

MỤC LỤC

Mục lục	1
Chương 1. Giới thiệu	8
1.1. Tổ chức và kiến trúc	8
1.2. Cấu trúc và chức năng	9
1.2.1. Chức năng	9
1.2.2. Cấu trúc	12
1.3. Câu hỏi	13
Chương 2. Sự phát triển của máy tính và hiệu năng	14
2.1. Lịch sử máy tính	14
2.1.1. Thé hệ đầu tiên: Đèn óng chân không	14
2.1.2. Thé hệ thứ hai: Transistor	20
2.1.3. Thé hệ thứ ba: Mạch tích hợp	21
2.1.4. Các thé hệ tiếp theo	26
2.2. Các đặc điểm thiết kế máy tính	30
2.2.1. Tốc độ vi xử lý	30
2.2.2. Cân bằng Hiệu suất	31
2.2.3. Cài tiến kiến trúc và tổ chức Chip	32
2.3. Đa nhân, mic và gpu	32
2.4. Kiến trúc intel x86	33
2.5. Hệ thống nhúng và kiến trúc arm	34
2.5.1. Hệ thống nhúng	34
2.5.2. Sự phát triển của ARM	36
2.6. Câu hỏi	37
Chương 3. Chức năng và kết nối máy tính	39
3.1. Các thành phần máy tính	39
3.2. Chức năng của máy tính	42
3.2.1. Truy xuất và thực thi lệnh	42
3.2.2. Ngắt	46
3.2.3. Chức năng vào/ra	53
3.3. Cấu trúc kết nối	54

3.4. Kết nối bus	56
3.4.1. Cấu trúc bus.....	56
3.4.2. Mô hình phân cấp đa bus	58
3.4.3. Các thành phần của thiết kế bus	60
3.5. Kết nối điểm-điểm	63
3.5.1. Lớp vật lý QPI	66
3.5.2. Lớp liên kết QPI	67
3.5.3. Lớp định tuyến QPI	68
3.5.4. Lớp giao thức QPI	68
3.6. PCI Express	68
3.6.1. Kiến trúc vật lý và logic của PCI.....	69
3.6.2. Lớp vật lý PCIe.....	71
3.6.3. Lớp giao vận PCIe	72
3.6.4. Lớp liên kết dữ liệu PCIe	75
3.7. Câu hỏi	75
Chương 4. Bộ nhớ cache	76
4.1. Tổng quan hệ thống bộ nhớ máy tính	76
4.1.1. Các đặc tính của hệ thống bộ nhớ.....	76
4.1.2. Phân cấp bộ nhớ.....	78
4.2. Nguyên lý bộ nhớ cache.....	80
4.3. Các yếu tố thiết kế cache.....	83
4.3.1. Địa chỉ cache	83
4.3.2. Kích thước cache	85
4.3.3. Hàm ánh xạ.....	86
4.3.4. Thuật toán thay thế	96
4.3.5. Chính sách ghi	97
4.3.6. Kích thước line	97
4.3.7. Số lượng cache	98
4.4. Tổ chức cache pentium 4	100
4.5. Câu hỏi	101
Chương 5. Bộ nhớ trong.....	103
5.1. Bộ nhớ bán dẫn.....	103

5.1.1.	Tổ chức	103
5.1.2.	DRAM và SRAM	103
5.1.3.	Các loại ROM – Bộ nhớ chỉ đọc	106
5.1.4.	Chip Logic	107
5.1.5.	Đóng gói chip	109
5.1.6.	Tổ chức module nhớ	110
5.1.7.	Tổ chức bộ nhớ đan xen	112
5.2.	Sửa lỗi.....	112
5.3.	Tổ chức DRAM mở rộng	117
5.3.1.	SDRAM – DRAM đồng bộ	117
5.3.2.	Rambus DRAM	120
5.3.1.	DDR SDRAM	120
5.3.1.	Cache DRAM	121
5.4.	Câu hỏi	122
Chương 6. Bộ nhớ ngoài.....		123
6.1.	Đĩa từ.....	123
6.1.1.	Cơ chế đọc và ghi từ	123
6.1.2.	Tổ chức dữ liệu	124
6.1.3.	Đặc tính vật lý.....	126
6.1.4.	Thông số hiệu suất đĩa.....	128
6.2.	Raid	129
6.2.1.	RAID cấp 0.....	130
6.2.2.	RAID cấp 1	134
6.2.3.	RAID cấp 5	135
6.2.4.	RAID cấp 6.....	136
6.3.	Ô đĩa bán dẫn SSD	136
6.3.1.	Bộ nhớ flash.....	136
6.3.2.	SSD so với HDD	137
6.3.3.	Tổ chức SSD	138
6.3.4.	Ván đè thực tế	139
6.4.	Bộ nhớ quang	140
6.4.1.	Đĩa CD.....	141

6.4.2.	CD-R	143
6.4.3.	CD-RW	143
6.4.4.	DVD	144
6.4.5.	Đĩa quang độ phân giải cao	145
6.5.	Băng từ	146
6.6.	Câu hỏi	147
Chương 7.	Module vào/ra (I/O)	148
7.1.	Thiết bị ngoại vi	148
7.1.1.	Bàn phím/Màn hình	150
7.1.2.	Ổ cứng	151
7.2.	Các module I/O	151
7.2.1.	Chức năng của module	151
7.2.2.	Cấu trúc Module I/O	152
7.3.	Các kỹ thuật vào/ra	153
7.3.1.	I/O chương trình	154
7.3.2.	I/O điều khiển ngắt	157
7.3.3.	Cơ chế DMA – Truy cập bộ nhớ trực tiếp	165
7.4.	Kênh vào/ra và Bộ xử lý vào/ra	171
7.4.1.	Quá trình phát triển của các chức năng I/O	171
7.4.2.	Đặc điểm của các Kênh I/O	171
7.5.	Giao Tiếp ngoài	173
7.5.1.	Các loại giao tiếp	173
7.5.2.	Cấu hình Điểm – Điểm và Đa điểm	174
7.6.	Câu hỏi	174
Chương 8.	Hệ đếm	175
8.1.	Hệ thập phân	175
8.2.	Hệ đếm có vị trí	176
8.3.	Hệ nhị phân	176
8.4.	Chuyển đổi giữa nhị phân và thập phân	177
8.4.1.	Phản nguyên	177
8.4.2.	Phản phân số	178
8.5.	Ký hiệu thập lục phân	180

8.6. Câu hỏi	181
Chương 9. Tính toán số học.....	183
9.1. Khối tính toán số học và logic	183
9.2. Biểu diễn số nguyên	183
9.2.1. Biểu diễn dấu – độ lớn	184
9.2.2. Biểu diễn bù hai	184
9.2.3. Mở rộng phạm vi.....	187
9.3. Các phép toán số học với số nguyên.....	189
9.3.1. Phép phủ định	189
9.3.2. Phép toán cộng và phép toán trừ.....	190
9.3.1. Phép nhân	192
9.3.2. Phép chia	201
9.4. Biểu diễn dấu chấm động	204
9.4.1. Nguyên tắc	204
9.4.2. Chuẩn IEEE cho biểu diễn số nhị phân dấu chấm động	207
9.5. Các phép toán với dấu chấm động	212
9.5.1. Phép cộng và phép trừ	213
9.5.2. Phép nhân và phép chia	214
9.5.3. Độ chính xác của phép toán.....	217
9.5.4. Chuẩn IEEE với các phép toán số học nhị phân dấu chấm động	218
9.6. Câu hỏi	219
Chương 10. Tập lệnh: Đặc điểm và chức năng	221
10.1. Đặc điểm của lệnh máy.....	221
10.1.1. Các thành phần của lệnh máy.....	221
10.1.2. Biểu diễn lệnh.....	222
10.1.3. Phân loại lệnh	223
10.1.4. Số lượng địa chỉ trong lệnh	224
10.1.5. Các vấn đề thiết kế tập lệnh	227
10.2. Các kiểu toán hạng	227
10.2.1. Số	228
10.2.2. Ký tự	228
10.2.3. Dữ liệu logic	229

10.3.	Kiểu dữ liệu trong Intel x86 và arm	229
10.3.1.	Các kiểu dữ liệu trong Intel x86.....	229
10.3.1.	Các kiểu dữ liệu trong ARM.....	231
10.3.2.	Hỗ trợ Endian.....	232
10.4.	Các loại hành động	232
10.4.1.	Truyền dữ liệu	235
10.4.2.	Xử lý số học	235
10.4.3.	Xử lý logic	236
10.4.4.	Điều khiển vào/ra	238
10.4.5.	Điều khiển hệ thống	238
10.4.6.	Chuyển điều khiển (rẽ nhánh)	239
10.5.	Câu hỏi.....	243
Chương 11. Tập lệnh: chế độ địa chỉ và định dạng		245
11.1.	Chế độ địa chỉ	245
11.1.1.	Địa chỉ tức thì.....	247
11.1.2.	Địa chỉ trực tiếp.....	247
11.1.3.	Địa chỉ gián tiếp	248
11.1.4.	Địa chỉ thanh ghi	249
11.1.5.	Địa chỉ gián tiếp thanh ghi	250
11.1.6.	Địa chỉ dịch chuyển	250
11.1.7.	Địa chỉ ngăn xếp.....	253
11.2.	Định dạng lệnh	253
11.2.1.	Kích thước lệnh.....	253
11.2.2.	Phân bổ các bit	254
11.2.3.	Các lệnh có độ dài thay đổi.....	258
11.3.	Câu hỏi.....	261
Chương 12. Cấu trúc và chức năng của bộ xử lý		262
12.1.	Tổ chức của bộ xử lý	262
12.2.	Tổ chức thanh ghi.....	263
12.2.1.	Thanh ghi hiển thị với người dùng	264
12.2.2.	Thanh ghi điều khiển và trạng thái.....	265
12.2.3.	Ví dụ tổ chức thanh ghi vi xử lý.....	266

12.3.	CHU KÌ LỆNH.....	267
12.3.1.	Chu kì gián tiếp	267
12.3.2.	Luồng dữ liệu	268
12.4.	Pipeline lệnh.....	271
12.4.1.	Chiến lược pipelining	271
12.4.2.	Hazard trong pipeline.....	276
12.4.3.	Đối phó với hazard rẽ nhánh	278
12.4.4.	Pipeline trong Intel 80486.....	282
12.5.	Câu hỏi.....	284

Chương 1. GIỚI THIỆU

Giáo trình này giới thiệu với các bạn sinh viên về cấu trúc và chức năng của máy tính. Chúng tôi mong muốn trình bày một cách rõ ràng, cụ thể nhất có thể về các đặc điểm và tính chất của các hệ máy tính hiện đại ngày nay. Tuy nhiên, có hai vấn đề gặp phải như sau

Thứ nhất, hiện nay trên thị trường có rất nhiều các dòng sản phẩm máy tính khác nhau từ những máy tính đơn chip giá rẻ đến các siêu máy tính giá hàng chục triệu đô. Không chỉ giá thành, chúng còn khác nhau từ kích thước, hiệu năng và ứng dụng. Thứ hai, công nghệ máy tính thay đổi một cách nhanh chóng từ việc sử dụng các mạch tích hợp để chế tạo các thành phần máy tính đến các khái niệm tổ chức song song trong việc kết hợp các thành phần đó.

Mặc dù có sự đa dạng cũng như tốc độ thay đổi công nghệ nhanh chóng, các hệ máy tính ngày nay vẫn tuân theo một số khái niệm cơ bản xuyên suốt. Trong cuốn sách này, chúng tôi trình bày một cách kỹ lưỡng về các khái niệm cơ bản của tổ chức và kiến trúc máy tính cũng như một số các vấn đề thiết kế máy tính hiện đại

1.1. TỔ CHỨC VÀ KIẾN TRÚC

Trong mô tả máy tính, có hai khái niệm cần phân biệt là *kiến trúc máy tính* và *tổ chức máy tính*. Mặc dù khá khó để có thể đưa ra định nghĩa tóm lược của hai khái niệm này, chúng ta có thể hiểu như sau

Kiến trúc máy tính bao gồm các thuộc tính của một hệ thống mà người lập trình có thể nhìn thấy được, hay nói cách khác, đây là các thuộc tính có tác động lên việc thực thi một chương trình máy tính. **Tổ chức máy tính** có thể hiểu là các khối chức năng trong máy tính và sự kết nối giữa chúng để thực hiện các đặc tính của kiến trúc. Ví dụ, các thuộc tính của kiến trúc bao gồm tập lệnh, số bit để biểu diễn các kiểu dữ liệu khác nhau (vd: ký tự, số,...), cơ chế vào/ra (I/O) và cách kỹ thuật định địa chỉ bộ nhớ. Các thuộc tính của tổ chức bao gồm các đặc điểm phần cứng như: các tín hiệu điều khiển, giao diện giữa máy tính và các thiết bị ngoại vi và các công nghệ bộ nhớ được sử dụng.

Ví dụ, về mặt thiết kế kiến trúc máy tính: ta muốn thực hiện lệnh Nhân. Vậy, về mặt tổ chức sẽ như sau: lệnh này sẽ được thực hiện bởi một đơn vị phần cứng đặc biệt hoặc một cơ chế cho phép thực hiện lặp đi lặp lại phép cộng. Việc lựa chọn một trong hai phương án trên phụ thuộc vào tần suất dự kiến lệnh nhân được sử dụng, tương quan tốc độ tính toán của hai phương pháp, chi phí và kích thước vật lý mỗi một phương án tổ chức.

Hiện nay, sự khác nhau giữa kiến trúc và tổ chức vẫn còn khá rõ nét. Nhiều nhà sản xuất máy tính đưa ra một họ các model máy tính, tất cả chúng đều có kiến trúc tương tự nhau nhưng lại khác nhau về tổ chức. Các model có giá cả và hiệu suất khác nhau. Một kiến trúc đặc biệt có thể tồn tại trong nhiều năm và với nhiều model máy tính khác nhau nhưng tổ chức của nó thay đổi theo sự thay đổi của công nghệ. Ví dụ kiến trúc IBM System/370 được đưa ra lần đầu tiên vào năm 1970 với một số model khác nhau. Khách hàng có nhu cầu thấp, đơn giản có thể mua một model rẻ hơn, chậm hơn và nếu có nhu cầu nâng cấp thì sau đó có thể nâng cấp lên một model đắt tiền hơn và nhanh hơn mà không cần phải bỏ phần mềm đã được phát triển trên đó. Trong những năm qua, IBM đã giới thiệu nhiều model mới với công nghệ cải tiến để thay thế các model cũ, cung cấp cho khách

hàng các hệ máy tính tốc độ cao hơn, chi phí thấp hơn hoặc cả hai. Những model mới hơn vẫn có cùng kiến trúc với model cũ để phần mềm của khách hàng được bảo vệ. Kiến trúc System/370 vẫn tồn tại đến ngày nay và là kiến trúc của dòng sản phẩm máy tính mainframe của hãng IBM.

Với một lớp máy tính khác gọi là máy vi tính, mỗi quan hệ giữa kiến trúc và tổ chức rất chặt chẽ. Những thay đổi trong công nghệ không chỉ ảnh hưởng đến tổ chức mà còn dẫn đến việc các kiến trúc máy tính mạnh hơn và phức tạp hơn. Tuy nhiên, yêu cầu tương thích giữa các thế hệ máy tính này khá thấp vì vậy việc thiết kế tổ chức và kiến trúc máy tính có nhiều tương tác hơn.

Cuốn sách này trình bày cả tổ chức máy tính và kiến trúc máy tính, nhấn mạnh nhiều hơn về mặt tổ chức. Tuy nhiên, vì một tổ chức máy tính được thiết kế để thực hiện một đặc tả kiến trúc cụ thể, việc đưa ra được một tổ chức toàn diện đòi hỏi phải có sự nghiên cứu chi tiết về kiến trúc máy tính.

1.2. CẤU TRÚC VÀ CHỨC NĂNG

Máy tính là một hệ thống phức tạp; một máy tính hiện đại chứa hàng triệu thành phần điện tử cơ bản. Làm thế nào ta có thể mô tả chúng một cách đơn giản, rõ ràng nhất? Câu trả lời là ta cần xác định tính phân cấp của các hệ thống phức tạp bao gồm cả máy tính. Một hệ thống phân cấp là một tập các hệ thống con tương quan với nhau, mỗi hệ thống con lại có cấu trúc phân cấp bên trong nó cho đến khi ta đạt đến cấp hệ thống thấp nhất.

Tính phân cấp của các hệ thống phức tạp hết sức quan trọng trong cả thiết kế và mô tả chúng. Các nhà thiết kế chỉ cần đối phó với một cấp độ nhất định của hệ thống tại một thời điểm. Ở mỗi cấp độ, hệ thống bao gồm một bộ các thành phần và các mối quan hệ tương tác của chúng. Hành vi ở mỗi cấp chỉ phụ thuộc vào một đặc tính đơn giản, trừu tượng của hệ thống ở cấp dưới nó. Khi làm việc với mỗi cấp độ, nhà thiết kế sẽ tập trung đến cấu trúc và chức năng của cấp đó:

- **Cấu trúc:** Cách thức mà các thành phần quan hệ tương tác với nhau.
- **Chức năng:** Hoạt động của mỗi bộ phận riêng như một phần của cấu trúc.

Để mô tả một hệ thống, chúng ta thường sử dụng một trong hai cách như sau: bắt đầu từ dưới lên (từ các thành phần cơ bản) sau đó xây dựng dần lên thành một bản mô tả đầy đủ, hoặc bắt đầu từ trên xuống sau đó phân tách hệ thống thành các thành phần nhỏ hơn. Từ trong thực tiễn, người ta thấy rằng phương pháp mô tả từ trên xuống là phương pháp rõ ràng và hiệu quả nhất và sẽ được sử dụng trong cuốn sách này.

Như vậy, với hệ thống máy tính, chúng ta bắt đầu với các thành phần chính của máy tính, mô tả cấu trúc và chức năng của chúng và tiếp tục tới các lớp thấp hơn trong phân cấp máy tính.

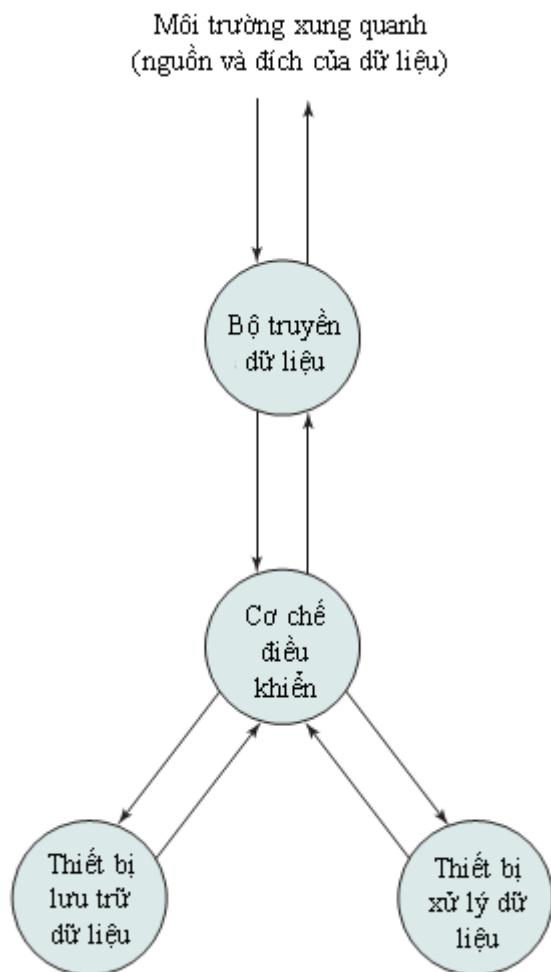
1.2.1. Chức năng

Về cơ bản, cấu trúc và chức năng của một máy tính khá đơn giản. Hình 1.1 mô tả các chức năng cơ bản mà một máy tính có thể thực hiện gồm có bốn chức năng chính sau:

- Xử lý dữ liệu
- Lưu trữ dữ liệu

10 Chương 1. Giới thiệu

- Di chuyển dữ liệu
- Điều khiển

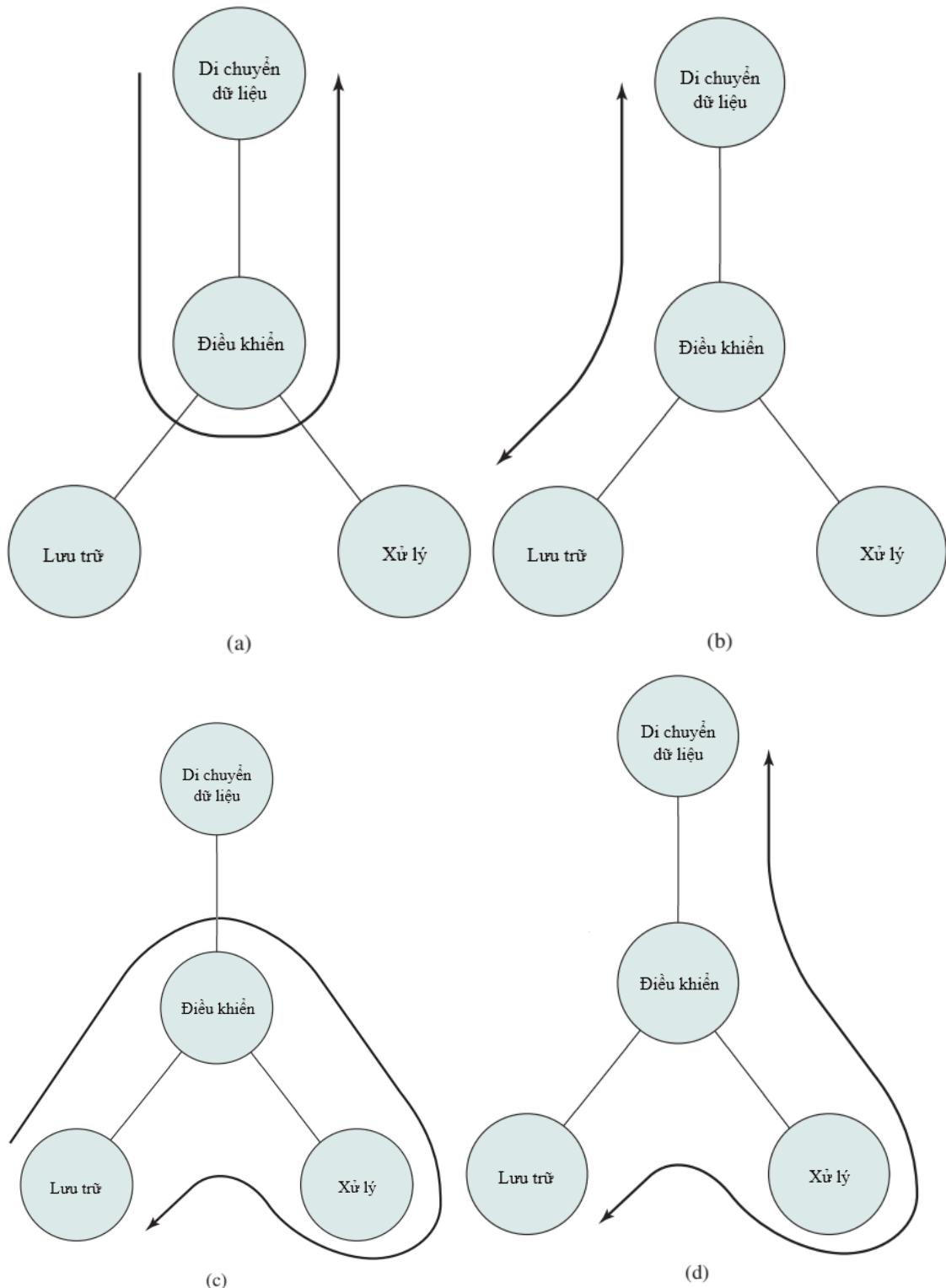


Hình 1.1 Các chức năng chính của máy tính

Đầu tiên, máy tính phải có khả năng **xử lý dữ liệu**. Dữ liệu có thể có nhiều dạng khác nhau và có rất nhiều loại yêu cầu xử lý khác nhau. Tuy nhiên, chúng ta sẽ thấy rằng chỉ có một số phương pháp xử lý dữ liệu cơ bản.

Một chức năng không kém phần quan trọng của máy tính là **lưu trữ dữ liệu**. Ngay cả khi đang trong quá trình xử lý dữ liệu, máy tính cũng phải lưu trữ tạm thời những dữ liệu đang được làm việc tại bất kỳ thời điểm nào. Do đó, phải có ít nhất một chức năng lưu trữ dữ liệu ngắn hạn. Ngoài ra, máy tính còn thực hiện chức năng lưu trữ dữ liệu dài hạn. Các tập tin dữ liệu được lưu trữ trên máy tính với mục đích phục hồi và cập nhật.

Máy tính phải có **di chuyển dữ liệu** giữa nó và thế giới bên ngoài. Việc trao đổi dữ liệu được thực hiện giữa hai bên: một bên là thiết bị nguồn và một bên là thiết bị đích đến của dữ liệu. Khi dữ liệu được nhận vào hoặc gửi đến một thiết bị kết nối trực tiếp với máy tính, quá trình này được gọi là *quá trình vào/ra* (I/O) dữ liệu, và thiết bị đó được gọi là *thiết bị ngoại vi*. Khi dữ liệu được truyền qua các khoảng cách dài hơn, gửi đến hoặc nhận từ một thiết bị từ xa ta gọi là *quá trình truyền thông dữ liệu*.



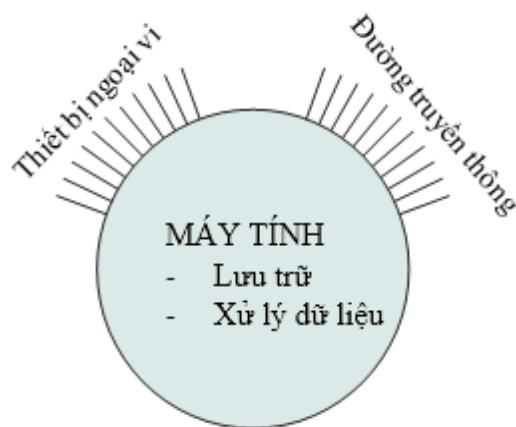
Hình 1.2 Một số hoạt động của máy tính

Cuối cùng, phải một chức năng thực hiện việc **điều khiển** ba chức năng này. Chức năng này được thực hiện thông qua các *lệnh* máy. Một khối điều khiển (control unit) quản lý các tài nguyên của máy tính, sắp xếp hoạt động của các phần chức năng để đáp ứng các lệnh đó.

Hình 1.2 mô tả bốn loại hoạt động có thể thực hiện được. Với Hình 1.2a máy tính có thể hoạt động như một thiết bị truyền dữ liệu, dữ liệu được truyền từ một thiết bị ngoại vi hoặc một đường truyền thông sang một bên khác. Máy tính cũng có thể hoạt động như một thiết bị lưu trữ dữ liệu (Hình 1.2b): dữ liệu truyền từ môi trường bên ngoài vào và được lưu trữ trong máy tính (đọc) và ngược lại (viết). Hai hình cuối cùng là các hoạt động liên quan đến xử lý dữ liệu: dữ liệu được xử lý có thể là dữ liệu đang được lưu trữ (hình 1.2c) hoặc dữ liệu đang trên đường trao đổi giữa bộ nhớ và môi trường bên ngoài (Hình 1.2d).

1.2.2. Cấu trúc

Hình 1.3 là dạng mô tả đơn giản nhất của máy tính. Máy tính tương tác với môi trường bên ngoài. Nói chung, tất cả các liên kết của nó với môi trường bên ngoài có thể được phân thành các loại như thiết bị ngoại vi hoặc đường truyền thông tin.

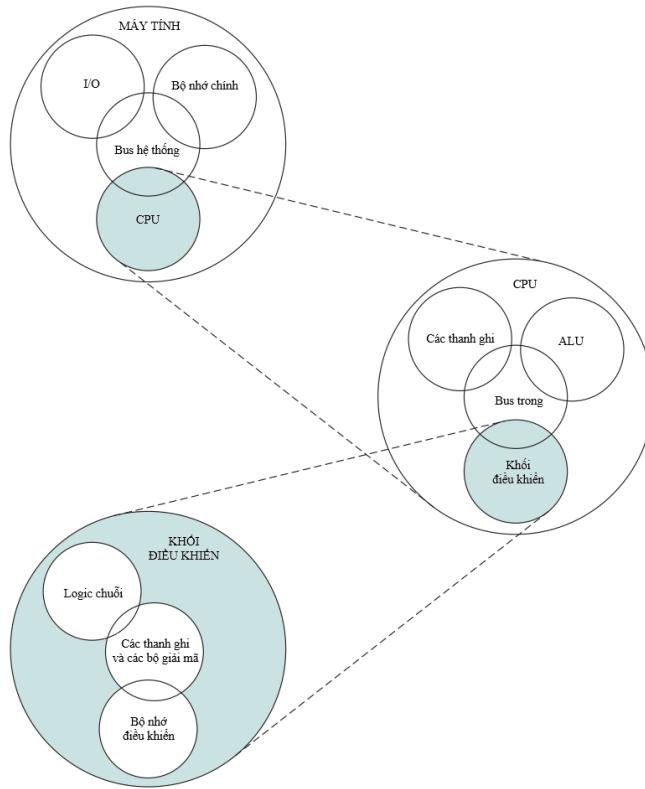


Hình 1.3 Máy tính

Nhưng mục tiêu chủ yếu của cuốn sách này là cấu trúc bên trong của máy tính, được thể hiện trong hình 1.4. Cấu trúc này có bốn cấu phần chính:

- **Bộ phận xử lý trung tâm (CPU):** Điều khiển hoạt động của máy tính và thực hiện các chức năng xử lý dữ liệu; thường được gọi là bộ vi xử lý.
- **Bộ nhớ chính:** Lưu trữ dữ liệu.
- **I/O (Vào/ra):** Trao đổi dữ liệu giữa máy tính và môi trường bên ngoài.
- **Hệ thống kết nối:** Một số cơ chế cho phép truyền thông tin giữa CPU, bộ nhớ chính, và I/O. Một ví dụ về hệ thống kết nối phổ biến nhất là hệ thống bus: gồm có một số dây dẫn kết nối tất cả các thành phần của máy tính.

Một hệ thống máy tính có thể có một hoặc nhiều thành phần kể trên. Trước đây, chỉ có một bộ xử lý duy nhất cho một hệ máy tính. Tuy nhiên trong những năm gần đây, người ta đã thiết kế sử dụng nhiều bộ xử lý trong một máy tính. Với các mục đích chính của chúng ta, giáo trình này đi sâu vào trình bày kiến trúc và tổ chức cơ bản nhất của một hệ thống máy tính nên các vấn đề liên quan đến bộ xử lý đa nhân chúng tôi sẽ cập nhật ở các bản sách tham khảo sau này. Quay lại với tổ chức của CPU, các thành phần chính của nó như sau:



Hình 1.4 Cấu trúc phân cấp của máy tính

- **Bộ điều khiển:** Điều khiển hoạt động của CPU và máy tính.
- **Khối tính toán số học và logic – ALU (Arithmetic and logic unit):** Thực hiện các chức năng xử lý dữ liệu của máy tính.
- **Thanh ghi:** Lưu trữ nội bộ trong CPU.
- **Cấu trúc kết nối trong CPU:** Một số cơ chế cho phép truyền thông tin giữa các đơn vị Điều khiển, ALU và Thanh ghi.

Mỗi thành phần này sẽ được trình bày chi tiết trong các Chương cuối của quyển Giáo trình này.

1.3. CÂU HỎI

1. Phân biệt tổ chức và kiến trúc máy tính.
2. Phân biệt cấu trúc và chức năng máy tính.
3. Các chức năng chính của máy tính là gì?
4. Liệt kê và mô tả ngắn gọn các thành phần chính của máy tính.
5. Liệt kê và mô tả ngắn gọn các thành phần chính của bộ xử lý.

Chương 2. SỰ PHÁT TRIỂN CỦA MÁY TÍNH VÀ HIỆU NĂNG

Chúng ta bắt đầu nghiên cứu về máy tính thông qua tìm hiểu lịch sử ra đời và phát triển của máy tính. Điều này sẽ giúp cho chúng ta có cái nhìn tổng thể về cấu trúc và chức năng của máy tính. Tiếp theo, vấn đề về hiệu suất sẽ được trình bày. Cuối cùng, sự phát triển của hai hệ thống sử dụng họ vi xử lý Intel x86 và ARM sẽ được giới thiệu ngắn gọn. Để tiện cho việc trình bày, hai hệ thống này được sử dụng làm ví dụ xuyên suốt trong các nội dung tiếp theo trong tài liệu này.

2.1. LỊCH SỬ MÁY TÍNH

2.1.1. Thẻ hệ đầu tiên: Đèn ống chân không

2.1.1.1. ENIAC

ENIAC (Electronic Numerical Integrator And Computer) được thiết kế và chế tạo tại trường Đại học Pennsylvania, là máy tính điện tử số đa năng đầu tiên trên thế giới. Dự án này đáp ứng nhu cầu của Hoa Kỳ trong chiến tranh thế giới II. Phòng thí nghiệm Nghiên cứu đạn đạo Quân đội (BRL), tổ chức chịu trách nhiệm phát triển phạm vi và bảng quy đao cho một loại vũ khí mới, đã gặp khó khăn trong việc đưa ra các bảng quy đao chính xác trong khoảng thời gian ngắn. Nếu không có những bảng bắn này, loại vũ khí mới trở nên vô dụng với các tay súng. BRL đã thuê hơn 200 người sử dụng bàn tính để giải các phép toán cần thiết. Một người làm việc riêng lẻ mất nhiều giờ (thậm chí cả ngày) để chuẩn bị xong bảng đạn đao cho một vũ khí đơn.

John Mauchly, giáo sư ngành kỹ thuật điện của Đại học Pennsylvania, cùng với một sinh viên của ông là John Eckert đã đề xuất xây dựng một máy tính đa năng sử dụng đèn ống chân không cho bài toán của BRL. Năm 1943, Quân đội chấp nhận đề xuất này, và công việc bắt đầu được triển khai trên ENIAC. Máy ENIAC rất to, nặng tới 30 tấn, chiếm hơn 450m² mặt sàn và chứa hơn 18000 đèn chân không. Khi hoạt động, nó tiêu thụ 140 kilowatt điện. Nó cũng nhanh hơn hẳn so với máy tính cơ điện, có khả năng thực hiện 5000 phép cộng trên giây.

ENIAC là máy thập phân chứ không phải một máy nhị phân. Tức là, số được biểu diễn dưới dạng thập phân, và phép tính số học thực hiện trong hệ thập phân. Bộ nhớ của nó bao gồm 20 bộ tích luỹ (accumulator), mỗi bộ tích luỹ có thể trữ một số thập phân 10 chữ số. Mỗi chữ số lại được biểu diễn bởi một vòng gồm 10 đèn ống chân không. Tại một thời điểm, chỉ có một đèn ống chân không ở trạng thái ON, thể hiện một trong 10 chữ số. Hạn chế lớn nhất của ENIAC là nó phải được lập trình thủ công bằng cách thiết lập các công tắc chuyển mạch và cắm rút dây cáp.

ENIAC hoàn thành vào năm 1946, quá muộn để được sử dụng phục vụ chiến tranh. Thay vào đó, nhiệm vụ đầu tiên của nó là thực hiện một loạt các phép tính phức tạp để xác định tính khả thi của bom hydro. Việc sử dụng ENIAC cho một mục đích khác với mục đích xây dựng ban đầu đã chứng minh được tính đa năng của máy này. ENIAC tiếp tục hoạt động dưới sự quản lý của BRL cho đến năm 1955 thì nó bị tháo rời.

2.1.1.2. Máy tính Von Neumann

Nhiệm vụ nhập và thay đổi chương trình cho ENIAC là cực kỳ tẻ nhạt. Nhưng giả sử nếu có một chương trình được thể hiện dưới dạng phù hợp để lưu trữ trong bộ nhớ cùng

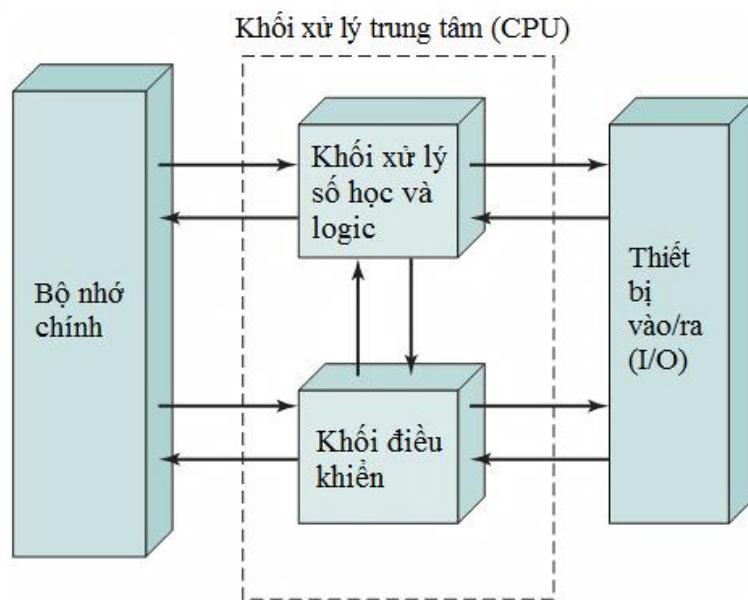
với dữ liệu. Khi đó, máy tính có thể đọc được lệnh từ bộ nhớ, và một chương trình có thể được thiết lập hoặc thay đổi bằng cách thiết lập giá trị của một phần trong bộ nhớ.

Ý tưởng này được gọi là **chương trình lưu trữ được**, thường được biết đến là ý tưởng của các nhà thiết kế ENIAC, đặc biệt là nhà toán học John von Neumann, chuyên gia tư vấn cho dự án ENIAC. Cùng thời điểm đó Alan Turing cũng phát triển ý tưởng này. Công bố đầu tiên của ý tưởng này là do von Neumann đưa ra năm 1945 trong một máy tính mới, EDVAC (Electronic Discrete Variable Computer).

Năm 1946, von Neumann và các cộng sự của ông bắt đầu thiết kế một máy tính chương trình lưu trữ mới, gọi là máy tính IAS, tại Viện Nghiên cứu cao cấp Princeton. Mặc dù cho đến năm 1952 vẫn không hoàn thành, máy tính IAS được ghi nhận là nguyên mẫu của tất cả các máy tính đa năng sau này.

Hình 2.1 cho thấy cấu trúc chung của máy tính IAS (so với Hình 1.4). Nó bao gồm

- Một **bộ nhớ chính**, lưu trữ cả dữ liệu và lệnh
- Một **khối số học và logic (ALU)** có khả năng xử lý dữ liệu nhị phân
- Một **khối điều khiển (CU)**, biên dịch các lệnh trong bộ nhớ và cho phép lệnh được thực hiện
- **Thiết bị vào/ra (I/O)** vận hành bởi khối điều khiển



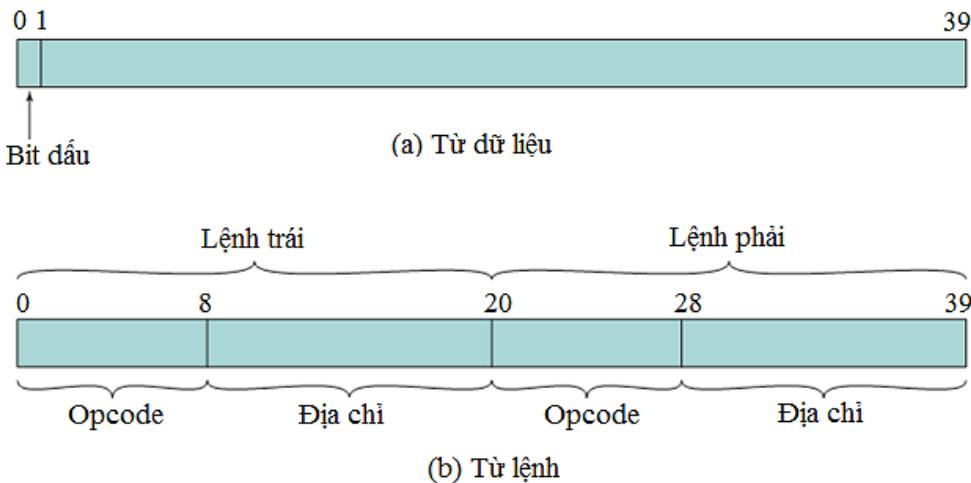
Hình 2.1 Cấu trúc máy tính IAS

Hầu như tất cả các máy tính ngày nay đều có cấu trúc và chức năng chung giống như trên và do đó được gọi là máy von Neumann. Vì vậy, ta sẽ mô tả một cách ngắn gọn hoạt động của máy tính IAS.

Bộ nhớ của IAS bao gồm 1000 vị trí lưu trữ, được gọi là **tù (word)**, mỗi vị trí có 40 chữ số (bit) nhị phân. Cả dữ liệu và lệnh được lưu trữ ở đó. Số được biểu diễn dưới dạng nhị phân, và mỗi lệnh là một mã nhị phân. Hình 2.2 mô tả các định dạng này. Mỗi số được biểu diễn bởi 1 bit dấu và 39 bit giá trị. Một từ cũng có thể chứa hai lệnh, mỗi lệnh gồm

16 Chương 2. Sự phát triển của máy tính và hiệu năng

20 bit. Trong 20 bit lệnh, 8 bit dành cho mã lệnh (**opcode**) chỉ định hành động nào được thực hiện và 12 bit dành cho địa chỉ để xác định một từ cụ thể trong bộ nhớ (đánh số từ 0 đến 999).

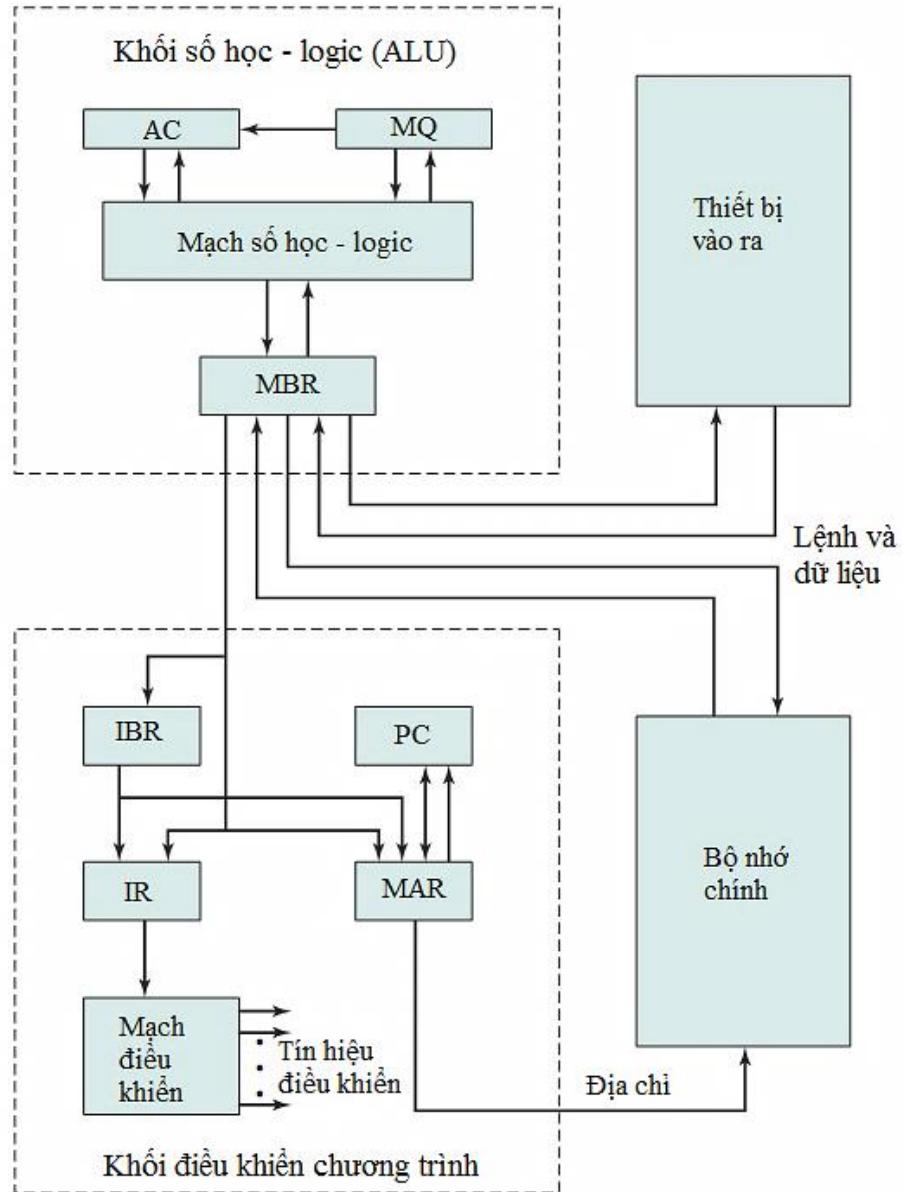


Hình 2.2 Định dạng bộ nhớ IAS

Khối điều khiển vận hành máy IAS bằng cách truy xuất các lệnh từ bộ nhớ và thực hiện lần lượt từng lệnh. Hình này cho thấy cả bộ điều khiển và ALU đều chứa các vị trí lưu trữ, được gọi là thanh ghi. Để giải thích rõ hơn, ta xét một sơ đồ cấu trúc chi tiết hơn ở hình 2.3. Hình vẽ này cho thấy cả khối điều khiển và khối ALU đều có các vị trí lưu trữ, được gọi là *thanh ghi* và được định nghĩa như sau:

- **Thanh ghi đệm dữ liệu (MBR):** Chứa một từ sắp được lưu vào bộ nhớ hoặc sắp được gửi ra khỏi I/O, hoặc được dùng để nhận một từ đến từ bộ nhớ hoặc từ các khôi I/O.
- **Thanh ghi địa chỉ bộ nhớ (MAR):** Chứa địa chỉ bộ nhớ của từ sắp được đọc hoặc ghi vào MBR.
- **Thanh ghi tập lệnh (IR):** Chứa lệnh với mã lệnh 8 bit sắp được thực hiện.
- **Thanh ghi đệm chứa tập lệnh (IBR):** Được sử dụng để tạm thời lưu trữ lệnh nằm bên phải của 1 từ trong bộ nhớ.
- **Bộ đếm chương trình (PC):** Chứa địa chỉ của cặp lệnh tiếp theo cần truy xuất từ bộ nhớ.
- **Bộ cộng tích luỹ (AC) và bộ nhân chia (MQ):** Được sử dụng để tạm thời lưu trữ các toán hạng và kết quả của các phép tính trong ALU. Ví dụ, kết quả của phép nhân hai số 40 bit là một số 80 bit; 40 bit quan trọng hơn được lưu trữ trong AC và 40 bit ít quan trọng hơn lưu trong MQ.

IAS hoạt động bằng cách thực hiện lặp đi lặp lại từng **chu kỳ lệnh**, như thể hiện trong hình 2.4. Mỗi chu kỳ lệnh bao gồm hai chu kỳ con. Trong **chu kỳ truy xuất**, opcode của lệnh tiếp theo được nạp vào IR và phần địa chỉ được nạp vào MAR. Lệnh này có thể được lấy từ IBR, hoặc có thể được lấy từ bộ nhớ bằng cách nạp một từ vào MBR, và rồi đưa xuống IBR, IR, và MAR.



