	122
4.3. Những mạch Logic số cơ bản	135
4.3.1. Mạch tích hợp (IC-Intergrate Circuit)	135
4.3.2. Mạch kết hợp (Combinational Circuit)	137
4.3.3. Bộ dồn kênh-bộ phân kênh	138
4.3.4. Mạch cộng (Adder)	142
4.3.5. Mạch giải mã và mã hóa	144
Câu hỏi và bài tập chương IV	151
<b>Chương V: Mạch tuần tự</b>	155
5.1. Xung đồng hồ	155
5.2. Mạch lật (chốt – latch)	156
5.2.1. Mạch lật SR (SR-latch)	157
5.2.2. Mạch lật D	160
5.2.3. Mạch lật JK	161
5.3.4. Mạch lật T	161
	276

5.3. Mạch lật lẻ (Flip-flop)	162
5.4. Mạch tuần tự	165
Bài tập chương V	173
<b>Chương VI: Kiến trúc bộ lệnh</b>	175
6.1. Phân loại kiến trúc bộ lệnh	175
6.2. Địa chỉ bộ nhớ	179
6.3. Mã hóa tập lệnh	181
6.3.1. Các tiêu chuẩn thiết kế dạng thức lệnh	182
6.3.2. Opcode mở rộng	184
6.3.3. Ví dụ về dạng thức lệnh	187
6.3.4. Các chế độ lập địa chỉ	191
6.4. Bộ lệnh	201
6.4.1. Nhóm lệnh truyền dữ liệu	203
6.4.2. Nhóm lệnh tính toán số học	204
6.4.3. Nhóm lệnh Logic	206
6.4.4. Nhóm các lệnh dịch chuyển	207
6.4.5. Nhóm các lệnh có điều kiện và lệnh nhảy	209
6.5. Cấu trúc lệnh CISC và RISC	212
Câu hỏi và bài tập chương VI	216
<b>Chương VII: Tổ chức bộ xử lý</b>	218
7.1. Tổ chức bộ xử lý trung tâm	218
7.2. Bộ điều khiển	221
7.3. Bộ thanh ghi	224
7.4. Đường đi dữ liệu (Datapath)	227
7.4.1. Tổ chức One-Bus	228
7.4.2. Tổ chức Two-Bus, Three-Bus	229
7.5. Diễn tiến thi hành lệnh mã máy	231
7.6. Xử lý ngắt (Interrupt Handling)	236
7.7. Kỹ thuật ống dẫn (Pipeline)	238
Câu hỏi và bài tập chương VII	244
<b>Chương VIII: Hệ thống bộ nhớ</b>	245
8.1. Các cấp bộ nhớ	245
8.2. Bộ nhớ cache (Cache memory)	248
8.2.1. Tổ chức bộ nhớ cache	251
	277

8.2.2. Kỹ thuật thay thế khối nhớ	259
8.2.3. Chiến thuật ghi	260
8.2.4. Các loại cache	262
8.3. Bộ nhớ trong (Main memory)	263
8.4. Bộ nhớ ảo (Virtual memory)	268
Câu hỏi và bài tập chương VIII	273
Phụ lục	275
Tài liệu tham khảo	279

## TÀI LIỆU THAM KHẢO

1. Nguyễn Minh Tuấn, Kiến trúc máy tính (giáo trình lược giản), V3.7, Trường ĐH Khoa học tự nhiên tp. HCM
2. Cấu trúc máy tính cơ bản, tổng hợp và biên dịch VN-Guide, nhà xuất bản thống kê.
3. Võ Đức Khánh ,Kiến trúc máy tính, ThS. Võ Đức Khánh
4. Võ Văn Chín, Nguyễn Hồng Vân, Phạm Hữu Tài. Giáo trình kiến trúc máy tính. ĐH Cần Thơ, 2003.
5. M. Abd-El-Barr, H. El-Rewini, Fundamentals of Computer Organization and Architecture, Wiley, 2005
6. Patterson, D. A., and J. L. Hennessy. *Computer Organization and Design: The Hardware/Software Interface*, 3rd ed. San Mateo, CA: Morgan Kaufman, 2004
7. Robert Bruce Thompson, Barbara Fritchman Thompson, Repairing and Upgrading Your PC, O'Reilly, 2006
8. William Stallings. Computer Organization & Architecture (Designing for performance), Sixth edition, Pearson Education, 2003
9. M. Abd-El-Barr, H. El-Rewini, Advanced Computer Architecture and Parallel Processing, Wiley, 2005