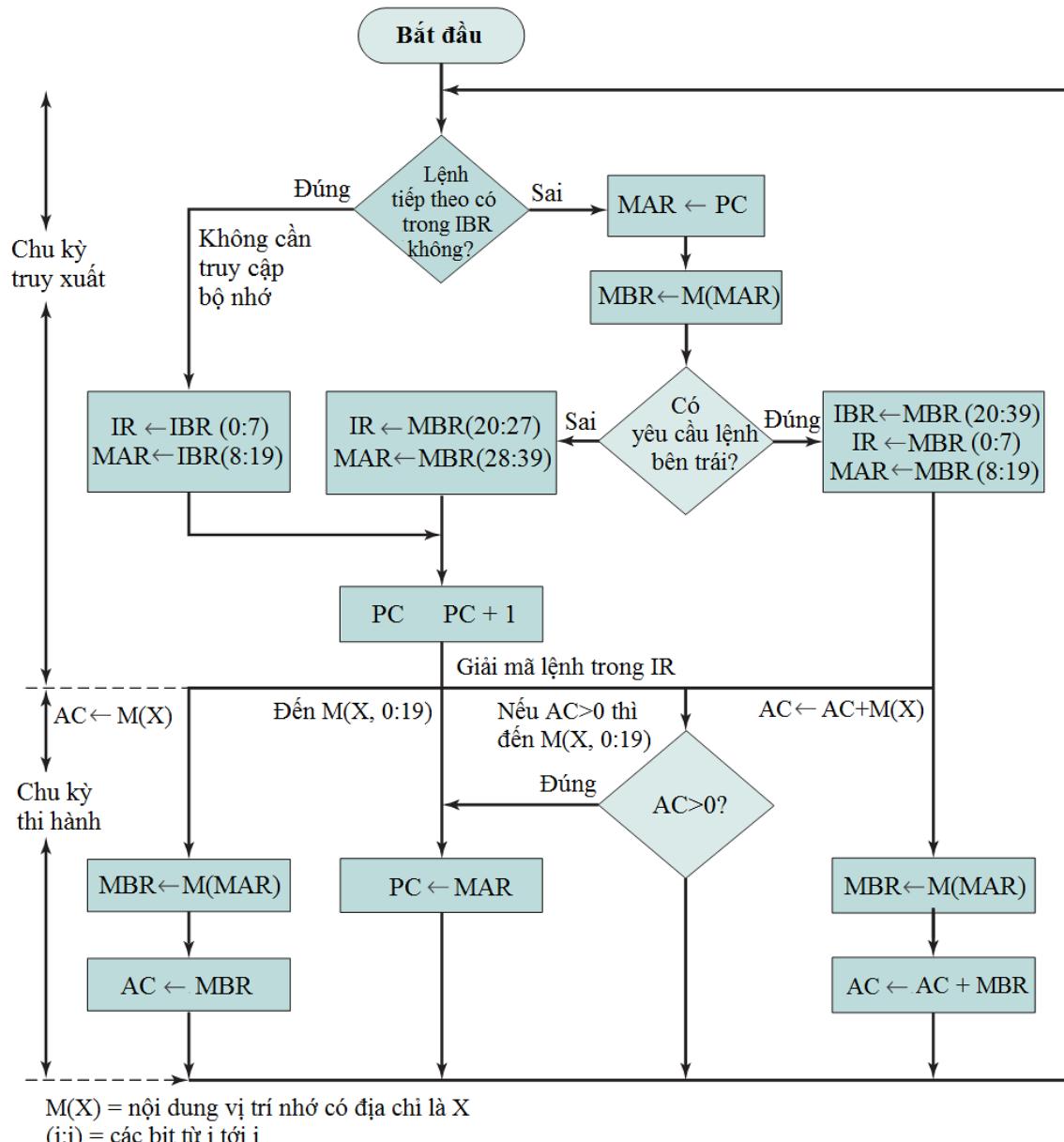
Hình 2.3 Cấu trúc mở rộng của máy IAS

Một khi opcode đã được nạp vào IR, **chu kỳ thực thi** được diễn ra. Mạch điều khiển giải mã opcode và thực hiện lệnh bằng cách gửi các tín hiệu điều khiển phù hợp để di chuyển dữ liệu hoặc thực hiện phép tính.

Máy tính IAS có tổng cộng 21 lệnh, được liệt kê trong Bảng 2.1. Các lệnh này có thể được phân thành các nhóm như sau:

- **Truyền dữ liệu:** Di chuyển dữ liệu giữa bộ nhớ và các thanh ghi ALU hoặc giữa hai thanh ghi ALU.
- **Rẽ nhánh không điều kiện:** Thông thường, khối điều khiển lần lượt thực hiện các lệnh trong bộ nhớ. Trình tự thực hiện này có thể được thay đổi bởi một lệnh rẽ nhánh, nhằm hỗ trợ cho các hành động lặp lại.

- Rẽ nhánh có điều kiện:** Nhánh có thể được tạo ra tùy thuộc vào điều kiện, do đó sẽ có các điểm quyết định.
- Số học:** Các phép tính được thực hiện bởi ALU.
- Sửa đổi địa chỉ:** Cho phép tính toán địa chỉ trong ALU rồi sau đó chèn địa chỉ vào các lệnh đã lưu trữ trong bộ nhớ. Điều này cho phép một chương trình định địa chỉ một cách linh hoạt hơn.



Hình 2.4 Một phần lưu đồ hoạt động của IAS

Bảng 2.1 Tập lệnh trong IAS

Kiểu lệnh	Opcode	Biểu diễn ký tự	Mô tả
Truyền dữ liệu	00001010	LOAD MQ	Truyền nội dung thanh ghi MQ tới thanh ghi AC
	00001001	LOAD MQ,M(X)	Truyền nội dung vị trí nhớ X tới MQ
	00100001	STOR M(X)	Truyền nội dung trong AC tới vị trí nhớ X
	00000001	LOAD M(X)	Truyền M(X) tới thanh ghi AC
	00000010	LOAD -M(X)	Truyền -M(X) tới thanh ghi AC
	00000011	LOAD M(X)	Truyền giá trị tuyệt đối của M(X) tới AC
	00000100	LOAD - M(X)	Truyền - M(X) tới thanh ghi AC
Nhánh không điều kiện	00001101	JUMP M(X,0:19)	Lấy lệnh tiếp theo từ nửa trái của M(X)
	00001110	JUMP M(X,20:39)	Lấy lệnh tiếp theo từ nửa phải của M(X)
Nhánh có điều kiện	00001111	JUMP +M(X,0:19)	Nếu số trong AC không âm, lấy lệnh tiếp theo từ nửa trái của M(X)
	00010000	JUMP +M(X,20:39)	Nếu số trong AC không âm, lấy lệnh tiếp theo từ nửa phải của M(X)
Số học	00000101	ADD M(X)	Cộng M(X) với AC; kết quả đặt vào AC
	00000111	ADD M(X)	Cộng M(X) với AC; kết quả đặt vào AC
	00000110	SUB M(X)	Lấy AC trừ M(X); kết quả đặt vào AC
	00001000	SUB M(X)	Lấy AC trừ M(X) ; kết quả đặt vào AC
	00001011	MUL M(X)	Nhân M(X) với MQ; các bit quan trọng nhất của kết quả đặt vào AC, các bit ít quan trọng nhất của kết quả đặt vào MQ
	00001100	DIV M(X)	Chia AC cho M(X); thương số đặt vào MQ, phần dư đặt vào AC
	00010100	LSH	Nhân AC với 2; tức là dịch trái một vị trí bit
	00010101	RSH	Chia AC cho 2; tức là dịch phải một vị trí
Sửa đổi địa chỉ	00010010	STOR M(X,8:19)	Thay trường địa chỉ bên trái của M(X) bằng 12 bit ngoài cùng bên phải của AC
	00010011	STOR M(X,28:39)	Thay trường địa chỉ bên phải của M(X) bằng 12 bit ngoài cùng bên phải của AC

Bảng 2.1 trình bày các lệnh dưới dạng ký hiệu ký tự dễ đọc. Trên thực tế, mỗi lệnh phải tuân theo định dạng ở hình 2.2b. Phần opcode (8 bit đầu tiên) xác định lệnh nào trong số 21 lệnh sẽ được thực hiện. Phần địa chỉ (12 bit còn lại) xác định vị trí nào trong 1000 vị trí bộ nhớ liên quan đến việc thực hiện lệnh.

2.1.1.3. Máy tính thương mại

Những năm 1950 đã chứng kiến sự ra đời của ngành công nghiệp máy tính với hai công ty lớn là Sperry và IBM chiếm lĩnh thị trường.

Năm 1947, Eckert và Mauchly thành lập Công ty Máy tính Eckert-Mauchly để sản xuất máy tính với mục đích thương mại. Máy tính thành công đầu tiên của họ là UNIVAC I (Universal Automatic Computer). Công ty Máy tính Eckert-Mauchly trở thành một nhánh UNIVAC của tập đoàn Sperry-Rand, và tiếp tục sản xuất một loạt máy tính sau này.

UNIVAC I là máy tính thương mại thành công đầu tiên. Nó được dùng cho cả các ứng dụng khoa học và thương mại. Bài báo đầu tiên mô tả hệ thống này đã liệt kê ra các mẫu nhiệm vụ có thể thực hiện là tính toán đại số ma trận, vấn đề về thống kê, hóa đơn bảo hiểm cho công ty bảo hiểm nhân thọ, và các vấn đề hậu cần khác.

UNIVAC II, có dung lượng lưu trữ lớn hơn và hiệu năng cao hơn UNIVAC I, đã được ra mắt vào những năm cuối của thập niên 1950 và cho thấy một số xu hướng của ngành công nghiệp máy tính. Thứ nhất, những tiến bộ trong công nghệ cho phép các công ty tiếp tục chế tạo các máy tính lớn hơn, mạnh hơn. Thứ hai, mỗi công ty cố gắng làm cho các máy mới *tương thích ngược* với các máy cũ. Điều này có nghĩa là các chương trình viết cho các máy cũ vẫn có thể thực hiện được trên máy mới. Chiến lược này được công nhận với nỗ lực duy trì nền tảng khách hàng; tức là, khi một khách hàng quyết định mua một máy tính mới hơn, anh ta có xu hướng mua sản phẩm mới hơn của cùng một công ty để tránh mất thêm tiền đầu tư vào các chương trình.

Chi nhánh UNIVAC cũng bắt đầu phát triển dòng máy tính 1100, đây là nguồn thu nhập chính của hãng. Mẫu sản phẩm đầu tiên, UNIVAC 1103 chủ yếu dành cho các ứng dụng khoa học, liên quan đến tính toán phức tạp. Các mẫu khác tập trung vào ng dụng kinh doanh, bao gồm việc xử lý lượng lớn dữ liệu văn bản. Hiện nay, sự phân biệt chức năng này hầu như không còn tồn tại nữa.

IBM, khi ấy là một công ty sản xuất thiết bị xử lý thẻ đục lỗ, cho ra đời máy tính điện tử chương trình lưu trữ đầu tiên – 701 vào năm 1953. Máy 701 phục vụ chính cho ứng dụng tính toán khoa học. Năm 1955, IBM giới thiệu sản phẩm 702 với một số tính năng phần cứng phù hợp với các ứng dụng kinh doanh. Đó là những mẫu sản phẩm đầu tiên thuộc dòng máy tính 700/7000 giúp IBM vươn lên trở thành nhà sản xuất máy tính hàng đầu.

2.1.2. Thê hệ thứ hai: Transistor

Bước tiến lớn đầu tiên trong nền công nghiệp sản xuất máy tính điện tử gắn liền với việc sử dụng bóng bán dẫn transistor thay thế cho đèn ống chân không. Transistor nhỏ hơn, rẻ hơn, và tỏa nhiệt ít hơn đèn ống chân không nhưng lại có thể được sử dụng giống như đèn ống chân không để chế tạo ra máy tính. Nếu như ống chân không kích thước lớn, đòi

hỏi dây dẫn, tám kim loại, bóng thủy tinh và chân không, thì transistor là một thiết bị bán dẫn khá nhỏ, được làm từ silic.

Transistor được phát minh tại Bell Labs vào năm 1947 và tạo ra một cuộc cách mạng điện tử những năm 1950. Tuy nhiên, cho đến những năm cuối của thập niên 1950, các máy tính bán dẫn hoàn toàn mới được bán trên thị trường. IBM không phải là công ty đầu tiên đưa ra công nghệ mới. NCR và RCA là những công ty dẫn đầu với một số máy tính transistor nhỏ. IBM theo ngay sau với dòng 7000.

Việc sử dụng transistor định nghĩa *thế hệ thứ hai* của máy tính. Có thể nói máy tính được phân loại thành các thế hệ dựa trên công nghệ phần cứng cơ bản được sử dụng (Bảng 2.2). Nếu so sánh với thế hệ máy trước, mỗi thế hệ máy mới lại có hiệu suất xử lý tốt hơn, dung lượng bộ nhớ lớn hơn, và kích thước nhỏ hơn.

Bảng 2.2 Các thế hệ máy tính

Thế hệ	Thời gian	Công nghệ	Tốc độ điển hình (hành động trên giây)
1	1946–1957	Ông chân không	40,000
2	1958–1964	Transistor	200,000
3	1965–1971	Mạch tích hợp cỡ nhỏ và vừa	1,000,000
4	1972–1977	Mạch tích hợp cỡ lớn	10,000,000
5	1978–1991	Mạch tích hợp cỡ rất lớn	100,000,000
6	1991–	Mạch tích hợp cỡ cực kỳ lớn	1,000,000,000

Bên cạnh đó, cũng có những sự thay đổi khác nữa. Thế hệ thứ hai đã chứng kiến sự ra đời của khối số học và logic phức tạp hơn, sử dụng các ngôn ngữ lập trình bậc cao và sự xuất hiện của phần mềm hệ thống. Phần mềm hệ thống cung cấp khả năng tải chương trình, di chuyển dữ liệu tới các thiết bị ngoại vi, và các thư viện để thực hiện các phép tính phổ biến, giống với các hệ điều hành hiện đại như Windows và Linux.

Thế hệ thứ hai cũng đánh dấu sự xuất hiện của Tập đoàn Thiết bị Số (DEC). DEC được thành lập vào năm 1957 và cho ra mắt máy tính đầu tiên, PDP-1. Máy tính PDP-1 của DEC tạo nên hiện tượng máy tính mini và sau này trở nên cực kì nổi bật trong thế hệ thứ ba.

2.1.3. Thế hệ thứ ba: Mạch tích hợp

Một transistor được đóng gói riêng được gọi là một *linh kiện rời*. Trong suốt những năm 1950 và đầu những năm 1960, linh kiện điện tử chủ yếu được sản xuất thành các linh kiện rời – transistor, điện trở, tụ điện, v.v. Các linh kiện rời được sản xuất riêng rẽ, đóng gói trong vỏ riêng, và được hàn hoặc nối dây với nhau trong các bảng mạch điện. Các bảng mạch này sau đó được đặt vào trong máy tính, máy hiện sóng oscilloscope và các thiết bị

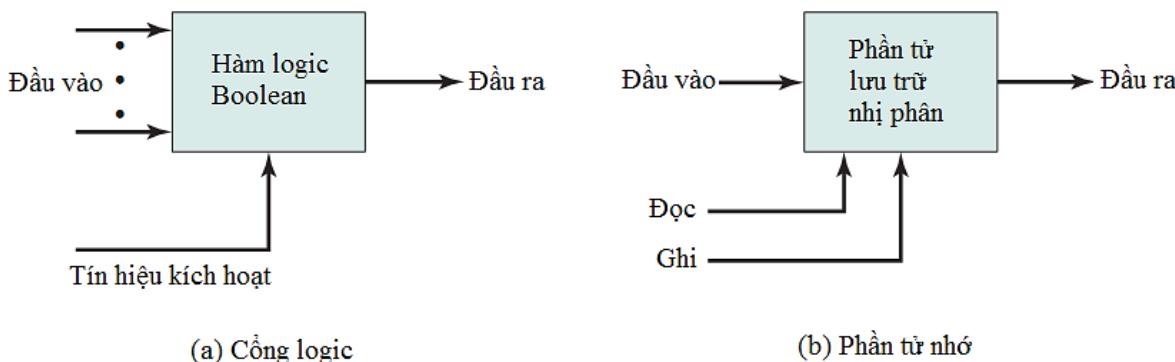
điện tử khác. Toàn bộ quá trình sản xuất, từ transistor tới bảng mạch điện, là rất đắt tiền và phức tạp.

Các máy tính thế hệ thứ hai ban đầu có khoảng 10.000 transistor. Con số này dần tăng lên hàng trăm ngàn, khiến cho việc sản xuất các máy tính mới hơn, mạnh hơn ngày càng trở nên khó khăn.

Năm 1958 đánh dấu thành tựu cách mạng điện tử và điểm bắt đầu của kỷ nguyên vi điện tử: phát minh ra mạch tích hợp. **Mạch tích hợp** là linh kiện xác định thế hệ máy tính thứ ba. Trong phần này, ta tìm hiểu sơ lược về công nghệ mạch tích hợp. Sau đó, ta xét hai thành viên quan trọng nhất của thế hệ thứ ba: IBM System/360 và DEC PDP-8.

2.1.3.1. Vi điện tử

Vi điện tử là chỉ "thiết bị điện tử nhỏ". Ngành điện tử số và công nghiệp máy tính luôn có một xu hướng liên tục và nhất quán, đó là kích thước mạch điện tử số ngày càng giảm. Trước khi xem xét tác động và lợi ích của xu hướng này, chúng ta cần tìm hiểu về bản chất của điện tử số.



Hình 2.5 Các linh kiện máy tính cơ bản

Như chúng ta biết, các chức năng cơ bản của máy tính số là lưu trữ, di chuyển, xử lý và điều khiển. Để thực hiện được các chức năng đó, chỉ cần hai loại linh kiện cơ bản (Hình 2.5) là: cổng logic và phản tử nhớ. Cổng logic là linh kiện thực hiện một hàm Boolean đơn giản, chẳng hạn như IF A AND B TRUE THEN C IS TRUE (cổng AND). Các linh kiện như vậy được gọi là cổng bởi vì chúng kiểm soát lưu lượng dữ liệu giống như cách mà cổng kênh kiểm soát lưu lượng nước. Phản tử nhớ là linh kiện có thể nhớ một bit dữ liệu; có nghĩa là bất kỳ lúc nào linh kiện đó cũng có thể ở một trong hai trạng thái ổn định. Bằng cách kết nối một số lượng lớn các linh kiện cơ bản này, chúng ta có thể xây dựng nên một chiếc máy tính. Chúng ta có thể liên hệ điều này với bốn chức năng cơ bản của máy tính như sau:

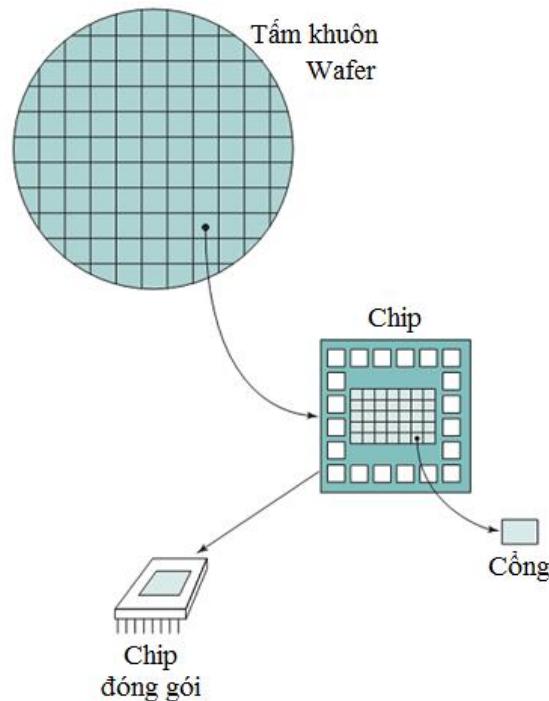
- **Lưu trữ dữ liệu:** Thực hiện bởi phản tử nhớ.
- **Xử lý dữ liệu:** Thực hiện bởi cổng logic.
- **Di chuyển dữ liệu:** Đường nối giữa các bộ phận của máy tính được sử dụng để di chuyển dữ liệu đi vào/ra bộ nhớ và từ bộ nhớ qua cổng logic tới bộ nhớ.
- **Điều khiển:** Đường nối giữa các bộ phận của máy tính có thể truyền tín hiệu điều khiển. Ví dụ, một cổng logic sẽ có một hoặc hai đầu vào dữ liệu và một đầu vào tín

hiệu điều khiển kích hoạt cổng. Khi tín hiệu điều khiển là ON, cổng thực hiện chức năng của nó trên đầu vào dữ liệu và tạo ra một đầu ra dữ liệu. Tương tự, phần tử nhớ sẽ nhớ bit thông tin ở đầu vào của nó khi tín hiệu điều khiển WRITE là ON và sẽ đặt bit nó đang nhớ lên đầu ra của nó khi tín hiệu điều khiển READ là ON.

Do đó, một máy tính bao gồm các cổng logic, phần tử nhớ và đường kết nối giữa các bộ phận. Cổng và phần tử nhớ lại được chế tạo từ những linh kiện điện tử số đơn giản.

Các linh kiện như điện trở, transistor và dây dẫn có thể cùng được chế tạo từ chất bán dẫn như silic. Do đó, hoàn toàn có thể chế tạo ra cả một mạch điện hoàn chỉnh trong một miếng silic nhỏ xíu thay vì lắp ráp các linh kiện rời được chế tạo từ các miếng silic riêng biệt vào cùng một mạch điện. Nhiều transistor có thể được sản xuất cùng lúc trên một tấm khuôn silic. Hơn nữa, các transistor này có thể được kết nối với nhau để hình thành các mạch điện.

Hình 2.6 mô tả các khái niệm chính trong mạch tích hợp. Một tấm khuôn silic mỏng (còn gọi là **wafer**) được chia thành một ma trận các ô vuông nhỏ (vài milimet vuông). Các mẫu mạch giống hệt nhau được chế tạo trong mỗi ô, sau đó tấm khuôn wafer được chia thành các **chip**. Mỗi chip bao gồm nhiều cổng logic và/hoặc phần tử nhớ cùng với một số điểm đầu vào và đầu ra. Chip này sau đó được đóng gói kín và được gắn thêm chân để có thể lắp vào các thiết bị ngoài chip. Một vài con chip đã đóng gói này có thể được kết nối với nhau trên một bảng mạch in để tạo ra mạch điện lớn hơn và phức tạp hơn.



Hình 2.6 Mối quan hệ giữa tấm khuôn Wafer, Chip và Cổng logic

Ban đầu, chỉ có thể sản xuất và đóng gói một vài cổng logic hoặc phần tử nhớ cùng nhau. Các mạch tích hợp ban đầu này được gọi là tích hợp cỡ nhỏ (SSI – small-scale integration). Sau đó, ngày càng nhiều linh kiện có thể được đóng gói trên cùng một chip. Sự tăng trưởng mật độ tích hợp được minh họa trong Hình 2.7; đây là một trong những xu

24 Chương 2. Sự phát triển của máy tính và hiệu năng

hướng công nghệ đáng chú ý nhất từng được ghi lại. Hình vẽ này phản ánh định luật Moore nổi tiếng, do Gordon Moore, người đồng sáng lập của Intel, đưa ra vào năm 1965. Ông Moore quan sát thấy số lượng transistor có thể đặt trên một con chip đã tăng gấp đôi sau mỗi năm và dự đoán chính xác tốc độ này sẽ duy trì trong tương lai gần. Đáng ngạc nhiên, tốc độ tăng trưởng này tiếp tục năm này qua năm khác, thập kỷ này qua thập kỷ khác. Tốc độ sau đó chậm lại còn tăng gấp đôi sau mỗi 18 tháng vào những năm 1970 và duy trì cho đến ngày nay.

Hệ quả của quy luật Moore được đưa ra như sau:

1. Giá của một con chip hầu như không thay đổi trong giai đoạn mật độ tăng trưởng nhanh chóng này. Điều này có nghĩa là giá của logic máy tính và mạch nhớ đã giảm rất mạnh.
2. Do các bộ phận logic và nhớ được đặt gần nhau hơn trên con chip mật độ cao hơn, chiều dài đường dẫn điện được rút ngắn, tăng tốc độ hoạt động.
3. Máy tính trở nên nhỏ gọn hơn và phù hợp để sử dụng ở nhiều môi trường khác nhau.
4. Giảm yêu cầu về điện năng tiêu thụ và bộ làm mát.
5. Các kết nối trên mạch tích hợp đáng tin cậy hơn nhiều so với các mối hàn. Mạch điện trên mỗi chip càng nhiều, kết nối giữa các con chip càng ít.



Hình 2.7 Mức tăng số lượng transistor trên mạch tích hợp

2.1.3.2. IBM System/360

Đến năm 1964, IBM đã chiếm lĩnh thị trường máy tính với dòng máy 7000. Trong năm đó, IBM đã cho ra mắt một dòng sản phẩm máy tính mới là System/360. Sự ra mắt sản phẩm mới này dường như là tin không vui đối với khách hàng hiện tại của IBM: bởi dòng sản phẩm 360 không tương thích với các máy IBM trước đó. Do đó, việc chuyển sang sử dụng máy 360 sẽ đem lại khó khăn cho những khách hàng hiện tại. Đây là bước đi táo bạo của IBM, tuy nhiên IBM muốn vượt qua một số khó khăn của kiến trúc 7000 và tạo ra một hệ thống có khả năng phát triển với công nghệ mạch tích hợp mới. Chiến lược táo bạo này đem lại thành quả cả về tài chính và kỹ thuật. 360 là sản phẩm thành công của thập kỷ và đã củng cố vị thế nhà cung cấp máy tính hàng đầu cho IBM, với hơn 70% thị

phần. Với một số sửa đổi và mở rộng, kiến trúc của 360 vẫn còn được duy trì cho đến ngày nay trong kiến trúc máy mainframe của IBM.

System/360 là họ máy tính được lập kế hoạch đầu tiên của ngành công nghiệp máy tính. Họ máy này bao trùm một dải rộng hiệu năng và chi phí. Bảng 2.3 liệt kê một số đặc điểm chính của các mẫu máy vào năm 1965 (mỗi thành viên trong họ máy tính được phân biệt bằng một con số). Các mẫu máy này tương thích với nhau, tức là một chương trình viết cho mẫu này có thể chạy được trên một mẫu khác trong cùng dòng máy, chỉ khác biệt về thời gian thực hiện.

Bảng 2.3 Các đặc điểm chính của họ System/360

Đặc điểm	Mẫu 30	Mẫu 40	Mẫu 50	Mẫu 65	Mẫu 75
Kích thước bộ nhớ tối đa (byte)	64K	256K	256K	512K	512K
Tốc độ dữ liệu từ bộ nhớ (Mbyte/s)	0.5	0.8	2.0	8.0	16.0
Chu kỳ xử lý (μ s)	1.0	0.625	0.5	0.25	0.2
Tốc độ tương đối	1	3.5	10	21	50
Số kênh dữ liệu tối đa	3	3	4	6	6
Tốc độ dữ liệu tối đa trên một kênh (Kbyte/s)	250	400	800	1250	1250

System/360 không chỉ quyết định tương lai của IBM mà còn có tác động sâu sắc đến toàn bộ ngành công nghiệp máy tính. Nhiều tính năng của nó đã trở thành tiêu chuẩn cho các sản phẩm máy tính lớn khác.

2.1.3.3. DEC PDP-8

Trong cùng năm mà IBM ra mắt System/360, Công ty thiết bị số (DEC – Digital Equipment Corporation) cũng giới thiệu máy tính PDP-8. Vào thời điểm mà mỗi chiếc máy tính đòi hỏi một phòng máy lạnh, PDP-8 (được gọi là máy tính cỡ nhỏ - minicomputer) đủ nhỏ để đặt vừa trên một chiếc bàn trong phòng thí nghiệm hoặc lắp được vào thiết bị khác. Dù PDP-8 không thể thực hiện được tất cả mọi chức năng như một chiếc máy mainframe, nhưng với mức giá 16.000 USD, nó đủ rẻ để mỗi kỹ thuật viên phòng thí nghiệm có thể sở hữu một chiếc. Trái lại, dòng máy mainframe System/360 được giới thiệu chỉ một vài tháng trước có giá lên tới hàng trăm nghìn đô la.

Giá rẻ và kích thước nhỏ của PDP-8 cho phép một nhà sản xuất khác mua PDP-8 và tích hợp nó vào một hệ thống tổng thể để bán lại. Những nhà sản xuất kiểu này được gọi là **nhà sản xuất thiết bị gốc (OEM)**, và thị trường OEM đã trở thành một phân khúc chính của thị trường máy tính.

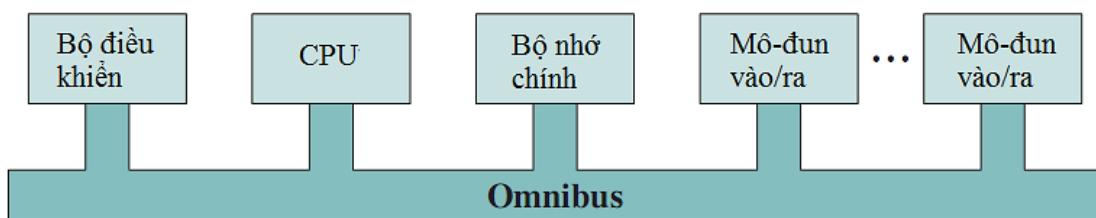
PDP-8 ngay lập tức trở thành hit và đem lại lợi nhuận lớn cho DEC. Chiếc máy này và các thành viên sau này thuộc họ máy PDP-8 (xem Bảng 2.4) đã được sản xuất với số lượng lớn không kém các máy tính IBM, với khoảng 50.000 máy được bán ra trong vòng vài năm tiếp theo. Theo DEC, PDP-8 đã "thiết lập khái niệm máy tính mini, dẫn tới ngành công nghiệp trị giá nhiều tỷ đô la". Nó cũng đã đưa DEC trở thành nhà cung cấp máy tính mini

số một, và cho tới khi PDP-8 hết thời, DEC là nhà sản xuất máy tính số hai, chỉ đứng sau IBM.

Trái ngược với kiến trúc chuyển mạch trung tâm mà IBM sử dụng trên các hệ thống 700/7000 và 360 của mình, các mẫu PDP-8 sau này đã sử dụng một kiến trúc phổ biến đối với các máy vi tính: cấu trúc bus. Điều này được minh họa trong Hình 2.8. Bus PDP-8, được gọi là Omnibus, bao gồm 96 đường dẫn tín hiệu riêng biệt, được sử dụng để truyền tín hiệu điều khiển, địa chỉ và dữ liệu. Bởi vì tất cả các thành phần hệ thống chia sẻ một nhóm đường dẫn tín hiệu chung, việc sử dụng đường dẫn chung có thể được kiểm soát bởi CPU. Kiến trúc này rất linh hoạt, cho phép các mô-đun được cắm vào bus để tạo ra các cấu hình khác nhau.

Bảng 2.4 Sự phát triển của PDP-8

Mẫu	Ra mắt	Giá của bộ xử lý + Bộ nhớ 4K từ 12 bit (\$1000)	Tốc độ dữ liệu từ Bộ nhớ (tùy/μs)	Thể tích (feet ³)	Cải tiến
PDP-8	1965	16.2	1.26	8.0	Sản xuất dây chuyền tự động
PDP-8/5	1966	8.79	0.08	3.2	Thực hiện lệnh nối tiếp
PDP-8/1	1968	11.6	1.34	8.0	Mạch tích hợp cỡ vừa
PDP-8/L	1968	7.0	1.26	2.0	Tủ chứa nhỏ hơn
PDP-8/E	1971	4.99	1.52	2.2	Omnibus
PDP-8/M	1972	3.69	1.52	1.8	Tủ chứa nhỏ bằng nửa 8/E
PDP-8/A	1975	2.6	1.34	1.2	Bộ nhớ bán dẫn; bộ xử lý dấu chấm động



Hình 2.8 Cấu trúc bus PDP-8

2.1.4. Các thế hệ tiếp theo

Sau thế hệ thứ ba, có ít sự đồng thuận chung hơn về việc định nghĩa các thế hệ máy tính. Bảng 2.2 cho thấy đã có một số thế hệ tiếp theo, dựa trên những tiến bộ trong công nghệ mạch tích hợp. Với sự ra đời của mạch tích hợp cỡ lớn (LSI – large-scale integration), hơn 1000 linh kiện có thể được đặt trên một con chip tích hợp. Mạch tích hợp cỡ rất lớn (VLSI – very large-scale integration) có hơn 10.000 linh kiện trên mỗi con chip,

trong khi chip tích hợp cỡ cực kỳ lớn (ULSI – ultra-large-scale integration) hiện tại có thể chứa hơn một tỷ linh kiện.

Với nhịp độ phát triển công nghệ nhanh, tốc độ giới thiệu sản phẩm mới nhanh, tầm quan trọng của phần mềm và truyền thông cũng như phần cứng, việc phân loại máy tính theo thế hệ trở nên không rõ ràng và ít có ý nghĩa. Có thể nói rằng ứng dụng thương mại của các sản phẩm mới đã đem đến sự thay đổi chính vào đầu những năm 1970. Trong phần này, chúng ta đề cập đến hai trong số những kết quả quan trọng nhất.

2.1.4.1. Bộ nhớ bán dẫn

Ứng dụng đầu tiên của công nghệ mạch tích hợp vào máy tính là xây dựng bộ xử lý (khối điều khiển và khối xử lý số học và logic) từ chip tích hợp. Tuy nhiên, công nghệ này cũng có thể được sử dụng để xây dựng bộ nhớ.

Trong những năm 1950 và 1960, hầu hết bộ nhớ máy tính được chế tạo từ các vòng kim loại nhỏ, mỗi vòng có đường kính khoảng 2 mm. Những vòng kim loại này được xâu vào lối dây đan treo trên tấm màng nhôm bên trong máy tính. Từ hóa theo một chiều, một vòng kim loại (được gọi là lõi – core) biểu diễn bit 1; từ hóa theo chiều khác, nó biểu diễn bit 0. Bộ nhớ lõi từ khá nhanh; chỉ mất một phần triệu giây để đọc một bit lưu trong bộ nhớ. Nhưng nó đắt tiền, cồng kềnh, và mỗi lần đọc dữ liệu trong lõi sẽ xóa luôn dữ liệu được lưu trữ. Vì vậy, cần phải có thêm mạch điện để khôi phục lại dữ liệu ngay khi nó được truy xuất.

Sau đó, vào năm 1970, Fairchild đã sản xuất bộ nhớ bán dẫn dung lượng tương đối lớn đầu tiên. Chip này, với kích thước gần bằng một lõi đơn, có thể chứa 256 bit bộ nhớ. Nó không làm mất dữ liệu và nhanh hơn lõi. Chỉ mất 1/70 tỷ giây để đọc một bit. Tuy nhiên, chi phí cho mỗi bit cao hơn so với lõi.

Đến năm 1974, chi phí cho mỗi bit bộ nhớ bán dẫn giảm xuống thấp hơn chi phí cho mỗi bit của bộ nhớ lõi từ. Sau đó, giá bộ nhớ liên tục giảm mạnh khi mật độ bộ nhớ vật lý tăng nhanh. Điều này dẫn đến việc các máy nhỏ hơn, nhanh hơn với bộ nhớ lớn hơn và đắt hơn. Sự phát triển của công nghệ bộ nhớ, cùng với sự phát triển của công nghệ bộ xử lý đã thay đổi bản chất của máy tính trong vòng chưa đầy một thập kỷ.

Kể từ năm 1970, bộ nhớ bán dẫn đã trải qua 13 thế hệ: 1K, 4K, 16K, 64K, 256K, 1M, 4M, 16M, 64M, 256M, 1G, 4G và 16GB trên một con chip ($1K = 2^{10}$, $1M = 2^{20}$, $1G = 2^{30}$). Mỗi thế hệ lại cung cấp mật độ lưu trữ gấp bốn lần thế hệ trước, cùng với việc giảm chi phí cho mỗi bit và giảm thời gian truy cập.

2.1.4.2. Vi xử lý

Giống như việc mật độ linh kiện trên chip nhớ ngày càng tăng, mật độ linh kiện trên chip xử lý cũng tăng liên tục. Dần dần, ngày càng có nhiều linh kiện được đặt trên một con chip, do đó để xây dựng một bộ xử lý máy tính thì cần ít chip hơn.

Năm 1971 đã chứng kiến một bước tiến đột phá, khi Intel phát triển sản phẩm 4004. 4004 là chip đầu tiên chứa tất cả các thành phần của CPU trên một con chip đơn: Vi xử lý ra đời.

28 Chương 2. Sự phát triển của máy tính và hiệu năng

Vi xử lý 4004 có thể cộng hai số 4 bit và có thể thực hiện phép nhân chỉ bằng việc lặp đi lặp lại phép cộng. Theo tiêu chuẩn ngày nay, 4004 quả là thô sơ, nhưng nó đánh dấu điểm khởi đầu trong quá trình phát triển liên tục về khả năng và hiệu suất của vi xử lý.

Sự phát triển này có thể nhận thấy dễ dàng nhất qua số lượng bit mà bộ xử lý xử lý được cùng một lúc. Tham số đánh giá tốt nhất là chiều rộng của bus dữ liệu: số bit dữ liệu có thể đưa vào hoặc gửi ra khỏi bộ xử lý tại một thời điểm. Một tham số khác là số bit trong thanh ghi tích luỹ (accumulator) hoặc trong tập hợp thanh ghi đa năng. Thông thường, các tham số này trùng nhau, nhưng cũng có trường hợp khác biệt. Ví dụ, một số vi xử lý được phát triển để xử lý các số 16 bit trong thanh ghi nhưng mỗi lần chỉ có thể đọc và ghi 8 bit.

Bước tiến lớn tiếp theo trong quá trình phát triển của vi xử lý là sự ra đời của Intel 8008 vào năm 1972. Đây là vi xử lý 8 bit đầu tiên và độ phức tạp gần gấp đôi so với 4004.

Tuy nhiên, sự kiện lớn tiếp theo có sức ảnh hưởng lớn hơn cả: sự ra đời của Intel 8080 vào năm 1974. Đây là vi xử lý đa năng đầu tiên. Trong khi 4004 và 8008 được thiết kế cho các ứng dụng cụ thể, 8080 được thiết kế để trở thành CPU của máy vi tính đa năng. Giống như 8008, 8080 là vi xử lý 8 bit. Tuy nhiên, 8080 nhanh hơn, có tập lệnh phong phú hơn, và có khả năng định địa chỉ lớn.

Bảng 2.5 Quá trình phát triển của vi xử lý Intel

(a) Bộ xử lý những năm 1970

	4004	8008	8080	8086	8088
Ra mắt	1971	1972	1974	1978	1979
Tốc độ đồng hồ	108 kHz	108 kHz	2 MHz	5 MHz, 8 MHz, 10 MHz	5 MHz, 8 MHz
Độ rộng bus	4 bit	8 bit	8 bit	16 bit	8 bit
Số transistor	2300	3500	6000	29,000	29,000
Kích thước (μm)	10		6	3	6
Bộ nhớ	640 byte	16 kB	64 kB	1 MB	1 MB

(b) Bộ xử lý những năm 1980

	80286	386TM DX	386TM SX	486TM DX CPU
Ra mắt	1982	1985	1988	1989
Tốc độ đồng hồ	6 – 12.5 MHz	16 – 33 MHz	16 – 33 MHz	25 – 50 MHz
Độ rộng bus	16 bit	32 bit	16 bit	32 bit
Số transistor	134,000	275,000	275,000	1.2 tỷ
Kích thước (μm)	1.5	1	1	0.8 – 1

Bộ nhớ	16 MB	4 GB	16 MB	4 GB
Bộ nhớ ảo	1 GB	64 TB	64 TB	64 TB
Cache	-	-	-	8 kB

(c) Bộ xử lý những năm 1990

	486TM SX	Pentium	Pentium Pro	Pentium II
Ra mắt	1991	1993	1995	1997
Tốc độ đồng hồ	16 – 33 MHz	60 – 166 MHz	150 – 200 MHz	200 – 300 MHz
Độ rộng bus	32 bit	32 bit	64 bit	64 bit
Số transistor	1.185 tỷ	3.1 tỷ	5.5 tỷ	7.5 tỷ
Kích thước (μm)	1	0.8	0.6	0.35
Bộ nhớ	4 GB	4 GB	64 GB	64 GB
Bộ nhớ ảo	64 TB	64 TB	64 TB	64 TB
Cache	8 kB	8 kB	512 kB L1 và 1 MB L2	kB L2

(d) Bộ xử lý hiện tại

	Pentium III	Pentium 4	Core 2 Duo	Core i7 EE 990
Ra mắt	1999	2000	2006	2011
Tốc độ đồng hồ	450 – 660 MHz	1.3 – 1.8 GHz	1.06 – 1.2 GHz	3.5 GHz
Độ rộng bus	64 bit	64 bit	64 bit	64 bit
Số transistor	9.5 tỷ	42 tỷ	167 tỷ	1170 tỷ
Kích thước (μm)	250	180	65	32
Bộ nhớ	64 GB	64 GB	64 GB	64 GB
Bộ nhớ ảo	64 TB	64 TB	64 TB	64 TB
Cache	512 kB L2	256 kB L2	2 MB L2	1.5 MB L2/ 12MB L3

Cũng trong thời gian đó, vi xử lý 16 bit đã bắt đầu được phát triển. Tuy nhiên, cho đến những năm cuối thập kỷ 1970, vi xử lý 16 bit đa năng mới xuất hiện. Một trong số đó là 8086. Bước tiếp theo diễn ra vào năm 1981, khi cả Bell Labs và Hewlett-Packard cùng phát triển bộ vi xử lý 32 bit. Intel đã giới thiệu vi xử lý 32 bit của mình, 80386, vào năm 1985 (Bảng 2.5).

2.2. CÁC ĐẶC ĐIỂM THIẾT KẾ MÁY TÍNH

Qua từng năm, giá thành của các hệ thống máy tính giảm dần, trong khi hiệu suất và dung lượng tiếp tục tăng lên đáng kể. Máy tính xách tay ngày nay đạt được sức mạnh tính toán như máy tính mainframe IBM từ 10 đến 15 năm trước.

Các ứng dụng máy tính được phát triển ngày càng phức tạp và tinh vi. Hệ thống máy trạm hỗ trợ các ứng dụng khoa học và kỹ thuật, các hệ thống mô phỏng, và có khả năng hỗ trợ các ứng dụng hình ảnh và video. Ngoài ra, các doanh nghiệp sử dụng các máy chủ ngày càng mạnh mẽ để xử lý giao dịch, xử lý cơ sở dữ liệu và hỗ trợ các mạng client/server lớn thay thế cho các trung tâm máy tính mainframe trước đây.

Điều hấp dẫn từ quan điểm của tổ chức và kiến trúc máy tính là, một mặt, các khái niệm cơ bản của máy tính hiện nay hầu như giống với các máy tính IAS từ hơn 50 năm trước, trong khi mặt khác, các kỹ thuật ngày càng trở nên tinh vi. Trong mục này, chúng ta sẽ tìm hiểu một số yếu tố thúc đẩy để thiết kế máy tính cho hiệu suất cao.

2.2.1. Tốc độ vi xử lý

Việc bổ sung các mạch mới với khoảng cách giữa các mạch được rút ngắn đã nâng cao hiệu suất vi xử lý lên 4 hoặc 5 lần sau mỗi ba năm kể từ khi Intel cho ra đời dòng chip x86 vào năm 1978. Nhưng tốc độ của vi xử lý sẽ không đạt được hết tiềm năng của nó trừ khi nó được cung cấp một luồng công việc liên tục để thực hiện dưới dạng lệnh máy. Bất cứ điều gì cản trở luồng hoạt động ấy đều làm suy giảm sức mạnh của bộ xử lý. Vì lẽ đó, trong khi nhà sản xuất chip tìm cách chế tạo ra con chip có mật độ lớn và lớn hơn nữa, các nhà thiết kế vi xử lý phải nỗ lực đưa ra các kỹ thuật tinh vi hơn để có thể khai thác hết tiềm năng của nó. Một số kỹ thuật được áp dụng trong các bộ xử lý hiện đại là:

- **Pipeline – Kỹ thuật đường ống:** Với pipeline, một bộ xử lý có thể xử lý cùng lúc nhiều lệnh. Bộ xử lý ghép nối các hành động bằng cách di chuyển dữ liệu hoặc lệnh vào một “đường ống” với tất cả các giai đoạn trong đường ống được xử lý đồng thời. Ví dụ, trong khi một lệnh đang được thi hành, máy tính giải mã lệnh tiếp theo.
- **Dự đoán nhánh:** Bộ xử lý tra mã lệnh tìm được từ bộ nhớ và dự đoán nhánh nào hoặc nhóm lệnh nào được xử lý tiếp theo. Nếu hầu hết mọi lần bộ xử lý đều đoán đúng, nó có thể truy xuất trước các lệnh và đệm các lệnh đó lại để giữ cho bộ xử lý luôn bận rộn. Hay nói cách khác, dự báo nhánh tăng lượng công việc có sẵn cho bộ xử lý thực thi.
- **Phân tích luồng dữ liệu:** Bộ xử lý phân tích xem lệnh nào phụ thuộc vào kết quả của lệnh khác, hoặc lệnh nào phụ thuộc vào dữ liệu nào, để đưa ra lịch trình xử lý lệnh tối ưu nhất. Thực tế, các lệnh được lên kế hoạch thực hiện khi đã sẵn sàng, không phụ thuộc vào trình tự chương trình ban đầu. Điều này giúp tránh sự chậm trễ không cần thiết.
- **Thi hành lệnh theo dự đoán:** Sử dụng dự đoán nhánh và phân tích luồng dữ liệu, một số bộ xử lý thi hành lệnh trước khi nó xuất hiện, lưu kết quả trong một vùng tạm thời. Điều này giữ cho bộ máy luôn hoạt động bận rộn.

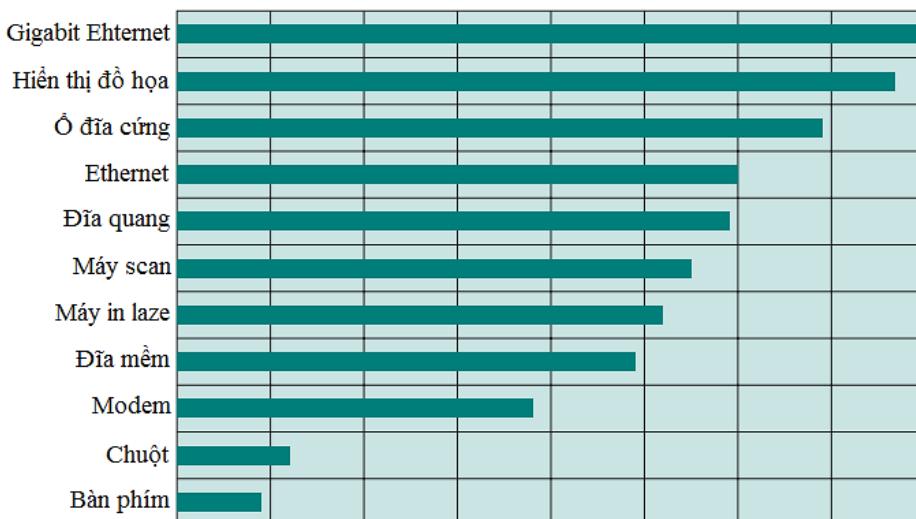
2.2.2. Cân bằng Hiệu suất

Trong khi sức mạnh của bộ xử lý tăng với tốc độ chóng mặt, các thành phần quan trọng khác trong máy tính vẫn chưa phát triển theo kịp. Do đó cần phải có giải pháp cân bằng hiệu suất: một sự điều chỉnh trong tổ chức và kiến trúc để bù đắp cho sự chênh lệch khả năng của các thành phần khác nhau trong máy tính.

Sự chênh lệch cần được chú trọng hơn cả là chênh lệch ở giao diện **giữa bộ vi xử lý và bộ nhớ chính**. Trong khi tốc độ bộ xử lý tăng lên rất nhanh, tốc độ truyền dữ liệu giữa bộ nhớ chính và bộ xử lý lại khá chậm. Giao diện giữa bộ xử lý và bộ nhớ chính là đường dẫn quan trọng nhất trong toàn bộ máy tính vì nó đảm nhận việc vận chuyển dòng lệnh và dữ liệu liên tục giữa các chip nhớ và chip vi xử lý. Nếu bộ nhớ hoặc đường dẫn không theo kịp tốc độ của bộ xử lý, bộ xử lý sẽ phải dừng lại trong trạng thái đợi và lãng phí thời gian xử lý đáng giá.

Có nhiều giải pháp điều chỉnh kiến trúc khắc phục được vấn đề này, tất cả đều được phản ánh trong các thiết kế máy tính hiện đại. Có thể kể tới một số ví dụ sau đây:

- Tăng số lượng bit được lấy ra tại 1 thời điểm bằng cách làm cho DRAMs “rộng hơn” thay vì “sâu hơn” và bằng cách sử dụng đường bus dữ liệu rộng.
- Thay đổi giao diện DRAM hiệu quả hơn bằng cách thêm vào 1 bộ nhớ cache hoặc 1 cơ chế đệm khác trên chip DRAM.
- Giảm tần suất truy cập bộ nhớ bằng cách kết hợp sử dụng cache giữa bộ xử lý và bộ nhớ chính.
- Tăng băng thông kết nối giữa bộ xử lý và bộ nhớ bằng cách sử dụng các bus tốc độ cao và phân cáp bus để đệm và tổ chức dòng dữ liệu.



Hình 2.9 Tốc độ dữ liệu điển hình của thiết bị vào/ra

Một mảng khác được tập trung thiết kế là xử lý các thiết bị vào ra. Khi máy tính chạy nhanh hơn và có dung lượng lớn hơn, các ứng dụng phức tạp hơn được phát triển để hỗ trợ cho việc sử dụng các thiết bị ngoại vi với các mệnh lệnh dành riêng cho hoạt động vào ra. Hình 2.9 đưa ra một số thiết bị ngoại vi điển hình được sử dụng trên máy tính cá nhân và máy trạm. Các thiết bị này tạo ra nhu cầu dữ liệu không lồ. Mặc dù các bộ xử lý hiện tại có

thể xử lý tốt dữ liệu được đưa tới từ các thiết bị này, việc tiếp nhận dữ liệu di chuyển giữa bộ xử lý và thiết bị ngoại vi vẫn còn vấn đề vướng mắc. Giải pháp cho vấn đề này bao gồm phương pháp lưu trữ đệm (cache) cùng với việc sử dụng bus kết nối tốc độ cao và bus có cấu trúc phức tạp hơn. Ngoài ra, việc sử dụng cấu hình nhiều bộ xử lý có thể hỗ trợ đáp ứng lượng lớn yêu cầu vào ra.

2.2.3. Cải tiến kiến trúc và tổ chức Chip

Khi các nhà thiết kế vật lộn với việc cân bằng hiệu suất giữa bộ xử lý với bộ nhớ chính và các thành phần máy tính khác, vẫn tồn tại nhu cầu tăng tốc độ xử lý. Có ba cách tăng tốc độ xử lý:

- Tăng tốc độ phần cứng của bộ xử lý: Việc tăng tốc độ phần cứng về cơ bản là do việc thu hẹp kích thước cổng logic trên chip xử lý, sao cho nhiều cổng được đóng gói cùng nhau hơn, gần sát nhau hơn và do tăng tốc độ đồng hồ. Khi các cổng gần nhau hơn, thời gian truyền tín hiệu giảm, cho phép tăng tốc bộ xử lý. Tăng tốc độ đồng hồ nghĩa là các hành động riêng lẻ được thi hành nhanh hơn.
- Tăng kích thước và tốc độ cache đặt giữa bộ xử lý và bộ nhớ chính. Cụ thể là, dành riêng một phần chip vi xử lý cho cache, thời gian truy cập cache sẽ giảm đáng kể.
- Thay đổi cấu trúc và tổ chức bộ xử lý để làm tăng tốc độ thực hiện lệnh. Điều này liên quan đến việc xử lý song song.

Thông thường, yếu tố chi phối việc tăng hiệu suất chính là việc tăng tốc độ đồng hồ và mật độ logic. Tuy nhiên, khi tốc độ đồng hồ và mật độ logic tăng, xuất hiện một số trở ngại về:

- Năng lượng: Khi mật độ cổng logic và tốc độ đồng hồ trên chip tăng. Mật độ năng lượng (W/cm^2) cũng tăng. Khó khăn trong việc tản nhiệt sinh ra trên chip mật độ lớn, tốc độ cao trở thành một vấn đề thiết kế nghiêm trọng.
- Trễ RC (R: điện trở – C: điện dung): Tốc độ dòng electron chạy giữa các transistor trên chip bị giới hạn bởi điện dung và điện trở của đường dây dẫn kim loại kết nối chúng; đặc biệt, trễ tăng lên khi tích RC tăng. Khi kích thước các linh kiện trên chip giảm, các dây nối mảnh hơn, điện trở tăng. Ngoài ra, dây đặt gần nhau hơn, điện dung tăng.
- Trễ bộ nhớ: Tốc độ bộ nhớ thường chậm hơn tốc độ bộ xử lý.

2.3. ĐA NHÂN, MIC VÀ GPU

Một phương pháp mới có thể giúp cải thiện hiệu suất: sử dụng nhiều bộ xử lý trên cùng một chip với một cache lớn dùng chung. Việc sử dụng nhiều bộ xử lý trên cùng một chip, còn được gọi là **đa nhân**, đem lại khả năng tăng hiệu suất cho máy mà không cần phải tăng tốc độ đồng hồ. Nhiều nghiên cứu chỉ ra rằng, trong một bộ xử lý, mức tăng hiệu suất tỷ lệ với căn bậc hai mức tăng độ phức tạp. Nhưng nếu nhiều bộ xử lý được sử dụng hiệu quả, thì việc tăng gấp đôi số lượng bộ xử lý sẽ làm tăng gấp đôi hiệu suất. Do đó, sử dụng hai bộ xử lý đơn giản trên một chip sẽ tốt hơn là một bộ xử lý phức tạp.

Khi mật độ logic trên chip tiếp tục tăng, xu hướng sử dụng nhiều nhân và nhiều cache hơn trên một chip vẫn tiếp tục được duy trì. Các chip 2 nhân ra đời sau đó là các chip 4, 8 và rồi 16 nhân. Khi cache lớn hơn, dần dần xuất hiện hai rồi ba cấp độ cache trên một

chip, trong đó cache cấp một dành riêng cho một bộ xử lý còn cache cấp hai và ba được chia sẻ chung cho tất cả các bộ xử lý.

Các nhà sản xuất chip hiện đang trong quá trình tạo ra một bước nhảy vọt về số lượng nhân trên mỗi chip (hơn 50 nhân/chip). Bước nhảy vọt trong hiệu suất cũng như những thách thức trong phát triển phần mềm để khai thác tối đa khả năng xử lý của nhiều nhân như vậy đã dẫn tới việc cho ra đời một khái niệm mới: **đa nhân tích hợp (MIC)**.

Giải pháp đa nhân và MIC gắn với một tập hợp các bộ xử lý đa năng giống nhau trên một con chip. Cùng lúc đó, các nhà sản xuất chip cũng theo đuổi một lựa chọn thiết kế khác: một con chip với nhiều bộ xử lý đa năng cùng các **bộ xử lý đồ họa (GPU – graphics processing unit)** và các nhân chuyên dụng để xử lý video và các tác vụ khác. Nói chung, GPU là một nhân được thiết kế để thực hiện các hành động song song dữ liệu đồ họa. Thường được tìm thấy trên một card đồ họa plug-in (card màn hình), GPU được sử dụng để mã hóa và dựng đồ họa 2D và 3D cũng như xử lý video.

2.4. KIẾN TRÚC INTEL X86

Trong mục này, chúng ta tìm hiểu sơ lược về hai họ vi xử lý: kiến trúc Intel x86 và ARM. Họ vi xử lý x86 hiện nay là kết quả sau nhiều thập kỷ nghiên cứu máy tính tập lệnh phức tạp (CISC – Complex instruction set computer). X86 kết hợp các nguyên tắc thiết kế phức tạp chỉ có ở các siêu máy tính hoặc máy mainframe và được xem như là một ví dụ xuất sắc của thiết kế CISC.

Xét về thị phần, Intel đã chiếm vị trí số một trong các nhà sản xuất chip xử lý cho hệ thống không nhúng trong nhiều thập kỷ. Sự phát triển của vi xử lý Intel gắn liền với các cột mốc trong sự phát triển của công nghệ máy tính nói chung. Bảng 2.5 đã thể hiện sự phát triển đó.

Một số điểm nhấn trong tiến trình phát triển dòng sản phẩm của Intel được ghi nhận như sau:

- **8080:** Vi xử lý đa năng đầu tiên trên thế giới. Đây là máy 8 bit với một đường 8 bit đưa dữ liệu tới bộ nhớ. 8080 được sử dụng trong máy tính cá nhân đầu tiên Altair.
- **8086:** Máy 16 bit mạnh hơn rất nhiều. Ngoài đường dữ liệu rộng hơn, thanh ghi lớn hơn, 8086 sử dụng một cache lệnh, hay hàng đợi, cho phép truy xuất trước một vài lệnh trước khi chúng được thi hành. Một biến thể của bộ xử lý này là 8088, được sử dụng trong máy tính cá nhân đầu tiên của IBM, bảo vệ chuỗi thành công của Intel. 8086 là hiện diện đầu tiên của kiến trúc x86.
- **80286:** Phiên bản mở rộng của 8086 ho phép định địa chỉ bộ nhớ 16 MByte thay vì chỉ 1MByte ở phiên bản trước.
- **80386:** Máy 32 bit đầu tiên của Intel. Với kiến trúc 32 bit, 80386 đã vươn tới độ phức tạp và sức mạnh của máy tính mini và máy mainframe được giới thiệu chỉ vài năm trước đó. Đây là bộ xử lý Intel đầu tiên hỗ trợ đa nhiệm, tức là nó có thể chạy nhiều chương trình cùng một lúc.
- **80486:** 80486 đã giới thiệu việc sử dụng công nghệ cache tinh vi, mạnh mẽ hơn và công nghệ pipelining. 80486 cũng có một bộ đồng xử lý toán học tích hợp sẵn, giảm gánh nặng xử lý các phép tính toán học phức tạp cho CPU.

- **Pentium:** Sử dụng kỹ thuật siêu vô hướng (superscalar) cho phép nhiều lệnh được xử lý đồng thời.
- **Pentium Pro:** Tiếp tục tổ chức siêu vô hướng, với việc sử dụng tích cực các kỹ thuật đổi tên thanh ghi, dự đoán rẽ nhánh, phân tích luồng dữ liệu và thi lệnh theo suy đoán.
- **Pentium II:** Kết hợp với kỹ thuật MMX của Intel, thiết kế riêng để xử lý dữ liệu video, âm thanh và đồ họa.
- **Pentium III:** Kết hợp thêm các lệnh dấu chấm động để hỗ trợ phần mềm đồ họa 3D.
- **Pentium 4:** Có thêm các lệnh dấu chấm động và những cải tiến khác cho đa phương tiện.
- **Core:** Vi xử lý Intel x86 đầu tiên có hai nhân – tức là triển khai hai bộ xử lý trên một chip đơn.
- **Core 2:** Mở rộng kiến trúc lên 64 bit. Core 2 Quad có 4 bộ xử lý trên một chip. Gần đây, có thể có tới 10 bộ xử lý trên một chip.

Hơn 30 năm sau khi được giới thiệu vào năm 1978, kiến trúc x86 tiếp tục thống lĩnh thị trường vi xử lý cho các hệ thống không nhúng. Mặc dù tổ chức và công nghệ của các máy x86 đã thay đổi đáng kể trong nhiều thập kỷ, kiến trúc tập lệnh được phát triển để vẫn tương thích ngược với các phiên bản trước đó. Do đó, bất kỳ chương trình nào được viết cho phiên bản cũ hơn của kiến trúc x86 đều có thể chạy được trên các phiên bản mới hơn. Tất cả sự thay đổi trong kiến trúc tập lệnh đều là thêm lệnh mới vào tập lệnh, không có bớt đi. Tốc độ thay đổi là bổ sung thêm khoảng một lệnh mỗi tháng vào kiến trúc trong suốt 30 năm, do đó hiện có tổng cộng hơn 500 lệnh trong tập lệnh.

2.5. HỆ THỐNG NHÚNG VÀ KIẾN TRÚC ARM

Một phương pháp khác để thiết kế vi xử lý là máy tính tập lệnh rút gọn (RISC – reduced instruction set computer). Kiến trúc ARM được sử dụng trong các hệ thống nhúng và là một trong những hệ thống RISC mạnh mẽ và được thiết kế tốt nhất trên thị trường.

2.5.1. Hệ thống nhúng

Thuật ngữ *hệ thống nhúng* đề cập đến việc sử dụng điện tử và phần mềm bên trong một sản phẩm, trái ngược với máy tính đa năng như máy tính xách tay hoặc máy tính để bàn. Hệ thống nhúng được định nghĩa như sau:

“Hệ thống nhúng. Sự kết hợp của phần cứng và phần mềm máy tính, có thể cùng với các thành phần khác, được thiết kế để thực hiện một chức năng cụ thể. Trong nhiều trường hợp, hệ thống nhúng là một phần của một hệ thống hoặc sản phẩm lớn hơn, ví dụ như một hệ thống chống bó cứng phanh trong ô tô.”

Hệ thống nhúng đa dạng hơn so với các hệ thống máy tính đa năng, bao gồm một loạt các ứng dụng (Bảng 2.6). Các hệ thống này có nhiều yêu cầu và ràng buộc khác nhau, chẳng hạn như sau:

- Các hệ thống từ nhỏ đến lớn, kéo theo nhiều ràng buộc về giá thành và nhiều yêu cầu để tối ưu hóa và tái sử dụng khác nhau
- Độ bền ngắn hoặc dài

- Kết hợp các yêu cầu chất lượng khác nhau như độ an toàn, tin cậy, thời gian thực và độ linh hoạt.
- Điều kiện môi trường khác nhau về bức xạ, độ rung và độ ẩm.
- Đặc tính ứng dụng khác nhau dẫn đến tải tĩnh hay động, tốc độ nhanh hay chậm, tác vụ về tính toán hay giao diện, hoặc kết hợp cả hai.
- Các mô hình tính toán khác nhau từ hệ thống sự kiện rời rạc đến hệ thống lai.

Bảng 2.6 Ví dụ về các hệ thống nhúng và thị trường của chúng

Thị trường	Thiết bị nhúng
Công nghiệp ô tô	Hệ thống đánh lửa Điều khiển động cơ Hệ thống phanh
Điện tử tiêu dùng	Tivi kỹ thuật số và tương tự Set-top box (DVD, VCR, hộp cáp) Thiết bị số hỗ trợ cá nhân (PDA) Thiết bị nhà bếp (tủ lạnh, lò nướng, lò vi sóng) Ôtô Đồ chơi/trò chơi Điện thoại/điện thoại di động/máy nhắn tin Máy ảnh Hệ thống định vị toàn cầu
Điều khiển công nghiệp	Robot và hệ thống điều khiển cho sản xuất Cảm biến
Y học	Bơm tiêm truyền Máy chạy thận Thiết bị bộ phận giả Màn hình theo dõi nhịp tim
Tự động hóa văn phòng	Máy fax Máy photocopy Máy in Màn hình Máy scan

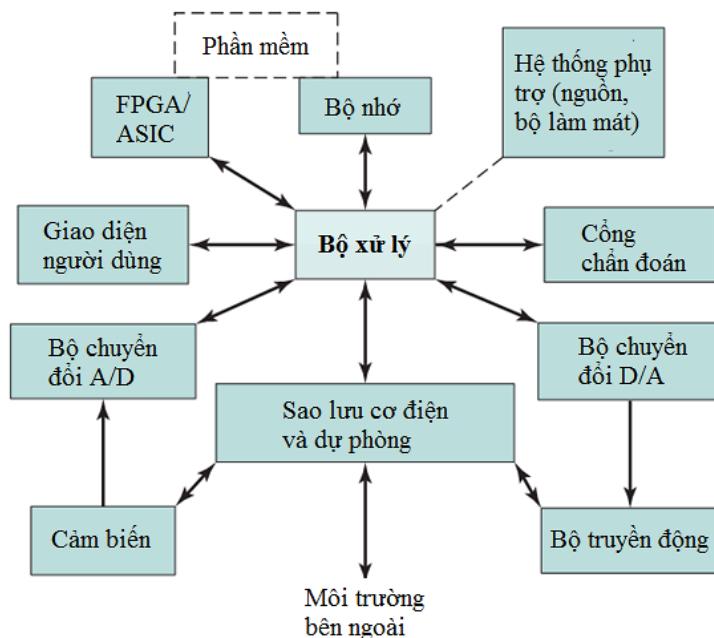
Hình 2.10 mô tả tổng quát một tổ chức hệ thống nhúng. Ngoài bộ xử lý và bộ nhớ, có một số điểm khác với máy tính để bàn hoặc máy tính xách tay thông thường:

- Có nhiều kiểu giao diện cho phép hệ thống đo đạc, xử lý và tương tác với môi trường bên ngoài.
- Giao diện với con người có thể đơn giản như ánh sáng nhấp nháy hoặc phức tạp như thị giác robot thời gian thực.
- **Công chẩn đoán có thể được sử dụng để chẩn đoán hệ thống đang được kiểm soát không chỉ để chẩn đoán máy tính.**
- Mảng công có thể lập trình (FPGA), mạch tích hợp ứng dụng riêng (ASIC), hoặc thậm chí phần cứng không phải kỹ thuật số có thể được sử dụng để tăng hiệu suất hoặc độ an toàn.
- Phần mềm thường có chức năng cố định và cụ thể cho từng ứng dụng.

2.5.2. Sự phát triển của ARM

ARM là một họ vi xử lý và vi điều khiển RISC được thiết kế bởi Công ty ARM, Anh. Chip ARM là bộ xử lý tốc độ cao nổi tiếng với kích thước nhỏ và đòi hỏi ít năng lượng. Chúng được sử dụng rộng rãi trong PDA và các thiết bị cầm tay khác, bao gồm cả máy chơi game, điện thoại và nhiều sản phẩm tiêu dùng khác. Chip ARM là bộ vi xử lý trong iPod của Apple và các thiết bị iPhone. ARM là kiến trúc bộ xử lý nhúng phổ biến nhất và thực sự là kiến trúc bộ xử lý được sử dụng rộng rãi nhất trên thế giới.

Công nghệ ARM bắt nguồn từ công ty Acorn Computers của Anh. Vào đầu những năm 1980, Acorn đã được Tập đoàn truyền thông Anh (BBC) ký hợp đồng để phát triển một kiến trúc máy vi tính mới cho dự án BBC Computer Literacy. Sự thành công của hợp đồng này cho phép Acorn tiếp tục phát triển bộ vi xử lý RISC thương mại đầu tiên, Acorn RISC Machine (ARM). Phiên bản đầu tiên, ARM1, bắt đầu hoạt động vào năm 1985 và được sử dụng để nghiên cứu và phát triển nội bộ đồng thời được sử dụng làm bộ đồng xử lý trong máy BBC. Cũng trong năm 1985, Acorn cho ra mắt ARM2, có chức năng và tốc độ tốt hơn. Sau đó, ARM3 với những cải tiến khác được ra mắt vào năm 1989.



Hình 2.10 Tổ chức tổng quát của một hệ thống nhúng

Trong thời kỳ này, chip vi xử lý của Acorn thực tế được trực tiếp sản xuất bởi công ty Công nghệ VLSI. VLSI đã được cấp phép để bán chip mà họ làm ra và đạt được thành công trong việc giới thiệu các công ty khác sử dụng ARM làm vi xử lý nhúng trong các sản phẩm của họ.

Thiết kế ARM phù hợp nhu cầu thị trường ngày càng tăng đối với bộ xử lý hiệu suất cao, tiêu thụ ít năng lượng, kích thước nhỏ và giá thành rẻ cho các ứng dụng nhúng. Nhưng việc phát triển tiếp nữa vượt quá khả năng của Acorn. Do đó, một công ty mới với tên gọi ARM Ltd được sáng lập nên bởi các đối tác là Acorn, VLSI, và Apple Computer. Máy Acorn RISC Machine trở thành Advanced RISC Machine. Sản phẩm đầu tiên của công ty mới, cải tiến lên từ ARM3, là ARM6. Sau đó, công ty đã giới thiệu một số họ sản phẩm mới, với chức năng và hiệu suất ngày càng nâng cao. Bảng 2.7 cho thấy một số đặc điểm của các họ kiến trúc ARM khác nhau.

Bảng 2.7 Sự phát triển của ARM

Họ	Đặc điểm chính	Cache	MIPS @ MHz
ARM1	RISC 32 bit	Không	
ARM2	Nhân bản và tráo đổi lệnh; Khối quản lý bộ nhớ tích hợp, bộ xử lý đồ họa và vào/ra	Không	7 MIPS @ 12 MHz
ARM3	Họ đầu tiên sử dụng cache bộ xử lý	4 kB thống nhất	12 MIPS @ 25 MHz
ARM6	Họ đầu tiên hỗ trợ địa chỉ 32 bit; dấu chấm động	4 kB thống nhất	28 MIPS @ 33 MHz
ARM7	SoC tích hợp	8 kB thống nhất	60 MIPS @ 60 MHz
ARM8	Pipeline 5 giai đoạn; dự đoán rẽ nhánh tĩnh	8 kB thống nhất	84 MIPS @ 72 MHz
ARM9		16 kB/16 kB	300 MIPS @ 300 MHz
ARM9E	Lệnh DSP nâng cao	16 kB/16 kB	220 MIPS @ 200 MHz
ARM10E	Pipeline 6 giai đoạn	32 kB/32 kB	
ARM11	Pipeline 9 giai đoạn	Biến đổi	740 MIPS @ 665 MHz
Cortex	Pipeline siêu vô hướng 13 giai đoạn	Biến đổi	2000 MIPS @ 1 GHz
XScale	Bộ xử lý ứng dụng; Pipeline 7 giai đoạn	32 kB/32 kB L1 512 kB L2	1000 MIPS @ 1.25 GHz

2.6. CÂU HỎI

1. Máy tính chương trình lưu trữ là gì?

38 Chương 2. Sự phát triển của máy tính và hiệu năng

2. Bốn thành phần chính của máy tính đa năng tổng quát là gì?
3. Ba yếu tố chính khi thiết kế hiệu suất của hệ thống máy tính là gì?
4. Giải thích luật Moore?
5. Liệt kê và giải thích các đặc tính chính của một họ máy tính?

Chương 3. CHỨC NĂNG VÀ KẾT NỐI MÁY TÍNH

Sau khi đọc xong chương này, sinh viên sẽ:

- ❖ Hiểu được các thành phần cơ bản của chu kỳ lệnh và nguyên lý ngắt
- ❖ Mô tả nội dung của hệ thống kết nối trong máy tính
- ❖ Hiểu sự khác nhau giữa định thời bus đồng bộ và không đồng bộ
- ❖ Giải thích được tại sao cần phải thực hiện tổ chức đa bus phân cấp
- ❖ Đánh giá được ưu điểm của kết nối điểm đến so với kết nối bus
- ❖ Trình bày được tổng quan của QPI
- ❖ Trình bày được tổng quan của PCIe

Về mặt tổng quát, một máy tính gồm có CPU (bộ xử lý trung tâm), bộ nhớ và các thành phần vào/ra (I/O). Các thành phần trên được kết nối với nhau theo một quy tắc nhất định để thực thi các chức năng cơ bản của máy tính: thực thi các chương trình máy tính. Vậy, về mặt tổng quát, chúng tôi có thể mô tả một hệ thống máy tính bằng cách mô tả (1) *hành vi bên ngoài của mỗi thành phần*: dữ liệu và các tín hiệu điều khiển mà nó trao đổi với các thành phần khác và (2) *cấu trúc kết nối và điều khiển* cần thiết để quản lý việc sử dụng cấu trúc kết nối.

Việc tìm hiểu tổng quan về cấu trúc và chức năng hết sức quan trọng để có thể hiểu hết bản chất của máy tính. Ngoài ra, nó giúp ta hiểu các vấn đề phức tạp trong việc đánh giá hiệu suất. Việc nắm được cấu trúc và chức năng cho ta hiểu rõ về một số hiện tượng như: tắc nghẽn hệ thống (hiện tượng nút cỗ chai), luồng luân phiên, mức độ hỏng hóc của hệ thống nếu một thành phần hỏng và việc thêm vào các công nghệ mới để cải thiện hiệu năng. Trong nhiều trường hợp, người ta thường cải thiện hiệu năng và độ tin cậy của toàn hệ thống bằng cách thay đổi thiết kế thay vì tăng tốc độ và độ tin cậy của từng thành phần.

Chương này chúng tôi sẽ tập trung trình bày các cấu trúc cơ bản của hệ thống kết nối các thành phần máy tính. Trong đó, đầu tiên chúng tôi sẽ trình bày ngắn gọn về các thành phần cơ bản, giao diện và chức năng của chúng. Sau đó, chúng tôi sẽ mô tả kỹ lưỡng về hệ thống kết nối sử dụng bus trong máy tính.

3.1. CÁC THÀNH PHẦN MÁY TÍNH

Trong Chương 2 ta đã biết thiết kế máy tính được dựa trên kiến trúc cơ bản của John von Neumann: *kiến trúc von Neumann* với ba nội dung chính như sau:

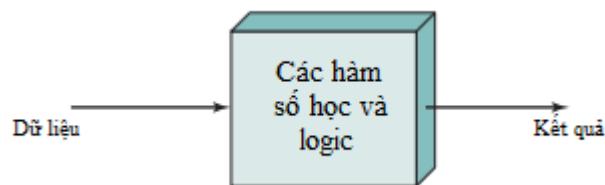
- Dữ liệu và lệnh được lưu trữ trên một bộ nhớ đọc-ghi
- Nội dung trong bộ nhớ này được đánh địa chỉ theo vị trí (không phân biệt kiểu dữ liệu được lưu trữ trong đó)
- Thực thi lệnh được thực hiện một cách tuần tự (lần lượt từng lệnh một)

Lý do đằng sau các nội dung này đã được trình bày trong Chương 2, tuy nhiên chúng tôi cũng tóm tắt lại như sau. Ta có một tập hợp các thành phần

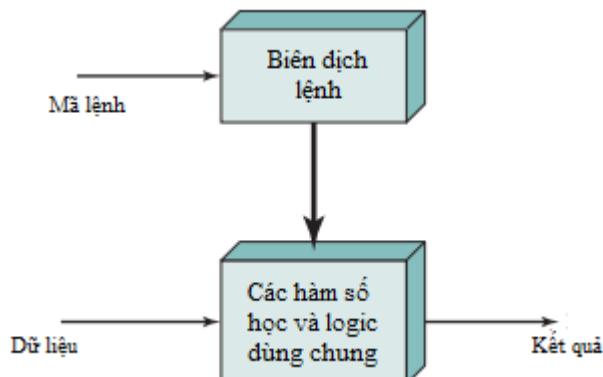
logic cơ bản được kết nối theo các cách khác nhau để lưu trữ dữ liệu nhị phân và thực hiện các phép toán số học và logic trên dữ liệu này. Nếu có một phép toán cần phải được tính toán, ta sẽ phải xây dựng một cấu hình các thành phần logic để thực hiện phép toán này. Có thể hiểu việc nối các thành phần khác nhau để tạo ra cấu hình mong muốn chính là việc lập trình. Kết quả là ta thu được một “*chương trình*” phần cứng.

Giả thiết thứ hai, giả sử ta xây dựng một cấu hình dùng chung cho tất cả các hàm toán học và logic. Bộ phận cứng này sẽ có thể thực hiện được các hàm khác nhau với dữ liệu tùy thuộc vào tín hiệu điều khiển đưa vào phần cứng. Trong trường hợp trên, đầu vào và đầu ra của mạch là dữ liệu và kết quả (Hình 3.1a). Ở trường hợp này, hệ thống nhận dữ liệu và tín hiệu điều khiển và đưa ra kết quả. Như vậy, thay vì phải nối lại dây cho mỗi một chương trình, người lập trình chỉ cần đưa ra một tập các tín hiệu điều khiển hoạt động của phần cứng.

Vậy các tín hiệu điều khiển được đưa vào như thế nào? Thực tế, mỗi chương trình là một chuỗi các bước. Tại mỗi bước, một vài phép toán số học và logic được thực hiện, do vậy cần phải có một tập các tín hiệu điều khiển tương ứng. Vì vậy, ta sẽ dùng một mã (code) duy nhất cho mỗi tập tín hiệu điều khiển và thiết lập thêm một khối phần cứng có nhiệm vụ nhận mã này và đưa ra các tín hiệu điều khiển tương ứng (Hình 3.1b).



(a) Lập trình phần cứng



(b) Lập trình phần mềm

Hình 3.1. Lập trình phần cứng và phần mềm

Bây giờ, công việc lập trình trở nên đơn giản hơn rất nhiều. Thay vì phải nối lại các bộ phận phần cứng cho mỗi chương trình mới, chúng ta chỉ cần đưa ra một chuỗi các từ mã mới. Mỗi một mã là một câu lệnh được đưa qua một

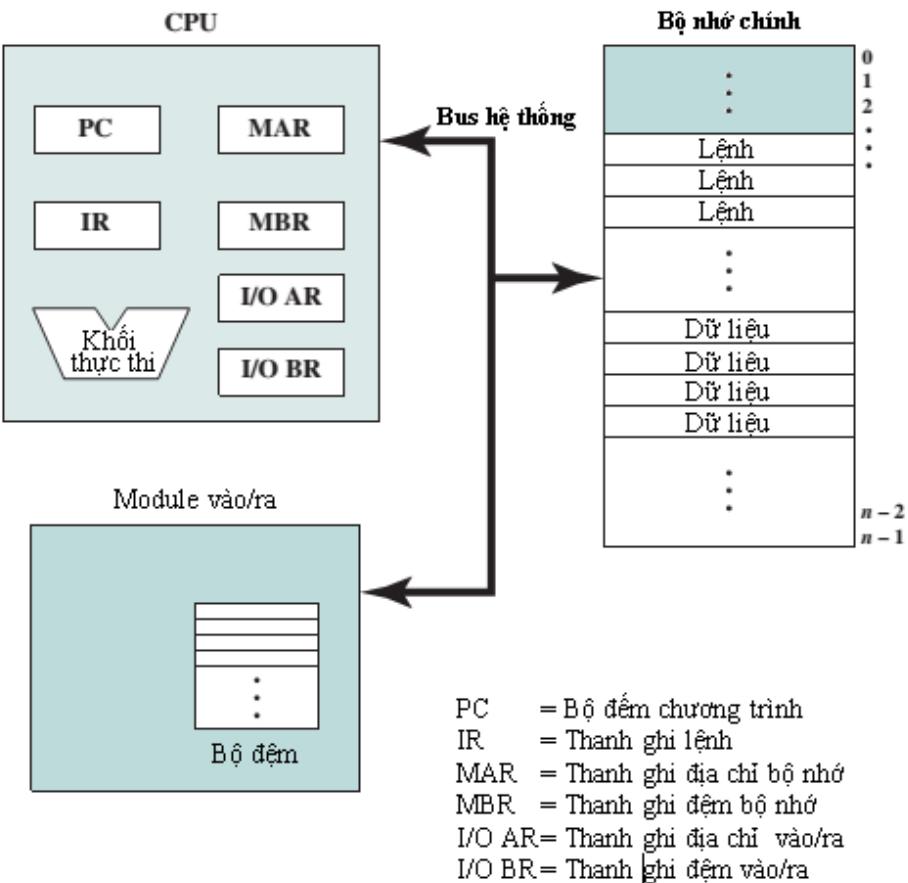
phần cứng để biên dịch thành các tín hiệu điều khiển. Chuỗi các mã hoặc lệnh này được gọi là *phần mềm*.

Trong hình 3.1b ta có thể thấy có hai thành phần chính của hệ thống: bộ biên dịch lệnh và một module thực thi các hàm tính toán số học và logic chung. Chúng cấu thành lên CPU. Ngoài ra, cần có một vài thành phần khác nữa mới có thể tạo nên một bộ máy tính. Dữ liệu và lệnh cần được đưa vào hệ thống. Để làm được việc đó, chúng ta cần module vào. Module này chứa các thành phần cơ bản để nhận dữ liệu và lệnh ở một vài dạng ban đầu, biến đổi chúng thành dạng tín hiệu bên trong có thể sử dụng được bởi hệ thống. Để đưa dữ liệu ra, ta cần một module ra. Vậy, chúng ta gọi chung là các *thành phần vào/ra (I/O module)*

Thêm vào đó, thiết bị vào sẽ đưa lệnh và dữ liệu vào một cách tuần tự, nhưng chương trình đôi khi không thực hiện một cách tuần tự, nó có thể nhảy đến vị trí khác (vd: lệnh *nhảy (JUMP)* trong IAS). Tương tự, có thể cần phải tham chiếu đến nhiều toán hạng và dữ liệu tại một thời điểm. Do vậy, ta cần phải có một nơi để lưu trữ tạm thời dữ liệu và lệnh. Module đó được gọi là *bộ nhớ* hay *bộ nhớ chính* để phân biệt với bộ nhớ ngoài hay các thiết bị ngoại vi. Von Neumann đã chỉ ra rằng dữ liệu và lệnh có thể cùng lưu trữ trên một bộ nhớ.

Hình 3.2 minh họa các thành phần chính của máy tính và giao tiếp giữa chúng. CPU trao đổi dữ liệu với bộ nhớ thông qua hai thanh ghi: MAR (memory address register – thanh ghi địa chỉ bộ nhớ) và MBR (memory buffer register – thanh ghi đệm bộ nhớ). Thanh ghi MAR chứa địa chỉ bộ nhớ cho thao tác đọc hoặc ghi tiếp theo, thanh ghi MBR chứa dữ liệu được đọc từ bộ nhớ hoặc chuẩn bị được ghi vào bộ nhớ. Tương tự, thanh ghi địa chỉ vào/ra (I/OAR) chỉ ra một thiết bị vào/ra cụ thể. Thanh ghi đệm vào/ra (I/OBR) được sử dụng để chuyển tiếp dữ liệu giữa một module vào/ra và CPU

Module bộ nhớ gồm một tập các vị trí được đánh địa chỉ dạng số liên tục. Mỗi vị trí chứa một số nhị phân có thể là dữ liệu hoặc lệnh. Một module vào/ra truyền dữ liệu từ thiết bị ngoài vào CPU và bộ nhớ và ngược lại. Các bộ đệm bên trong module vào/ra lưu trữ dữ liệu tạm thời cho đến khi nó được gửi đi.



3.2. CHỨC NĂNG CỦA MÁY TÍNH

Chức năng chính của máy tính là thực thi chương trình gồm nhiều lệnh lưu trữ trong bộ nhớ máy tính. Bộ xử lý sẽ lần lượt thực hiện các lệnh theo một chu trình nhất định. Ở dạng đơn giản nhất, một câu lệnh được thực thi qua hai bước: Bộ xử lý *đọc* (*truy xuất*) lệnh từ bộ nhớ và *thực thi* lệnh đó. Việc đó được thực hiện lặp đi lặp lại với tất cả các lệnh trong chương trình.

Quá trình thực hiện một lệnh được gọi là **chu kỳ lệnh**. Một chu kỳ lệnh được minh họa trong Hình 3.3 gồm có hai bước như đã nói ở trên đó là: **chu kỳ truy xuất** và **chu kỳ thực thi**.

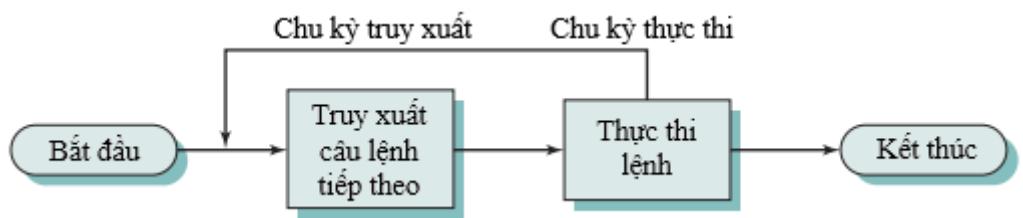
3.2.1. Truy xuất và thực thi lệnh

Bắt đầu một chu kỳ lệnh, bộ xử lý truy xuất lệnh từ bộ nhớ. Trong các máy tính thông thường, thanh ghi PC (program counter: bộ đếm chương trình) chứa địa chỉ của lệnh sẽ được truy xuất tiếp theo. Nếu không có gì đặc biệt, bộ xử lý tự động tăng PC sau mỗi lần truy xuất lệnh để nó sẽ truy xuất đến câu lệnh tiếp theo của chương trình (các câu lệnh được lưu trữ lần lượt tại các địa chỉ bộ nhớ liên tục, câu lệnh sau có địa chỉ tiếp theo lớn hơn câu lệnh trước). Ví dụ, ta có một máy tính trong đó mỗi từ lệnh có 16 bit lưu trữ trong bộ nhớ. Bộ xử lý truy xuất đến câu lệnh tại địa chỉ 300. Theo trình tự thông thường, các câu lệnh tiếp theo truy xuất đến sẽ lần lượt là 301, 302, 303, ...

Lệnh sau khi được truy xuất sẽ được ghi vào thanh ghi IR (instruction register – thanh ghi lệnh) trong bộ xử lý. Lệnh chứa các bit chỉ ra hoạt động bộ xử lý cần thực hiện. Bộ xử lý giải mã các bit này và thực hiện hoạt động được yêu cầu. Thông thường, các hoạt động được chia thành bốn nhóm:

- **Bộ xử lý – bộ nhớ:** Dữ liệu được truyền từ bộ xử lý đến bộ nhớ hoặc từ bộ nhớ đến bộ xử lý.
- **Bộ xử lý – Vào/ra:** Dữ liệu được truyền đến và đi từ thiết bị ngoại vi bằng cách truyền dữ liệu giữa bộ xử lý và module vào/ra.
- **Xử lý dữ liệu:** Bộ xử lý thực hiện các phép toán số học và logic lên dữ liệu
- **Điều khiển:** Một lệnh có thể chứa chỉ thị làm thay đổi tuần tự thực hiện các câu lệnh. Ví dụ: bộ xử lý truy xuất câu lệnh ở địa chỉ 149, trong đó chứa chỉ thị yêu cầu truy xuất câu lệnh tiếp theo ở địa chỉ 182. Bộ xử lý sẽ phải thiết lập thanh ghi PC về giá trị 182 thay vì 150 theo trình tự ban đầu.

Việc thực thi một lệnh có thể kết hợp tất cả các hoạt động trên.

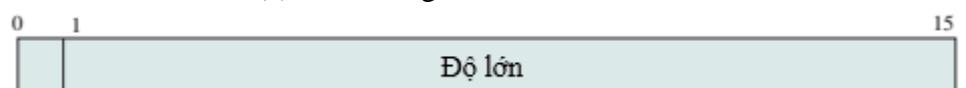


Hình 3.3 Chu kỳ lệnh cơ bản

Xét một ví dụ đơn giản trong Hình 3.4 như sau: ta có một máy giả thiết có các đặc điểm như trong hình. Bộ xử lý chứa một thanh ghi dữ liệu được gọi là thanh ghi AC (accumulator). Cả lệnh và dữ liệu có 16 bit chiều dài do vậy sẽ rất thuận tiện nếu ta tổ chức bộ nhớ thành các từ nhớ 16b. Về cấu trúc lệnh: 4 bit dùng cho phần mã lệnh, như vậy ta có thể có đến $2^4 = 16$ mã lệnh khác nhau và tối đa $2^{12} = 4096$ (4K) từ nhớ trong bộ nhớ có thể được truy cập trực tiếp.



(a) Định dạng lệnh



(b) Định dạng số nguyên

Thanh ghi PC = Địa chỉ của lệnh

Thanh ghi IR = Lệnh sẽ được thực hiện

Thanh ghi AC = Thanh ghi lưu trữ dữ liệu tạm thời

(c) Các thanh ghi trong CPU

0001 = Đọc dữ liệu từ bộ nhớ vào thanh ghi AC

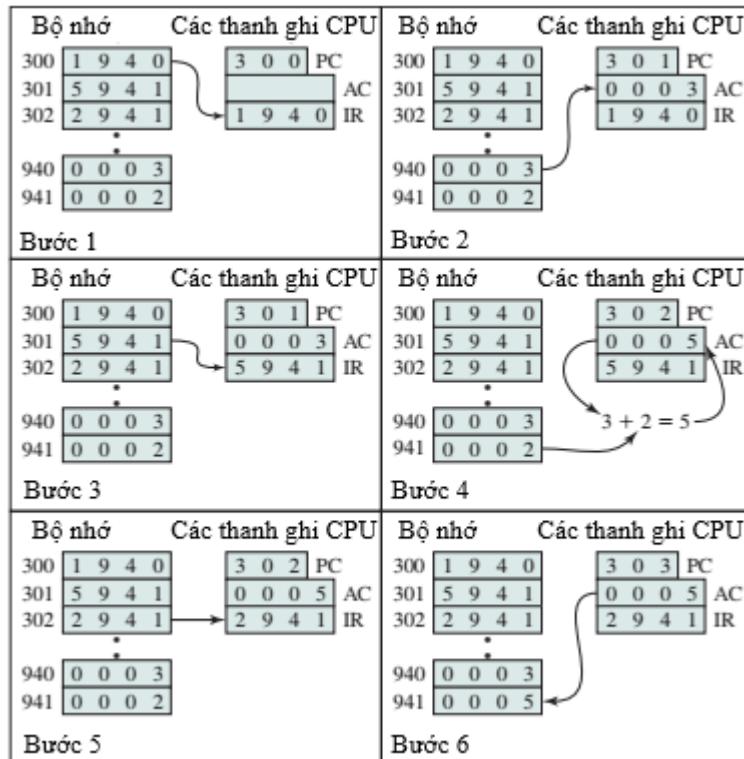
44 Chương 3. Chức năng và kết nối máy tính

0010 = Lưu trữ dữ liệu từ thanh ghi AC vào bộ nhớ

0101 = Đọc dữ liệu từ bộ nhớ và cộng vào thanh ghi AC

(d) Các mã lệnh được sử dụng

Hình 3.4 Đặc tính của máy giả thiết

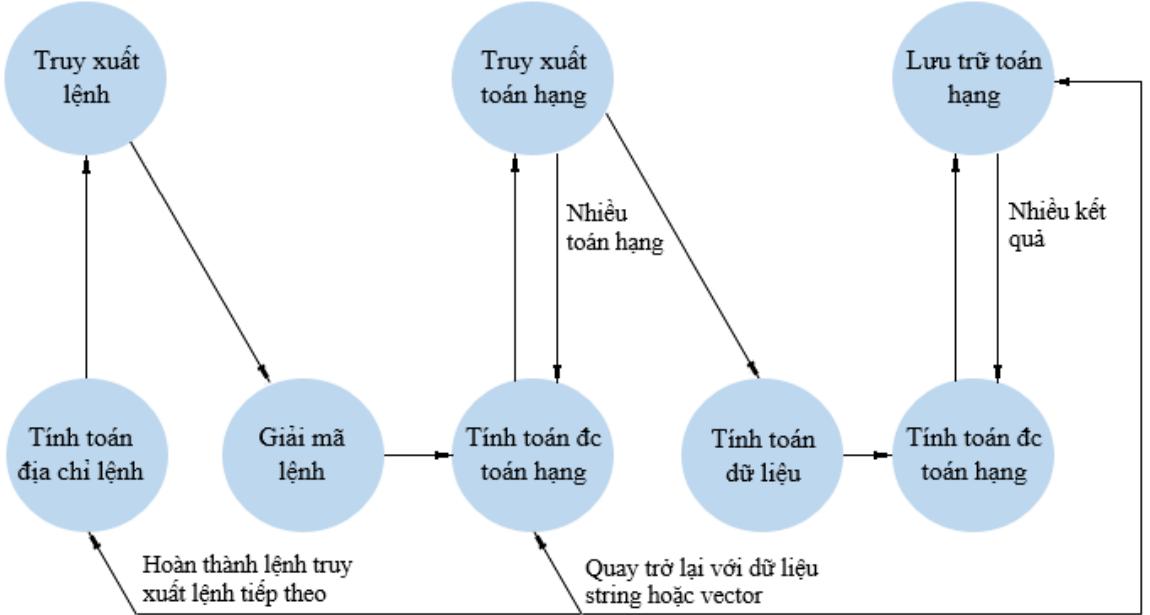


Hình 3.5 Ví dụ về thực hiện chương trình (dữ liệu trong bộ nhớ và thanh ghi biểu diễn dưới dạng số thập lục phân)

Hình 3.5 minh họa hoạt động của bộ xử lý thực hiện một chương trình và trạng thái của các thanh ghi trong quá trình đó. Chương trình này cộng giá trị tại địa chỉ 940 và 941 và ghi kết quả vào ô nhớ 941. Với các lệnh này, ta sẽ có ba chu kỳ truy xuất và ba truy kỳ thực thi như sau:

- Thanh ghi PC có giá trị 300 (địa chỉ của câu lệnh đầu tiên của chương trình). Do vậy, lệnh (có giá trị 1940 theo hệ thập lục phân) được tải vào thanh ghi IR, sau đó thanh ghi PC tăng thêm một đơn vị. Chú ý rằng, quá trình truy xuất này có sự tham gia của hai thanh ghi là MAR và MBR. Tuy nhiên để đơn giản, chúng ta tạm thời bỏ qua các thanh ghi này.
- 4 bit đầu tiên (tương ứng với chữ số hệ thập lục phân đầu tiên) trong thanh ghi IR chỉ thị rằng dữ liệu sẽ được tải vào thanh ghi AC. 12 bit còn lại là địa chỉ (940) của dữ liệu được tải vào.
- Lệnh tiếp theo (5941) được đọc vào từ địa chỉ 301, sau đó thanh ghi PC tăng lên một đơn vị.

4. Cộng dữ liệu từ địa chỉ 941 với dữ liệu cũ trong thanh ghi AC, kết quả được lưu trữ lại vào thanh ghi AC.
5. Lệnh tiếp theo (2941) được truy xuất từ địa chỉ 302, thanh ghi PC tăng lên một đơn vị.
6. Nội dung của thanh ghi AC được lưu trữ vào địa chỉ 941.



Hình 3.6 Đồ thị trạng thái chu kỳ lệnh

Trong ví dụ này, mỗi lệnh bao gồm một chu kỳ truy xuất và chu kỳ thực thi. Trong các thế hệ máy tính sau này, một lệnh có thể có nhiều hơn một địa chỉ, do đó chu kỳ lệnh sẽ có nhiều hơn một lần truy cập bộ nhớ. Ngoài ra, không chỉ tham chiếu đến bộ nhớ, các lệnh còn có thể thực hiện các hoạt động vào/ra thiết bị ngoại vi. Hình 3.6 minh họa chi tiết hơn về chu kỳ lệnh dưới dạng đồ thị trạng thái. Với mỗi lệnh cụ thể, một số trạng thái có thể không có, một số trạng thái có thể xuất hiện nhiều hơn một lần. Các trạng thái bao gồm:

- **Tính toán địa chỉ lệnh (iac):** Xác định địa chỉ lệnh tiếp theo được thực hiện. Thông thường, việc này được thực hiện bằng cách cộng một số cố định vào địa chỉ của lệnh trước đó. Ví dụ, nếu lệnh dài 16 bit và bộ nhớ được tổ chức dưới dạng các từ nhớ 16b thì ta sẽ cộng 1 vào địa chỉ lệnh trước đó. Nếu bộ nhớ được tổ chức dưới dạng các từ nhớ 8b thì sẽ phải cộng 2 vào địa chỉ trước lệnh trước đó (do một lệnh sẽ được lưu trữ trên 2 từ nhớ).
- **Truy xuất lệnh (if):** Đọc lệnh từ địa chỉ của nó vào bộ xử lý.
- **Giải mã lệnh (iod):** Phân tích lệnh để xác định loại hoạt động cần thực hiện và các toán hạng được sử dụng.
- **Tính toán địa chỉ toán hạng (oac):** Nếu hoạt động cần phải tham chiếu đến một toán hạng trong bộ nhớ hay qua thiết bị ngoại vi thì xác định địa chỉ của toán hạng.

46 Chương 3. Chức năng và kết nối máy tính

- **Truy xuất toán hạng (of):** Truy xuất toán hạng từ bộ nhớ hoặc đọc từ thiết bị ngoại vi.
- **Tính toán dữ liệu (do):** Thực hiện phép toán được chỉ ra trong lệnh.
- **Lưu trữ toán hạng (os):** Ghi kết quả vào bộ nhớ hoặc đưa ra thiết bị ngoại vi.

Các trạng thái trong phần trên của Hình 3.6 là các trạng thái có sự truyền dữ liệu giữa bộ xử lý và bộ nhớ hoặc module vào/ra. Các trạng thái ở dưới của Hình chỉ được thực hiện bên trong bộ xử lý. Mỗi lệnh có thể có nhiều toán hạng hoặc nhiều kết quả tùy thuộc vào các hệ máy tính khác nhau

3.2.2. Ngắt

Hầu như các máy tính đều có cơ chế cho phép các module khác (bộ nhớ, vào/ra) có thể **ngắt** tiến trình thông thường của bộ xử lý. Bảng 3.1 liệt kê một số loại ngắt phổ biến nhất. Trong chương 7 chúng tôi sẽ trình bày rõ hơn về ngắt, tuy nhiên, để hiểu rõ hơn về chu kỳ lệnh và việc thực hiện ngắt trong cấu trúc kết nối, chúng tôi sẽ trình bày sơ qua về ngắt. Người đọc không cần phải đi sâu về quá trình đưa ra ngắt và xử lý nó, ta tập trung vào việc truyền tín hiệu giữa các module khi có ngắt.

Bảng 3.1 Các loại ngắt

Chương trình	Được tạo ra trong quá trình thực hiện lệnh, chẳng hạn như: tràn số học, chia cho không, cố gắng thực thi một lệnh không được phép hoặc tham chiếu đến bên ngoài không gian bộ nhớ của người dùng.
Định thời	Được tạo ra bởi bộ định thời trong bộ xử lý. Chức năng này cho phép hệ điều hành thực hiện định kỳ một hoạt động nào đó.
Vào/ra	Được tạo ra bởi bộ điều khiển vào/ra để báo hiệu khi một hoạt động vào/ra đã hoàn thành, yêu cầu sự phục vụ của bộ xử lý hoặc để báo lỗi.
Lỗi phần cứng	Được tạo ra bởi một lỗi như lỗi nguồn điện hoặc lỗi chấn lě bộ nhớ

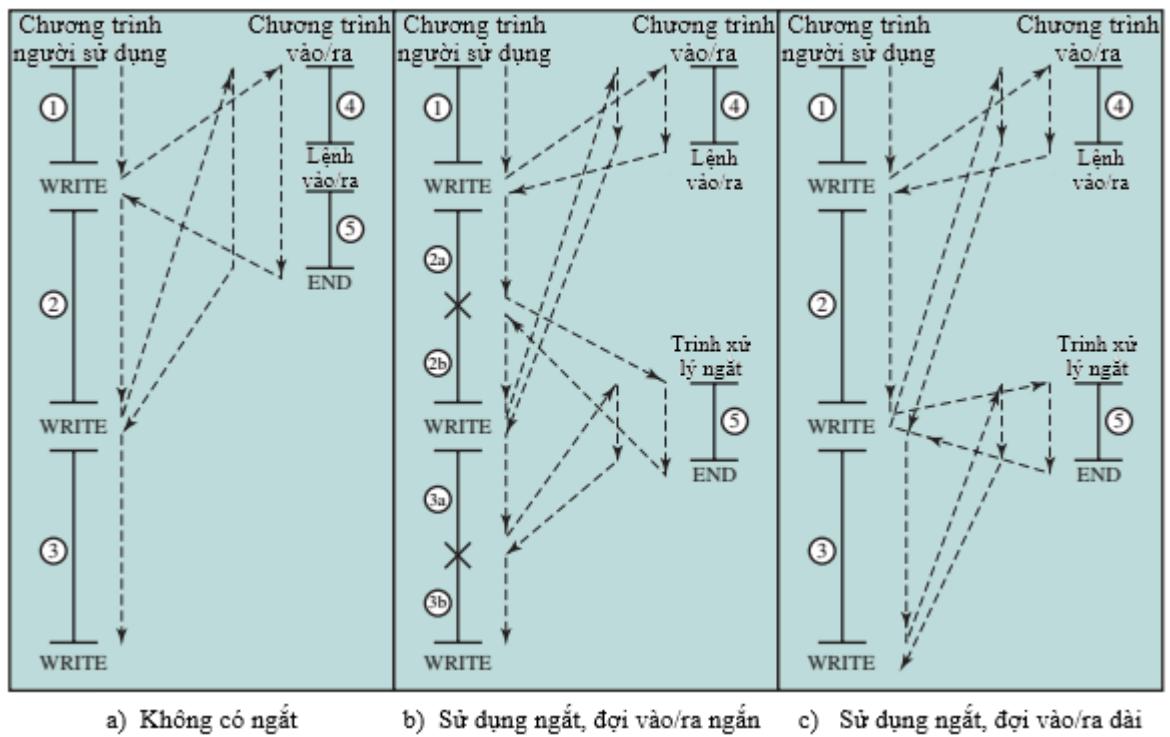
Ngắt được đưa ra với mục đích cải thiện hiệu suất của bộ xử lý. Ví dụ, hầu như các thiết bị ngoại đều chậm hơn bộ xử lý. Giả sử bộ xử lý truyền dữ liệu đến một máy in sử dụng chu kỳ lệnh như trong Hình 3.3. Sau mỗi lệnh ghi, bộ xử lý phải dừng lại và chờ đợi máy in làm việc. Khoảng thời gian này có thể gấp hàng trăm hoặc hàng nghìn lần một chu kỳ lệnh. Rõ ràng, điều này sẽ làm lãng phí hiệu suất của bộ xử lý.

Hình 3.7a minh họa quá trình này. Một chương trình người sử dụng (user program) thực hiện một loạt lời gọi WRITE. Các đoạn mã 1, 2, 3 là các chuỗi câu lệnh không có giao tiếp với thiết bị ngoại vi. Lời gọi WRITE này gọi một

chương trình vào/ra để thực hiện một hoạt động vào/ra. Hoạt động vào/ra này gồm ba phần như sau:

- Một chuỗi lệnh có nhãn 4 trong hình, chuẩn bị cho hoạt động vào/ra. Nó có thể bao gồm việc sao chép dữ liệu vào một bộ đệm và chuẩn bị các tham số cho các lệnh của thiết bị.
- Lệnh vào/ra. Nếu không có ngắt, khi lệnh này được thực hiện, chương trình phải đợi thiết bị ngoại vi thực hiện chức năng được yêu cầu hoặc thực hiện lặp lại hoạt động kiểm tra để xác định xem liệu hoạt động vào/ra đã hoàn thành chưa.
- Một chuỗi lệnh nhãn 5 hoàn thành nốt hoạt động. Chuỗi lệnh này có thể bao gồm việc thiết lập một cờ để báo cho bộ xử lý biết là hoạt động đã hoàn thành hoặc lỗi.

Vì hoạt động vào/ra mất một khoảng thời gian khá lớn để hoàn thành, chương trình vào/ra phải treo để đợi hoạt động hoàn tất, do đó chương trình người sử dụng phải dừng lại ở lời gọi WRITE một khoảng thời gian nhất định.



Hình 3.7 Tiến trình của chương trình không có và có ngắt

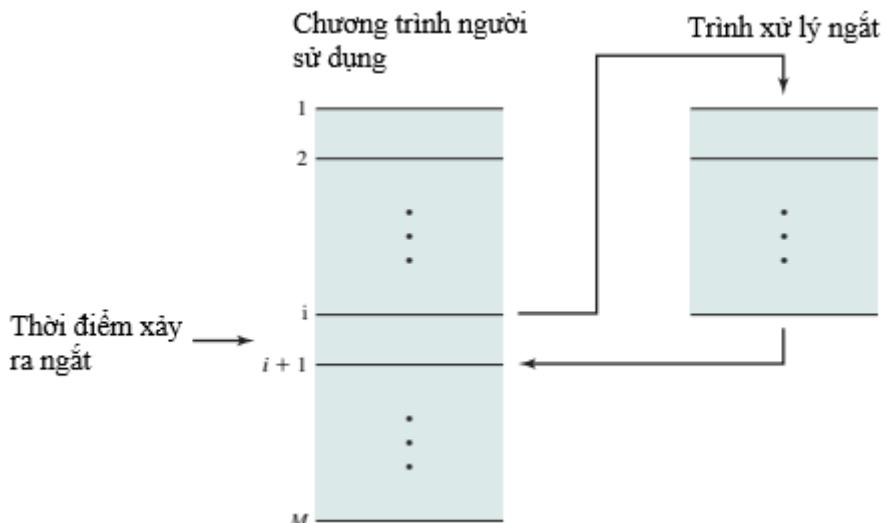
3.2.2.1. Ngắt và chu kỳ lệnh

Với ngắt, bộ xử lý có thể thực hiện các lệnh khác trong khi một tiến trình vào/ra đang thực hiện. Xét luồng điều khiển trong Hình 3.7b. Khác với phần a, chương trình người sử dụng thực hiện lời gọi WRITE, lời gọi này gọi đến chương trình vào/ra thực hiện các mã chuẩn bị và lệnh vào/ra thực tế (nhãn 4). Sau khi nhãn này thực hiện xong, luồng điều khiển trở về chương trình người sử dụng. Thiết bị ngoại vi nhận dữ liệu từ bộ nhớ máy tính và thực hiện công

việc in, trong khi đó, bộ xử lý tiếp tục thực hiện nhãn 2 của chương trình người sử dụng.

Sau khi máy in in xong và giải phóng bộ nhớ đệm, nghĩa là nó sẵn sàng nhận thêm dữ liệu từ bộ xử lý, module vào/ra sẽ gửi một tín hiệu ngắn đến bộ xử lý. Bộ xử lý đáp ứng bằng cách ngưng hoạt động của chương trình hiện tại (nhãn 2), rẽ nhánh sang một chương trình phục vụ thiết bị vào/ra tương ứng (được gọi là **trình xử lý ngắn**) và tiếp tục điều khiển quá trình thực hiện Chương trình vào/ra. Các thời điểm bộ xử lý bị ngắn được ký hiệu bằng dấu sao trong hình 3.7b.

Đối với chương trình người dùng, khi có yêu cầu ngắn được gửi đến trong lúc chương trình đang được thực thi, chương trình sẽ bị dừng lại (ngắn). Khi việc xử lý ngắn xong, chương trình sẽ được khôi phục lại và tiếp tục được thực thi (Hình 3.8). Do vậy, chương trình người sử dụng không bao giờ đoạn mã đặc biệt nào chứa ngắn, bộ xử lý và hệ điều hành sẽ treo chương trình người sử dụng và khôi phục lại nó tại cùng thời điểm.



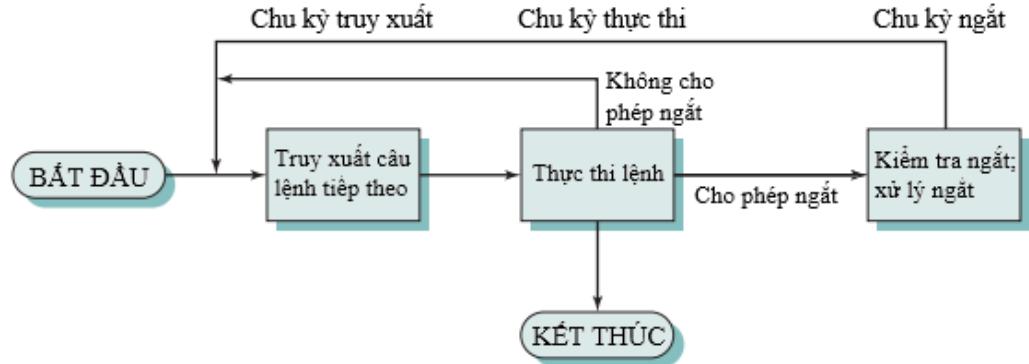
Hình 3.8 Luồng điều khiển trong trường hợp có ngắn

Để thực hiện ngắn, *chu kỳ ngắn* được thêm vào chu kỳ lệnh như trong Hình 3.9. Trong chu kỳ ngắn, bộ xử lý kiểm tra xem liệu có yêu cầu ngắn nào được gửi tới hay không (bằng cách kiểm tra tín hiệu báo ngắn). Nếu không có ngắn nào đang chờ, bộ xử lý thực thi chu kỳ truy xuất và truy xuất đến câu lệnh tiếp theo của chương trình hiện tại. Nếu có ngắn đang đợi, bộ xử lý thực hiện các công việc sau:

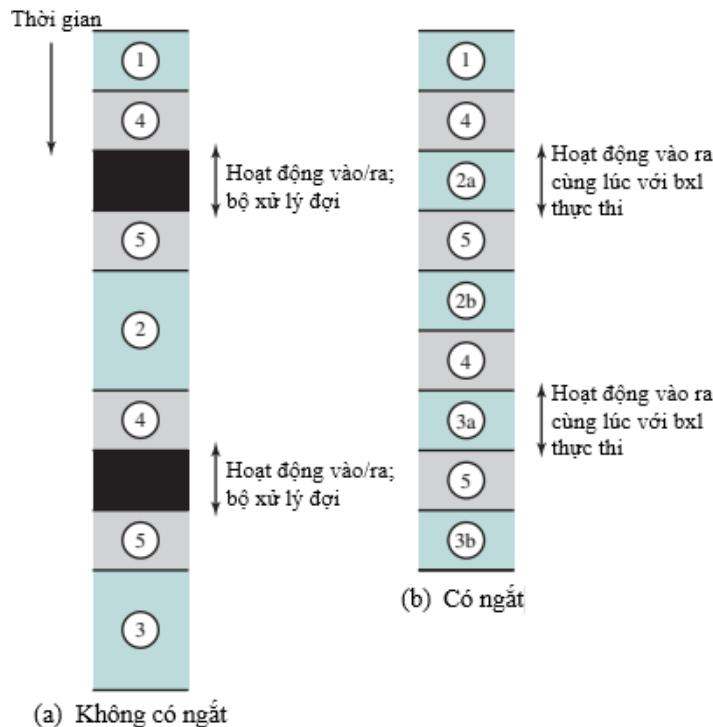
- Treo chương trình hiện tại đang thực thi và lưu nội dung của nó bằng cách lưu địa chỉ của lệnh tiếp theo sẽ được thực thi (nội dung thanh ghi PC) và tắt cả các dữ liệu liên quan khác.
- Thiết lập thanh ghi PC về địa chỉ đầu tiên của *trình xử lý ngắn*.

Bộ xử lý thực hiện chu kỳ truy xuất và truy xuất đến lệnh đầu tiên của chương trình xử lý ngắn để phục vụ ngắn. Chương trình xử lý ngắn thường là

một chương trình có trong hệ điều hành. Chương trình này sẽ xác định loại ngắt và thực hiện tất cả các hành động cần thiết. Như trong ví dụ trên, trình xử lý ngắt xác định module vào/ra nào đã phát ra ngắt và rẽ nhánh đến chương trình ghi dữ liệu ra module vào/ra. Khi trình xử lý ngắt hoàn thành, bộ xử lý sẽ khôi phục lại chương trình người sử dụng tại thời điểm bị ngắt.



Để hiểu rõ hơn về hiệu quả của việc xử lý ngắt, xét sơ đồ thời gian trong Hình 3.10 dựa trên luồng điều khiển trong Hình 3.7a và 3.7b. Các đoạn mã của chương trình người sử dụng được tô màu xanh, các đoạn mã của chương trình vào/ra được tô màu ghi. Hình 3.10a là trường hợp khi không sử dụng ngắt. Bộ xử lý sẽ phải đợi đến khi hoạt động vào/ra thực hiện xong.

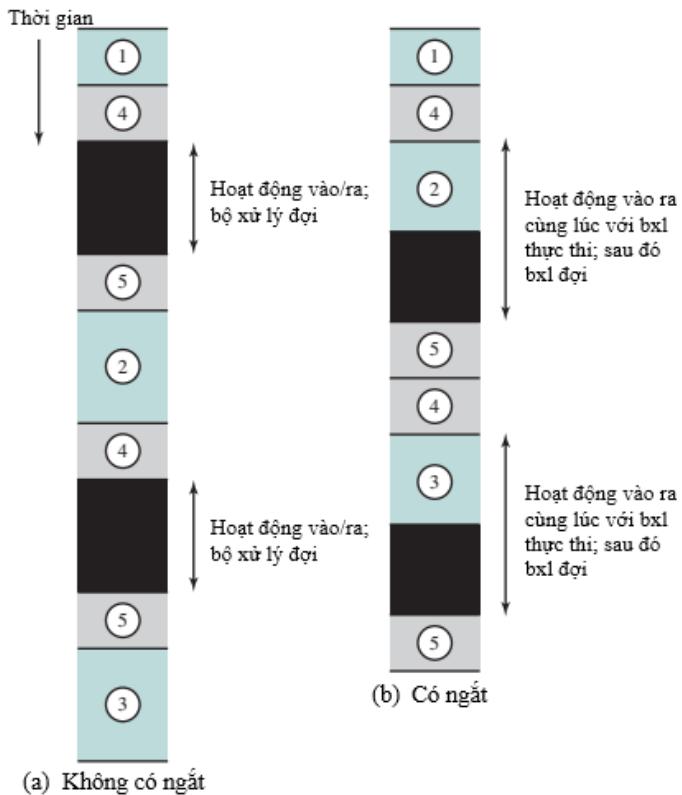


Hình 3.10 Tiến trình thời gian của chương trình: trường hợp thời gian đợi hoạt động Vào/ra ngắn

Hình 3.7b và 3.10b minh họa trường hợp giả thiết rằng khoảng thời gian thực hiện hoạt động vào/ra khá ngắn, ngắn hơn thời gian bộ xử lý thực hiện

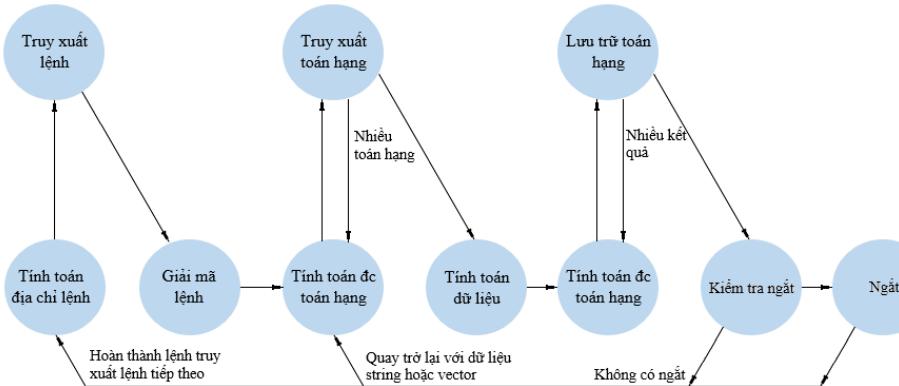
50 Chương 3. Chức năng và kết nối máy tính

nhận 2. Do đó, nhận 2 sẽ bị ngắt để phục vụ thiết bị ngoại vi và khôi phục lại sau khi xử lý ngắt xong.



Hình 3.11 Tiến trình thời gian của chương trình: trường hợp thời gian đợi hoạt động Vào/ra dài

Một trường hợp thường gặp khác (Hình 3.7c) là khi thiết bị ngoại vi chậm hơn nhiều so với bộ xử lý, ví dụ như máy in (printer), hoạt động vào/ra mất nhiều thời gian hơn so với việc bộ xử lý thực hiện các lệnh nhận 2, chương trình người sử dụng đi đến lời gọi GHI (WRITE) trước khi hoạt động vào/ra đầu tiên thực hiện xong. Chương trình người sử dụng tạm thời dừng lại và đợi đến khi chương trình vào/ra hoàn thành thì lời gọi hàm GHI mới tiếp tục được xử lý. Hình 3.11 là sơ đồ thời gian của trường hợp này và mặc dù không cần sử dụng ngắt nhưng cơ chế xử lý này vẫn cải thiện hiệu suất của hệ thống

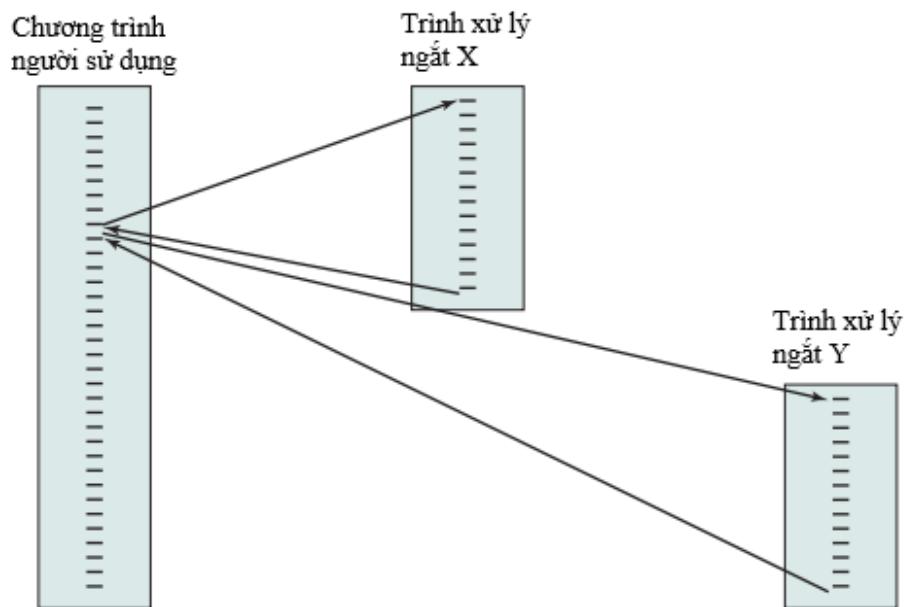


Hình 3.12 Sơ đồ trạng thái chu kỳ lệnh có ngắt

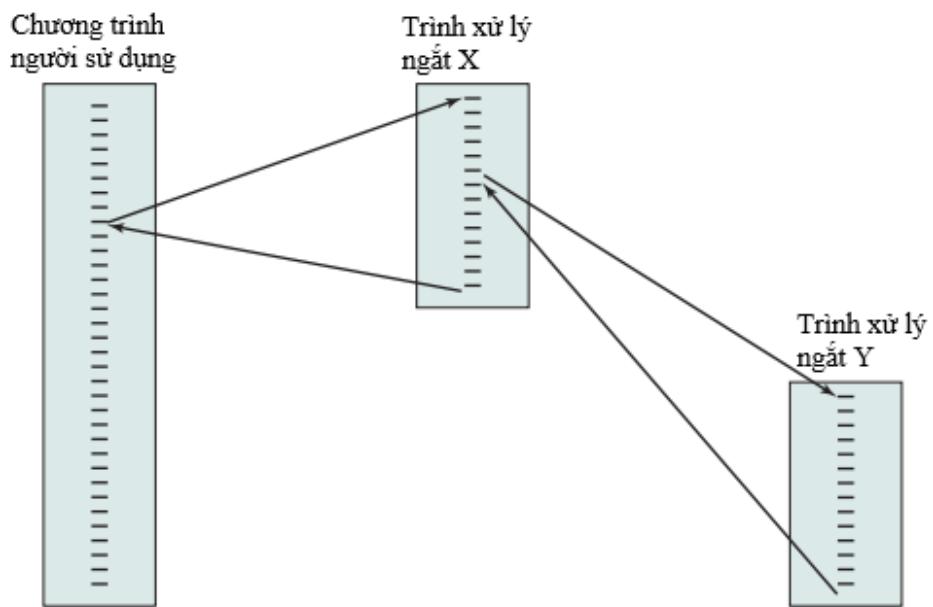
Hình 3.12 là sơ đồ trạng thái chu kỳ lệnh bao gồm cả việc xử lý ngắt (gồm có chu kỳ ngắt)

3.2.2.2. Xử lý nhiều ngắt (da ngắt)

Các trường hợp ở phần trên chỉ có một ngắt duy nhất tại một thời điểm. Vậy, giả thiết nếu nhiều ngắt được yêu cầu cùng một lúc. Ví dụ như một máy tính có thể đang nhận dữ liệu từ một đường truyền thông tin và nhận kết quả từ máy in. Máy in sẽ đưa ra một ngắt mỗi lần hoàn tất thao tác in. Bộ điều khiển đường truyền cũng gửi yêu cầu ngắt mỗi lần một đơn vị dữ liệu được gửi đến. Giả sử, yêu cầu ngắt này xuất hiện ngay tại thời điểm bộ xử lý đang xử lý ngắt máy in.



(a) Xử lý ngắt tuần tự



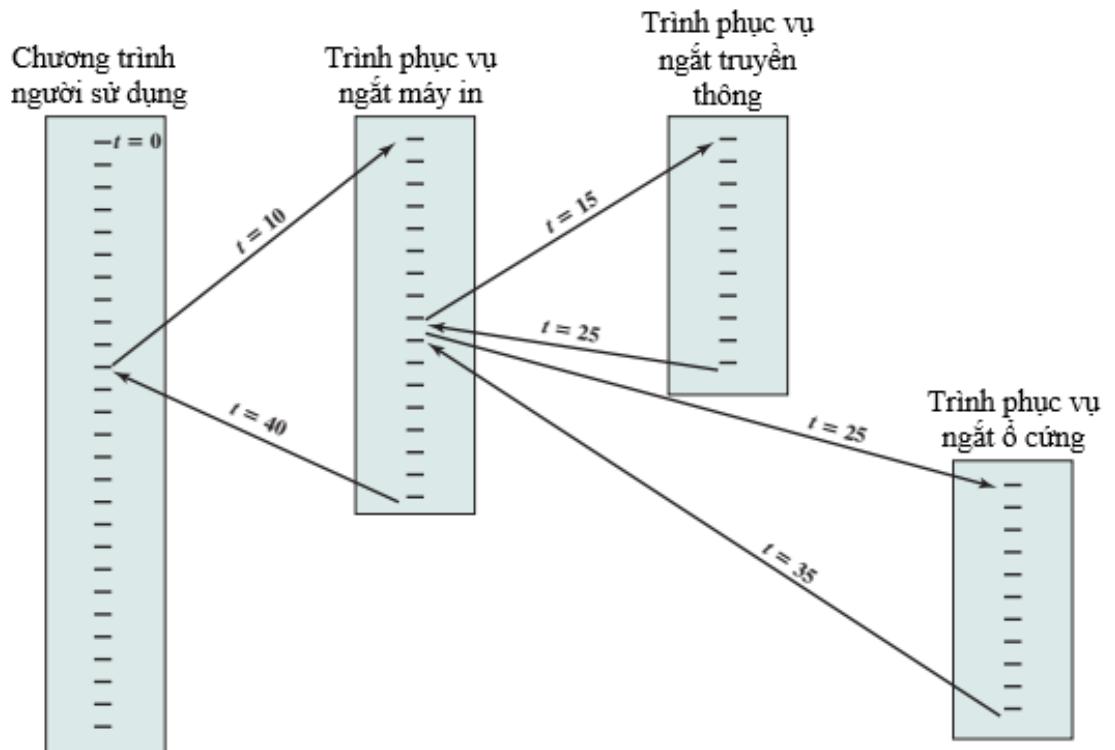
(b) Xử lý ngắt ưu tiên

Hình 3.13 Truyền điều khiển trong trường hợp đa ngắt

Có hai giải pháp cho vấn đề này. Thứ nhất, ta vô hiệu hóa tất cả các ngắt được gửi đến trong khi bộ xử lý đang xử lý một ngắt bất kỳ. Như vậy, nếu có yêu cầu ngắt trong khoảng thời gian này, nó sẽ được lưu lại, đến khi nào trình xử lý ngắt hoàn thành, bộ xử lý sẽ kiểm tra xem có ngắt nào đang chờ hay không trước khi quay lại chương trình người sử dụng. (Hình 3.13a).

Nhược điểm của phương pháp trên là nó không giải quyết được vấn đề ưu tiên hoặc yêu cầu thời gian của ngắt. Ví dụ, dữ liệu được gửi đến từ đường truyền thông, nó cần phải được nhận một cách nhanh chóng để giải phóng bộ nhớ đệm cho dữ liệu khác đến. Nếu đợt dữ liệu đầu tiên đi vào chưa được xử lý trước khi đợt thứ hai đến, dữ liệu có thể bị mất.

Phương pháp thứ hai là xác định ưu tiên cho ngắt. Phương pháp này cho phép ngắt có mức ưu tiên cao hơn có thể ngắt quá trình xử lý ngắt có mức ưu tiên thấp hơn (Hình 3.13b). Ví dụ, xét một hệ thống với ba thiết bị vào/ra: máy in, ổ đĩa và một đường truyền thông với các mức ưu tiên lần lượt là 2, 4 và 5. Hình 3.14 minh họa trường hợp một chương trình người dùng bắt đầu ở thời điểm $t = 0$. Tại $t = 10$, một ngắt máy in xảy ra; thông tin chương trình người sử dụng được đặt trên bộ nhớ ngăn xếp hệ thống và bộ xử lý thực thi **trình phục vụ ngắt (ISR)** máy in. Mặc dù thủ tục này vẫn đang được thực thi, tại $t = 15$, một ngắt đường truyền thông xảy ra. Do đường truyền thông có mức ưu tiên cao hơn so với máy in, ngắt này được thực hiện. ISR máy in bị ngắt, trạng thái của nó được đẩy lên bộ nhớ ngăn xếp, bộ xử lý thực thi ISR cho đường truyền thông. Trong khi ISR này được thực hiện, một yêu cầu ngắt được gửi đến từ ổ đĩa ($t = 20$). Bởi vì ngắt này có mức ưu tiên thấp hơn, nó bị giữ lại để đợi ISR của đường truyền thông tin kết thúc.



3.2.3. Chức năng vào/ra

Chúng ta vừa xem xét hoạt động của máy tính với sự điều khiển của bộ xử lý cũng như sự giao tiếp giữa bộ xử lý và bộ nhớ. Một phần không kém quan trọng khác là sự tương tác giữa bộ xử lý và các thành phần vào/ra. Vấn đề này sẽ được giải thích rõ hơn trong Chương 7, tuy nhiên, ở đây chúng tôi sẽ tóm tắt ngắn gọn một số điểm như sau:

Một module vào/ra (ví dụ: bộ điều khiển ổ đĩa) có thể trao đổi dữ liệu trực tiếp với bộ vi xử lý. Tương tự như cách bộ vi xử lý đọc hoặc ghi dữ liệu với bộ nhớ, nó cũng có thể đọc hoặc ghi dữ liệu với module vào/ra thông qua việc định một địa chỉ cụ thể và trình tự các lệnh thực hiện việc trao đổi dữ liệu này giống như trong Hình 3.5 với các lệnh vào/ra thay vì các lệnh tham chiếu bộ nhớ.

Ngoài ra, các hệ thống máy tính có cơ chế cho phép trao đổi dữ liệu từ module vào/ra trực tiếp với bộ nhớ. Bộ xử lý sẽ cấp cho một module vào/ra quyền đọc hoặc ghi dữ liệu vào bộ nhớ, từ đó việc truyền giữa module vào/ra-memory được thực hiện mà không cần sự tham gia của bộ xử lý - giảm trách nhiệm của bộ xử lý. Cơ chế này được gọi là cơ chế truy cập bộ nhớ trực tiếp (DMA) sẽ được trình bày cụ thể trong Chương 7.

3.3. CẤU TRÚC KẾT NỐI

Một máy tính bao gồm một tập các thành phần hoặc module theo ba loại cơ bản (bộ xử lý, bộ nhớ, vào/ra) trao đổi thông tin với nhau. Thực chất, một máy tính giống như một mạng lưới các module cơ bản. Do đó, ta cần một bộ phận để kết nối các module với nhau.

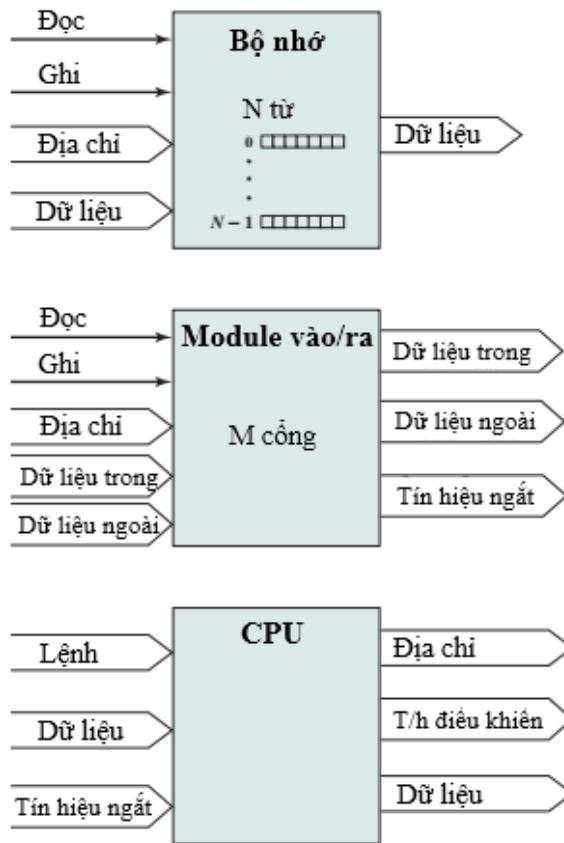
Tập hợp các bộ phận kết nối này được gọi là cấu trúc kết nối. Việc thiết kế cấu trúc này phụ thuộc vào sự trao đổi dữ liệu giữa các module.

Hình 3.15 minh họa các loại thông tin đầu vào và đầu ra của mỗi loại module

- **Bộ nhớ:** module bộ nhớ bao gồm N từ nhớ có kích thước bằng nhau. Mỗi từ nhớ được gán cho một địa chỉ dạng số $(0, 1, 2, \dots, N - 1)$. Tín hiệu điều khiển *đọc/ghi* sẽ quyết định *dữ liệu* (từ nhớ) được đọc hoặc ghi vào bộ nhớ. Vị trí của hoạt động (đọc/ghi) được định ra thông qua tín hiệu *địa chỉ*.
- **Module vào/ra:** Nhìn từ bên trong, module vào/ra có chức năng tương tự như bộ nhớ: gồm có hai hoạt động là đọc và ghi.Thêm vào đó, một module vào/ra có thể điều khiển nhiều thiết bị ngoại vi, mỗi thiết bị được giao tiếp với máy tính thông qua một *cổng*, cổng này sẽ được đánh một *địa chỉ* duy nhất (ví dụ: $0, 1, \dots, M - 1$). Các đường dẫn *dữ liệu ngoài* để giao tiếp dữ liệu với thiết bị ngoại vi này. Ngoài ra, module vào/ra có thể gửi các *tín hiệu ngắn* tới bộ xử lý.
- **Bộ xử lý:** Bộ xử lý đọc *lệnh* và *dữ liệu*, ghi dữ liệu ra sau khi xử lý và sử dụng các *tín hiệu điều khiển* để điều khiển hoạt động chung của hệ thống. Nó cũng nhận *tín hiệu ngắn* từ module vào/ra.

Tùy các loại dữ liệu cần trao đổi giữa các module ở trên, cấu trúc kết nối được thiết kế phải đảm bảo hỗ trợ các loại truyền dữ liệu sau đây:

- **Bộ nhớ đến bộ vi xử lý:** Bộ xử lý đọc lệnh hoặc một đơn vị dữ liệu từ bộ nhớ.
- **Bộ xử lý đến bộ nhớ:** Bộ xử lý ghi một đơn vị dữ liệu vào bộ nhớ.



Hình 3.14 Các module máy tính

- **Vào/ra đến bộ vi xử lý:** Bộ xử lý đọc dữ liệu từ một thiết bị ngoại vi thông qua một module vào/ra.
- **Bộ xử lý đến Vào/ra:** Bộ vi xử lý gửi dữ liệu đến thiết bị ngoại vi.
- **Vào/ra đến hoặc đi từ bộ nhớ:** Đối với trường hợp này, một module vào/ra được phép trao đổi dữ liệu trực tiếp với bộ nhớ, không qua bộ vi xử lý, sử dụng cơ chế truy cập bộ nhớ trực tiếp – DMA.

Nhiều năm qua, có một số cấu trúc kết nối được thử nghiệm. Phổ biến nhất là (1) cấu trúc bus, cấu trúc đa bus và cấu trúc kết nối điểm-điểm kết hợp truyền dữ liệu dạng gói. Phần còn lại của chương chúng tôi sẽ trình bày về các cấu trúc này.

3.4. KẾT NỐI BUS

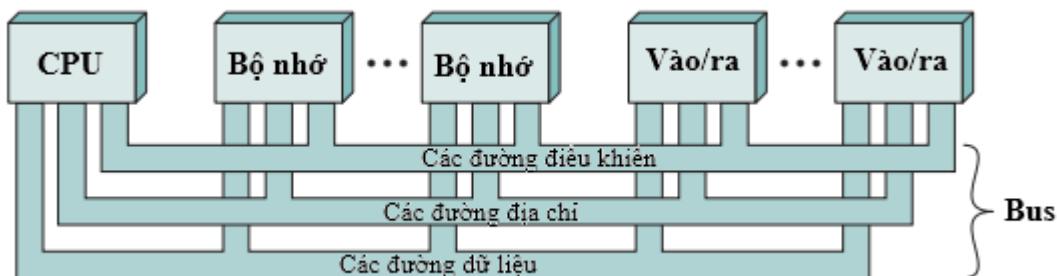
Bus là một đường kết nối giữa hai hoặc nhiều thiết bị. Đặc điểm chính của bus là nó là một **phương tiện truyền dẫn chia sẻ**: nhiều thiết bị kết nối với bus, khi một thiết bị gửi tín hiệu trên bus, tất cả các thiết bị khác gắn vào bus đều nhận được. Nếu hai thiết bị cùng truyền dữ liệu cùng một thời điểm, các tín hiệu sẽ bị chồng chéo lên nhau và bị méo. Do đó, tại một thời điểm chỉ một thiết bị có thể truyền dữ liệu thành công.

Thông thường, một bus gồm nhiều đường truyền. Mỗi đường có khả năng truyền các tín hiệu được biểu diễn dưới dạng số nhị phân 1 và 0. Các tín hiệu có thể được truyền nhiều đợt trên một dòng (truyền nối tiếp). Kết hợp nhiều đường trong bus ta có thể truyền nhiều số nhị phân tại cùng một thời điểm (truyền song song). Ví dụ, một khối 8 bit ta có thể truyền đồng thời trong 8 đường bus.

Thông thường, một hệ thống máy tính sẽ có nhiều bus khác nhau cung cấp các đường liên kết thành phần ở cấp độ khác nhau của hệ thống máy tính. Bus kết nối các thành phần chính của máy tính (gồm có: bộ xử lý, bộ nhớ, các module vào/ra) được gọi là bus hệ thống. Các cấu trúc kết nối phổ biến nhất thường sử dụng một hoặc nhiều bus hệ thống.

3.4.1. Cấu trúc bus

Thông thường, một bus hệ thống bao gồm từ năm mươi cho đến hàng trăm đường riêng biệt. Mỗi đường sẽ đảm nhiệm một ý nghĩa hoặc chức năng cụ thể. Mặc dù có rất nhiều thiết kế bus khác nhau, chức năng của các đường bus thường được chia thành ba nhóm như sau: dữ liệu, địa chỉ, điều khiển. Ngoài ra, bus có thể có thêm một số đường cung cấp nguồn cho các module được nối với nó.



Hình 3.15 Sơ đồ kết nối bus

Các **đường dữ liệu** cho phép truyền tải dữ liệu giữa các module của hệ thống. Các đường này tập hợp lại thành **bus dữ liệu**. Bus dữ liệu có thể gồm: 32, 64, 128 đường hoặc nhiều hơn, số đường này được hiểu là *độ rộng* của bus dữ liệu. Vì mỗi đường chỉ có thể truyền 1 bit tại một thời điểm, số lượng đường sẽ quyết định số lượng bit có thể truyền tại một thời điểm. Vậy độ rộng của bus dữ liệu là một yếu tố quan trọng để đánh giá hiệu suất toàn hệ thống. Ví dụ, nếu bus hệ thống có độ rộng 32 bit, chiều dài của lệnh là 64 bit, vậy bộ xử lý cần truy xuất bộ nhớ hai lần trong mỗi chu kỳ lệnh.

Các **đường địa chỉ** được dùng để xác định nguồn và đích của dữ liệu trong bus dữ liệu. Ví dụ, nếu bộ xử lý cần đọc một từ (8, 16, 32 bit) từ bộ nhớ, nó sẽ đặt địa chỉ của từ đó lên đường địa chỉ. Rõ ràng, độ rộng của **bus địa chỉ** quyết định dung lượng tối đa của bộ nhớ.Thêm vào đó, các đường địa chỉ cũng thường được dùng cho địa chỉ các cổng vào/ra. Thông thường, các bit trọng số cao được dùng để xác định một module, các bit trọng số thấp xác định một vị trí trong bộ nhớ hoặc một cổng vào/ra. Ví dụ, bus địa chỉ 8 bit có các giá trị nhỏ hơn 01111111 sẽ tham chiếu đến 128 vị trí trong module bộ nhớ (module 0), các địa chỉ từ 10000000 trở lên tham chiếu đến các thiết bị được nối với một module vào/ra (module 1).

Các **đường điều khiển** được sử dụng để điều khiển việc truy cập vào và việc sử dụng các đường dữ liệu và địa chỉ. Vì các đường dữ liệu và địa chỉ được chia sẻ với tất cả các thành phần của hệ thống nên cần phải có một phương tiện để kiểm soát việc sử dụng của chúng. Tín hiệu điều khiển truyền cả lệnh và các thông tin định thời giữa các module hệ thống. Tín hiệu định thời chỉ định tính hợp lệ của dữ liệu và địa chỉ. Các tín hiệu lệnh cho biết các thao tác cần thực hiện. Các đường điều khiển bao gồm:

- **Ghi bộ nhớ:** ghi dữ liệu trên bus vào vị trí bộ nhớ có địa chỉ tương ứng trong bus địa chỉ
- **Đọc bộ nhớ:** lấy dữ liệu từ vị trí được định địa chỉ và đặt lên bus
- **Ghi vào/ra:** khiến cho dữ liệu trên bus được đưa ra đầu ra của cổng vào/ra được định địa chỉ
- **Đọc vào/ra:** khiến cho dữ liệu từ cổng vào/ra được định địa chỉ được đặt vào bus
- **Truyền ACK:** chỉ thị rằng dữ liệu đã được nhận vào hoặc được đặt lên bus

- **Yêu cầu bus:** chỉ ra rằng một module đang cần chiếm quyền điều khiển bus
- **Cấp quyền bus:** chỉ ra rằng module yêu cầu đã được cấp quyền điều khiển bus
- **Yêu cầu ngắn:** chỉ ra rằng có một ngắn đang chờ
- **ACK ngắn:** xác nhận yêu cầu ngắn đã được công nhận
- **Đồng hồ:** được sử dụng để đồng bộ hóa các hoạt động
- **Reset:** khởi tạo tất cả các module.

Hoạt động của bus như sau: nếu một module muốn truyền dữ liệu đến một điểm khác, nó cần phải làm hai việc: (1) yêu cầu được sử dụng bus và (2) truyền dữ liệu qua bus. Nếu một module muốn yêu cầu dữ liệu từ một module khác, nó phải (1) yêu cầu được sử dụng bus và (2) truyền yêu cầu đến module khác qua các đường điều khiển và địa chỉ thích hợp. Sau đó nó phải đợi module thứ hai gửi dữ liệu trở lại.

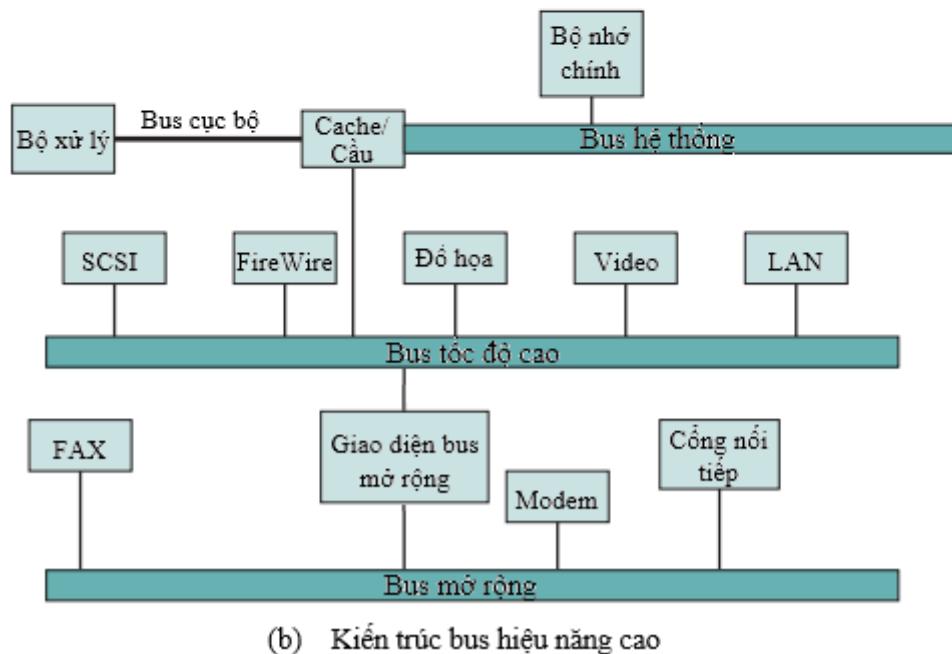
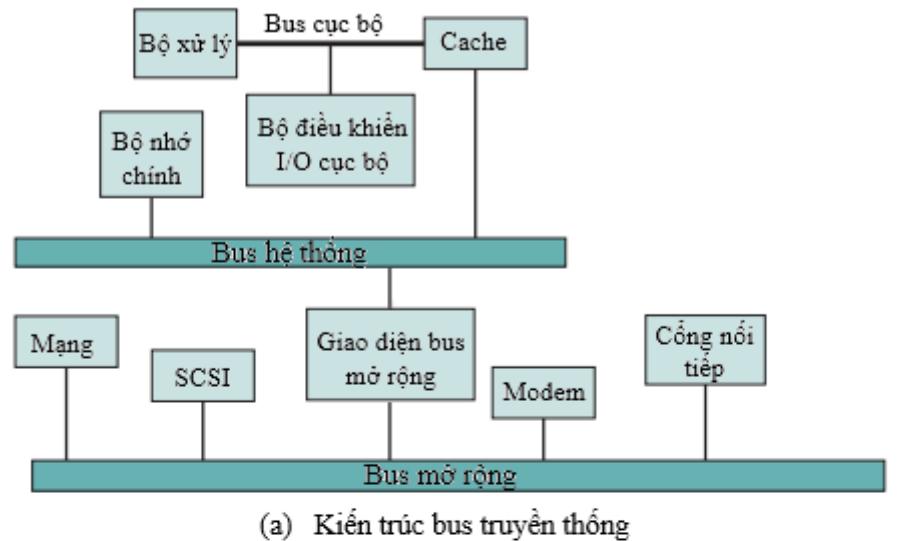
3.4.2. Mô hình phân cấp đa bus

Nếu có một số lượng lớn các thiết bị được nối vào bus, hiệu suất sẽ giảm vì hai nguyên nhân chính sau:

1. Rõ ràng, càng nhiều thiết bị nối vào bus thì chiều dài bus càng tăng gây ra trễ truyền càng lớn. Trễ này quyết định khoảng thời gian cần thiết để các thiết bị phối hợp sử dụng bus. Khi quyền điều khiển bus thường xuyên được chuyển từ một thiết bị này đến thiết bị khác, trễ truyền này sẽ gây ảnh hưởng đáng kể đến hiệu suất.
2. Bus có thể bị hiện tượng nút cổ chai khi nhu cầu truyền dữ liệu vượt quá khả năng của xe buýt. Vấn đề này có thể giải quyết một mức độ nào đó bằng cách tăng tốc độ truyền dữ liệu của nó bằng cách sử dụng bus có độ rộng lớn hơn (ví dụ: tăng bus dữ liệu từ 32 bit lên 64 bit). Tuy nhiên, các thiết bị được kết nối với bus có yêu cầu truyền dữ liệu với tốc độ ngày càng cao (ví dụ: các vi điều khiển video và đồ họa, các giao diện mạng) hệ thống bus ngày càng ít có khả năng đáp ứng kịp.

Theo đó, hầu hết các hệ thống máy tính có cấu trúc kết nối bus thường sử dụng nhiều bus khác nhau (đa bus) dưới dạng hệ thống phân cấp. Hình 3.17a biểu diễn một cấu trúc truyền thông điển hình. Trong đó, một bus cục bộ kết nối bộ vi xử lý với bộ nhớ cache và có thể hỗ trợ một hoặc nhiều thiết bị cục bộ. Bộ điều khiển cache kết nối bộ nhớ cache với cả bus cục bộ và bus hệ thống. Trong các hệ thống hiện đại, bộ nhớ cache thường được đặt trên cùng chip bộ xử lý nên bus bên ngoài không cần thiết. Như sẽ trình bày trong Chương 4, việc sử dụng cấu trúc cache sẽ làm giảm số lần bộ vi xử lý truy cập vào bộ nhớ chính. Do đó, bộ nhớ chính không cần phải nối với bus cục bộ mà chỉ cần nối với bus hệ thống. Bằng cách này, một module vào/ra giao tiếp dữ liệu với bộ nhớ chính qua bus hệ thống sẽ không làm ảnh hưởng tới hoạt động của bộ xử lý.

Các bộ điều khiển vào/ra hoàn toàn có thể được nối trực tiếp vào bus hệ thống. Ngoài ra, một giải pháp hiệu quả khác là sử dụng một hoặc nhiều bus mở rộng. Giao diện bus mở rộng đảm dữ liệu truyền giữa bus hệ thống và bộ điều khiển vào/ra trên bus mở rộng. Bằng cách đó cho phép hệ thống hỗ trợ nhiều thiết bị vào/ra và đồng thời bảo vệ lưu lượng bộ nhớ-bộ xử lý không bị ảnh hưởng bởi lưu lượng vào/ra.



Hình 3.16 Cấu hình bus

Hình 3.17a là một ví dụ điển hình của việc nối các thiết bị vào/ra với bus mở rộng. Các kết nối mạng bao gồm mạng cục bộ (LANs) (ví dụ Ethernet 10 Mbps) và mạng diện rộng (WAN) (mạng chuyển mạch gói). SCSI (giao diện hệ thống máy tính nhỏ) cũng là một loại bus hỗ trợ trao đổi dữ liệu với các ổ

đĩa cục bộ và các thiết bị ngoại vi khác. Cổng nối tiếp hỗ trợ giao tiếp máy in hoặc máy quét.

Kiến trúc bus truyền thông này có hiệu quả khá tốt nhưng lại dàn không đáp ứng được nhu cầu trao đổi dữ liệu ngày càng cao của các thiết bị ngoại vi. Người ta đưa ra một giải pháp là xây dựng một loại bus tốc độ cao (high speed bus) kết hợp với phần còn lại của hệ thống thông qua một thiết bị cầu nối với bus của bộ xử lý. Cấu hình này được gọi là kiến trúc tầng lửng như trong hình 3.17b. Trong đó, bus cục bộ liên kết bộ xử lý với bộ điều khiển cache. Bus hệ thống hỗ trợ bộ nhớ chính. Bộ điều khiển cache tích hợp một **cầu** có nhiệm vụ đệm dữ liệu từ bus tốc độ cao. Các thiết bị vào/ra có yêu cầu truyền dữ liệu tốc độ cao được nối với bus này. Các thiết bị có tốc độ truyền dữ liệu thấp hơn được hỗ trợ bởi bus mở rộng qua một giao diện đệm dung lượng giữa bus mở rộng và bus tốc độ cao.

Ưu điểm của cấu trúc này là bus tốc độ cao cho phép các thiết bị yêu cầu cao tích hợp chặt chẽ hơn với bộ vi xử lý đồng thời không phụ thuộc vào bộ xử lý. Do đó, sự chênh lệch giữa tốc độ bộ xử lý và bus tốc độ cao giảm thiểu. Sự thay đổi trong kiến trúc bộ xử lý không ảnh hưởng đến bus tốc độ cao và ngược lại.

3.4.3. Các thành phần của thiết kế bus

Mặc dù có rất nhiều cấu trúc bus khác nhau, có một số thông số hay thành phần thiết kế cơ bản để phân loại và phân biệt các bus (Bảng 3.2)

Bảng 3.2 Các yếu tố trong thiết kế bus

Loại bus	Độ rộng bus
Chuyên dụng	Địa chỉ
Ghép kênh	Dữ liệu
Phương pháp phân xử	Loại truyền dữ liệu
Tập trung	Đọc
Phân tán	Ghi
Định thời	Đọc thay đổi ghi
Đồng bộ	Đọc sau khi ghi
Bất đồng bộ	Khối

3.4.3.1. Loại bus

Các đường bus có thể chia ra thành hai loại cơ bản: chuyên dụng và ghép kênh. Các đường bus *chuyên dụng* có một chức năng duy nhất vĩnh viễn hoặc nối với một số các thành phần máy tính nhất định.

Ví dụ về bus chuyên dụng là các đường bus địa chỉ hoặc bus dữ liệu. Tuy nhiên, trong một số hệ thống người ta thiết kế cho phép các tín hiệu địa chỉ và dữ liệu có thể truyền chung trên một đường bus bằng cách sử dụng đường điều khiển Address Valid. Nếu dữ liệu trên bus là tín hiệu địa chỉ thì đường Address Valid được kích hoạt. Lúc này, các module sẽ có một khoảng thời gian để sao chép địa chỉ và xác định xem nó có phải địa chỉ của module mình

không. Địa chỉ sao đó được loại bỏ khỏi bus, đường bus lúc này được sử dụng để truyền dữ liệu giữa module truyền và module có địa chỉ vừa được gửi. Phương pháp sử dụng cùng một đường bus cho việc truyền các mục đích khác nhau được gọi *ghép kênh thời gian*.

Ưu điểm của phương pháp ghép kênh thời gian là ta có thể sử dụng ít đường hơn, tiết kiệm không gian và dễ dàng cáp giá thành. Nhược điểm của phương pháp này là mạch điện sẽ phức tạp hơn và dễ dàng hiệu suất cũng sẽ giảm hơn do không thể truyền song song nhiều tín hiệu.

3.4.3.2. Phân xử bus

Trong tất cả các hệ thống đơn giản nhất, nhiều module có thể muốn điều khiển bus cùng một thời điểm. Ví dụ một module vào/ra muốn đọc hoặc ghi dữ liệu trực tiếp từ bộ nhớ mà không cần qua bộ xử lý. Bởi vì tại một thời điểm chỉ có một thiết bị có thể truyền thông tin qua bus nên ta cần một **cơ chế phân xử** để quyết định xem module nào được sử dụng bus. Có rất nhiều phương pháp để thực hiện điều này tuy nhiên các phương pháp này được chia thành hai loại chính: **phân xử tập trung** hoặc **phân xử phân tán**. Với cơ chế tập trung người ta sử dụng một thiết bị phần cứng được gọi là bộ điều khiển bus hay bộ phân xử bus để phân bổ thời gian trên bus. Thiết bị này có thể là một module riêng hoặc một bộ phận của bộ vi xử lý. Với cơ chế phân tán, mỗi module chứa logic điều khiển truy cập (access control logic), các module cùng hoạt động để chia sẻ bus. Mục đích của hai phương pháp phân xử là để chỉ một thiết bị: bộ vi xử lý hoặc module vào/ra chiếm quyền làm chủ bus và bắt đầu việc truyền dữ liệu với một module khác trong hệ thống.

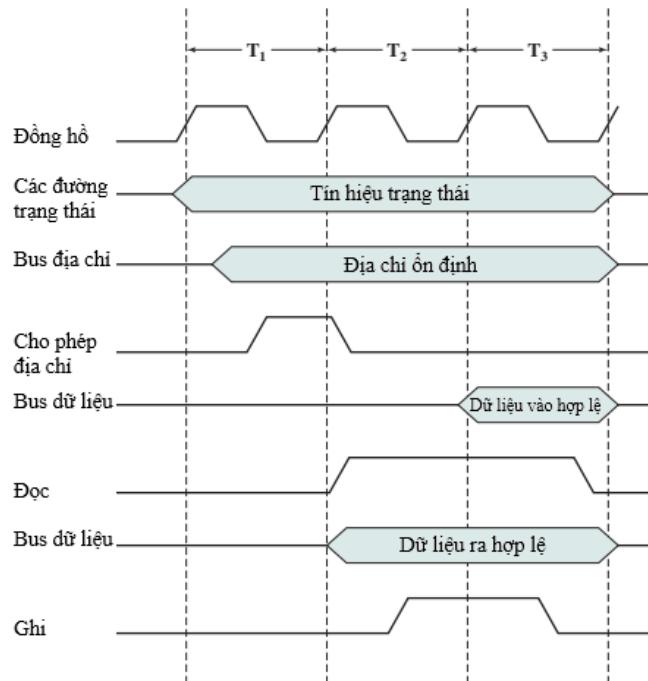
3.4.3.3. Định thời

Định thời được hiểu là cách các sự kiện phối hợp trên bus, bus sử dụng hoặc là định thời đồng bộ hoặc là định thời bất đồng bộ

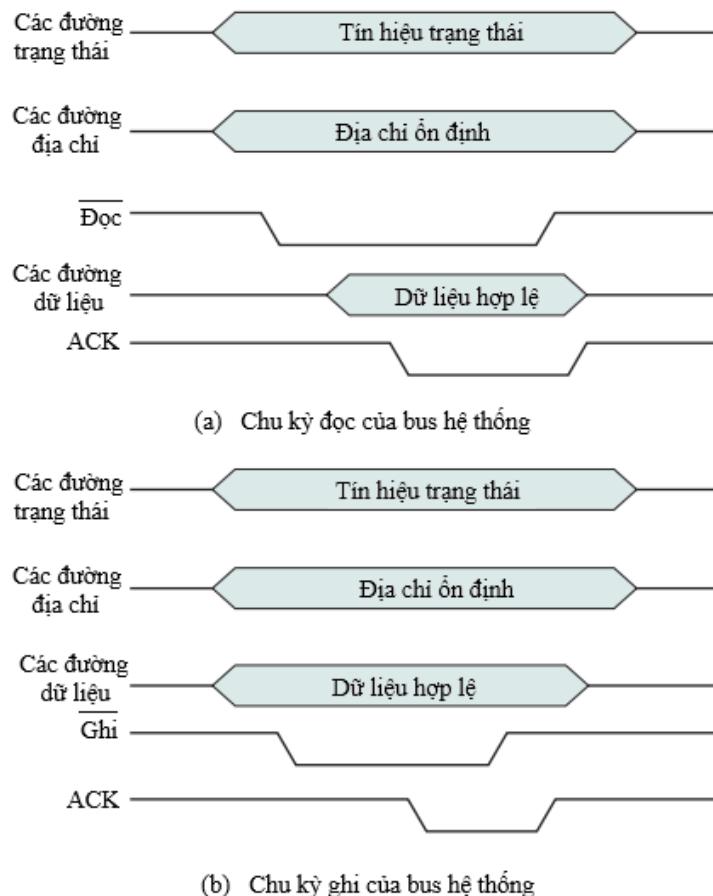
Với **định thời đồng bộ**, việc xuất hiện các sự kiện trên bus được quyết định bởi các xung đồng hồ. Bus có một đường truyền tín hiệu đồng hồ bằng cách truyền một chuỗi liên tục các bit 0, 1 xen kẽ nhau và trong một khoảng thời gian bằng nhau. Một cặp 1-0 được gọi là một chu kỳ đồng hồ hoặc chu kỳ bus. Tất cả các thiết bị nối với bus đều đọc được tín hiệu đồng hồ này. Tất cả các sự kiện truyền trên bus đều được bắt đầu vào thời điểm đầu chu kỳ đồng hồ. Hình 3.18 biểu diễn biểu đồ thời gian cho hoạt động đọc và ghi đồng bộ theo xung đồng hồ. Trong ví dụ này, bộ xử lý đặt địa chỉ bộ nhớ vào bus địa chỉ tại chu kỳ đồng hồ đầu tiên có thể cùng với một vài tín hiệu trạng thái. Khi đường địa chỉ đã ổn định, bộ xử lý sẽ phát ra tín hiệu *cho phép địa chỉ* (address enable). Đối với hoạt động đọc, bộ xử lý sẽ đưa ra lệnh đọc vào đầu chu kỳ thứ hai. Module bộ nhớ nhận biết địa chỉ và đặt dữ liệu vào bus dữ liệu (sau tín hiệu lệnh đọc một chu kỳ). Bộ xử lý đọc dữ liệu từ bus dữ liệu và triệt tiêu tín hiệu đọc. Đối với hoạt động ghi, bộ xử lý đặt dữ liệu vào bus dữ liệu tại đầu chu kỳ thứ hai và đưa ra lệnh ghi sau khi các đường dữ liệu đã ổn định. Module bộ nhớ sao chép thông tin từ bus dữ liệu trong chu kỳ đồng hồ thứ ba

62 Chương 3. Chức năng và kết nối máy tính

Với **định thời không đồng bộ**, các sự kiện sẽ xuất hiện lần lượt trên bus và phụ thuộc vào sự xuất hiện của một sự kiện trước đó. Ví dụ đơn giản như trong Hình 3.19a, bộ xử lý đặt địa chỉ và các tín hiệu trạng thái lên bus. Sau một khoảng thời gian ngắn để các tín hiệu này ổn định, nó phát ra lệnh đọc. Bộ nhớ giải mã địa chỉ và đáp ứng bằng cách đặt dữ liệu lên các đường dữ liệu. Khi các đường dữ liệu đã ổn định, module bộ nhớ gửi tín hiệu xác nhận (ACK) để báo cho bộ xử lý biết dữ liệu đã có trên bus. Khi bộ xử lý đọc dữ liệu xong, nó sẽ loại bỏ tín hiệu đọc, từ đó module bộ nhớ sẽ loại bỏ tín hiệu ACK và dữ liệu. Cuối cùng, sau khi ACK bị xóa bỏ, bộ xử lý sẽ xóa tín hiệu địa chỉ.



Hình 3.17 Hoạt động bus đồng bộ



Hình 3.18 Hoạt động của bus không đồng bộ

Hình 3.19b minh họa một hoạt động ghi không đồng bộ đơn giản. Trong trường hợp này, bộ xử lý đặt dữ liệu lên các đường dữ liệu cùng lúc với tín hiệu trạng thái và địa chỉ lên các đường trạng thái và địa chỉ. Module bộ nhớ sao chép dữ liệu từ các đường dữ liệu và sau đó gửi đi dòng xác nhận (ACK). Bộ xử lý sau đó loại bỏ tín hiệu ghi và module bộ nhớ loại bỏ ACK.

Định thời đồng bộ là phương pháp tương đối đơn giản trong thực hiện thực tế và thử nghiệm. Tuy nhiên nhược điểm là nó ít linh hoạt hơn định thời không đồng bộ. Vì tất cả các thiết bị trên một bus đồng bộ được gắn với một đồng hồ có tốc độ cố định, hệ thống không thể tận dụng những tiến bộ trong hiệu suất của từng thiết bị. Với định thời không đồng bộ, nhiều thiết bị có tốc độ nhanh, chậm khác nhau, sử dụng công nghệ cũ hoặc mới hơn có thể chia sẻ chung một bus.

3.5. KẾT NỐI ĐIỂM-ĐIỂM

Kiến trúc bus chia sẻ là kiến trúc cơ bản được sử dụng để kết nối giữa bộ vi xử lý và các thành phần khác (bộ nhớ, vào/ra, v.v...) trong nhiều thập kỷ. Nhưng với các hệ thống hiện đại ngày nay, kiến trúc kết nối điểm-điểm có xu hướng được sử dụng ngày càng tăng hơn so với kết nối bus.

64 Chương 3. Chức năng và kết nối máy tính

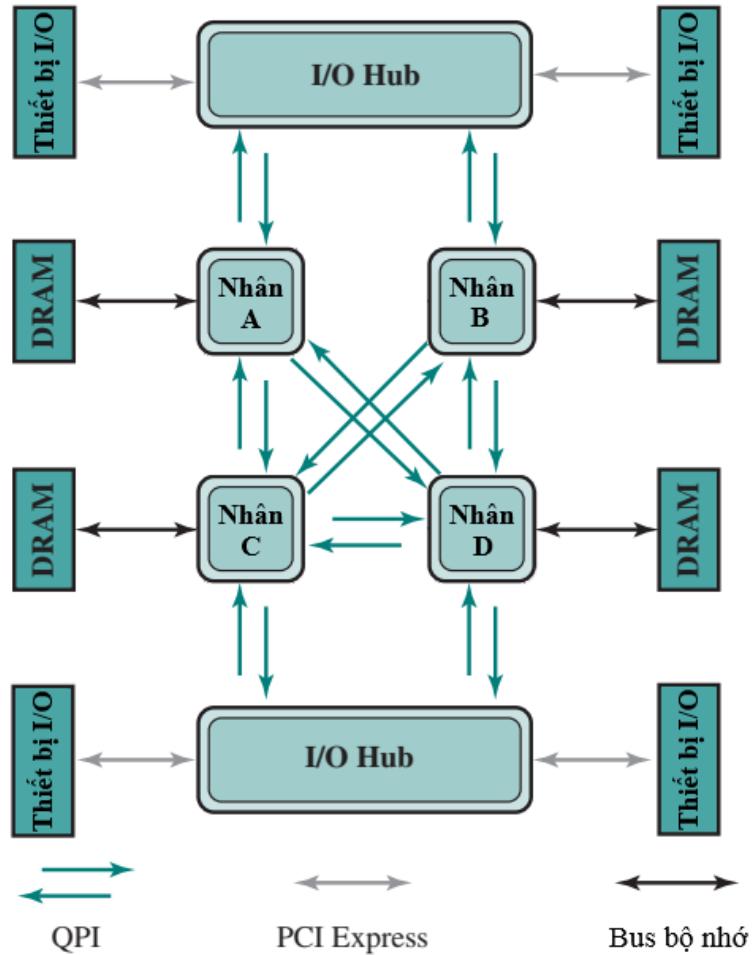
Lý do cơ bản của sự thay đổi này là do tần số tín hiệu đồng hồ của bus đồng bộ ngày càng cao thì việc cấp nguồn trở nên khó khăn. Với tốc độ dữ liệu ngày càng cao, việc thực hiện các chức năng đồng bộ và phân xử cũng trở nên khó khăn.Thêm vào đó, sự ra đời của công nghệ chip đa nhân với nhiều bộ vi xử lý và bộ nhớ tích hợp trên một con chip duy nhất, người ta thấy rằng việc sử dụng một cấu trúc bus chia sẻ thông thường làm tăng thêm khó khăn trong việc tăng tốc độ dữ liệu và giảm độ trễ của bus để theo kịp với tốc độ vi xử lý. So với kết nối bus, kết nối điểm-điểm có độ trễ thấp hơn, tốc độ dữ liệu cao hơn, khả năng mở rộng tốt hơn.

Trong phần này, chúng ta sẽ xét một ví dụ quan trọng của hệ thống kết nối điểm-điểm: **QuickPath Interconnect (QPI)** của Intel, được giới thiệu vào năm 2008.

Sau đây là một số đặc điểm quan trọng của QPI và các cấu trúc kết nối điểm-điểm khác:

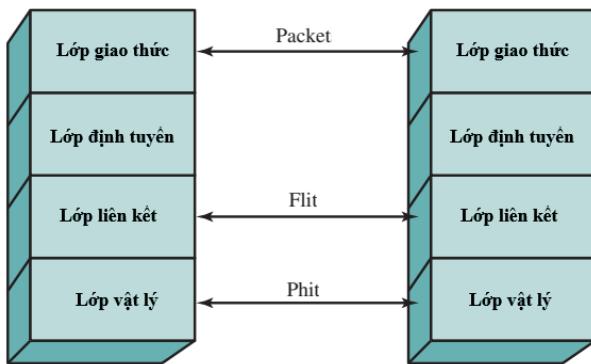
- **Nhiều kết nối trực tiếp:** Các thành phần được kết nối trực tiếp với nhau. Điều này giúp loại bỏ cơ chế phân xử như trong mô hình kết nối bus.
- **Kiến trúc giao thức lớp:** Giống như mô hình phân lớp TCP/IP của mạng máy tính, kết nối QPI sử dụng kiến trúc lớp chứ không phải là sử dụng các tín hiệu điều khiển như trong mô hình kết nối bus.
- **Truyền dữ liệu dạng gói tin:** Dữ liệu không được gửi dưới dạng chuỗi bit thô mà được gửi dưới dạng các gói tin, mỗi gói tin bao gồm một phần tiêu đề điều khiển và một mã kiểm soát lỗi.

Hình 3.20 là một ví dụ điển hình về kết nối QPI trên một máy tính đa nhân. Các liên kết QPI (chỉ thị bằng các mũi tên màu xanh lá cây) tạo thành một mạng lưới chuyển mạch cho phép dữ liệu di chuyển qua mạng. Kết nối QPI trực tiếp có thể được thiết lập giữa hai nhân. Nếu nhân A trong hình 3.20 cần truy cập đến bộ điều khiển bộ nhớ trong nhân D, nó sẽ phải gửi yêu cầu qua các lõi B hoặc C.



Hình 3.19 Cấu hình đa nhân sử dụng QPI

Ngoài ra, QPI có thể kết nối với module vào/ra thông qua một I/O Hub (IOH). Thông thường trong các hệ thống hiện đại, liên kết giữa IOH với bộ điều khiển thiết bị vào/ra sử dụng một công nghệ kết nối được gọi là PCI Express (PCIe) sẽ được đề cập đến phần sau của chương này. IOH có nhiệm vụ chuyển đổi các giao thức, định dạng QPI sang các giao thức, định dạng PCIe và ngược lại. Mỗi nhân cũng kết nối với một module bộ nhớ (các bộ nhớ này thường sử dụng công nghệ bộ nhớ DRAM (bộ nhớ truy cập ngẫu nhiên)) thông qua một bus chuyên dụng.



Hình 3.20 Các lớp QPI

Kiến trúc giao thức QPI gồm bốn lớp, bao gồm (Hình 3.21):

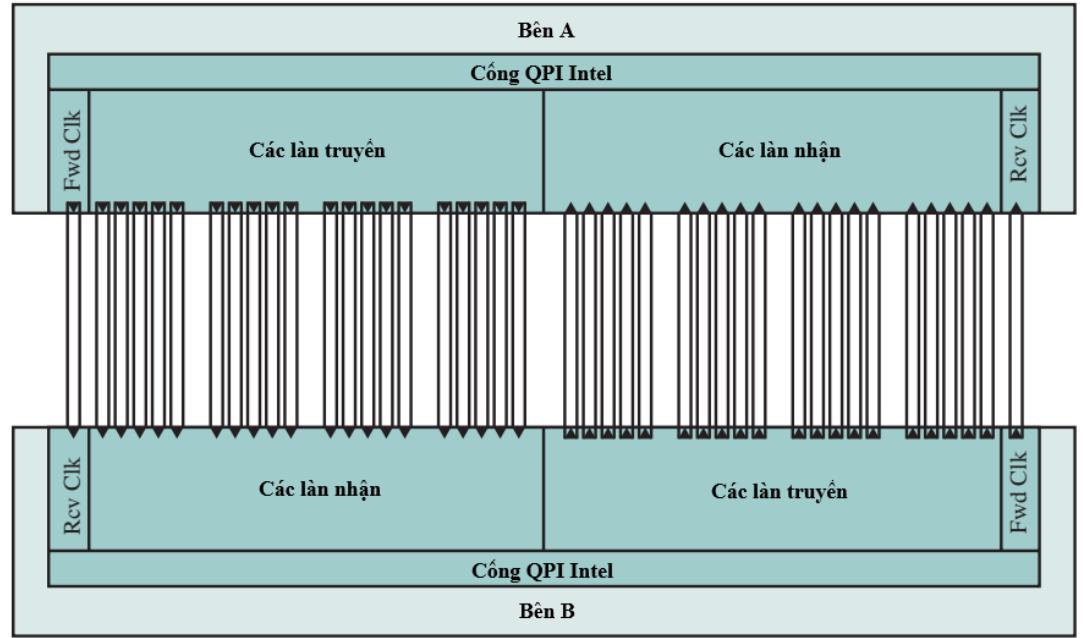
- **Lớp vật lý:** bao gồm các đường dây dẫn, các mạch điện và các logic để truyền tín hiệu cũng như hỗ trợ các tính năng phụ cần thiết trong việc truyền và nhận các số 1 và 0. Đơn vị truyền ở lớp này các gói là 20 bit, được gọi là **Phit**.
- **Lớp liên kết:** Chịu trách nhiệm truyền tin cậy và điều khiển luồng. Đơn vị dữ liệu của lớp Liên kết là một **Flit** 80-bit (flow control unit).
- **Lớp định tuyến:** Cung cấp một framework để truyền các gói dữ liệu
- **Lớp giao thức:** Bộ quy tắc để trao đổi các **gói tin** dữ liệu giữa các thiết bị. Một gói bao gồm một số không đổi các Flit.

3.5.1. Lớp vật lý QPI

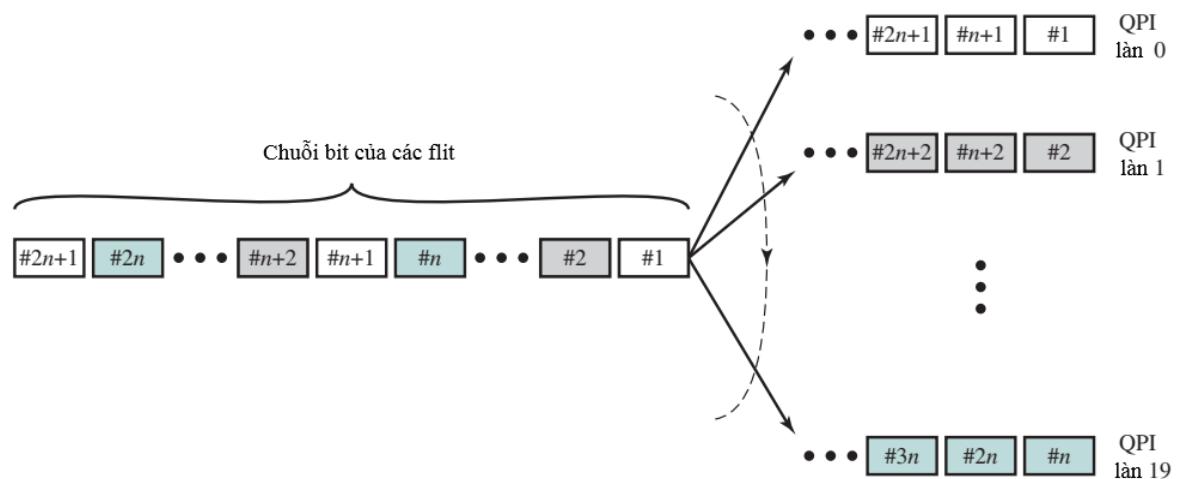
Hình 3.22 minh họa kiến trúc vật lý một cổng QPI. Cổng QPI gồm 84 đường liên kết riêng lẻ được phân thành các nhóm. Mỗi đường truyền dữ liệu gồm hai đường dây truyền mỗi bit tại một thời điểm, cặp dây này được gọi là **làn**. Có 20 làn mỗi hướng (hướng truyền và hướng nhận) cùng với một làn đồng hồ mỗi hướng. Tốc độ tín hiệu hiện nay của kết nối QPI là 6,4 GT/giây (tốc độ truyền/giây). Với 20 bit mỗi làn truyền, tốc độ sẽ lên tới 16 GB/s và với hai chiều chuyên dụng, tổng dung lượng sẽ là 32 GB/s.

Các làn mỗi hướng được chia thành bốn quadrant (mỗi quadrant có 5 làn). Trong một số ứng dụng, kết nối QPI có thể hoạt động ở dung lượng chỉ còn một nửa hoặc một phần tư để giảm mức tiêu thụ điện năng hoặc các lối quay vòng.

Một chức năng khác của lớp vật lý là quản lý việc chuyển đổi giữa các flit 80-bit và các phit 20-bit bằng cách sử dụng một kỹ thuật được gọi là phân bố đa lớp. Các flit được xem như là một chuỗi bit, được phân phối vào các làn theo cơ chế xoay vòng (bit thứ nhất đến làn thứ nhất, bit thứ hai đến làn thứ hai, ...), như minh họa trong hình 3.23. Phương pháp này cho phép QPI đạt được tốc độ dữ liệu rất cao nhờ cách truyền song song giữa hai cổng.



Hình 3.21 Giao diện vật lý của kết nối QPI Intel



Hình 3.22 Phân bổ bit vào các làn QPI

3.5.2. Lớp liên kết QPI

Lớp liên kết thực hiện hai chức năng chính: điều khiển luồng và kiểm soát lỗi. Các chức năng này được thực hiện trên các flit, mỗi flit gồm 72-bit chứa thông tin và một mã sửa lỗi 8-bit thường được gọi là mã CRC (Cyclic Redundancy Check) (mã này chúng tôi sẽ thảo luận trong chương 5).

Thông tin chứa trong flit có thể là dữ liệu hoặc thông báo. Flit dữ liệu chứa các bit dữ liệu cần truyền giữa các nhân hoặc giữa một nhân với IOH. Flit thông báo được sử dụng cho các chức năng như điều khiển luồng, điều khiển lỗi, và liên kết cache.

Chức năng điều khiển luồng hết sức cần thiết để đảm bảo rằng thực thể *QPI* gửi không vượt quá khả năng xử lý dữ liệu và giải phóng bộ đệm cho dữ

liệu mới của thực thể *QPI nhận*. Để điều khiển luồng dữ liệu, QPI sử dụng một **credit scheme**. Khi khởi tạo, phía gửi đưa ra một số **credit** để gửi các flit đến bên nhận. Mỗi khi một flit được gửi đến người nhận, bên gửi sẽ giảm bộ đếm **credit** đi một đơn vị. Khi nào bộ đếm của bên nhận được giải phóng hết, nó sẽ trả lại cho bên gửi một **credit**. Do đó, bên nhận có thể điều khiển được tốc độ truyền dữ liệu qua kết nối QPI.

Trong một số trường hợp, một bit truyền ở lớp vật lý bị thay đổi trong quá trình truyền có thể do lỗi trên đường truyền hoặc một số hiện tượng khác. Chức năng kiểm soát lỗi ở lớp liên kết có khả năng phát hiện và phục hồi các lỗi bit tránh trường hợp các lỗi này ảnh hưởng đến các lớp cao hơn. Cơ chế kiểm soát lỗi với một luồng dữ liệu từ hệ thống A đến hệ thống B gồm các bước như sau:

1. Mỗi flit 80-bit chứa một trường CRC 8-bit. CRC là một mã được tính toán dựa trên 72 bit còn lại. Về mặt truyền dẫn, khi một chuỗi bit được gửi đi, bên A sẽ tính CRC cho mỗi 72b của một flit và chèn giá trị đó vào flit.
2. Khi bên B nhận được một flit, bên B tính CRC cho 72-bit dữ liệu nhận được và so sánh giá trị này với giá trị CRC được gửi đến. Nếu hai giá trị CRC không khớp, flit đã bị lỗi.
3. Khi B phát hiện ra một lỗi, nó sẽ gửi một yêu cầu đến A để truyền lại flit lỗi. Tuy nhiên, vì A truyền liên tục nên trước khi nhận được yêu cầu truyền lại flit lỗi có thể nó đã truyền thêm một số flit khác. Vì vậy, khi nhận được yêu cầu truyền lại, A phải sao lưu và truyền lại flit lỗi cùng với các flit tiếp theo sau đó.

3.5.3. Lớp định tuyến QPI

Lớp định tuyến được sử dụng để xác định đường đi mà một gói tin sẽ đi qua trong hệ thống kết nối. Một bảng định tuyến mô tả các đường đi có thể mà một gói tin có thể đi theo. Với các mạng kết nối nhỏ số lượng các đường đi ít nên các bảng định tuyến khá đơn giản. Đối với các hệ thống lớn hơn, bảng định tuyến phức tạp hơn cho phép việc định tuyến linh hoạt hơn và định tuyến lại lưu lượng truy cập tùy thuộc vào (1) cách các thiết bị kết nối với nhau, (2) tài nguyên hệ thống được phân chia và (3) các sự kiện tin cậy.

3.5.4. Lớp giao thức QPI

Trong lớp này, gói tin được định nghĩa là đơn vị truyền. Nội dung gói tin được chuẩn hóa với một số tính linh hoạt được đáp ứng được các yêu cầu khác nhau. Chức năng quan trọng của lớp này là giao thức gắn kết bộ nhớ cache, đảm bảo rằng các giá trị của bộ nhớ chính được ánh xạ trong nhiều cache thống nhất với nhau.

3.6. PCI EXPRESS

PCI (Bộ ghép nối thiết bị ngoại vi – Peripheral Component Interconnect) là một bus tốc độ cao, độc lập với bộ xử lý, có thể hoạt động như một tầng lõng hoặc bus ngoại vi. So với bus thông thường, PCI cho hiệu

suất hệ thống phù hợp với các thiết bị ngoại vi tốc độ cao (ví dụ: các adapter đồ họa, card mạng và bộ điều khiển ổ cứng). Tuy nhiên, do yêu cầu trao đổi dữ liệu ngày càng lớn của các thiết bị ngoại vi, người ta đã đưa ra một phiên bản mới gọi là **PCI Express (PCIe)**. PCIe cũng như QPI là một kết nối điểm-điểm nhằm thay thế các kết nối bus thông thường.

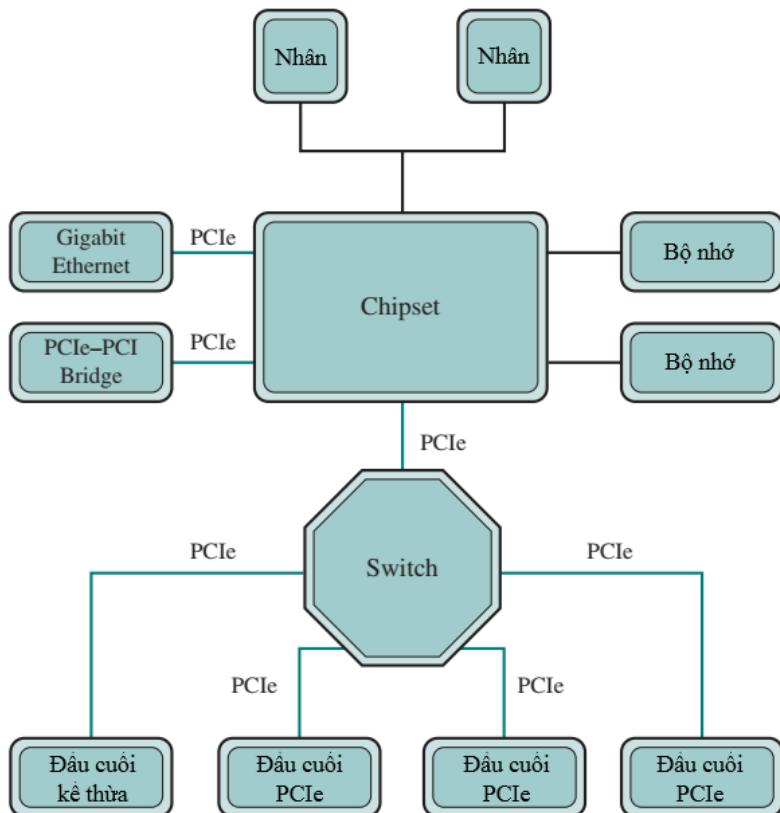
Một yêu cầu quan trọng của PCIe là có phải dung lượng cao để hỗ trợ nhu cầu của các thiết bị I/O tốc độ ngày càng cao như card Gigabit Ethernet. Ngoài ra, chuẩn này phải có khả năng hỗ trợ các luồng dữ liệu thời gian thực như các ứng dụng video-on-demand, .v.v... Ngoài ra, các nền tảng ngày nay cũng phải tương thích với nhiều luồng dữ liệu tốc độ cao cùng một thời điểm. Các hệ thống không thể xử lý tất cả các luồng dữ liệu với tốc độ như nhau - ví dụ với luồng dữ liệu thời gian thực, dữ liệu trễ sẽ không có giá trị, được coi là không có dữ liệu. Mỗi loại dữ liệu sẽ được gắn thẻ riêng sao cho một hệ thống I/O có thể nhận biết và thực hiện việc ưu tiên lưu lượng của nó.

3.6.1. Kiến trúc vật lý và logic của PCI

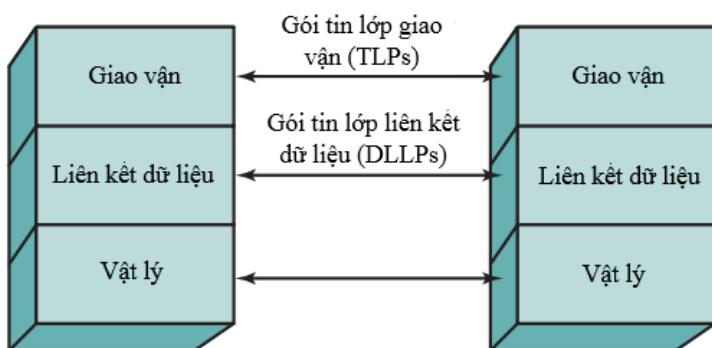
Trong Hình 3.24 là cấu hình điển hình của PCIe. Một thiết bị ở chính giữa là một *chipset* hoặc *host bridge* kết nối bộ vi xử lý và hệ thống bộ nhớ với một hoặc nhiều thiết bị chuyển mạch (*Switch*) PCIe. Chipset làm việc như một thiết bị đệm để xử lý sự khác biệt về tốc độ dữ liệu giữa bộ điều khiển I/O, bộ nhớ và bộ xử lý. Ngoài ra, nó còn có nhiệm vụ chuyên đổi giữa các định dạng PCIe với các định dạng tín hiệu của bộ xử lý và bộ nhớ. Chipset PCIe thường hỗ trợ nhiều cổng PCIe, một số thiết bị có thể gắn trực tiếp vào các cổng đó hoặc có thể qua một switch quản lý nhiều luồng PCIe.

Cũng giống như với QPI, giao tiếp PCIe cũng có kiến trúc giao thức. Kiến trúc giao thức PCIe bao gồm các lớp sau (Hình 3.25):

- Vật lý: Lớp này bao gồm các đường dây dẫn mang tín hiệu, các mạch và các logic hỗ trợ các tính năng cần thiết trong việc truyền và nhận các bit 1 và 0.



Hình 3.23 Cấu hình kết nối sử dụng PCIe



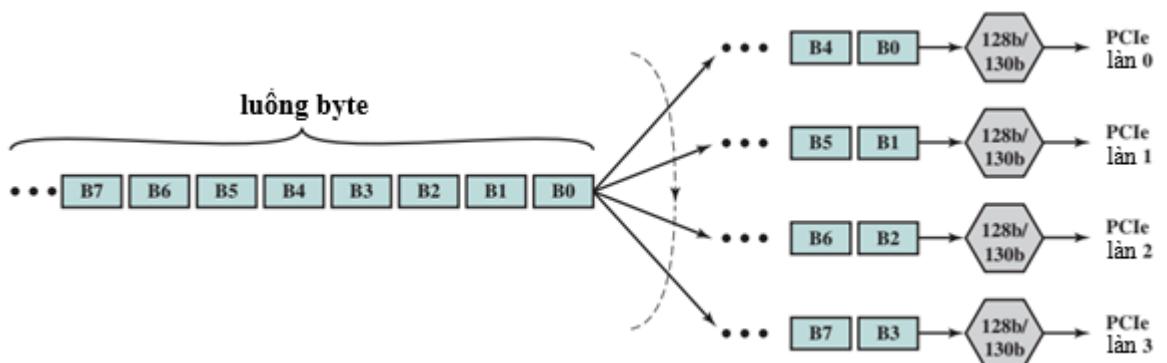
Hình 3.24 Các lớp giao thức PCIe

- **Liên kết dữ liệu:** Chịu trách nhiệm truyền tải tin cậy (kiểm soát lỗi) và điều khiển luồng. Các gói tin dữ liệu sinh ra và xử lý bởi lớp này được gọi là DLLP - gói dữ liệu lớp liên kết (DLLP – Data Link Layer Packets).
- **Giao vận:** Tạo ra và thu nhận các gói dữ liệu được sử dụng để thực hiện các cơ chế truyền dữ liệu và điều khiển luồng của các gói tin giữa hai thành phần trên một kết nối. Các gói tin dữ liệu được tạo ra, thu nhận và xử lý bởi lớp giao vận được gọi là Gói TL(TLP - Transaction Layer Packets).

3.6.2. Lớp vật lý PCIe

Tương tự như QPI, PCIe là một kiến trúc kết nối điểm-điểm. Mỗi cổng PCIe bao gồm một số làn truyền tin hai chiều (lưu ý rằng trong QPI, mỗi làn chỉ truyền được một chiều). Truyền theo từng hướng trong một đường bằng các tín hiệu khác nhau qua một cặp dây. Một cổng PCI có thể bao gồm 1, 4, 6, 16 hoặc 32 làn. Trong phần này, chúng tôi sẽ trình bày đặc tả PCIe 3.0 được giới thiệu vào cuối năm 2010.

Cũng như với QPI, PCIe sử dụng kỹ thuật phân phối đa làn. Hình 3.26 minh họa cho một cổng PCIe gồm bốn làn. Dữ liệu được phân phối cho bốn làn theo từng byte theo cơ chế xoay vòng. Tại mỗi làn vật lý, dữ liệu được lưu đệm lại và xử lý mỗi 16 byte (128 bit) tại một thời điểm. Mỗi khối 128 bit được mã hóa thành một từ mã 130 bit và truyền đi; đây được gọi là mã hóa 128b / 130b. Do đó, tốc độ dữ liệu hiệu dụng của một làn riêng lẻ giảm xuống bằng 128/130.



Hình 3.25 Phân bố đa làn PCIe

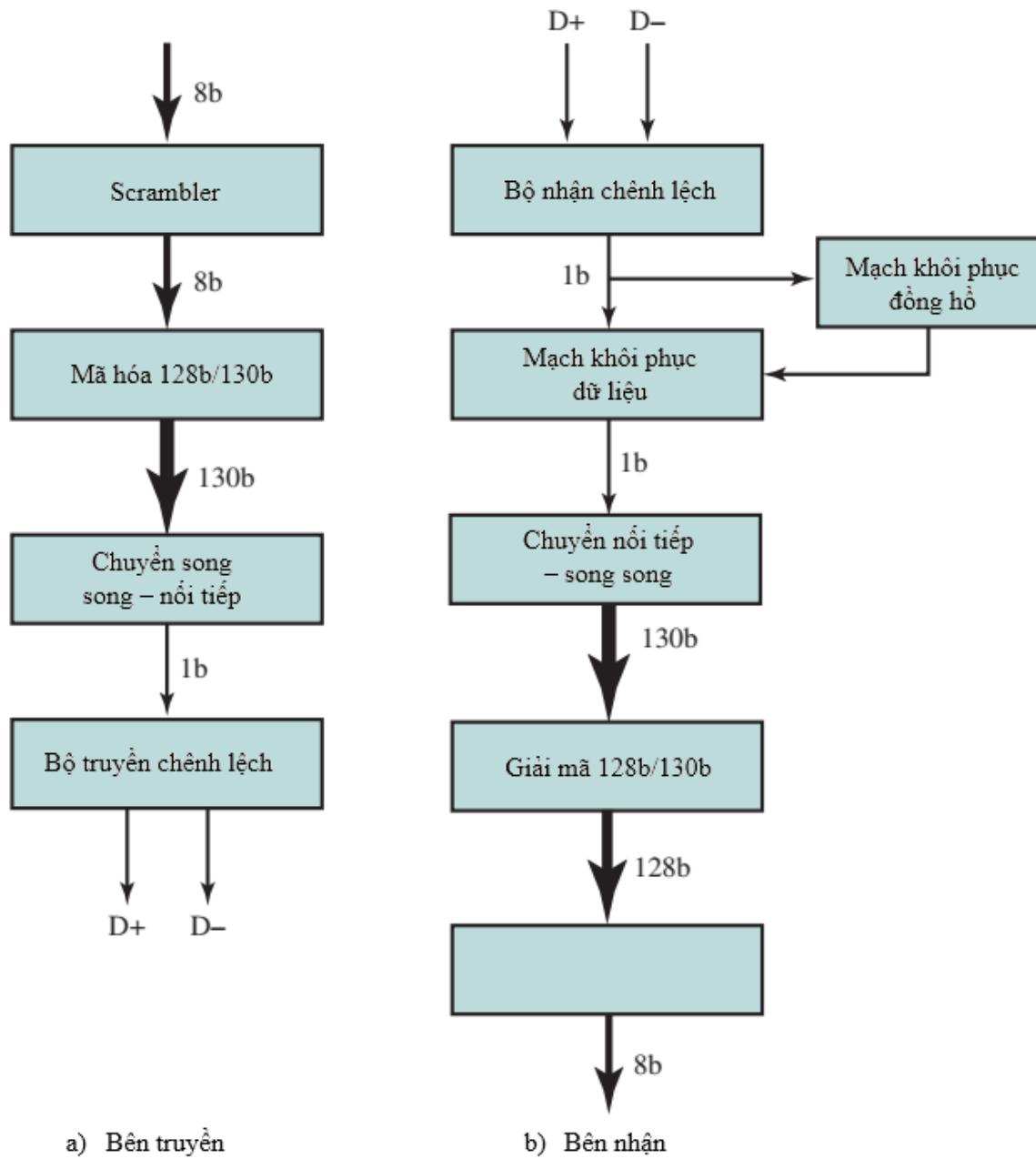
Về nguyên nhân phải thực hiện mã hóa 128b/130b như sau: không giống như QPI, PCIe không sử dụng đường đồng hồ để đồng bộ hóa luồng bit. Nghĩa là, đường đồng hồ không được sử dụng để xác định điểm bắt đầu và điểm kết thúc của mỗi bit đến. Tuy nhiên, bên nhận vẫn phải đồng bộ với bên phát để biết được thời điểm bắt đầu và kết thúc của mỗi bit tín hiệu vì nếu có bất kỳ độ lệch nhỏ nào giữa đồng hồ của bên truyền và bên nhận thì sẽ gây ra trường hợp lỗi bit đặc biệt trong trường hợp một chuỗi dài chỉ toàn các bit 0 hoặc 1 được truyền. Lúc đó, đầu ra sẽ có một mức điện áp trong một thời gian dài rất dễ gây mất đồng bộ giữa hai bên. Một giải pháp phổ biến và được sử dụng trong PCIe 3.0 là kỹ thuật scrambling.

Một kỹ thuật khác hỗ trợ đồng bộ là mã hóa: thêm vào một số bit vào chuỗi bit cần truyền. Đối với PCIe 3.0, mỗi khối 128 bit đầu vào được ánh xạ vào một khối 130-bit bằng cách thêm một header 2-bit với mục đích đồng bộ hóa. Giá trị của header là 10 nếu khối trên là khối dữ liệu và 01 nếu 128b đó là khối thông tin lớp liên kết.

Hình 3.27 mô tả việc truyền dữ liệu sử dụng mã hóa và scrambling. Dữ liệu truyền được đưa vào một scrambler. Đầu ra của scrambler sau đó được đưa vào bộ mã hóa 128b/130b, tại đó nó được lưu đệm lại và ánh xạ vào một

khối 130-bit. Khối này sau đó đi qua một bộ chuyển đổi song song-nối tiếp và được truyền một bit một tại một thời điểm.

Tại máy thu, đồng hồ được đồng bộ với dữ liệu được gửi đến để phục hồi chuỗi bit. Chuỗi bit này sau đó đi qua một bộ chuyển đổi nối tiếp-song song để tạo ra các khối 130-bit. Mỗi khối được truyền qua bộ giải mã 128b/130b để khôi phục 128-bit gốc ban đầu, cuối cùng đi qua bộ descrambler để khôi phục khôi bit gốc.



Hình 3.26 Sơ đồ khái quát quá trình truyền và nhận PCIe

3.6.3. Lớp giao vận PCIe

Lớp giao vận (TL) nhận các yêu cầu đọc và viết từ phần mềm phía trên lớp giao vận – TL và tạo ra các gói tin yêu cầu để truyền tới đích thông qua

lớp liên kết. Hầu hết các giao dịch sử dụng kỹ thuật *giao dịch phân chia*: việc truyền tin được phân ra thành hai giai đoạn là yêu cầu và hoàn thành. Ban đầu một bên gửi đi gói tin yêu cầu được gửi đi, bên nhận yêu cầu sẽ trả lời lại bằng cách gửi lại gói tin được gọi là *gói hoàn thành*. Gói hoàn thành được gửi lại chỉ khi dữ liệu đã sẵn sàng và nó có một định danh duy nhất cho phép các gói tin này định hướng đến bên yêu cầu một cách chính xác. Như vậy, khác với hệ thống bus thông thường là khi có một giao dịch truyền nhận giữa hai bên thì cần phải chiếm hoàn toàn bus, kỹ thuật phân chia, việc yêu cầu và hoàn thành có thể hoàn toàn tách riêng nhau do đó giữa chúng luồng dữ liệu PCIe khác có thể sử dụng đường truyền. Định dạng gói TL hỗ trợ địa chỉ bộ nhớ 32-bit và địa chỉ bộ nhớ mở rộng 64-bit.

CÁC LOẠI GIAO DỊCH VÀ KHÔNG GIAN BỘ NHỚ Lớp giao vận hỗ trợ bốn không gian địa chỉ:

- **Bộ nhớ:** Không gian bộ nhớ bao gồm bộ nhớ chính của hệ thống và các thiết bị I/O PCIe.
- **I/O:** Không gian địa chỉ này được sử dụng cho các thiết bị PCI kề thửa, với các dải địa chỉ bộ nhớ dành riêng được sử dụng cho các thiết bị I/O kề thửa.
- **Cáu hình:** Không gian địa chỉ này cho phép TL đọc/ghi các thanh ghi cấu hình kết hợp với các thiết bị I/O.
- **Bản tin:** Không gian địa chỉ này dành cho tín hiệu điều khiển liên quan đến ngắt, xử lý lỗi, và quản lý nguồn điện.

ĐỊNH DẠNG GÓI TIN TLP Gói tin lớp giao vận được minh họa trong Hình 3.28a gồm các trường sau:

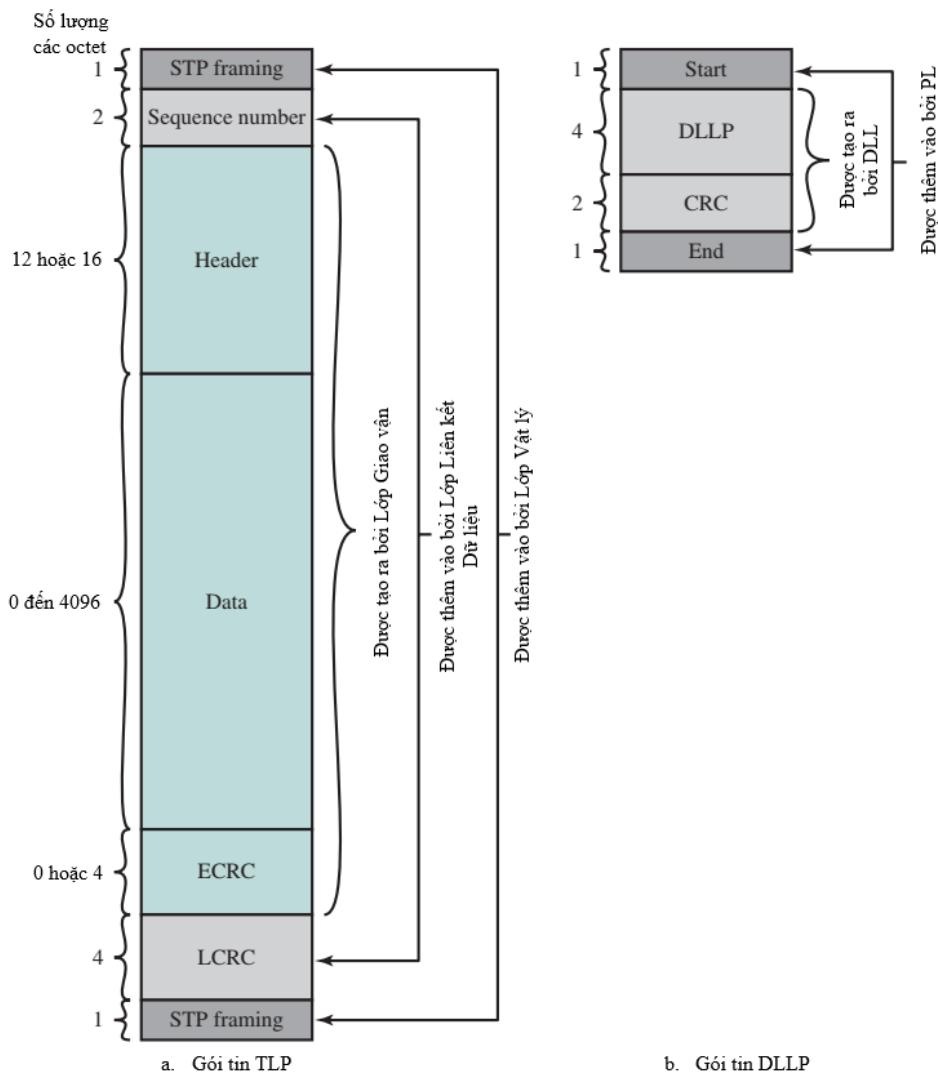
- **Header:** Trường header chứa các thông tin cần thiết để bên nhận để xử lý gói tin bao gồm các thông tin định tuyến.
- **Dữ liệu:** Trường dữ liệu có kích thước lên đến 4096 byte. Một số TLP không có trường dữ liệu.
- **ECRC:** Trường CRC cho phép lớp TL bên nhận kiểm tra lỗi trong phần Header và Dữ liệu của TLP.

Ví dụ trường Header của một bản tin yêu cầu giao dịch bộ nhớ như trong Hình 3.29. Các trường màu xanh lá cây là các trường có trong tất cả các Header của các bản tin khác. Các trường R là các trường dành riêng cho mục đích sử dụng trong tương lai. Ngoài ra, các trường khác như sau:

- **Length:** Chiều dài trường dữ liệu tính theo double word (DW), trong đó một DW = 4 byte.
- **Attr:** Gồm hai bit: *relaxed ordering bit* và *snoop bit*. Nếu *relaxed ordering bit* được thiết lập thì một giao dịch có thể được hoàn thành trước các giao dịch khác đang xếp hàng chờ. Nếu *snoop bit* được thiết lập nghĩa là không cần thống nhất cache với gói tin TLP này.

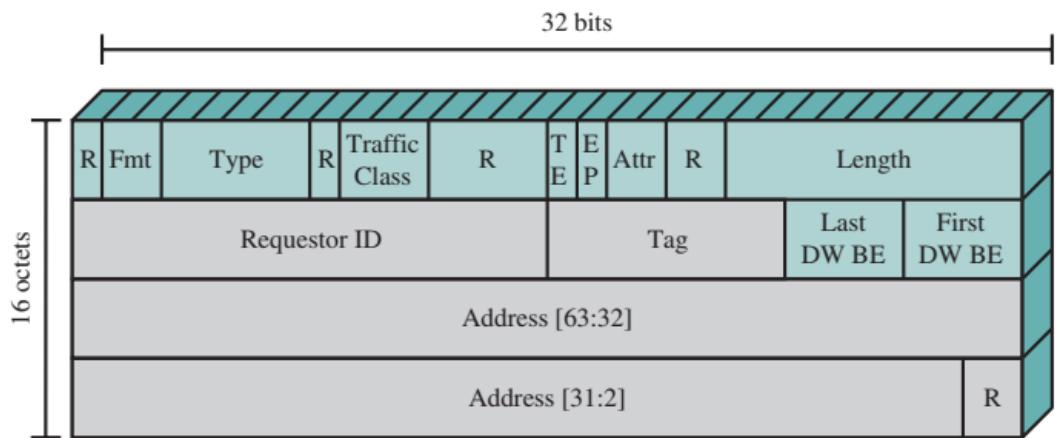
74 Chương 3. Chức năng và kết nối máy tính

- **EP:** Nếu bit này được thiết lập nó cho biết dữ liệu trong TLP này được coi là dữ liệu không hợp lệ mặc dù giao dịch vẫn hoàn thành bình thường.
- **TE:** Được thiết lập trong trường hợp có trường ECRC trong bản tin TLP.
- **Traffic Class:** Gồm 3-bit được gán cho một luồng lưu lượng để cho phép PCIe ưu tiên dịch vụ.
- **Type, Format:** Hai trường này có tất cả 7 bit xác định loại giao dịch, kích thước trường Header, và cho biết có hay không trường dữ liệu.
- **First DW Byte Enables:** Gồm bốn bit cho biết liệu byte tương ứng trong DW đầu tiên có hợp lệ hay không.



Hình 3.27 Định dạng đơn vị dữ liệu giao thức PCIe

- **Last DW Byte Enables:** Gồm bốn bit cho biết liệu byte tương ứng trong DW cuối cùng có hợp lệ hay không.



Hình 3.28 Định dạng yêu cầu bộ nhớ TLP

Các trường còn lại (màu ghi) trong Hình 3.29 cho ta biết Header này dành cho giao dịch yêu cầu bộ nhớ. Requestor ID xác định bên gửi yêu cầu đến bộ nhớ để nơi nhận có thể gửi phản hồi lại. Tag là một số được bên gửi yêu cầu gán cho giao dịch; bên hoàn thành phải đưa cả trường Tag này vào trong bản tin trả lời để bên gửi yêu cầu khớp yêu cầu và phản hồi. Trường Address chỉ ra địa chỉ bộ nhớ bắt đầu đọc.

3.6.4. Lớp liên kết dữ liệu PCIe

Mục đích của lớp liên kết dữ liệu PCIe là đảm bảo việc truyền tin cậy các gói dữ liệu qua kết nối PCIe. DLL tham gia vào việc định dạng các TLP và cũng truyền các DLLP.

GÓI DLLP Gói tin lớp liên kết dữ liệu. Có ba nhóm DLLP quan trọng được sử dụng để quản lý kết nối: các gói điều khiển luồng, các gói quản lý nguồn và các gói TLP ACK và NAK.

XỬ LÝ GÓI TIN TLP Lớp DLL thêm hai trường vào bản tin TLP (Hình 3.29): một số 16-bit và trường CRC lớp liên kết 32-bit. Khác với bản tin TLP chỉ được xử lý tại điểm cuối, hai trường thêm vào này được xử lý tại tất cả các nút trên đường truyền từ điểm đầu đến điểm cuối.

3.7. CÂU HỎI

1. Các nhóm chức năng chung được xác định bởi các lệnh máy tính?
2. Liệt kê và định nghĩa ngắn gọn các trạng thái có thể để thực hiện một lệnh.
3. Liệt kê và định nghĩa ngắn gọn hai phương pháp xử lý nhiều ngắt.
4. Những dạng truyền nào phải có hỗ trợ của cấu trúc kết nối (ví dụ: bus)?
5. Ưu điểm của kiến trúc đa bus so với kiến trúc đơn bus?
6. Liệt kê và định nghĩa ngắn gọn các lớp giao thức QPI.
7. 3.7 Liệt kê và định nghĩa ngắn gọn các lớp giao thức PCIe.

Chương 4. BỘ NHỚ CACHE

Mặc dù khái niệm có vẻ đơn giản nhưng bộ nhớ máy tính lại là thành phần đa dạng nhất về công nghệ, tổ chức, hiệu suất và giá thành trong hệ thống máy tính. Không có một công nghệ riêng lẻ nào là tối ưu để đáp ứng mọi yêu cầu về bộ nhớ cho một hệ thống máy tính. Kết quả là, hệ thống máy tính thường được trang bị một hệ thống bộ nhớ phân cấp, một số loại bộ nhớ bên trong hệ thống (truy cập trực tiếp bởi bộ xử lý) và một số bên ngoài (bộ xử lý truy cập được thông qua một mô-đun I/O).

Chương này và chương tiếp theo tập trung vào các thành phần bộ nhớ trong, còn Chương 6 bàn về bộ nhớ ngoài. Ta bắt đầu bằng việc thảo luận các đặc điểm chính của bộ nhớ máy tính. Phần còn lại của chương trình bày về một thành phần thiết yếu trong tất cả các hệ thống máy tính hiện đại: bộ nhớ cache.

4.1. TỔNG QUAN HỆ THỐNG BỘ NHỚ MÁY TÍNH

4.1.1. Các đặc tính của hệ thống bộ nhớ

Để trình bày về bộ nhớ máy tính dễ dàng hơn, ta phân loại các hệ thống bộ nhớ dựa theo các đặc tính chính của chúng. Các đặc tính quan trọng nhất được liệt kê trong Bảng 4.1.

Bảng 4.1 Đặc tính quan trọng của hệ thống bộ nhớ máy tính

Vị trí	Hiệu suất
Bên trong (thanh ghi, cache, bộ nhớ chính)	Thời gian truy cập
Bên trong (đĩa quang, đĩa từ, băng)	Chu kỳ nhở
Dung lượng	Tốc độ truyền tải
Số byte	Loại vật lý
Số từ	Bán dẫn
Đơn vị truyền	Từ
Word	Quang học
Block	Quang từ
Phương pháp truy cập	Tính chất vật lý
Tuần tự	Khả biến/Bất biến
Trực tiếp	Xóa được/Không xoá được
Ngẫu nhiên	Tổ chức
Kết hợp	Mô-đun bộ nhớ

Thuật ngữ **vị trí** trong Bảng 4.1 đề cập đến việc bộ nhớ nằm bên trong hay bên ngoài máy tính. Bộ nhớ trong thường được hiểu tương đương với bộ nhớ chính. Nhưng còn có những dạng bộ nhớ trong khác. Bộ xử lý cần có bộ nhớ cục bộ của nó, ở dạng thanh ghi (ví dụ, xem Hình 2.3). Hơn nữa, như chúng ta sẽ thấy, khói điều khiển của bộ xử lý cũng có thể yêu cầu bộ nhớ trong của riêng nó. Hai loại bộ nhớ trong này sẽ được trình bày ở các chương sau. Cache là một dạng khác của bộ nhớ trong. Bộ nhớ ngoài bao gồm các thiết bị

lưu trữ ngoài, chẳng hạn như đĩa và băng, có thể được bộ xử lý truy cập tới thông qua bộ điều khiển I/O.

Một đặc tính rõ ràng của bộ nhớ là **dung lượng**. Đối với bộ nhớ trong, dung lượng thường được biểu diễn theo byte (1 byte = 8 bit) hoặc từ. Độ dài từ phổ biến là 8, 16 và 32 bit. Dung lượng bộ nhớ ngoài thường được biểu diễn theo byte.

Một khái niệm liên quan khác là **đơn vị truyền tải**. Đối với bộ nhớ trong, đơn vị truyền tải bằng số đường dây điện đi vào và đi ra khỏi bộ nhớ. Nó có thể bằng độ dài từ, nhưng thường là lớn hơn, chẳng hạn 64, 128 hoặc 256 bit. Để hiểu rõ hơn, hãy xem xét ba khái niệm liên quan của bộ nhớ trong:

- **Word:** Đơn vị "tự nhiên" của tổ chức bộ nhớ. Kích thước của word thường bằng số bit được sử dụng để biểu diễn một số nguyên và bằng độ dài lệnh. Tuy nhiên, có nhiều trường hợp ngoại lệ. Ví dụ: CRAY C90 (mẫu siêu máy tính CRAY cũ) có độ dài từ 64 bit nhưng biểu diễn số nguyên bằng 46 bit. Kiến trúc Intel x86 có nhiều độ dài lệnh khác nhau còn kích thước word bằng 32 bit.
- **Đơn vị địa chỉ:** Trong một số hệ thống, đơn vị địa chỉ chính là word. Tuy nhiên, nhiều hệ thống định địa chỉ ở mức byte. Tổng quát, nếu địa chỉ dài A bit thì số lượng đơn vị địa chỉ $N = 2^A$.
- **Đơn vị truyền tải:** Đối với bộ nhớ chính, đơn vị truyền tải là số bit được đọc hoặc ghi vào bộ nhớ tại một thời điểm. Đơn vị truyền tải không nhất thiết phải bằng từ hoặc đơn vị địa chỉ. Đối với bộ nhớ ngoài, dữ liệu thường được truyền theo từng block (block lớn hơn nhiều so với một từ).

Phương pháp truy cập đơn vị dữ liệu cũng là một đặc điểm phân biệt các loại bộ nhớ khác nhau. Có những phương pháp truy cập sau:

- **Truy cập tuần tự:** Bộ nhớ được tổ chức thành các đơn vị dữ liệu, được gọi là bản ghi. Việc truy cập phải thực hiện theo một dãy tuần tự. Thông tin địa chỉ đã lưu được sử dụng để phân biệt các bản ghi và hỗ trợ cho quá trình **truy cập**. Phương pháp này sử dụng cơ chế đọc-ghi chia sẻ, tức là phải di chuyển từ vị trí hiện tại đến vị trí mong muốn, bỏ qua các bản ghi trung gian. Do đó, thời gian để truy cập tới một bản ghi tùy ý là rất biến động. Băng là một ví dụ sử dụng truy cập tuần tự.
- **Truy cập trực tiếp:** Sử dụng cơ chế đọc-ghi chia sẻ. Tuy nhiên, mỗi block hoặc bản ghi có một địa chỉ duy nhất dựa trên vị trí thực tế. Truy cập được thực hiện bằng cách truy cập trực tiếp sẽ tìm đến một vùng lân cận, đếm, hoặc chờ cho tới khi đến được vị trí mong muốn. Thời gian truy cập cũng là biến động. Đĩa là một ví dụ sử dụng truy cập trực tiếp.
- **Truy cập ngẫu nhiên:** Mỗi vị trí trong bộ nhớ có cơ chế định địa chỉ riêng. Thời gian truy cập tới một vị trí xác định là không đổi và không phụ thuộc vào chuỗi các truy cập trước đó. Do đó, một vị trí bất kỳ có thể được lựa chọn ngẫu nhiên, được định địa chỉ và truy cập trực tiếp. Bộ nhớ chính và một số hệ thống cache là truy cập ngẫu nhiên.
- **Kết hợp:** Đây là một kiểu bộ nhớ truy cập ngẫu nhiên cho phép so sánh các vị trí bit mong muốn trong một word để tìm ra cặp giống nhau và thực hiện việc này đồng thời trên tất cả các word. Vì vậy, một word được lấy ra dựa trên một phần nội

dung của nó chứ không dựa trên địa chỉ của nó. Giống như bộ nhớ truy cập ngẫu nhiên thông thường, mỗi vị trí có cơ chế định địa chỉ riêng; thời gian truy cập không đổi, không phụ thuộc vị trí hoặc các mẫu truy cập trước. Bộ nhớ cache có thể sử dụng kiểu truy cập kết hợp.

Theo quan điểm của người dùng, hai đặc điểm quan trọng nhất của bộ nhớ là dung lượng và **hiệu suất**. Ba tham số hiệu suất được sử dụng là:

- **Thời gian truy cập (độ trễ):** Đối với bộ nhớ truy cập ngẫu nhiên, đó là thời gian cần thiết để thực hiện một thao tác đọc hoặc ghi, tức là thời gian từ lúc bộ nhớ nhận được một địa chỉ đến lúc dữ liệu được lưu lại hoặc sẵn sàng để sử dụng. Đối với bộ nhớ không phải truy cập ngẫu nhiên, đó là thời gian cần thiết để cơ chế đọc-ghi đến được vị trí mong muốn.
- **Chu kỳ bộ nhớ:** Khái niệm này chủ yếu áp dụng cho bộ nhớ truy cập ngẫu nhiên và bằng thời gian truy cập cộng với thời gian cần thiết trước khi có thể bắt đầu truy cập thứ hai. Lưu ý rằng chu kỳ bộ nhớ là khái niệm liên quan đến hệ thống bus, không liên quan đến bộ xử lý.
- **Tốc độ truyền tải:** Đây là tốc độ truyền dữ liệu vào hoặc ra khỏi một khối nhớ. Đối với bộ nhớ truy cập ngẫu nhiên, tốc độ truyền tải bằng $1/\text{chu kỳ}$.

Bộ nhớ còn được phân biệt theo **kiểu vật lý**. Đó là bộ nhớ bán dẫn, bộ nhớ bề mặt từ (đĩa, băng), bộ nhớ quang học và bộ nhớ quang tử.

Một số **đặc tính vật lý** của lưu trữ dữ liệu là rất quan trọng. Trong bộ nhớ khả biến, thông tin bị phân rã một cách tự nhiên hoặc bị mất đi khi nguồn điện tắt. Trong bộ nhớ bất biến, thông tin một khi đã được ghi thì không bị suy giảm trừ khi có chủ đích thay đổi; không cần năng lượng điện để giữ lại thông tin. Các bộ nhớ bề mặt từ là bất biến. Bộ nhớ bán dẫn (bộ nhớ trên các mạch tích hợp) có thể là khả biến hoặc bất biến. Bộ nhớ không xóa được không thể bị thay đổi, trừ khi phá hủy khối lưu trữ. Bộ nhớ bán dẫn không xóa được được gọi là *bộ nhớ chỉ đọc* (ROM – read-only memory). Tất nhiên, bộ nhớ không xóa được trong thực tế cũng phải là bất biến.

Đối với bộ nhớ truy cập ngẫu nhiên, **tổ chức** là một vấn đề thiết kế chính. Ở đây, *tổ chức* là sự sắp xếp các bit để tạo thành word.

4.1.2. Phân cấp bộ nhớ

Các ràng buộc trong thiết kế bộ nhớ máy tính có thể được tóm gọn bằng ba câu hỏi: Dung lượng bao nhiêu là đủ? Tốc độ ra sao? Giá thành như thế nào?

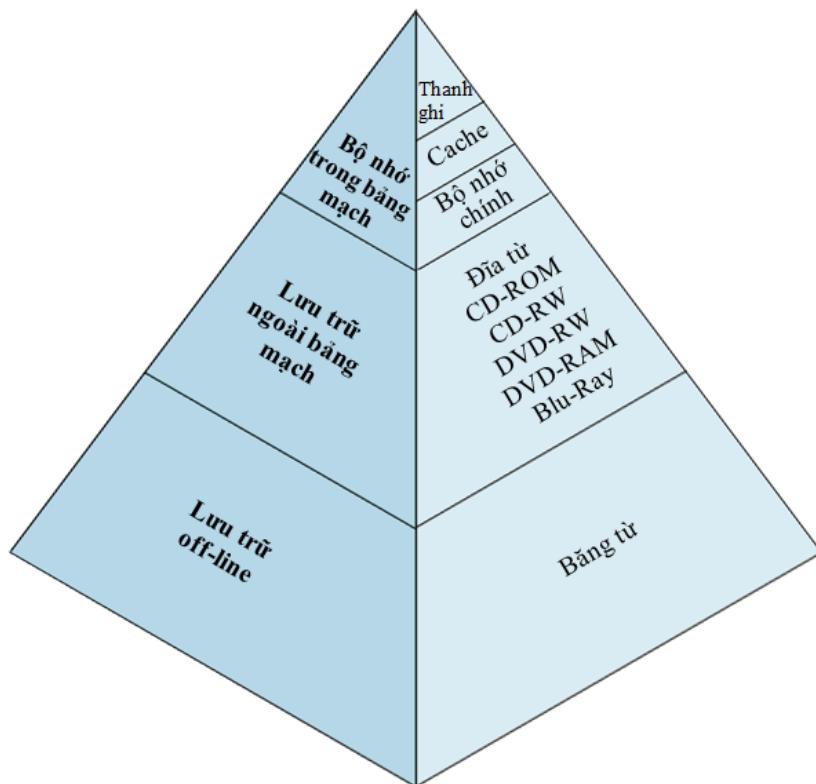
Câu hỏi đầu tiên là câu hỏi mở. Dù dung lượng lớn thế nào cũng sẽ có nhiều ứng dụng được phát triển để sử dụng hết dung lượng đó. Câu hỏi về tốc độ, theo nghĩa nào đó, dễ trả lời hơn. Để đạt được hiệu suất cao nhất, bộ nhớ phải có khả năng theo kịp bộ xử lý. Tức là, khi bộ xử lý đang thực thi lệnh, ta không muốn nó phải tạm dừng để chờ các lệnh hoặc toán hạng. Về câu hỏi cuối, đối với một hệ thống thực tế, giá thành bộ nhớ phải hợp lý cân đối với giá thành của các thành phần khác.

Cần có sự cân bằng giữa dung lượng, thời gian truy cập và giá thành bộ nhớ. Giữa ba đặc điểm này tồn tại mối quan hệ như sau:

- Truy cập nhanh hơn, chi phí cho mỗi bit lớn hơn

- Dung lượng lớn hơn, chi phí cho mỗi bit nhỏ hơn
- Dung lượng lớn hơn, truy cập chậm hơn

Rõ ràng lúc này, người thiết kế gấp phải tình thế tiến thoái lưỡng nan. Người thiết kế muốn sử dụng các công nghệ bộ nhớ cung cấp bộ nhớ dung lượng lớn, bởi vì dung lượng là rất cần thiết và đồng thời chi phí cho mỗi bit sẽ thấp. Tuy nhiên, để đáp ứng yêu cầu về hiệu suất, người thiết kế cần phải sử dụng những bộ nhớ đắt tiền, dung lượng tương đối nhỏ với thời gian truy cập ngắn.



Hình 4.1 Phân cấp bộ nhớ

Giải pháp cho vấn đề này là không phụ thuộc vào một thành phần hoặc công nghệ bộ nhớ riêng lẻ mà sử dụng một **hệ thống phân cấp bộ nhớ**. Một hệ thống phân cấp điển hình được mô tả trong Hình 4.1. Xét từ trên xuống dưới trong hệ thống phân cấp, xu hướng như sau:

- a. Chi phí cho mỗi bit giảm
- b. Dung lượng tăng
- c. Thời gian truy cập dài hơn
- d. Giảm tần suất truy cập bộ nhớ của bộ xử lý

Do đó, bộ nhớ nhỏ hơn, đắt hơn, nhanh hơn được bổ sung bởi bộ nhớ lớn hơn, rẻ hơn, chậm hơn. Mấu chốt của cách tổ chức này xuất phát từ ý (d): giảm tần suất truy cập bộ nhớ. Việc sử dụng hai cấp độ bộ nhớ để giảm thời gian truy cập trung bình về nguyên tắc có thể thực hiện được, nhưng chỉ khi áp dụng các điều kiện từ (a) đến (d). Bằng cách sử dụng các công nghệ khác nhau, sẽ có nhiều hệ thống bộ nhớ thỏa mãn các điều kiện từ (a) đến (c). Điều kiện (d) cũng thường có hiệu lực.

Cơ sở cho tính hợp lệ của điều kiện (d) là một nguyên tắc được gọi là **tính cục bộ của tham chiếu**. Trong quá trình thực thi một chương trình, bộ xử lý tham chiếu các lệnh và dữ liệu trong bộ nhớ. Các tham chiếu bộ nhớ này có xu hướng cụm. Các chương trình thường chứa một số vòng lặp và chương trình con. Mỗi khi vào một vòng lặp hoặc chương trình con, có nhiều tham chiếu lặp đi lặp lại tới một tập hợp nhỏ các lệnh. Tương tự, các thao tác trên bảng và mảng liên quan đến việc truy cập vào một cụm word dữ liệu. Trong một khoảng thời gian dài, các cụm được sử dụng sẽ thay đổi, nhưng trong một khoảng thời gian ngắn, bộ xử lý chủ yếu làm việc với các cụm tham chiếu bộ nhớ nhất định.

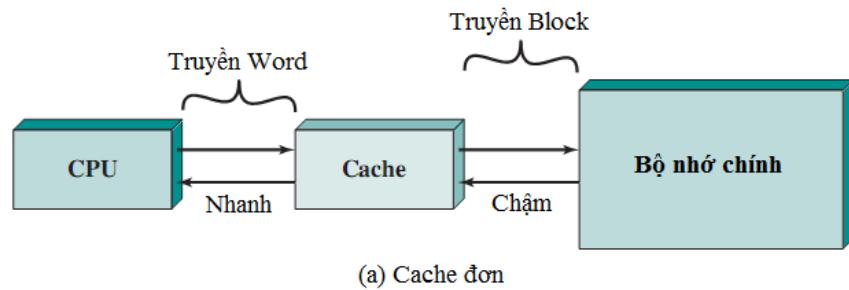
Theo đó, có thể tổ chức dữ liệu trên hệ thống phân cấp sao cho tỷ lệ truy cập vào các cấp thấp hơn nhỏ hơn đáng kể so với các cấp bộ nhớ phía trên. Xét ví dụ hệ thống nhớ hai cấp. Giả sử bộ nhớ cấp 2 chứa tất cả các lệnh và dữ liệu của chương trình. Các cụm hiện đang được tham chiếu có thể được đặt tạm thời ở bộ nhớ cấp 1. Đôi khi, một trong các cụm ở cấp 1 phải được đưa về cấp 2 để dành chỗ cho một cụm mới lên cấp 1. Tuy nhiên, phần lớn các tham chiếu sẽ là tham chiếu tới lệnh và dữ liệu trong bộ nhớ cấp 1.

Nguyên tắc này có thể được áp dụng trên nhiều hơn hai cấp độ bộ nhớ, như hệ thống phân cấp được thể hiện trong Hình 4.1. Loại bộ nhớ nhanh nhất, nhỏ nhất và đắt nhất là các thanh ghi bên trong bộ xử lý. Thông thường, một bộ xử lý chứa vài chục thanh ghi, tuy nhiên, một số khác có chứa hàng trăm thanh ghi. Bộ nhớ chính là hệ thống bộ nhớ chính bên trong máy tính. Mỗi vị trí trong bộ nhớ chính có một địa chỉ duy nhất. Bộ nhớ chính thường được mở rộng bằng bộ nhớ nhỏ hơn, tốc độ cao hơn. Cache thường không hiển thị với lập trình viên hoặc, thậm chí, với cả bộ xử lý. Nó là một thiết bị để di chuyển dữ liệu giữa bộ nhớ chính và thanh ghi của bộ xử lý để cải thiện hiệu suất.

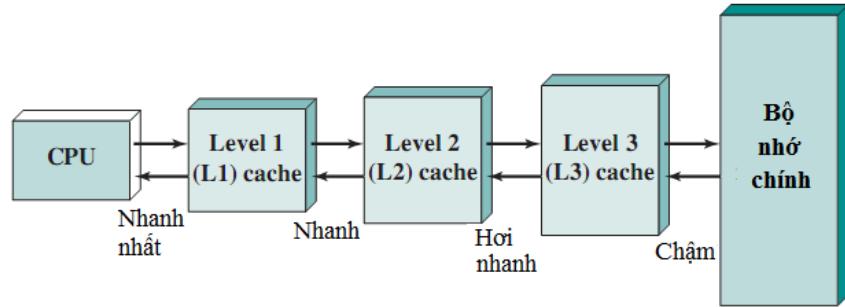
Ba dạng bộ nhớ vừa được mô tả thường là khả biến và sử dụng công nghệ bán dẫn. Việc sử dụng ba cấp độ bộ nhớ lợi dụng thực tế rằng bộ nhớ bán dẫn rất đa dạng chủng loại, khác nhau về tốc độ và chi phí. Dữ liệu được lưu trữ lâu hơn trên thiết bị lưu trữ bên ngoài, trong đó phổ biến nhất là ổ đĩa cứng, đĩa từ, băng và bộ nhớ quang học. Bộ nhớ ngoài bất biến còn được gọi là bộ nhớ thứ cấp hoặc bộ nhớ phụ. Các bộ nhớ này được sử dụng để lưu trữ các chương trình và các tập tin dữ liệu; thường hiển thị với lập trình viên ở mức tập tin và bản ghi. Đĩa cũng được sử dụng để cung cấp một phần mở rộng cho bộ nhớ chính được gọi là bộ nhớ ảo.

4.2. NGUYÊN LÝ BỘ NHỚ CACHE

Bộ nhớ Cache được thiết kế để kết hợp thời gian truy cập của bộ nhớ tốc độ cao, đắt tiền với dung lượng lớn của bộ nhớ tốc độ thấp và rẻ hơn. Điều này được minh họa trong Hình 4.2a. Trong đó, có một bộ nhớ chính tương đối lớn và chậm cùng với một bộ nhớ cache nhỏ hơn, nhanh hơn. Cache chứa bản sao của một phần bộ nhớ chính. Khi bộ xử lý muốn đọc một word của bộ nhớ chính, trước tiên phải kiểm tra để xác định xem word đó có nằm trong cache hay không. Nếu có, word này được truyền từ cache đến bộ xử lý. Nếu không, một block trong bộ nhớ chính, gồm một vài word có chứa word mong muốn, được đọc vào cache và rồi word đó được gửi đến bộ xử lý. Do tính cục bộ của tham chiếu, khi một block dữ liệu được truy xuất vào bộ nhớ cache để đáp ứng một tham chiếu bộ nhớ, có khả năng là sắp tới sẽ có sự tham chiếu tới cùng vị trí bộ nhớ đó hoặc tới các word khác trong block đó.



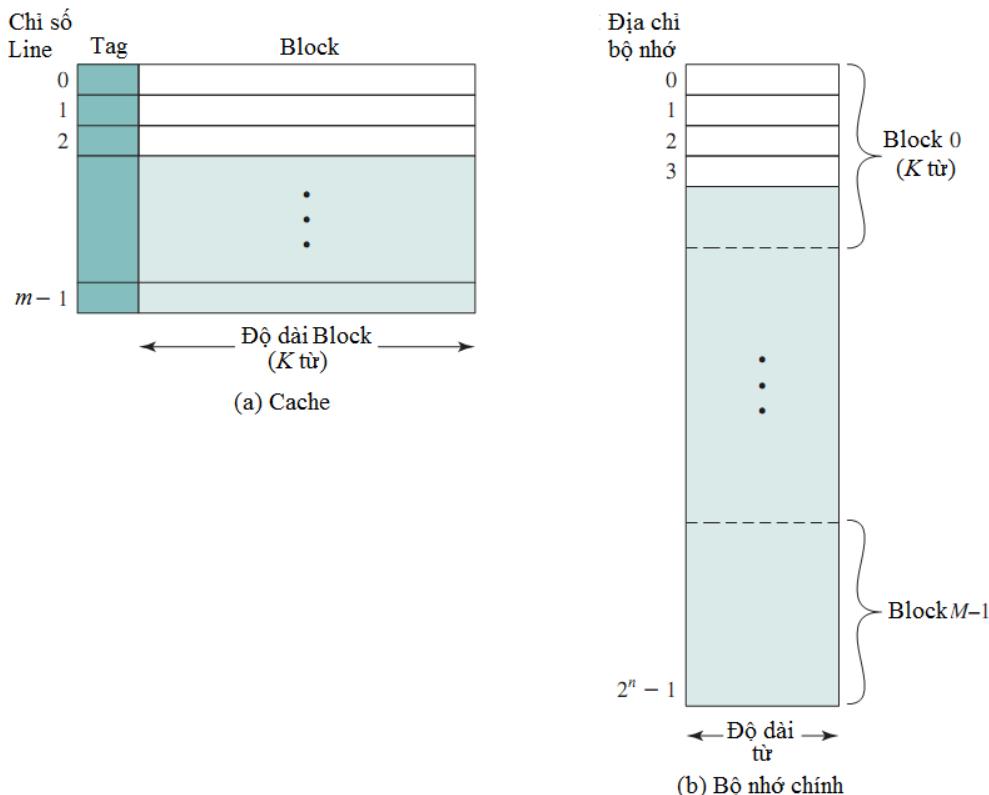
(a) Cache đơn



(b) Tô chức cache ba level

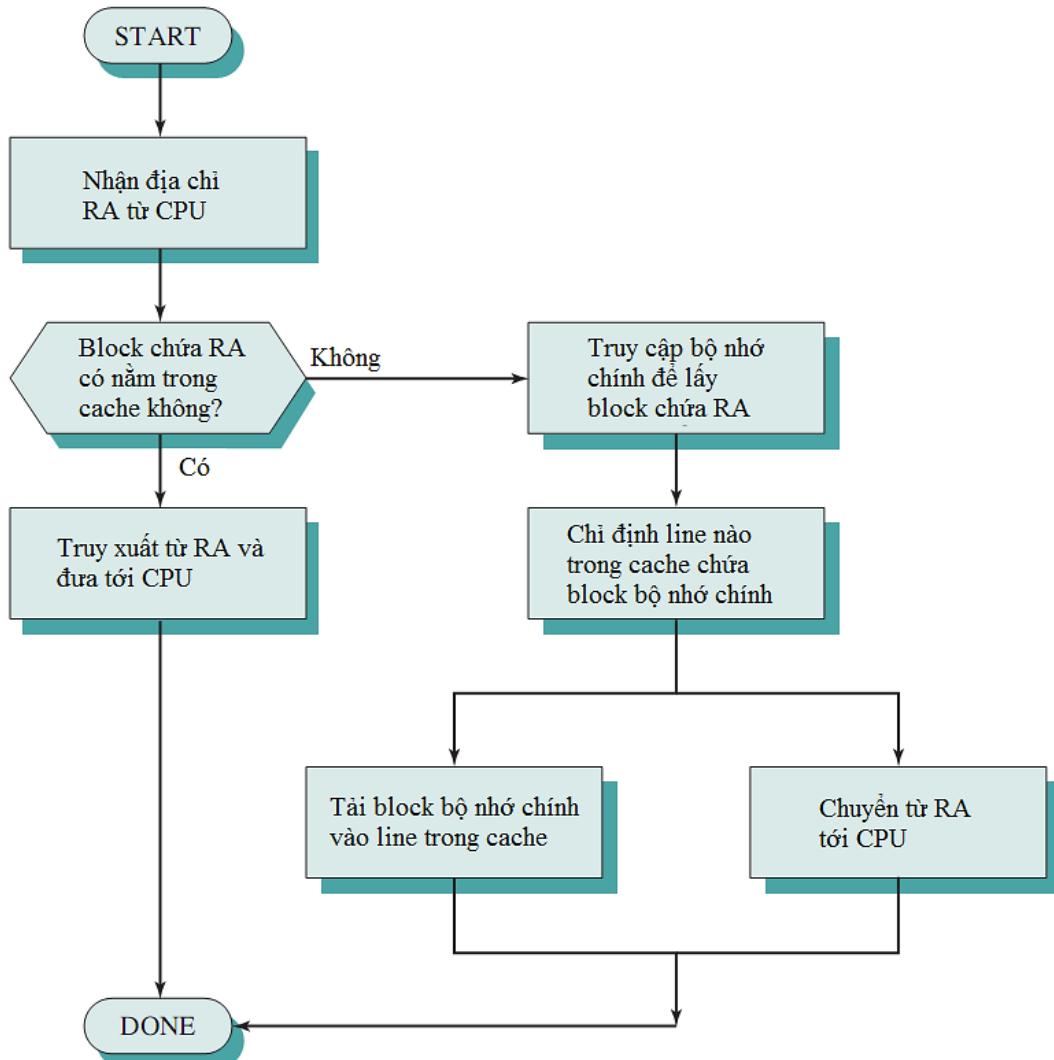
Hình 4.2 Cache và bộ nhớ chính

Hình 4.2b miêu tả việc sử dụng nhiều cấp bộ nhớ cache. Cache L2 chậm hơn và thường lớn hơn cache L1, cache L3 chậm hơn và thường lớn hơn cache L2.



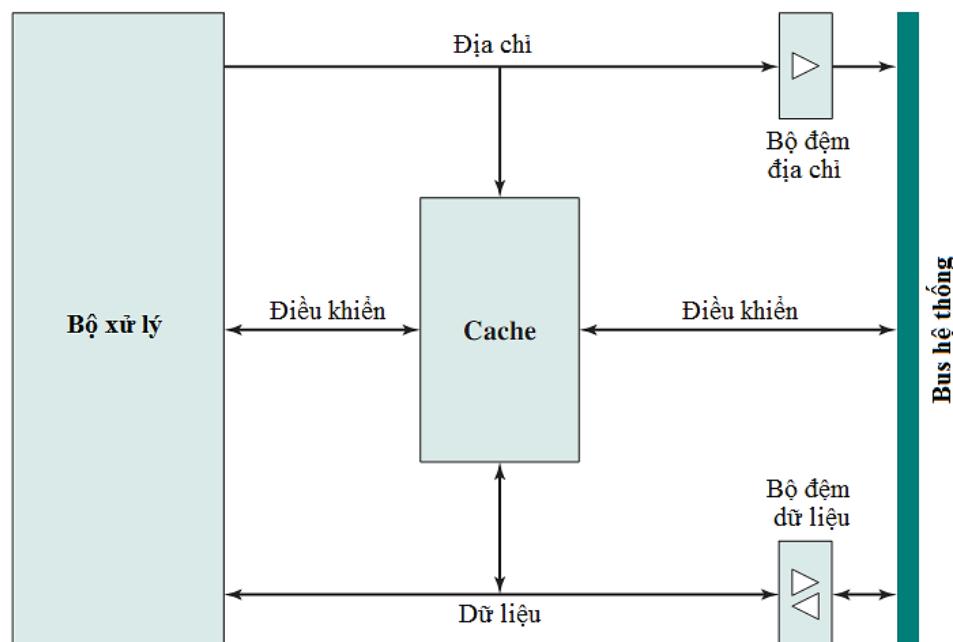
Hình 4.3 Cấu trúc Cache/Bộ nhớ chính

Hình 4.3 mô tả cấu trúc của một hệ thống cache/bộ nhớ chính. Bộ nhớ chính có tới 2^n word, mỗi word có một địa chỉ n bit duy nhất. Vì mục đích ánh xạ, bộ nhớ này được coi là bao gồm K block có độ dài cố định. Tức là, có $M = 2^n/K$ block trong bộ nhớ chính. Bộ nhớ cache bao gồm m block, được gọi là **line**. Mỗi line có K word, cộng với một vài bit của trường tag. Mỗi line còn chứa các bit điều khiển (không được thể hiện), chẳng hạn như một bit cho biết line đó kể từ khi được nạp vào cache đã được thay đổi hay chưa. Độ dài của một line, không bao gồm tag và các bit điều khiển, được gọi là **kích thước line**. Kích thước line nhỏ nhất là 32 bit, với mỗi "word" là một byte; trong trường hợp này kích thước line là 4 byte. Số lượng line ít hơn đáng kể so với số lượng block bộ nhớ chính ($m \ll M$). Tại một thời điểm, một vài tập con của các block bộ nhớ chính nằm trong các line của cache. Nếu một word trong một block của bộ nhớ chính được đọc, block đó được truyền đến một trong các line của cache. Bởi vì có nhiều block hơn line, một line sẽ không được dành riêng vĩnh viễn cho một block cụ thể. Do đó, mỗi line bao gồm một **tag** để xác định block nào đang được lưu trữ. Tag thường là một phần của địa chỉ bộ nhớ chính.



Hình 4.4 Hành động đọc cache

Hình 4.4 mô tả hành động đọc. Bộ xử lý tạo ra địa chỉ đọc (RA - read address) của một word cần đọc. Nếu word đã nằm trong bộ nhớ cache, nó sẽ được gửi đến bộ xử lý. Nếu không, block chứa word đó được nạp vào bộ nhớ cache, và word được gửi đến bộ xử lý. Hình 4.4 cho thấy hai hành động cuối cùng diễn ra song song và phản ánh tổ chức điển hình của bộ nhớ cache như mô tả trong hình 4.5. Trong tổ chức này, cache kết nối với bộ xử lý thông qua các đường dữ liệu, điều khiển và địa chỉ. Đường dữ liệu và địa chỉ còn nối tới bộ đệm dữ liệu và địa chỉ; các bộ đệm này lại nối tới bus hệ thống để từ đó nối tới bộ nhớ chính. Khi có một cache hit, bộ đệm dữ liệu và bộ đệm địa chỉ bị vô hiệu hóa, chỉ có giao tiếp giữa bộ xử lý và cache, không có lưu lượng trên bus hệ thống. Khi có một cache miss, địa chỉ mong muốn được tải lên bus hệ thống và dữ liệu được đưa ra qua bộ đệm dữ liệu tới cả bộ nhớ cache và bộ xử lý. Trong cách tổ chức khác, cache được đặt giữa bộ xử lý và bộ nhớ chính đối với tất cả đường dữ liệu, địa chỉ và điều khiển. Trong trường hợp này, đối với một cache miss, trước tiên word mong muốn được đọc vào cache rồi sau đó chuyển từ cache sang bộ xử lý.



Hình 4.5 Tổ chức cache điển hình

4.3. CÁC YẾU TỐ THIẾT KẾ CACHE

Mặc dù có nhiều cách thực hiện bộ nhớ cache, có một vài yếu tố thiết kế cơ bản được dùng để phân loại và phân biệt các kiến trúc bộ nhớ cache. Bảng 4.2 liệt kê các yếu tố chính đó.

4.3.1. Địa chỉ cache

Hầu hết các bộ xử lý không nhúng, và nhiều bộ vi xử lý nhúng, có hỗ trợ bộ nhớ ảo. Về cơ bản, bộ nhớ ảo là một tiện ích cho phép các chương trình định địa chỉ bộ nhớ từ quan điểm logic, mà không cần quan tâm dung lượng vật lý sẵn có của bộ nhớ chính. Khi bộ nhớ ảo được sử dụng, các trường địa chỉ của lệnh máy chứa các địa chỉ ảo. Để đọc và

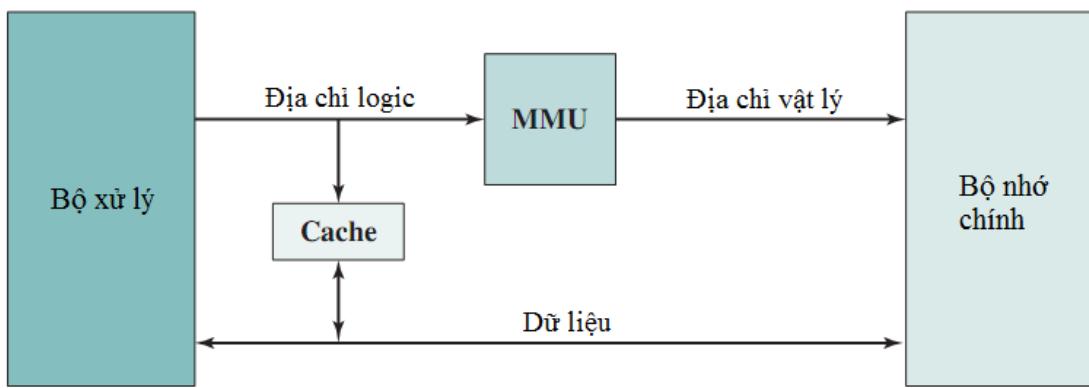
ghi ở bộ nhớ chính, một khối phần cứng quản lý bộ nhớ (MMU – memory management unit) sẽ dịch địa chỉ ảo sang địa chỉ vật lý trong bộ nhớ chính.

Khi địa chỉ ảo được sử dụng, nhà thiết kế hệ thống có thể lựa chọn để đặt cache giữa bộ xử lý và MMU hoặc giữa MMU và bộ nhớ chính (Hình 4.6). **Cache logic**, còn được gọi là **cache ảo**, lưu trữ dữ liệu bằng **địa chỉ ảo**. Bộ xử lý truy cập cache trực tiếp mà không phải đi qua MMU. **Cache vật lý** lưu trữ dữ liệu nhờ sử dụng **địa chỉ vật lý** của bộ nhớ chính.

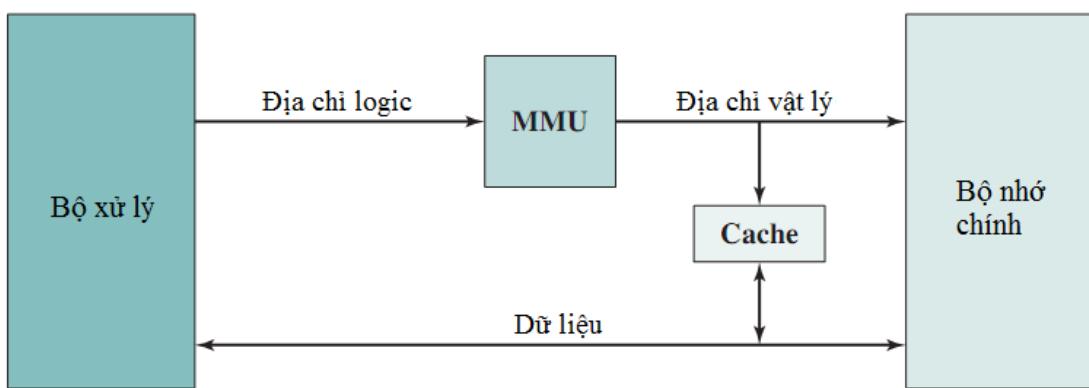
Một ưu điểm rõ ràng của cache logic là tốc độ truy cập bộ nhớ cache nhanh hơn so với tốc độ truy cập cache vật lý, vì cache có thể đáp ứng trước khi MMU thực hiện dịch địa chỉ. Nhược điểm là hầu hết các hệ thống bộ nhớ ảo cung cấp cho các ứng dụng cùng một không gian địa chỉ bộ nhớ ảo giống nhau. Tức là, mỗi ứng dụng đều thấy một bộ nhớ ảo bắt đầu ở địa chỉ 0. Do đó, cùng một địa chỉ ảo trong hai ứng dụng khác nhau tương ứng với hai địa chỉ vật lý khác nhau. Vì vậy, bộ nhớ cache phải được làm sạch hoàn toàn với mỗi chuyển đổi ngữ cảnh của ứng dụng hoặc phải thêm các bit bổ sung vào mỗi line của cache để xác định xem địa chỉ này tương ứng với không gian địa chỉ ảo nào.

Bảng 4.2 Các yếu tố thiết kế cache

Địa chỉ cache	Chính sách ghi
Logic	Write through
Vật lý	Write back
Kích thước cache	Kích thước line
Hàm ánh xạ	Số lượng cache
Trực tiếp	Một cấp hay hai cấp
Kết hợp	Thống nhất hay phân chia
Kết hợp tập hợp	
Thuật toán thay thế	
Ít được sử dụng gần đây nhất (LRU)	
Vào trước ra trước (FIFO)	
Ít được sử dụng nhất (LFU)	
Ngẫu nhiên	



(a) Cache logic



(b) Cache vật lý

Hình 4.6 Cache vật lý và cache logic

4.3.2. Kích thước cache

Chúng ta muốn kích thước cache đủ nhỏ để tổng chi phí trung bình cho mỗi bit gần bằng với chi phí đó khi chỉ sử dụng riêng bộ nhớ chính và đủ lớn để tổng thời gian truy cập trung bình gần bằng với thời gian đó khi chỉ sử dụng riêng cache. Có một số động lực khác để giảm thiểu kích thước cache. Cache càng lớn thì có càng nhiều công logic tham gia vào việc định địa chỉ cache. Kết quả là các bộ nhớ cache lớn có xu hướng chậm hơn cache nhỏ - kể cả khi được xây dựng bằng cùng công nghệ mạch tích hợp và đặt ở cùng một vị trí trên chip và bảng mạch. Diện tích chip và bảng mạch cũng giới hạn kích thước bộ nhớ cache. Có thể nói, không có một kích thước cache nào là "tối ưu". Bảng 4.3 liệt kê các kích thước bộ nhớ cache của một số bộ xử lý từ trước tới nay.

Bảng 4.3 Kích thước cache của một số bộ xử lý

Bộ xử lý	Loại	Năm	Cache L1 ^a	Cache L2	Cache L3
IBM 360/85	Mainframe	1968	16 – 32 kB		-
PDP-11/70	Máy tính mini	1975	1 kB	-	-
VAX 11/780	Máy tính mini	1978	16 kB	-	-

IBM 3033	Mainframe	1978	64 kB	-	-
IBM 3090	Mainframe	1985	128 – 256 kB	-	-
Intel 80486	PC	1989	8 kB	-	-
Pentium	PC	1993	8 kB/8 kB	256–512 kB	-
PowerPC 601	PC	1993	32 kB	-	-
PowerPC 620	PC	1996	32 kB/32 kB	-	-
PowerPC G4	PC/server	1999	32 kB/32 kB	256 kB–1 MB	2 MB
IBM S/390 G6	Mainframe	1999	256 kB	8 MB	-
Pentium 4	PC/server	2000	8 kB/8 kB	256 kB	-
IBM SP	High-end server/siêu máy tính	2000	64 kB/32 kB	8 MB	-
CRAY MTA ^b	Siêu máy tính	2000	8 kB	2 MB	-
Itanium	PC/server	2001	16 kB/16 kB	96 kB	4 MB
Itanium 2	PC/server	2002	32 kB	256 kB	6 MB
IBM POWER5	High-end server	2003	64 kB	1.9 MB	36 MB
CRAY XD-1	Siêu máy tính	2004	64 kB/64 kB	1 MB	-
IBM POWER6	PC/server	2007	64 kB/64 kB	4 MB	32 MB
IBM z10	Mainframe	2008	64 kB/128 kB	3 MB	24–48 MB
Intel Core i7 EE 990	Workstation/server	2011	6*32 kB/ 32kB	1.5 MB	12 MB
IBM zEnterprise 196	Mainframe/server	2011	24*64 kB/ 128 kB	24*1.5 MB	24 MB L3 192 MB L4

Lưu ý:

^a Hai giá trị được ngăn cách bởi dấu gạch chéo ứng với cache lệnh và cache dữ liệu.

^b Cả hai cache đều là cache lệnh; không có cache dữ liệu.

4.3.3. Hàm ánh xạ

Bởi vì số lượng line trong cache ít hơn số block bộ nhớ chính, cần có thuật toán ánh xạ các block bộ nhớ chính vào các line cache. Hơn nữa, cần có phương tiện để xác định block bộ nhớ chính nào đang chiếm một line cache. Hàm ánh xạ được sử dụng sẽ quyết định cách

thức tổ chức cache. Ba kỹ thuật ánh xạ có thể được sử dụng là: trực tiếp, kết hợp, và kết hợp tập hợp. Ta sẽ lần lượt tìm hiểu từng kỹ thuật thông qua ví dụ cụ thể.

Ví dụ 4.1 Đối với cả ba trường hợp, có một số giả thiết sau:

- Cache có thể chứa 64 Kbyte.
- Dữ liệu được truyền giữa bộ nhớ chính và cache theo từng block 4 byte. Do đó, cache gồm $16K = 2^{14}$ line, mỗi line có 4 byte.
- Bộ nhớ chính gồm 16 Mbytes, mỗi byte được đánh địa chỉ trực tiếp bởi một địa chỉ 24 bit ($2^{24} = 16M$). Do đó, đối với mục đích ánh xạ, có thể coi là bộ nhớ chính bao gồm 4M block 4 byte

4.3.3.1. Ánh xạ trực tiếp

Đây là kỹ thuật ánh xạ đơn giản nhất, ánh xạ mỗi block của bộ nhớ chính vào một line cache xác định duy nhất. Hàm ánh xạ được biểu diễn bởi

$$i = j \bmod m$$

trong đó

i = chỉ số line cache

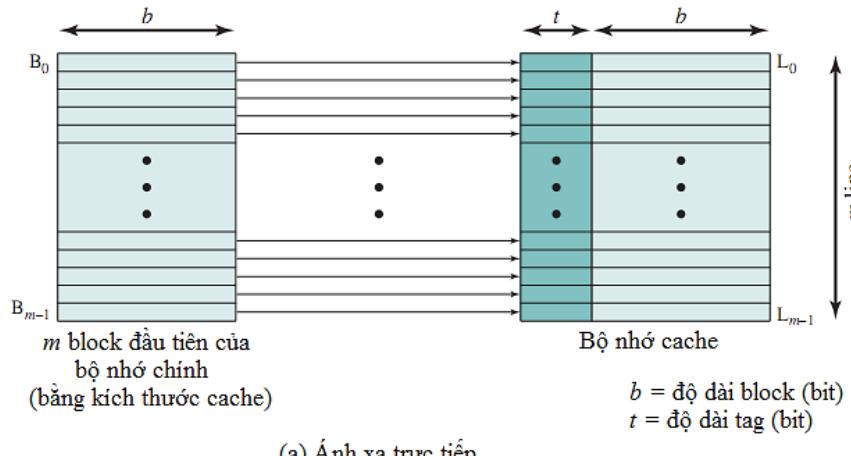
j = chỉ số block bộ nhớ chính

m = tổng số line trong cache

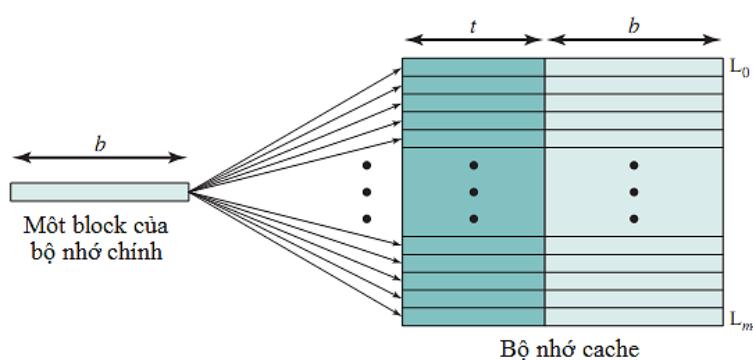
Hình 4.7a mô tả sự ánh xạ m block đầu tiên của bộ nhớ chính. Mỗi block trong bộ nhớ chính được ánh xạ vào một line duy nhất trong cache. m block tiếp theo của bộ nhớ chính cũng được ánh xạ vào cache theo cách tương tự; tức là, block B_m của bộ nhớ chính ánh xạ vào line L_0 of cache, block B_{m+1} ánh xạ vào line L_1 , v.v...

Hàm ánh xạ này được thực hiện dễ dàng bằng địa chỉ bộ nhớ chính. Hình 4.8 mô tả cơ chế tổng quát. Đối với mục đích truy cập bộ nhớ cache, mỗi địa chỉ bộ nhớ chính được xem như bao gồm ba trường. w bit có trọng số nhỏ nhất xác định một word hoặc một byte trong một block của bộ nhớ chính; trong hầu hết các máy tính hiện đại, địa chỉ được đánh ở mức byte. s bit còn lại xác định một block trong 2^s block của bộ nhớ chính. Logic cache chia s bit này thành trường tag $s - r$ bit (phần bit có trọng số lớn nhất) và trường line r bit. Trường line xác định một line trong $m = 2^r$ line của cache. Tóm lại,

- Độ dài địa chỉ = $(s + w)$ bits
- Số lượng đơn vị địa chỉ = 2^{s+w} word hoặc byte
- Kích thước block = kích thước line = 2^w word hoặc byte
- Số block trong bộ nhớ chính = $2^{s+w}/2^w = 2^s$
- Số line trong cache = $m = 2^r$
- Kích thước cache = 2^{r+w} word hoặc byte
- Kích thước tag = $(s - r)$ bit

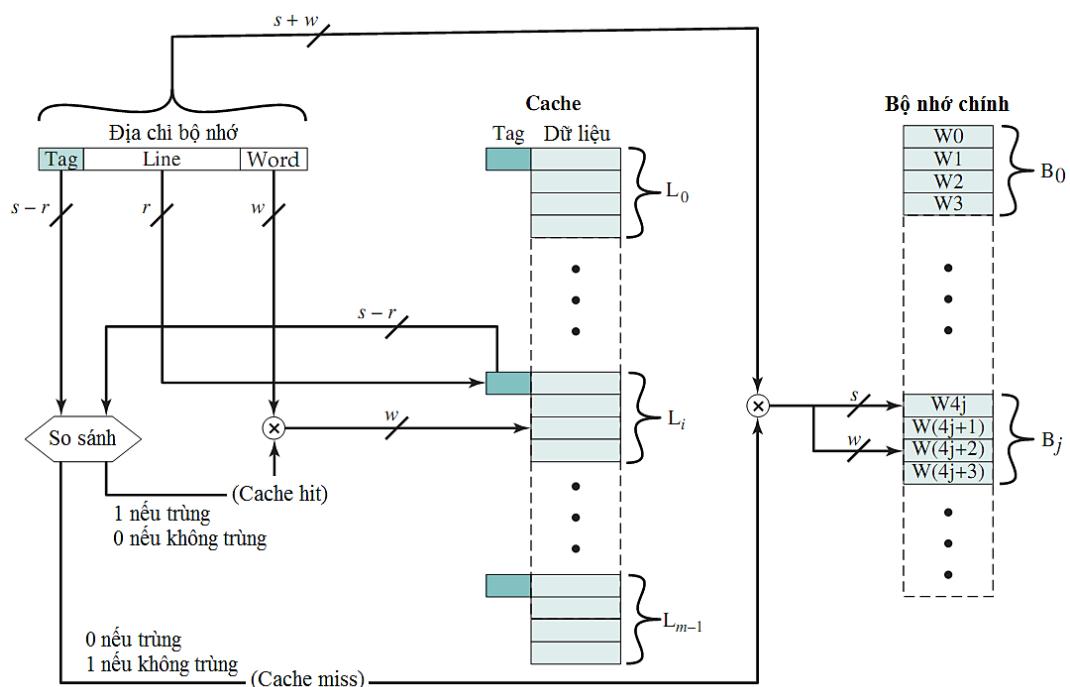


(a) Ánh xạ trực tiếp



(b) Ánh xạ kết hợp

Hình 4.7 Ánh xạ từ Bộ nhớ chính sang Cache: Trực tiếp và Kết hợp



Hình 4.8 Tổ chức cache ánh xạ trực tiếp

Ví dụ 4.1a Hình 4.9 mô tả hệ thống ví dụ của ta sử dụng ánh xạ trực tiếp¹. Trong ví dụ này, $m = 16K = 2^{14}$ và $i = j \bmod 2^{14}$. Sự ánh xạ trở thành:

Line cache	Địa chỉ bộ nhớ chính bắt đầu của block
0	000000, 010000, ..., FF0000
1	000004, 010004, ..., FF0004
:	:
$2^{14} - 1$	00FFFC, 01FFFC, ..., FFFFFC

Lưu ý rằng hai block có cùng chỉ số tag thì không bao giờ được ánh xạ vào cùng một line. Do đó, các block có địa chỉ bắt đầu 000000, 010000... FF0000 có chỉ số tag lần lượt là 00, 01... FF.

Xem lại Hình 4.4, một hành động đọc diễn ra như sau. Hệ thống cache sử dụng địa chỉ 24 bit. Chỉ số line 14 bit được sử dụng như một con trỏ trong cache để chỉ định truy cập vào một line cụ thể. Nếu chỉ số tag 8 bit trùng với chỉ số tag hiện đang được lưu trong một line, thì chỉ số word 2 bit được sử dụng để lựa chọn một byte trong 4 byte trong line đó. Nếu không có tag nào trùng, 22 bit của trường tag + line sẽ được sử dụng để truy xuất một block từ bộ nhớ chính. Địa chỉ thực tế được sử dụng để truy xuất là 22 bit tag + line nối với hai bit 0, sao cho 4 byte được lấy ra bắt đầu từ byte đầu tiên của block.

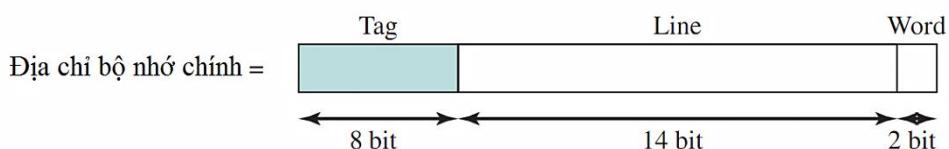
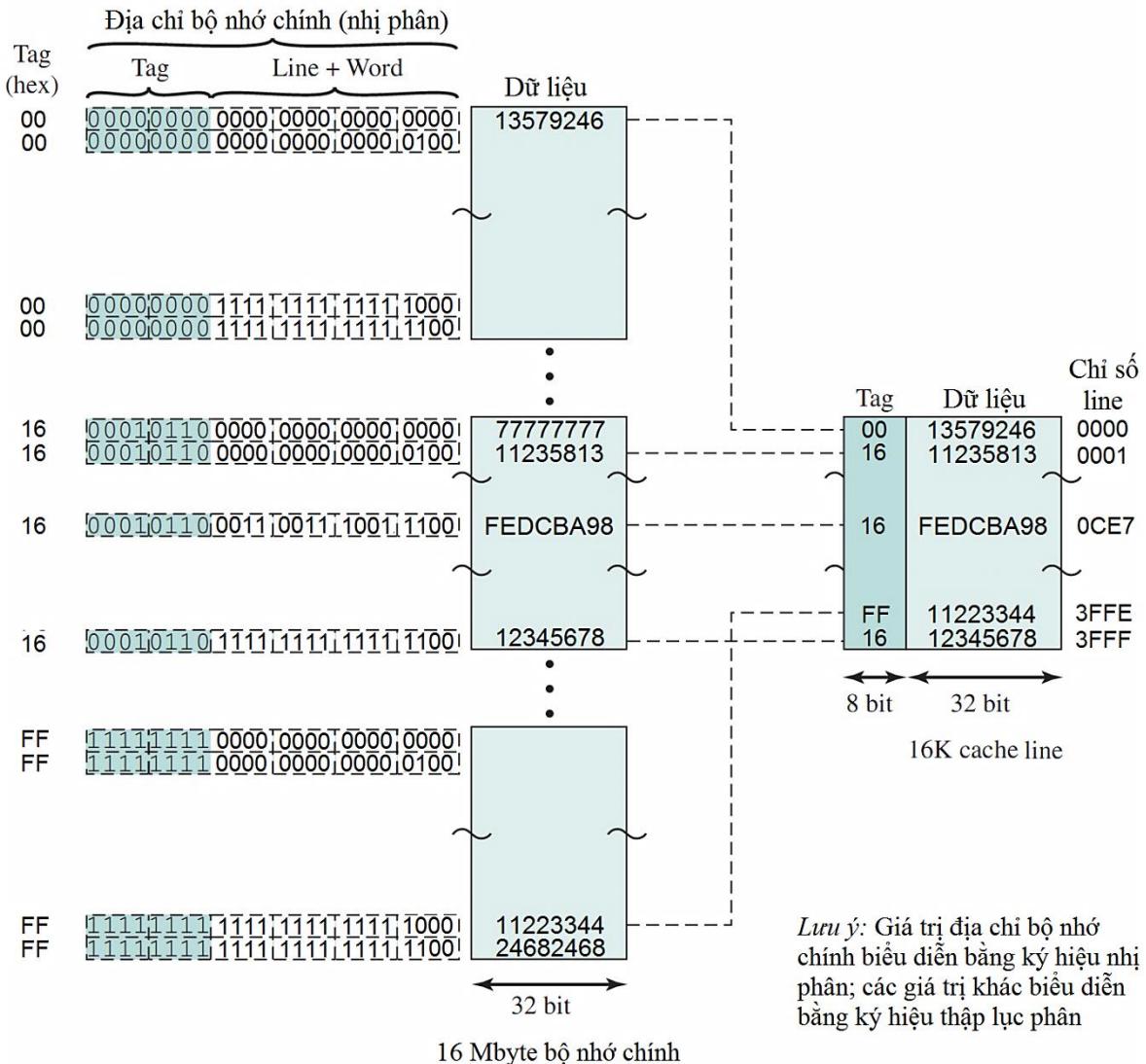
Kỹ thuật ánh xạ trực tiếp đơn giản và không tốn kém để thực hiện. Nhược điểm chính là chỉ có một vị trí cố định trong cache dành cho một block bất kỳ. Vì vậy, nếu một chương trình tham chiếu lặp đi lặp lại các word thuộc hai block khác nhau nhưng được ánh xạ vào cùng một line, khi đó các block sẽ liên tục bị đổi chỗ trong cache, và tỷ lệ truy cập cache thành công sẽ rất thấp.

Một phương pháp để giảm cache miss là ghi nhớ dữ liệu đã bị loại bỏ khỏi cache để phòng trường hợp sau đó lại cần nó lần nữa. Vì dữ liệu bị loại bỏ đã từng được truy xuất, nên nó có thể được sử dụng lại. Việc tái sử dụng này có thể khả thi nếu sử dụng một *cache nạp nhân*. Cache nạp nhân là bộ nhớ cache kết hợp toàn phần, có kích thước từ 4 đến 16 line, nằm giữa cache L1 ánh xạ trực tiếp và cấp độ bộ nhớ tiếp theo.

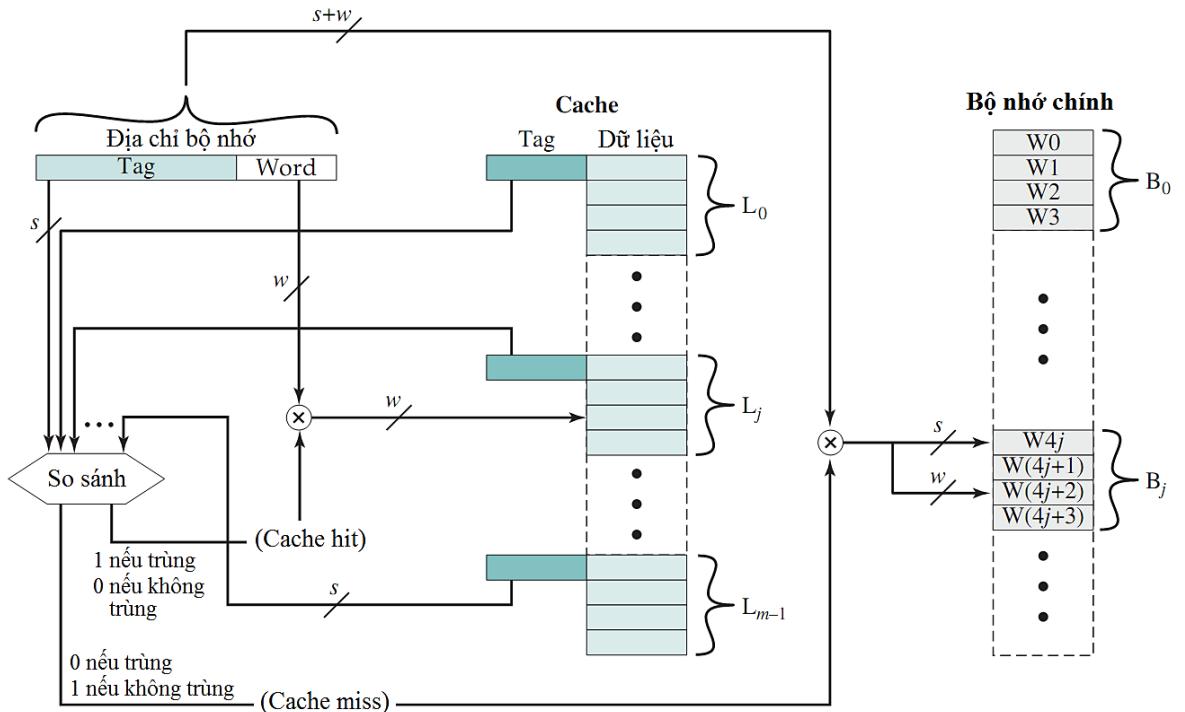
4.3.3.1. Ánh xạ kết hợp

Ánh xạ kết hợp khắc phục nhược điểm của ánh xạ trực tiếp bằng cách cho phép mỗi block bộ nhớ chính được nạp vào bất kỳ line nào trong cache (Hình 4.9b). Trong trường hợp này, địa chỉ bộ nhớ được coi là bao gồm trường Tag và trường Word. Trường Tag xác định một block bộ nhớ chính duy nhất. Để xác định một block có nằm trong bộ nhớ cache hay không, logic điều khiển cache phải kiểm tra tag của tất cả các line cùng lúc để tìm ra giá trị tag trùng. Hình 4.10 minh họa cho logic này.

¹ Trong hình này và các hình tiếp theo, giá trị bộ nhớ được biểu diễn bằng ký hiệu thập lục phân.



Hình 4.9 Ví dụ ánh xạ trực tiếp



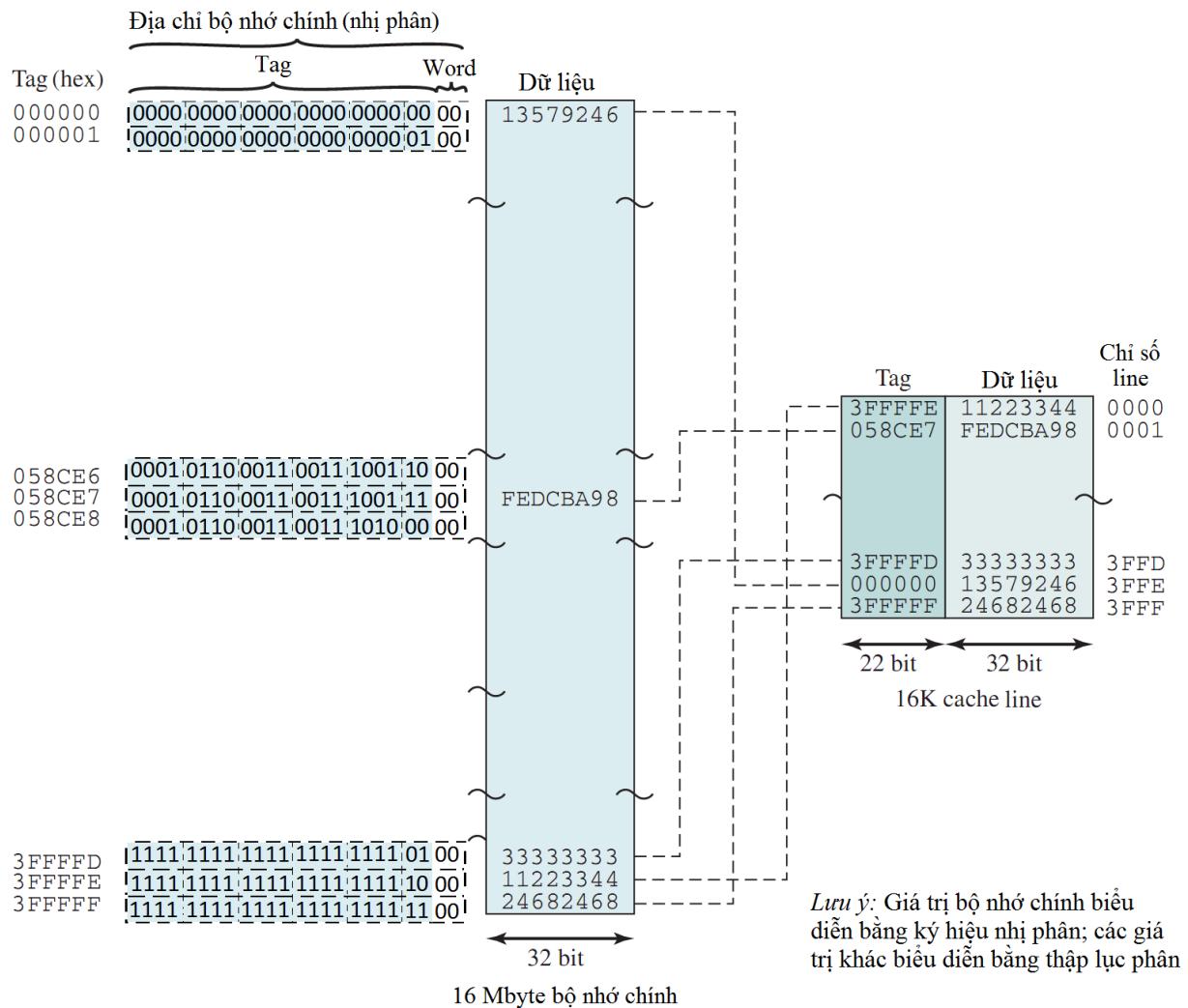
Hình 4.10 Tổ chức cache kết hợp toàn phần

Ví dụ 4.1b Hình 4.11 mô tả hệ thống ví dụ sử dụng ánh xạ kết hợp. Địa chỉ bộ nhớ chính bao gồm chỉ số tag 22 bit và chỉ số byte 2 bit. Tag 22 bit phải được lưu trữ cùng với block dữ liệu 32 bit trong mỗi line của bộ nhớ cache. Lưu ý rằng 22 bit ngoài cùng bên trái (có trọng số lớn nhất) của địa chỉ chính là tag. Do đó, địa chỉ thập lục phân 24 bit 16339C có tag 22 bit là 058CE7. Điều này có thể nhìn thấy dễ dàng hơn trong ký hiệu nhị phân:

Địa chỉ bộ nhớ	0011	0110	0011	0011	1001	1100	(nhị phân)
	1	6	3	3	9	C	(hex)
Tag (22 bit bên trái)	00	0101	1000	1100	1110	0111	(nhị phân)
	0	5	8	C	E	7	(hex)

Lưu ý rằng không có trường nào trong địa chỉ tương ứng với chỉ số line, do đó chỉ số line trong cache không được xác định bằng định dạng địa chỉ. Tóm lại,

- Độ dài địa chỉ = $(s + w)$ bit
- Số lượng đơn vị địa chỉ = 2^{s+w} word hoặc byte
- Kích thước block = Kích thước line = 2^w word hoặc byte
- Số lượng block trong bộ nhớ chính = $2^{s+w}/2^w = 2^s$
- Số lượng line trong cache = không xác định
- Kích thước tag = s bit



Hình 4.11 Ví dụ ánh xa kết hợp

Với ánh xạ kết hợp, việc lựa chọn block nào bị thay thế khi có một block mới được đọc vào bộ nhớ cache trở nên linh hoạt hơn. Các thuật toán thay thế được thiết kế sao cho tỷ lệ truy cập cache thành công là cao nhất. Nhược điểm chính của ánh xạ kết hợp là yêu cầu mạch điện phức tạp hơn để có thể kiểm tra tag của tất cả các line trong cache đồng thời.

4.3.3.2. Ánh xa kết hợp tập hợp

Ánh xạ kết hợp tập hợp thể hiện ưu điểm của cả phương pháp ánh xạ trực tiếp và kết hợp đồng thời lại giảm được nhược điểm của chúng.

Trong trường hợp này, bộ nhớ cache gồm một số tập hợp, mỗi tập hợp gồm một số line. Hàm ánh xạ được thể hiện bằng công thức:

$$m = v * k$$

$$i = j \bmod v$$

trong đó

i = chỉ số của tập hợp trong cache

j = chỉ số của block trong bộ nhớ chính

m = số lượng line trong cache

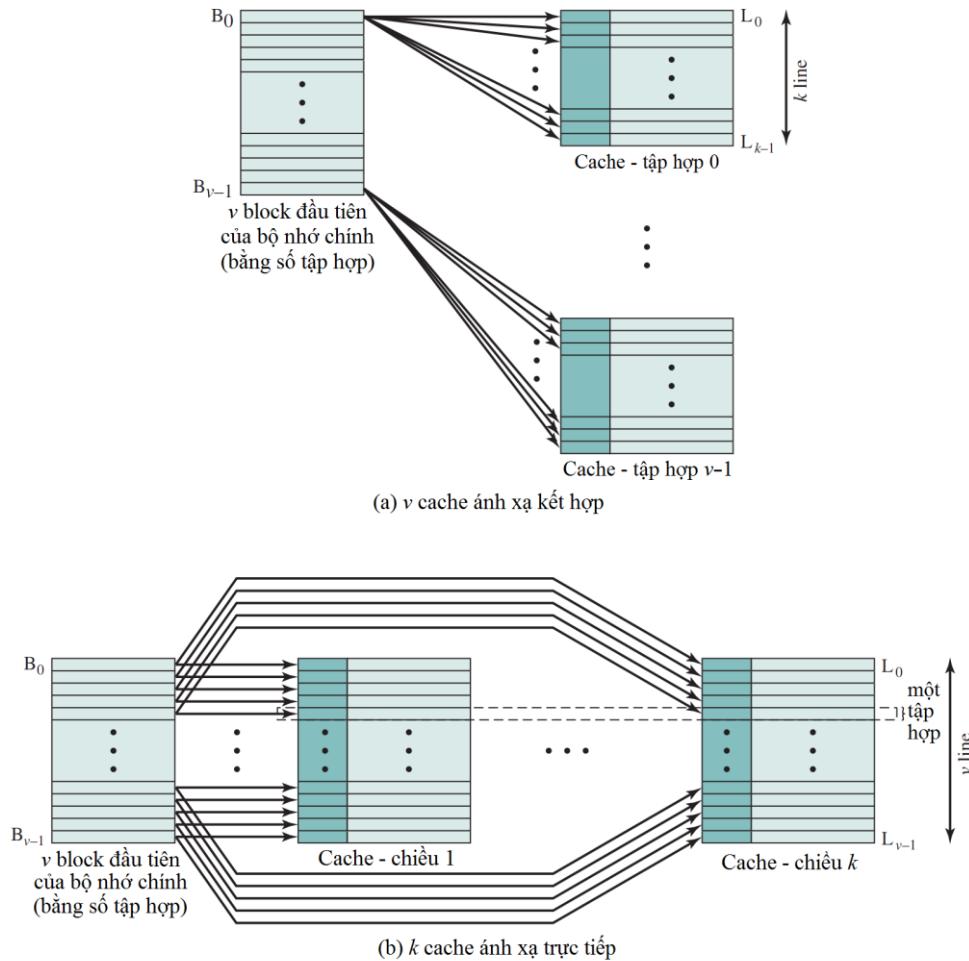
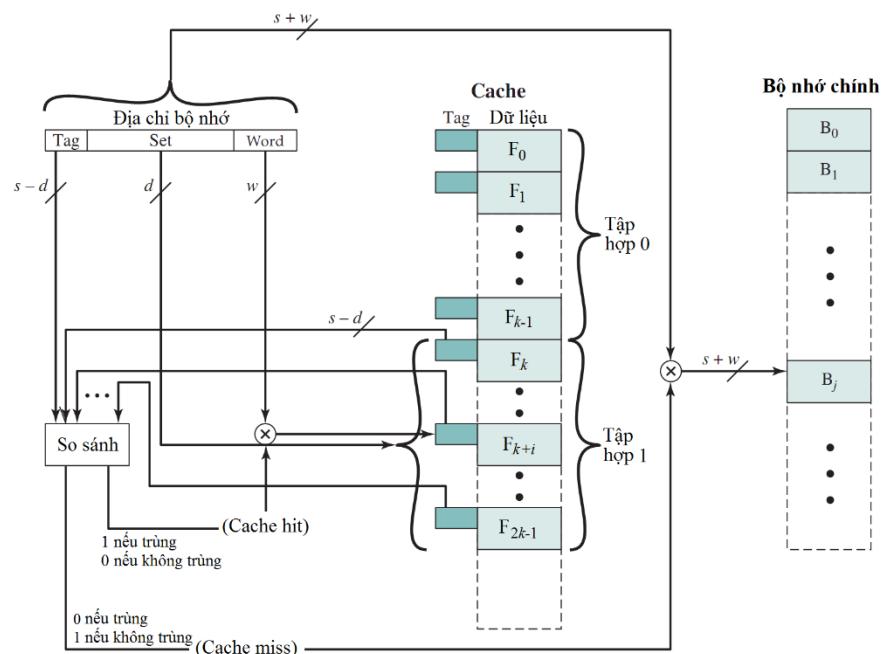
v = số lượng tập hợp

k = số lượng line trong một tập hợp

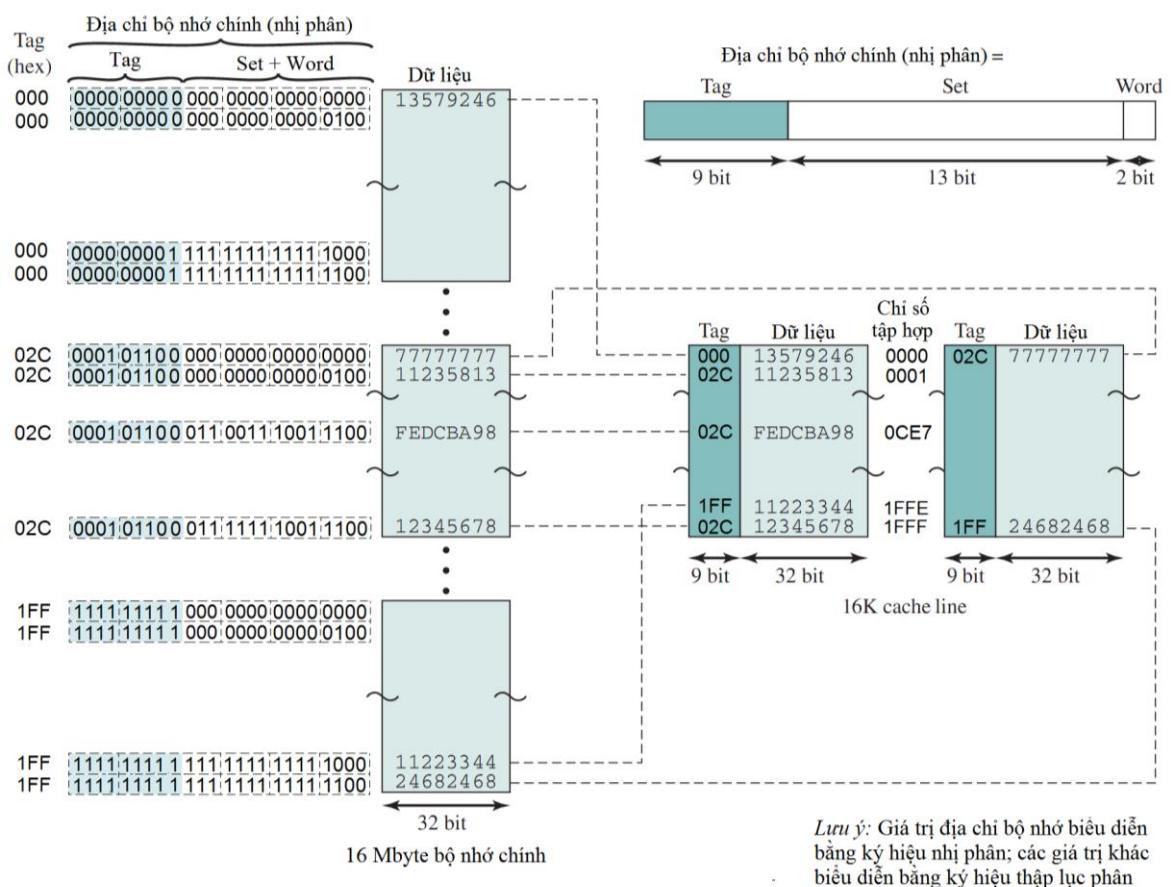
Ta gọi đây là ánh xạ kết hợp tập hợp k chiều. Với ánh xạ kết hợp tập hợp, block B_j có thể được ánh xạ vào bất kỳ line nào của tập i . Hình 4.12a minh họa sự ánh xạ v block đầu tiên của bộ nhớ chính. Giống với ánh xạ kết hợp, mỗi word có thể ánh xạ vào nhiều line trong cache. Đối với ánh xạ kết hợp tập hợp, một word có thể ánh xạ được vào tất cả các line trong một tập hợp nhất định, do đó block B_0 ánh xạ vào tập 0, block B_1 ánh xạ vào tập 1, v.v... Vì vậy, một bộ nhớ cache kết hợp tập hợp có thể được thực hiện như v bộ nhớ cache kết hợp. Cũng có thể thực hiện bộ nhớ cache kết hợp tập hợp như k cache ánh xạ trực tiếp, như thể hiện trong hình 4.12b. Mỗi cache ánh xạ trực tiếp được gọi là một *chiều*, gồm v line. Vậy v block đầu tiên của bộ nhớ chính được ánh xạ trực tiếp vào v line của mỗi chiều; nhóm v block tiếp theo của bộ nhớ chính được ánh xạ tương tự, v.v... Việc thực hiện ánh xạ trực tiếp thường được sử dụng khi mức độ kết hợp nhỏ (tức là giá trị k nhỏ) trong khi việc thực hiện ánh xạ kết hợp thường được sử dụng khi mức độ kết hợp cao hơn.

Đối với ánh xạ kết hợp tập hợp, logic điều khiển cache diễn giải địa chỉ bộ nhớ thành ba trường: Tag, Set và Word. Trong đó, d bit trường Set chỉ định một trong $v = 2^d$ tập hợp. s bit của trường Tag và Set xác định một trong 2^s block của bộ nhớ chính. Hình 4.13 minh họa logic điều khiển bộ nhớ cache. Với ánh xạ kết hợp toàn phần, trường tag trong một địa chỉ bộ nhớ khá lớn và phải được so sánh với tag của tất cả các line trong cache. Với ánh xạ kết hợp tập hợp k chiều, trường tag trong địa chỉ bộ nhớ nhỏ hơn nhiều và chỉ cần so sánh với k tag trong một tập hợp duy nhất. Tóm lại,

- Độ dài địa chỉ = $(s + w)$ bit
- Số lượng đơn vị địa chỉ = 2^{s+w} word hoặc byte
- Kích thước block = kích thước line = 2^w word hoặc byte
- Số block trong bộ nhớ chính = $2^{s+w}/2^w = 2^s$
- Số line trong tập hợp = k
- Số lượng set = $v = 2^d$
- Số lượng line trong cache = $m = kv = k * 2^d$
- Kích thước cache = $k * 2^{d+w}$ word hoặc byte
- Kích thước tag = $(s - d)$ bit

Hình 4.12 Ánh xạ từ Bộ nhớ chính sang Cache: Kết hợp tập hợp k chiềuHình 4.13 Tổ chức cache kết hợp tập hợp k chiều

Ví dụ 4.1c Hình 4.14 mô tả hệ thống ví dụ sử dụng ánh xạ kết hợp tập hợp với hai line trong mỗi tập hợp, được gọi là kết hợp tập hợp hai chiều. Trường Set 13 bit xác định một tập hợp hai line nhất định trong cache. Nó cũng cho biết số block trong bộ nhớ chính, mod 2^{13} . Điều này xác định việc ánh xạ block vào line. Do đó, các block 000000, 008000... FF8000 của bộ nhớ chính được ánh xạ vào tập hợp 0 trong cache. Lưu ý rằng không có hai block nào được ánh xạ vào cùng một tập hợp trong cache mà lại có cùng chỉ số tag. Đối với hoạt động đọc, trường set 13 bit được sử dụng để xác định tập hợp hai line nào sẽ được kiểm tra. Cả hai line trong tập hợp đó được kiểm tra xem có chứa tag trùng với tag trong địa chỉ bộ nhớ cần truy cập.

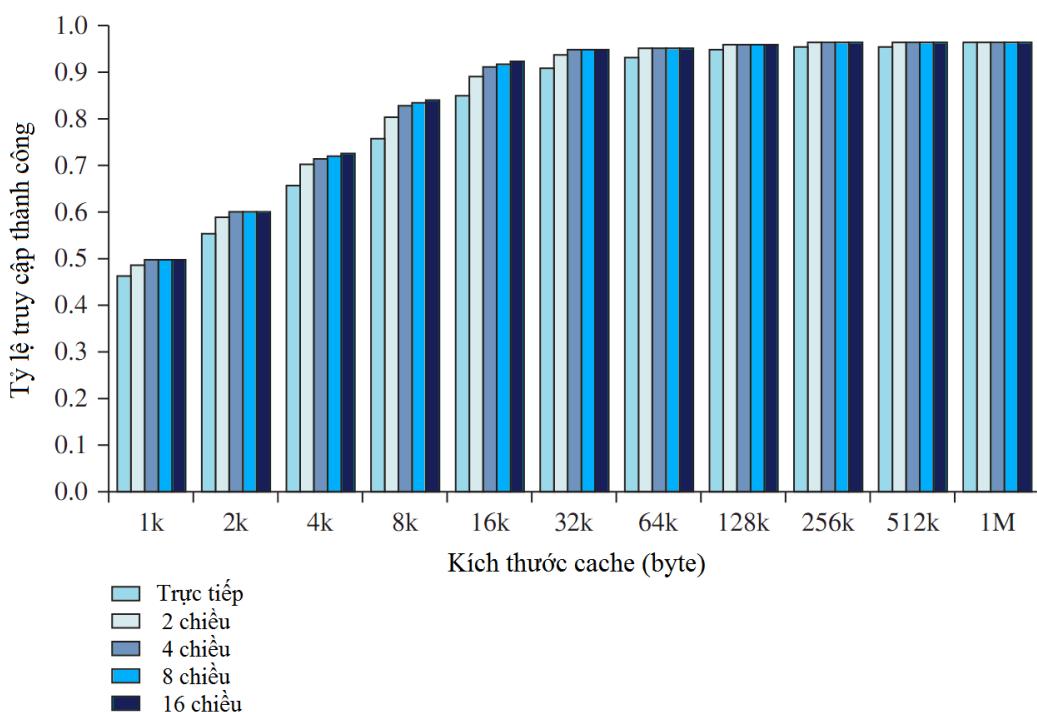


Hình 4.14 Ví dụ ánh xạ kết hợp tập hợp 2 chiều

Trong trường hợp đặc biệt khi $v = m$, $k = 1$, kỹ thuật kết hợp tập hợp trở thành ánh xạ trực tiếp, và khi $v = 1$, $k = m$, nó sẽ trở thành ánh xạ kết hợp. Việc sử dụng hai line trong một tập hợp ($v = m/2$, $k = 2$) là tổ chức kết hợp tập hợp phổ biến nhất. Nó cải thiện đáng kể tỷ lệ truy cập thành công so với ánh xạ trực tiếp. Kết hợp tập hợp bốn chiều ($v = m/4$, $k = 4$) đem lại sự cải thiện vừa phải. Tuy nhiên, tăng hơn nữa số line trong một tập hợp lại không có nhiều hiệu quả.

Hình 4.15 là kết quả của một nghiên cứu mô phỏng với hiệu suất cache kết hợp tập hợp là hàm biến thiên theo kích thước cache. Sự chênh lệch hiệu suất giữa kết hợp trực tiếp và kết hợp hai chiều là đáng kể khi kích thước cache nhỏ hơn hoặc bằng 64 kB. Cũng lưu ý rằng sự chênh lệch hiệu suất giữa hai chiều và bốn chiều ở 4 kB nhỏ hơn nhiều so với sự chênh lệch ở kích thước cache 8 kB. Độ phức tạp của cache tăng tỷ lệ với mức độ kết hợp, và thể hiện rõ khi kích thước cache nhỏ hơn 16 Kbyte. Điểm lưu ý cuối cùng là, ngoài khoảng 32 kB, việc tăng kích thước cache không làm tăng hiệu suất một cách đáng kể.

Kết quả của Hình 4.15 dựa trên mô phỏng quá trình thực thi của trình biên dịch GCC. Các ứng dụng khác nhau có thể mang lại kết quả khác nhau, tuy nhiên kết quả so sánh tỷ lệ truy cập cache thành công với kích thước cache vẫn thể hiện xu hướng giống như Hình 4.15.



Hình 4.15 Sự biến thiên của mức độ kết hợp theo kích thước cache

4.3.4. Thuật toán thay thế

Khi bộ nhớ cache đã đầy, nếu một block mới được đưa đến cache, một trong những block hiện đang nằm trong cache phải được thay thế. Đối với ánh xạ trực tiếp, chỉ có một line định sẵn cho một block bất kỳ và không có sự lựa chọn nào khác. Đối với các kỹ thuật kết hợp và kết hợp tập hợp, cần phải có thuật toán thay thế. Để đạt được tốc độ cao, thuật toán đó phải được thực hiện trong phần cứng. Ta sẽ đề cập đến bốn trong số các thuật toán phổ biến nhất. Thuật toán hiệu quả nhất là LRU (least recently used): Thay thế block nằm trong cache lâu nhất mà không có tham chiếu đến nó. Đối với kết hợp tập hợp hai chiều, thuật toán này được thực hiện dễ dàng. Mỗi line chứa một bit USE. Khi một line được tham chiếu, bit USE của nó được đặt lên 1 và bit USE của line còn lại trong tập hợp đó được đặt thành 0. Khi một block mới được đọc vào tập hợp, line có bit USE bằng 0 được thay thế. Do ta giả thiết rằng các vị trí bộ nhớ được sử dụng gần đây hơn có nhiều khả năng

được tham chiếu tiếp, LRU cho tỷ lệ truy cập thành công tốt nhất. LRU cũng tương đối dễ thực hiện trong cache kết hợp toàn phần. Cơ chế bộ nhớ cache duy trì một danh sách riêng chứa chỉ mục của tất cả các line trong cache. Khi một line được tham chiếu, nó được di chuyển lên phía trên của danh sách. Khi cần thay thế, line ở cuối danh sách được sử dụng. Do sự đơn giản trong thực hiện, LRU là thuật toán thay thế phổ biến nhất.

Một thuật toán khác là FIFO (first-in-first-out): Thay thế block trong tập hợp đã nằm trong cache lâu nhất. FIFO được thực hiện dễ dàng bằng kỹ thuật round-robin. Một thuật toán nữa là LFU (least frequently used): Thay thế block trong tập hợp mà số lần tham chiếu đến nó là ít nhất. LFU có thể được thực hiện bằng cách sử dụng một bộ đếm ở mỗi line. Một kỹ thuật khác không dựa trên sự sử dụng block (tức là không phải LRU, LFU, FIFO) là chọn một line ngẫu nhiên trong số các line. Các nghiên cứu đã chỉ ra rằng việc thay thế ngẫu nhiên đem lại hiệu suất chỉ kém hơn một chút so với thuật toán dựa trên sự sử dụng block.

4.3.5. Chính sách ghi

Khi một block đang nằm trong cache được thay thế, có 2 trường hợp cần được xem xét. Nếu block cũ trong cache chưa từng thay đổi giá trị thì có thể ghi đè block mới lên mà không cần ghi block cũ ra trước. Nếu có ít nhất một thao tác ghi đã được thực hiện trên một word trong line của cache thì bộ nhớ chính phải được cập nhật bằng cách ghi line đó vào một block của bộ nhớ chính trước khi đưa block mới vào cache. Có nhiều chính sách ghi khác nhau, với ràng buộc về hiệu suất và tính thương mại. Có hai vấn đề cần phải đối mặt. Một là, nhiều thiết bị có thể truy cập vào bộ nhớ chính. Ví dụ, một mô-đun I/O có thể đọc/ghi trực tiếp vào bộ nhớ. Nếu một word đã thay đổi chỉ được cập nhật trong cache, thì word tương ứng ở bộ nhớ chính không còn hợp lệ. Hơn nữa, nếu thiết bị I/O thay đổi word trong bộ nhớ chính, thì word trong cache lại không hợp lệ. Hai là, vấn đề phức tạp hơn xảy ra khi nhiều bộ xử lý được gắn vào cùng một bus và mỗi bộ xử lý lại có cache cục bộ riêng. Khi đó, nếu một word bị thay đổi trong một cache, nó có thể làm mất ý nghĩa một word trong các cache khác.

Kỹ thuật đơn giản nhất được gọi là **write through**. Sử dụng kỹ thuật này, tất cả các hành động ghi được thực hiện đồng thời ở cả bộ nhớ chính và cache, đảm bảo rằng nội dung bộ nhớ chính luôn luôn hợp lệ. Một mô-đun bộ xử lý – cache bất kỳ có thể giám sát lưu lượng truy cập vào bộ nhớ chính để duy trì tính nhất quán trong cache của chính nó. Nhược điểm chính của kỹ thuật này là nó tạo ra lưu lượng bộ nhớ đáng kể và có thể tạo ra nút nghẽn cỏ chai. Một kỹ thuật khác, được gọi là **write back**, giảm thiểu việc ghi vào bộ nhớ chính. Với write back, việc cập nhật giá trị mới chỉ được thực hiện trong cache. Khi có một cập nhật xảy ra, một bit được gọi là **dirty bit** hoặc **use bit** trong line được đặt lên 1. Khi một block cần được thay thế trong cache, nó được ghi trở lại bộ nhớ chính nếu và chỉ nếu dirty bit bằng 1. Vấn đề đối với write back là nhiều phần của bộ nhớ chính là không hợp lệ, và do đó các truy cập bằng mô-đun I/O chỉ được cho phép thực hiện thông qua cache. Điều này làm cho mạch phức tạp hơn và khả năng tạo nút nghẽn cỏ chai.

4.3.6. Kích thước line

Một yếu tố thiết kế khác là kích thước line. Khi 1 block dữ liệu được lấy ra và đặt vào cache, line sẽ chứa không chỉ word mong muốn mà còn một số word liền kề. Khi kích

thước block tăng dần, ban đầu tỷ lệ truy cập thành công cũng tăng do nguyên lý cục bộ của tham chiếu, theo đó dữ liệu trong vùng lân cận của một word được tham chiếu có thể sẽ được tham chiếu trong tương lai gần. Khi kích thước block tăng thêm, nhiều dữ liệu hữu ích hơn được đưa vào cache. Tuy nhiên, tỷ lệ truy cập thành công bắt đầu giảm khi block lớn hơn nữa và xác suất sử dụng thông tin mới truy xuất sẽ nhỏ hơn xác suất tái sử dụng thông tin phải được thay thế. Có hai ảnh hưởng rõ rệt là:

- Block lớn hơn làm giảm số lượng block trong cache. Bởi vì mỗi lần truy xuất block mới lại ghi đè lên nội dung cũ hơn trong cache, số lượng block càng ít thì dữ liệu càng nhanh chóng bị ghi đè ngay sau khi chúng được truy xuất.
- Khi 1 block lớn hơn, mỗi word thêm vào lại càng xa word mong muốn và do đó ít có khả năng là cần thiết trong tương lai gần.

Mối quan hệ giữa kích thước line và tỷ lệ truy cập thành công là rất phức tạp, tùy thuộc vào đặc tính cục bộ của mỗi chương trình cụ thể, và không có giá trị tối ưu nhất định. Kích thước line từ 8 đến 64 byte được xem là hợp lý, gần như tối ưu.

4.3.7. Số lượng cache

Khi cache mới ra mắt, hệ thống điển hình có một bộ nhớ cache đơn. Gần đây, việc sử dụng nhiều bộ nhớ cache đã trở nên phổ biến. Hai khía cạnh của vấn đề thiết kế này là số lượng cấp độ cache và việc sử dụng cache thống nhất hay phân chia.

4.3.7.1. Cache nhiều cấp độ

Khi mật độ logic tăng lên, việc đưa bộ nhớ cache lên trên chip cùng với bộ xử lý là khả thi: ta có *cache trên chip*. So với cache có thể truy cập thông qua bus bên ngoài, cache trên chip làm giảm bớt việc bộ xử lý phải sử dụng bus bên ngoài và do đó tăng tốc độ thực hiện và tăng hiệu suất tổng thể của hệ thống. Khi đã tìm thấy lệnh hoặc dữ liệu được yêu cầu trong cache trên chip, việc truy cập bus được loại bỏ. Do các đường dữ liệu trong bộ xử lý ngắn hơn so với độ dài của bus, việc truy cập bộ nhớ cache trên chip sẽ nhanh hơn rất nhiều so với chu kỳ bus. Hơn nữa, trong giai đoạn này, bus được tự do để hỗ trợ các nhu cầu truyền khác.

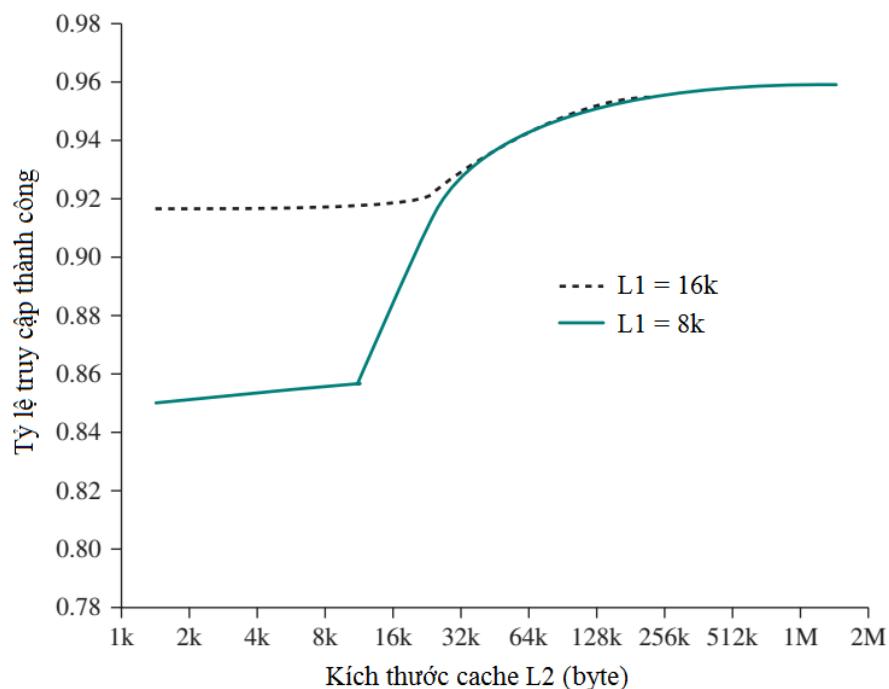
Việc sử dụng một bộ nhớ cache trên chip mở ra câu hỏi liệu có còn cần cache ngoài chip hay không. Câu trả lời là có, và hầu hết các thiết kế hiện đại có cả cache trên chip và cache bên ngoài. Tổ chức đơn giản nhất là bộ nhớ cache hai cấp, với cache trên chip được gán là cấp 1 (L1) và cache bên ngoài được gán là cấp 2 (L2). Lý do cho việc sử dụng cache L2 là: Nếu không có cache L2 và bộ xử lý yêu cầu truy cập vào một vị trí bộ nhớ không có trong cache L1, thì bộ xử lý phải truy cập vào bộ nhớ DRAM hoặc ROM thông qua bus. Do tốc độ bus thường chậm và thời gian truy cập bộ nhớ chính lâu, điều này dẫn đến hiệu suất kém. Mặt khác, nếu sử dụng bộ nhớ cache L2 SRAM (RAM tĩnh), thì thông tin bị thiếu có thể được truy xuất nhanh chóng. Nếu SRAM đủ nhanh để kịp với tốc độ của bus thì dữ liệu có thể được truy cập bằng một giao dịch trạng thái không đợi, kiểu truyền bus nhanh nhất.

Có hai đặc điểm thiết kế đáng lưu ý đối với các bộ nhớ cache nhiều cấp độ. Một là, đối với cache L2 ngoài chip, nhiều thiết kế không sử dụng bus hệ thống làm đường dẫn giữa cache L2 và bộ xử lý, mà sử dụng một đường dẫn dữ liệu riêng biệt, để giảm bớt gánh nặng

cho hệ thống bus. Hai là, khi các thành phần của bộ xử lý liên tục thu nhỏ kích thước, một số bộ xử lý hiện nay kết hợp cả cache L2 trên chip xử lý, cải thiện hiệu suất.

Hình 4.16 cho thấy kết quả của một nghiên cứu mô phỏng sự thay đổi hiệu suất bộ nhớ cache hai cấp theo kích thước cache. Giả thiết rằng cả hai cache có kích thước line nhau nhau. Hình vẽ thể hiện tỷ lệ truy cập cache thành công tổng cộng. Tức là, một lần truy cập thành công được tính nếu dữ liệu mong muốn xuất hiện trong cache L1 hoặc L2. Hình vẽ này cho thấy ảnh hưởng của L2 lên tổng số truy cập thành công khi kích thước L1 cố định. L2 bắt đầu có ảnh hưởng đáng kể đến tổng số lần truy cập cache thành công khi kích thước của nó gấp đôi kích thước cache L1. Lưu ý rằng, trong đường biểu diễn cho cache L1 có kích thước 8 Kbyte, phần dốc nhất tương ứng với cache L2 có kích thước là 16 Kbyte. Còn đối với cache L1 16 Kbyte, phần dốc nhất của đường biểu diễn ứng với kích thước cache L2 là 32 Kbyte. Trước đó, cache L2 có ảnh hưởng rất ít đến hiệu suất tổng cộng của bộ nhớ cache. Nhu cầu sử dụng cache L2 lớn hơn cache L1 để cải thiện hiệu suất là hoàn toàn hợp lý. Nếu cache L2 có kích thước line và dung lượng bằng với cache L1, nội dung của nó sẽ gần như giống hệt cache L1.

Khi diện tích sẵn có trên chip dành cho bộ nhớ cache ngày càng tăng, hầu hết các bộ xử lý hiện đại đã chuyển cache L2 lên chip xử lý và có thêm cache L3. Ban đầu, bộ nhớ cache L3 có thể truy cập được qua bus bên ngoài. Gần đây, hầu hết các bộ xử lý đã tích hợp cache L3 trên chip. Việc dùng thêm cấp độ cache thứ ba có thể góp phần cải thiện hiệu suất. Hơn nữa, các hệ thống lớn hiện nay, chẳng hạn như hệ thống mainframe IBM zEnterprise, kết hợp 3 cấp độ cache trên chip và cache thứ tư dùng chung cho nhiều chip.



Hình 4.16 Tỷ lệ truy cập thành công tổng cộng (L1 và L2) với L1 8 Kbyte và 16 Kbyte

4.3.7.2. Cache thống nhất và cache phân chia

Khi cache trên chip mới xuất hiện, có nhiều thiết kế cho bộ nhớ cache đơn để lưu trữ các tham chiếu tới cả dữ liệu và lệnh. Gần đây, việc chia tách bộ nhớ cache thành hai phần:

một dành cho lệnh và một dành cho dữ liệu dần trở nên phổ biến. Hai bộ nhớ cache này ngang hàng nhau, thường là hai cache L1. Khi bộ xử lý cần truy xuất lệnh từ bộ nhớ chính, trước tiên nó sẽ kiểm tra cache lệnh L1, còn khi bộ xử lý cần truy xuất dữ liệu từ bộ nhớ chính, trước tiên nó sẽ kiểm tra cache dữ liệu L1.

Cache thống nhất có hai ưu điểm:

- Đối với một kích thước cache xác định, cache thống nhất có tỷ lệ truy cập thành công cao hơn cache phân chia vì nó tự động cân bằng tải giữa các truy xuất lệnh và dữ liệu. Tức là, nếu một mẫu thực thi có nhiều lượt truy xuất lệnh hơn so với dữ liệu thì bộ nhớ cache sẽ có xu hướng chứa đầy các lệnh và nếu một mẫu thực thi có tương đối nhiều lượt truy xuất dữ liệu hơn, điều ngược lại sẽ xảy ra.
- Chỉ cần thiết kế và thực hiện một bộ nhớ cache.

Xu hướng sử dụng cache là cache chia sẻ tại cấp L1 và cache thống nhất cho các cấp cao hơn, đặc biệt đối với các máy siêu vô hướng chú trọng việc thực hiện lệnh song song và truy xuất trước các lệnh dự đoán được sử dụng trong tương lai. Ưu điểm chính của việc thiết kế bộ nhớ cache phân chia là loại trừ sự tranh chấp cache giữa khói truy xuất/giải mã lệnh và khói thi hành. Điều này rất quan trọng trong các thiết kế có sử dụng pipeline lệnh. Thông thường, bộ xử lý sẽ truy xuất trước các lệnh và điền chúng vào một bộ đệm, hoặc pipeline. Giả sử có một bộ nhớ cache lệnh/dữ liệu thống nhất. Khi khói thi hành truy cập bộ nhớ để tải và lưu trữ dữ liệu, yêu cầu được gửi đến cache thống nhất. Nếu cùng lúc đó, khói truy xuất lệnh đưa ra yêu cầu đọc một lệnh cho bộ nhớ cache, yêu cầu đó sẽ tạm thời bị chặn để cache có thể phục vụ khói thi hành trước, để hoàn thành lệnh hiện đang thi hành. Sự tranh chấp cache này có thể làm giảm hiệu suất. Cấu trúc cache phân chia khắc phục khó khăn này.

4.4. TỔ CHỨC CACHE PENTIUM 4

Có thể nhìn thấy rõ sự phát triển của tổ chức bộ nhớ cache qua quá trình phát triển vi xử lý Intel (Bảng 4.4). Vi xử lý 80386 không có cache trên chip. Vi xử lý 80486 có một cache trên chip 8 Kbyte, kích thước line 16 byte và tổ chức kết hợp tệp 4 chiều. Tất cả các vi xử lý Pentium đều có hai cache L1 trên chip, một cho dữ liệu và một cho lệnh. Đối với Pentium 4, cache L1 có kích thước 16 Kbyte, kích thước line 64 byte và tổ chức kết hợp tệp 4 chiều. Pentium II còn có thêm cache L2 đưa đầu vào tới cả hai bộ nhớ cache L1. Cache L2 là kết hợp tệp 8 chiều với kích thước 512 kB và kích thước line 128 byte. Bộ nhớ cache L3 bắt đầu được thêm vào ở Pentium III và được đưa lên trên chip ở Pentium 4.

Bảng 4.4 Sự phát triển của cache Intel

Vấn đề	Giải pháp	Xuất hiện lần đầu ở chip xử lý
Bộ nhớ bên ngoài chậm hơn bus hệ thống	Bổ sung thêm cache bên ngoài với công nghệ bộ nhớ nhanh hơn	386
Tăng tốc độ bộ xử lý khiến cho bus bên ngoài trở thành nút nghẽn đôi	Di chuyển cache bên ngoài lên chip, hoạt động cùng tốc độ của	486

với các truy cập cache	bộ xử lý	
Cache bên trong khá nhỏ do không gian trên chip hạn chế	Bổ sung thêm cache L2 với công nghệ nhanh hơn bộ nhớ chính	486
Xuất hiện sự tranh chấp khi cả khôi truy xuất lệnh trước và khôi thi hành cùng lúc yêu cầu truy cập vào cache. Trong trường hợp đó, khôi truy xuất lệnh trước bị tạm dừng để khôi thi hành truy cập dữ liệu.	Tách riêng cache dữ liệu và cache lệnh	Pentium
Tăng tốc độ bộ xử lý khiến cho bus bên ngoài trở thành nút nghẽn đối với các truy cập cache L2	Tạo bus riêng biệt ở mặt sau (BSB) chạy ở tốc độ cao hơn bus chính bên ngoài (mặt trước). BSB dành riêng cho cache L2.	Pentium Pro
Một số ứng dụng xử lý các cơ sở dữ liệu không lồ và phải có quyền truy cập nhanh vào số lượng lớn dữ liệu. Các bộ nhớ trên chip là quá nhỏ.	Di chuyển cache L2 lên trên chip xử lý.	Pentium II
	Thêm bộ nhớ cache L3 bên ngoài.	Pentium III
	Di chuyển cache L3 lên trên chip xử lý.	Pentium 4

Hình 4.18 cho thấy sơ đồ giản lược của tổ chức Pentium 4, nhấn mạnh sự có mặt của ba cấp độ cache. Bộ xử lý bao gồm bốn thành phần chính:

- Khối truy xuất/giải mã: Truy xuất các lệnh chương trình theo trình tự từ cache L2, giải mã chúng thành một chuỗi các vi lệnh, và lưu kết quả vào cache lệnh L1.
- Logic thi hành không theo trình tự: Lập kế hoạch thực hiện các hoạt động vi mô phụ thuộc vào dữ liệu và tính sẵn có của tài nguyên; do đó, các hoạt động vi mô có thể được lập kế hoạch để thực hiện theo một thứ tự khác với chúng đã được tìm nạp từ dòng lệnh. Theo thời gian cho phép, đơn vị này lập kế hoạch thực hiện đầu cơ của các hoạt động vi mô có thể được yêu cầu trong tương lai.

4.5. CÂU HỎI

1. Phân biệt ba kỹ thuật: truy cập tuần tự, truy cập trực tiếp, và truy cập ngẫu nhiên?
2. Mối quan hệ giữa thời gian truy cập, giá cả bộ nhớ, và dung lượng là gì?
3. Nguyên lý cục bộ của tham chiếu ảnh hưởng thế nào đến việc sử dụng nhiều cấp độ bộ nhớ?
4. Phân biệt ba kỹ thuật: ánh xạ trực tiếp, ánh xạ kết hợp, và ánh xạ kết hợp tập hợp là gì?
5. Đối với cache ánh xạ trực tiếp, một địa chỉ bộ nhớ chính gồm bao nhiêu trường? Là những trường gì?

6. Đối với cache kết hợp, một địa chỉ bộ nhớ chính gồm bao nhiêu trường? Là những trường gì?
7. Đối với cache kết hợp tập hợp, một địa chỉ bộ nhớ chính gồm bao nhiêu trường? Là những trường gì?
8. Phân biệt write through và write back?
9. Ưu và nhược điểm của cache phân chia là gì?

Chương 5. BỘ NHỚ TRONG

5.1. BỘ NHỚ BÁN DẪN

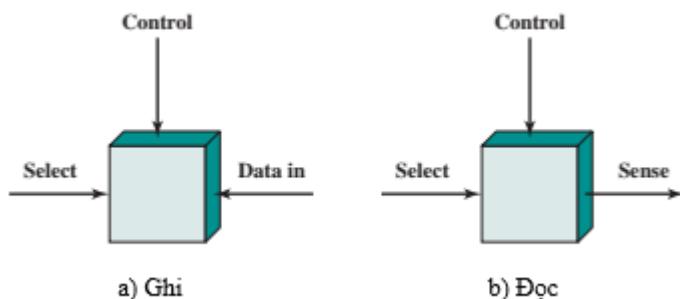
Trong các máy tính trước đây, bộ nhớ máy tính thường dưới dạng các vòng tròn từ tính được gọi là lõi (core). Ngày nay, với sự phát triển của công nghệ vi điện tử, các bộ nhớ chính được sản xuất dưới dạng chip bán dẫn đã thay thế bộ nhớ từ tính. Trong phần này, chúng ta sẽ tìm hiểu công nghệ bộ nhớ bán dẫn.

5.1.1. Tô chức

Thành phần cơ bản của **bộ nhớ bán dẫn** là ô nhớ (memory cell). Mặc dù có nhiều công nghệ điện tử được sử dụng trong đó, tất cả các tế bào bộ nhớ bán dẫn đều có các tính chất chung như sau:

- Chúng có hai trạng thái ổn định biểu diễn tương ứng các số nhị phân 1 và 0.
- Chúng có khả năng được *ghi* vào (ít nhất một lần) trước khi thiết lập trạng thái ổn định.
- Chúng có khả năng đọc để cảm nhận được trạng thái.

Hình 5.1 mô tả hoạt động của một ô nhớ. Thông thường, một ô nhớ có ba dây dẫn nối vào có khả năng truyền tải được tín hiệu điện. Đường **Select** được dùng để chọn một ô nhớ để đọc hoặc viết. Đường **Control** chỉ ra hoạt động đọc hoặc ghi đối với ô nhớ đó. Với hoạt động ghi, đường **Data-in** sẽ đưa vào trạng thái 0 hoặc 1. Với hoạt động đọc, đường **Sense** sẽ cảm nhận trạng thái của ô nhớ và chuyển thành tín hiệu tương ứng đi ra. Chi tiết về tô chức bên trong, chức năng của các ô nhớ phụ thuộc vào công nghệ mạch tích hợp sử dụng và nằm ngoài phạm vi của cuốn sách này. Ở đây, chúng ta sẽ coi nó là một ô nhớ riêng biệt có thể thực hiện các hoạt động đọc/ghi lên đó.



Hình 5.1 Các hoạt động của ô nhớ

5.1.2. DRAM và SRAM

Tất cả các loại bộ nhớ mà chúng ta sẽ tìm hiểu trong chương này là đều có cơ chế truy cập ngẫu nhiên: truy cập trực tiếp vào các vị trí trong bộ nhớ thông qua logic địa chỉ nối dây.

Bảng 5.1 liệt kê các loại bộ nhớ bán dẫn chính. Phổ biến nhất là bộ nhớ RAM – bộ nhớ truy cập ngẫu nhiên (*random-access memory*). Thực tế, đây chỉ là cách đặt tên, còn tất cả các loại bộ nhớ được liệt kê trong bảng đều có cơ chế truy cập ngẫu nhiên. Đặc điểm để phân biệt các loại bộ nhớ trong đó là bộ nhớ RAM cho phép đọc/ghi dữ liệu vào nó một

cách dễ dàng và nhanh chóng. Cả hai hoạt động đọc và ghi đều được thực hiện thông qua việc sử dụng tín hiệu điện.

Bảng 5.1 Các loại bộ nhớ bán dẫn

Loại bộ nhớ	Nhóm	Khả năng xóa	Cơ chế ghi	Tính chất	
Bộ nhớ truy cập ngẫu nhiên (RAM)	Bộ nhớ đọc-ghi	Bằng điện, mức byte	Điện	Điện động	
Bộ nhớ chỉ đọc (ROM)	Bộ nhớ chỉ đọc	Không được	Mặt nạ	Điện tĩnh	
ROM lập trình được (PROM)					
PROM xóa được	Bộ nhớ hầu như chỉ đọc	Tia UV, mức chip	Bằng điện		
PROM xóa được bằng điện		Bằng điện, mức byte			
Bộ nhớ Flash		Bằng điện, mức khối			

Một đặc điểm khác của bộ nhớ RAM khác với các bộ nhớ khác là dữ liệu ghi trên nó không ổn định khi không có nguồn điện duy trì (tính điện động). Do đó, RAM phải được cấp nguồn liên tục. Nếu nguồn cấp bị gián đoạn, dữ liệu trên RAM sẽ bị mất nên thông thường RAM chỉ có thể được sử dụng để lưu trữ dữ liệu tạm thời. Hai loại RAM truyền thống được sử dụng trong máy tính là DRAM và SRAM.

5.1.2.1. RAM ĐỘNG

Có hai công nghệ RAM chính là: RAM động và RAM tĩnh. RAM động (DRAM – Dynamic RAM) có các ô nhớ lưu trữ dữ liệu dưới dạng các mạch nạp điện vào tụ. Việc có hoặc không có điện tích trên tụ tương ứng với một giá trị nhị phân 1 hoặc 0. Bởi vì điện tích trên tụ điện có xu hướng xả ra tự nhiên, RAM động đòi hỏi phải có dòng làm tươi định kỳ để duy trì lưu trữ dữ liệu.

Hình 5.2a là sơ đồ một ô nhớ DRAM điển hình. Đường địa chỉ được kích hoạt khi có một hoạt động đọc hoặc ghi đối với ô nhớ này. Transistor hoạt động như một công tắc được đóng lại (cho phép dòng điện đi qua) nếu điện áp được đặt lên đường địa chỉ và mở (không có dòng điện đi qua) nếu không có điện áp trên đường địa chỉ.

Đối với hoạt động ghi, một tín hiệu điện áp được đặt lên đường Bit line; điện áp cao tương ứng với bit 1 còn điện áp thấp đại diện cho bit 0. Tín hiệu địa chỉ đưa vào đường địa chỉ sẽ đóng mạch công tắc cho phép nạp điện vào tụ.

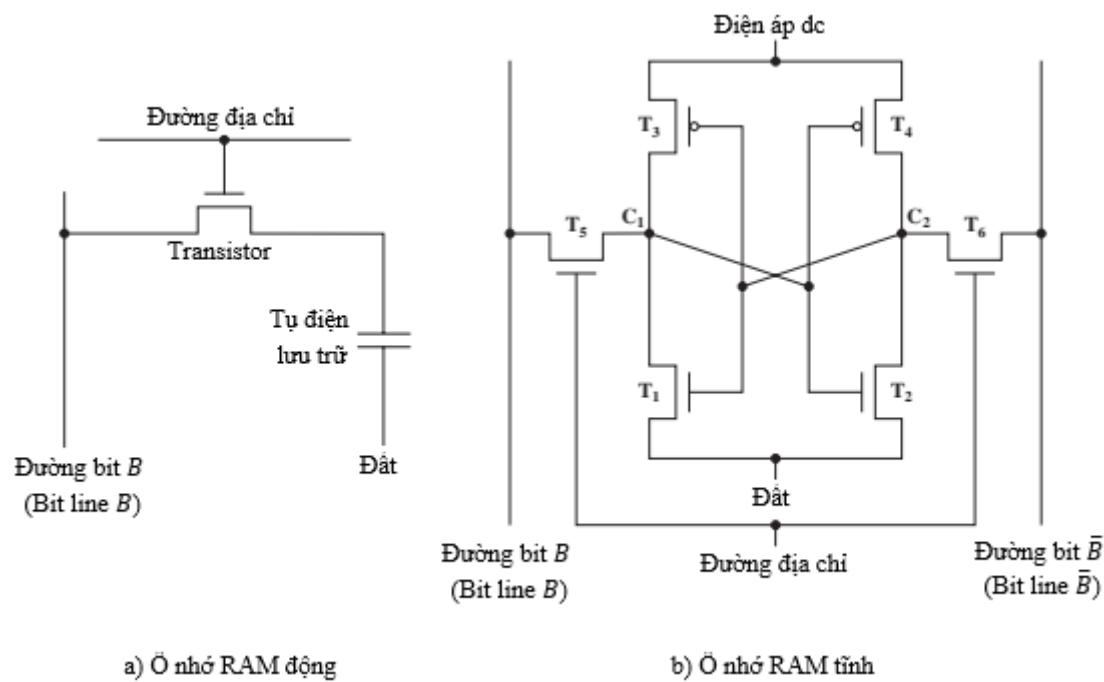
Đối với hoạt động đọc, khi đường địa chỉ có tín hiệu, transistor sẽ đóng, điện tích lưu trữ trên tụ điện sẽ phóng ra đường bit line và đi đến một bộ khuếch đại cảm nhận. Bộ khuếch đại cảm nhận so sánh điện áp tụ điện với giá trị tham chiếu và xác định xem ô nhớ

đó chứa giá trị logic 1 hay 0. Tín hiệu đọc ra từ ô nhớ sẽ giải phóng tụ điện do đó cần phải nạp lại điện áp cho tụ để khôi phục lại giá trị lưu trữ cho ô nhớ.

Mặc dù các ô nhớ DRAM lưu trữ một bit (0 hoặc 1), về cơ bản đây là một thiết bị tương tự. Các tụ điện có thể lưu trữ bất kỳ giá trị điện tích nào trong một khoảng nhất định; do đó cần có một giá trị ngưỡng xác định xem điện tích đó có giá trị 1 hoặc 0.

5.1.2.2. RAM TĨNH

RAM tĩnh (SRAM – Static RAM) là một thiết bị kỹ thuật số sử dụng các thành phần logic tương tự như trong bộ vi xử lý. Trong một SRAM, các giá trị nhị phân được lưu trữ bằng các mạch logic flip-flop truyền thống. Bộ nhớ RAM tĩnh lưu trữ dữ liệu lâu dài miễn là có nguồn điện được cấp cho nó.



Hình 5.2 Cấu trúc ô nhớ bộ nhớ thông thường

Hình 5.2b là một ô nhớ SRAM điển hình. Bốn transistor T_1, T_2, T_3, T_4 được kết nối với nhau tạo ra trạng thái logic ổn định. Trong trạng thái logic 1, điểm C_1 ở mức cao và điểm C_2 ở mức thấp; T_1 và T_4 tắt và T_2 và T_3 đang bật. Với trạng thái logic 0, điểm C_1 ở mức thấp và điểm C_2 ở mức cao; T_1 và T_4 bật và T_2 và T_3 tắt. Cả hai trạng thái đều ổn định miễn là điện áp dc được cấp liên tục cho ô nhớ.

Tương tự DRAM, đường địa chỉ SRAM được sử dụng để mở hoặc đóng công tắc. Đường địa chỉ điều khiển hai transistor T_5 và T_6 . Khi tín hiệu địa chỉ được đưa vào đường này, hai bóng bán dẫn được đóng, cho phép một hoạt động đọc hoặc ghi lên ô nhớ. Đối với hoạt động ghi, giá trị cần ghi vào được đưa vào đường B , trong khi giá trị bù của nó được đưa vào đường \bar{B} . Các giá trị này đặt các transistor T_1, T_2, T_3, T_4 ở vào một trạng thái thích hợp. Đối với hoạt động đọc, giá trị bit được đọc từ đường B .

5.1.2.3. SO SÁNH SRAM VÀ DRAM

Cả RAM tĩnh và RAM động đều là bộ nhớ điện động; nghĩa là nguồn điện phải liên tục được cấp cho bộ nhớ để duy trì các giá trị bit. Một ô nhớ của DRAM đơn giản hơn và nhỏ hơn ô nhớ của SRAM. Do đó, DRAM có mật độ dày hơn (do các ô nhớ nhỏ hơn = nhiều ô nhớ hơn trên một đơn vị diện tích) và ít tốn kém hơn SRAM tương ứng. Mặt khác, DRAM cần có các mạch làm tươi. Tuy nhiên, đối với những bộ nhớ lớn hơn, chi phí mạch làm tươi sẽ được bù lại bởi chi phí của các ô nhớ DRAM nhỏ hơn so với SRAM. Do đó, DRAM thường được sử dụng nhiều hơn để thực hiện các bộ nhớ có dung lượng lớn. Một điểm cuối cùng đó là tốc độ của SRAM nhanh hơn DRAM nên SRAM thường được sử dụng cho bộ nhớ cache (cả cache trên chip hoặc cache ngoài chip) còn DRAM được sử dụng cho bộ nhớ chính.

5.1.3. Các loại ROM – Bộ nhớ chỉ đọc

Bộ nhớ chỉ đọc (ROM – Read only memory) lưu trữ dữ liệu vĩnh viễn không thể thay đổi. ROM là bộ nhớ điện tĩnh do dữ liệu không bị mất đi ngay cả khi không có nguồn điện duy trì. Mặc dù có thể đọc dữ liệu từ ROM, nhưng không thể ghi dữ liệu mới vào đó. Ứng dụng quan trọng của ROM là vi chương trình và một số ứng dụng khác bao gồm:

- Thư viện các hàm thường xuyên được sử dụng
- Các chương trình hệ thống
- Các bảng chức năng

Đối với một số yêu cầu nhất định, ưu điểm của ROM là dữ liệu hoặc chương trình được lưu trữ vĩnh viễn và không bao giờ được nạp từ một thiết bị lưu trữ thứ hai.

ROM được chế tạo giống như các chip mạch tích hợp khác, dữ liệu được ghi vào chip như một phần của quá trình chế tạo. Điều này có phát sinh hai vấn đề:

- Chi phí cho việc ghi dữ liệu vào tương đối lớn dù rằng hàng ngàn bản ROM được sản xuất
- Không có chỗ cho sai sót. Nếu chỉ một bit bị ghi sai thì toàn bộ lô ROM sẽ bị hủy bỏ.

Khi chỉ cần một số lượng ít ROM lưu trữ nội dung nhất định, một loại ROM thay thế rẻ hơn là **PROM** (**ROM có thể lập trình được – Programmable ROM**). Giống như ROM, PROM là **bộ nhớ điện tĩnh** và chỉ có thể được ghi vào một lần. Quá trình ghi PROM được thực hiện bằng điện và có thể được thực hiện bởi nhà cung cấp hoặc khách hàng (không cần ghi trong quá trình sản xuất). Cần có một thiết bị đặc biệt để ghi hoặc "lập trình" lên chip. PROM linh hoạt và tiện lợi còn ROM thì phù hợp với việc sản xuất hàng loạt.

Một biến thể khác của bộ nhớ chỉ đọc là **bộ nhớ hầu như chỉ đọc (Read-mostly memory)**. Bộ nhớ dạng này phù hợp với các ứng dụng trong đó hoạt động đọc nhiều hơn rất nhiều so với hoạt động ghi nhưng cần lưu trữ dạng điện tĩnh. Có ba loại phổ biến: EPROM, EEPROM và bộ nhớ flash.

Bộ nhớ EPROM (Erasable Programmable Read-Only Memory - Bộ nhớ chỉ đọc, Lập trình được, Xóa được) có thể đọc và ghi bằng điện giống như PROM. Tuy nhiên, trước khi thực hiện hoạt động ghi, tất cả các ô nhớ phải được xoá về trạng thái ban đầu bằng cách

chiếu một tia cực tím mạnh qua một cửa sổ được thiết kế trên chip. Quá trình xóa này có thể thực hiện liên tục; mỗi lần xóa mất đến khoảng 20 phút. Do đó, nội dung trong EPROM có thể xóa đi, ghi lại nhiều lần và, giống như ROM và PROM, dữ liệu của nó có thể lưu trữ vĩnh viễn. Với dung lượng lưu trữ tương đương, EPROM đắt hơn PROM, nhưng ưu điểm là khả năng cập nhật lại dữ liệu nhiều lần.

Một công nghệ ROM khác là cũng rất phổ biến là bộ nhớ EEPROM (Bộ nhớ chỉ đọc, xóa được bằng điện). Với bộ nhớ này, các nội dung mới được ghi vào mà không cần xóa các nội dung trước đó; chỉ cập nhật địa chỉ byte hoặc các byte. Quá trình ghi diễn ra lâu hơn đáng kể so với quá trình đọc khoảng vài trăm micro giây cho mỗi byte. EEPROM kết hợp ưu điểm của tính điện tĩnh với tính linh hoạt trong việc cập nhật dữ liệu, sử dụng các đường bus dữ liệu, địa chỉ, điều khiển thông thường. EEPROM đắt hơn EPROM và mật độ bit cũng thưa hơn (ít bit hơn trong mỗi chip).

Một dạng bộ nhớ bán dẫn khác là bộ nhớ flash. Được giới thiệu lần đầu vào giữa những năm 1980, bộ nhớ flash nằm giữa EPROM và EEPROM về cả chi phí và chức năng. Giống như EEPROM, bộ nhớ flash sử dụng công nghệ xóa bằng điện có thể xóa toàn bộ bộ nhớ trong một hoặc vài giây, nhanh hơn nhiều so với EPROM. Ngoài ra, có thể xóa một khối bộ nhớ chứ không nhất định phải xóa toàn bộ chip. Tuy nhiên, bộ nhớ này không cho phép xóa mức byte. Giống như EPROM, bộ nhớ flash sử dụng chỉ một transistor cho mỗi bit do đó chip flash mật độ cao hơn EEPROM.

5.1.4. Chip Logic

Cũng như các sản phẩm mạch tích hợp, bộ nhớ bán dẫn được đóng gói trong chip (Hình 2.7) dưới dạng một ma trận các ô nhớ.

Trong hệ thống phân cấp bộ nhớ nói chung, chúng ta thấy rằng có sự tương quan giữa tốc độ, dung lượng và chi phí. Sự tương quan này cũng đóng một vai trò quan trọng trong việc thiết kế tổ chức các ô nhớ trên một chip và logic chức năng của chip đó. Đối với bộ nhớ bán dẫn, một trong những vấn đề thiết kế chính là số bit dữ liệu có thể được đọc/ghi cùng một lúc. Về mặt tổ chức, sự bố trí các ô nhớ trong ma trận giống như sự sắp xếp các từ trong bộ nhớ. Ma trận được tổ chức thành W từ, mỗi từ B bit. Ví dụ, một chip nhớ dung lượng 16-Mbit có thể được tổ chức thành 1M từ 16-bit. Một loại tổ chức bộ nhớ khác là 1-bit-mỗi-chip, trong đó dữ liệu được đọc/ghi một 1 bit tại một thời điểm. Phản tiếp theo chúng ta sẽ tìm hiểu tổ chức bộ nhớ DRAM, tổ chức ROM tương tự nhưng đơn giản hơn.

Hình 5.3 minh họa một tổ chức điển hình của bộ nhớ DRAM 16-Mbit với 4-bit được đọc/ghi tại một thời điểm. Về mặt logic, ma trận bộ nhớ được tổ chức thành bốn ma trận vuông 2048×2048 phần tử. Các phần tử của ma trận được nối với nhau theo cả hai chiều ngang (hàng) và chiều dọc (cột). Một đường ngang nối tất cả các đầu Select của tất cả các ô nhớ; một đường thẳng đứng nối các đầu Data-In/Sense của các ô nhớ.

Đường địa chỉ chứa thông tin địa chỉ của từ cần được chọn. Số lượng các đường cần thiết sẽ là $\log_2 W$ (với W là số lượng từ nhớ trong bộ nhớ). Như trong ví dụ trên của chúng tôi, cần phải có ít nhất 11 đường địa chỉ để chọn ra được một trong 2048 hàng, 11 đường này được đưa vào bộ giải mã hàng; thực hiện việc chuyển từ 11 đường đầu vào thành 2048 đường đầu ra (nghĩa là từ 11 đường đầu vào, bộ giải mã này sẽ kích hoạt một đường đầu ra tương ứng trong 2048 đường nối hàng) ($2^{11} = 2048$).

11 đường địa chỉ nữa để chọn ra một trong 2048 cột (mỗi cột 4 bit). Bốn đường dữ liệu đưa dữ liệu vào hoặc ra từ ô nhớ thông qua một bộ đệm dữ liệu. Với đầu vào (ghi), bộ điều khiển bit của mỗi đường bit được kích hoạt giá trị 1 hoặc 0 theo giá trị của đường dữ liệu tương ứng. Với đầu ra (đọc), giá trị của mỗi đường bit được truyền qua bộ khuếch đại cảm nhận và đưa ra đường dữ liệu. Đường hàng chọn ra một dòng các ô nhớ trong ma trận để đọc hoặc ghi.

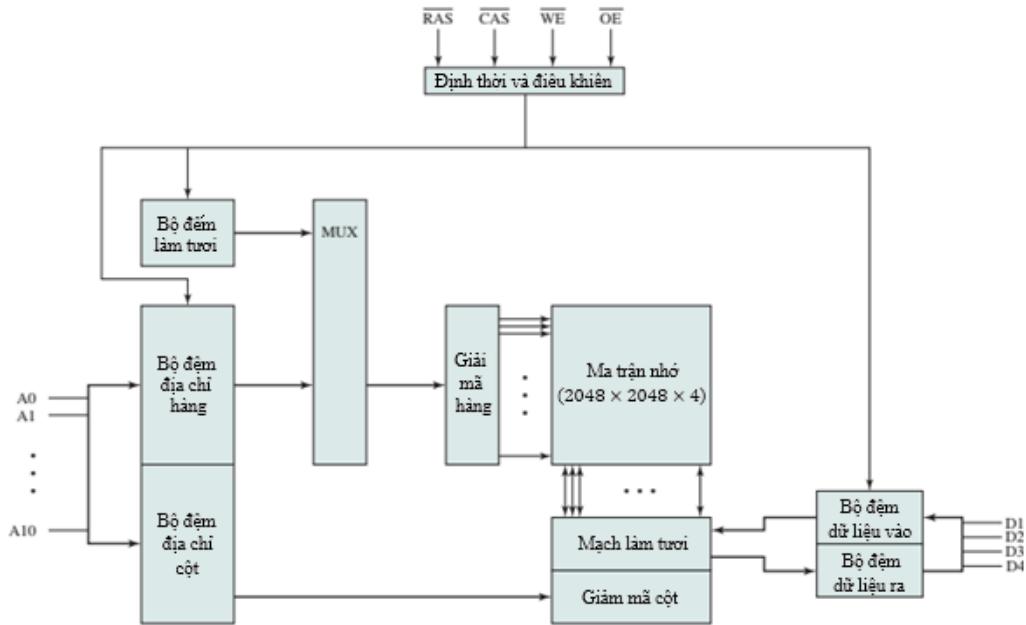
Bởi vì DRAM này chỉ có 4 bit được đọc/ghi cùng lúc tại một thời điểm nên thường ta phải kết hợp nhiều DRAM bằng cách sử dụng một bộ điều khiển bộ nhớ (memory controller) để có thể thực hiện việc đọc/ghi một từ nhớ từ bus.

Lưu ý rằng, đầu vào chip DRAM chỉ có 11 đường địa chỉ (A0-A10), một nửa số lượng cần thiết cho ma trận nhớ 2048×2048 . Việc này được thực hiện để tiết kiệm cho số chân. 22 đường địa chỉ được đưa qua một bộ chọn bên ngoài chip và được ghép thành 11 đường địa chỉ và đưa vào chip theo hai nhịp. Lần đầu, 11 tín hiệu địa chỉ được truyền cho chip để xác định địa chỉ hàng của ma trận, và sau đó là 11 tín hiệu địa chỉ khác được đưa vào cho địa chỉ cột. Các tín hiệu \overline{RAS} (tín hiệu chọn địa chỉ hàng) và \overline{CAS} (tín hiệu chọn địa chỉ cột) sẽ được đưa vào cùng với hai nhịp trên để chip có thể xác định được địa chỉ hàng hay cột.

Chân \overline{WE} (cho phép ghi) và \overline{OE} (cho phép đọc) tương ứng với việc chỉ thị thực hiện hoạt động ghi/đọc lên chip. Hai chân khác không được thể hiện trong Hình 5.3 là chân Vss (chân nối đất) và chân Vcc (chân nối nguồn).

Ngoài ra, ta có thể thấy việc ghép kênh địa chỉ cùng với việc sử dụng ma trận nhớ vuông giúp cho việc gấp bốn lần kích thước bộ nhớ với mỗi thế hệ mới của chip bộ nhớ. Thêm một chân vào (pin) nữa dành cho đường địa chỉ làm tăng gấp đôi số hàng và số cột vì vậy kích thước của bộ nhớ tăng lên 4 lần.

Trong Hình 5.3 cũng minh họa mạch làm tươi các ô nhớ của DRAM. Ta đã biết, tất cả các ô nhớ DRAM đòi hỏi phải làm tươi sau một thời gian nhất định. Trong quá trình làm tươi, các chip DRAM sẽ bị vô hiệu hóa một khoảng thời gian trong khi tất cả các ô nhớ dữ liệu được làm tươi. Bộ đếm làm tươi sẽ đến tất cả các giá trị hàng. Đôi với mỗi hàng, dữ liệu thông qua mạch làm tươi sẽ được đọc ra và ghi vào cùng một vị trí, nhờ đó, dữ liệu trong mỗi ô nhớ được khôi phục lại mức điện áp chính xác.



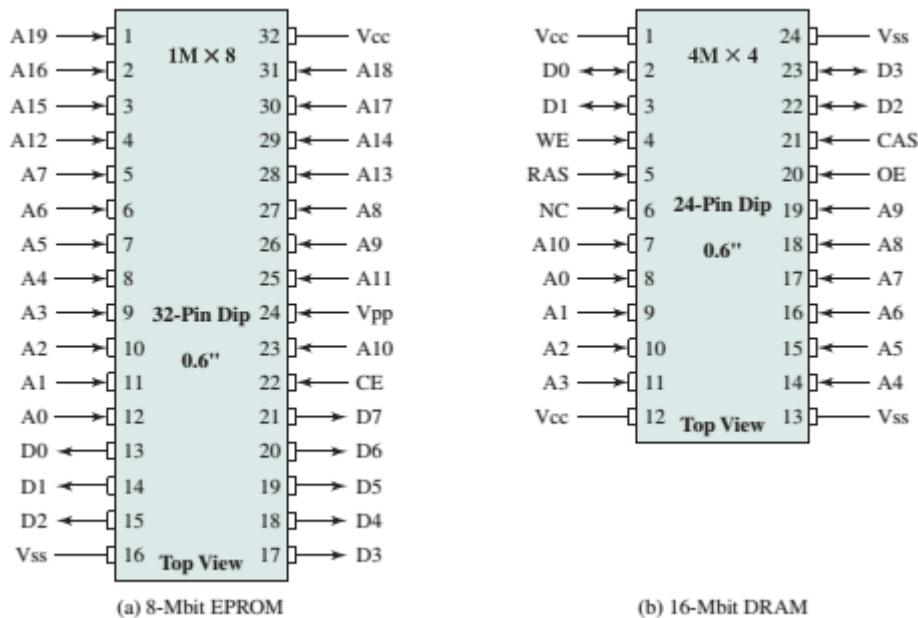
Hình 5.3 Tổ chức bộ nhớ DRAM 16Mb ($4M \times 4$)

5.1.5. Đóng gói chip

Như đã đề cập trong Chương 2, các mạch tích hợp được đóng gói trong các chip và có các chân (pin) để giao tiếp với bên ngoài.

Hình 5.4a minh họa một chip EPROM. Chip có dung lượng 8Mbit được tổ chức thành $1M \times 8$. Số chân của chip là 32 chân với kích thước tiêu chuẩn. Các chân hỗ trợ các đường tín hiệu sau:

- Địa chỉ từ nhớ được truy cập. Với 1M từ nhớ cần tổng cộng 20 chân địa chỉ: $A_0 - A_{19}$
- 8 đường dữ liệu: $D_0 - D_7$
- Chân cấp nguồn V_{cc}
- Chân nối đất: V_{ss}
- Chân CE (Chip Enable – cho phép chip): vì có thể có nhiều chip cùng kết nối với cùng bus địa chỉ, chân CE sẽ chỉ ra địa chỉ đó có dành cho chip này hay chip khác. Chân CE được kích hoạt bởi một bộ logic nối với các bit trọng số cao của bus địa chỉ (ví dụ trong trường hợp này các bit địa chỉ trên A_{19})
- Chân V_{pp} (Điện áp chương trình) được cung cấp trong quá trình lập trình (các hoạt động ghi).



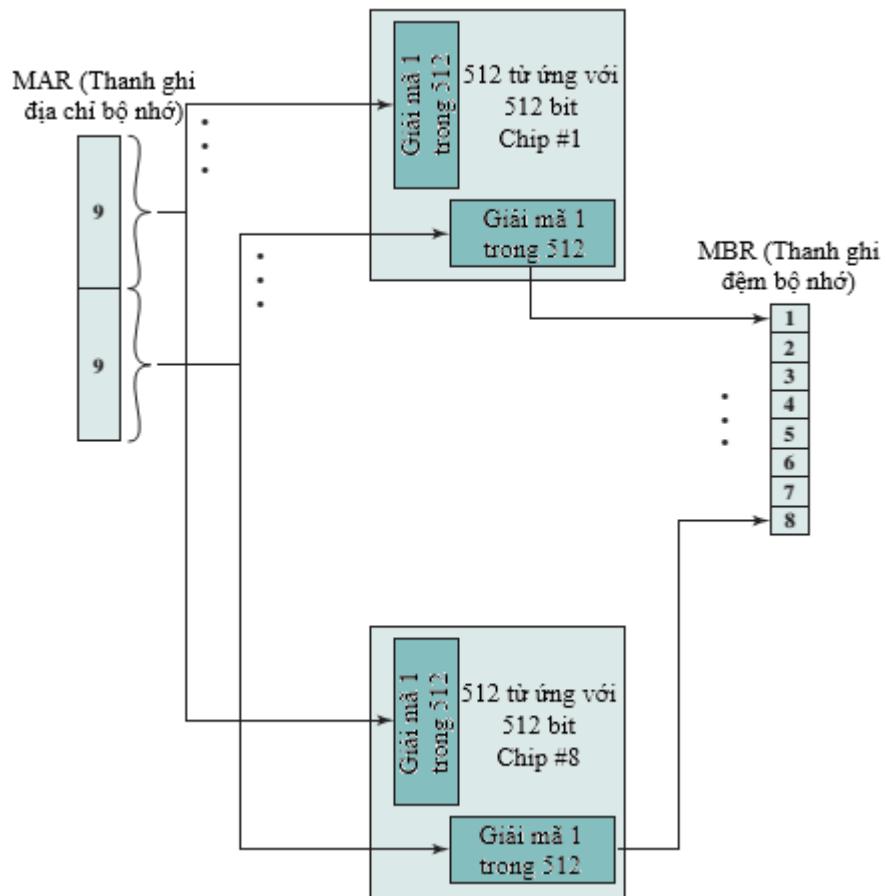
Hình 5.4 Sơ đồ chân và tín hiệu của các chip bộ nhớ

Hình 5.4b là một cấu hình DRAM 16-Mbit điển hình. Chip này được tổ chức dưới dạng $4M \times 4$. Có một số điểm khác biệt so với một chip ROM ở hình a như sau: Do RAM là bộ nhớ đọc/ghi nên các chân dữ liệu ($D_0 \div D_3$) có dạng đầu vào/đầu ra (mũi tên hai chiều). Các chân WE (Write Enable – Cho phép ghi) và OE (Output Enable – cho phép đưa dữ liệu ra) sẽ là các tín hiệu quyết định hoạt động ghi hay đọc. Do DRAM được truy cập theo hàng và cột và dữ liệu địa chỉ đưa vào theo phương thức ghép kênh nên chỉ cần có 11 chân địa chỉ vào để xác định một vị trí bộ nhớ ($2^{11} \times 2^{11} = 2^{22} = 4M$). Chức năng các chân RAS và CAS đã được trình bày ở trên. Cuối cùng, chân NC (No Connect – không có kết nối) được thêm vào để số chân của chip là số chẵn.

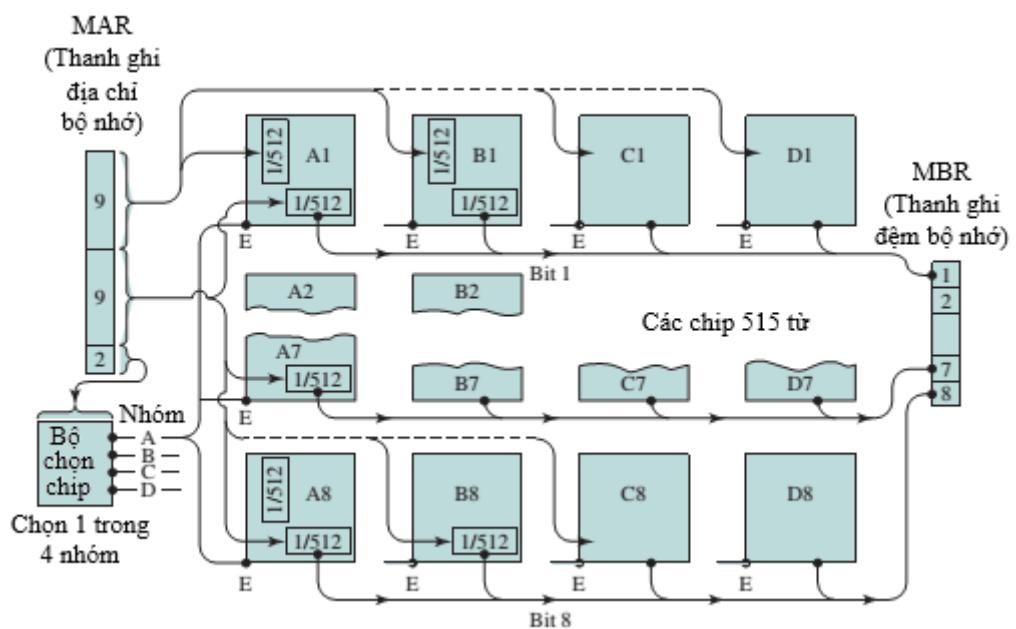
5.1.6. Tổ chức module nhớ

Nếu một chip RAM tổ chức dưới dạng 1 bit/một vị trí nhớ, thì rõ ràng chúng ta sẽ cần ít nhất số lượng chip bằng số bit của một từ nhớ. Ví dụ tổ chức một module bộ nhớ trong hình 5.5 có dung lượng $256K \times 8b$ ($256K$ từ nhớ, mỗi từ 8-bit). Với $256K$ từ nhớ ta cần phải có 18-bit địa chỉ. Địa chỉ này được đưa vào 8 chip $256K \times 1b$ (mỗi chip cung cấp một đầu dữ liệu vào/ra 1 bit).

Tổ chức này hoạt động miễn là kích thước của bộ nhớ bằng số bit trong mỗi chip. Trong trường hợp cần phải xây dựng một module bộ nhớ có kích thước lớn hơn, ta phải tổ chức dưới dạng một ma trận chip. Hình 5.6 minh họa một tổ chức bộ nhớ kích thước $1M$ từ nhớ, mỗi từ 8b. Trong đó ta có bốn cột, mỗi cột có 8 chip kích thước $256K \times 1b$. Như vậy, kích thước một cột là $256K \times 8b$, hay $256K$ từ nhớ (mỗi từ 8b). Với $1M$ từ nhớ ta cần 20 đường địa chỉ, trong đó: 18 bit thấp được chuyển đến tất cả 32 chip, 2 bit cao được đưa vào một bộ giải mã ra thành một tín hiệu CS tới một trong bốn cột của module.



Hình 5.5 Tô chức bộ nhớ 256KByte



Hình 5.6 Tô chức bộ nhớ 1MByte

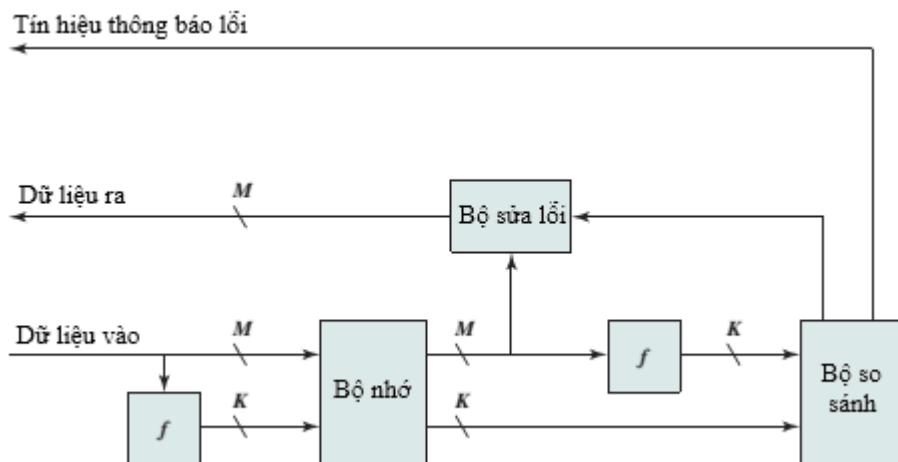
5.1.7. Tổ chức bộ nhớ đan xen

Bộ nhớ chính gồm một tập các chip nhớ DRAM. Một số chip có thể được nhóm lại với nhau để tạo thành một dải bộ nhớ. Các dải bộ nhớ này có thể được tổ chức theo cách “đan xen”: Mỗi dải nhớ có thể phục vụ yêu cầu đọc hoặc ghi một cách độc lập, như vậy một hệ thống nhớ gồm K dải nhớ có thể phục vụ K yêu cầu cùng một lúc, tăng hiệu năng đọc và ghi của bộ nhớ lên K lần. Nếu các từ liên tiếp của bộ nhớ được lưu giữ tại các dải nhớ khác nhau thì rõ ràng việc truyền một khối dữ liệu sẽ được thực hiện song song trên cả K dải nhớ, như vậy tốc độ truyền nhanh hơn rất nhiều.

5.2. SỬA LỖI

Trong các hệ thống nhớ, một vấn đề có thể xảy ra đó là lỗi. Có hai loại lỗi là: lỗi cứng và lỗi mềm. **Lỗi cứng** là lỗi vật lý vĩnh viễn do đó một hoặc nhiều ô nhớ không thể lưu trữ dữ liệu: bị kẹt ở một giá trị 0 hoặc 1 hoặc không thể chuyển giữa 0 hoặc 1. Nguyên nhân của lỗi cứng là do tác động của môi trường, lỗi sản xuất hoặc do hao mòn dần trong quá trình sử dụng. **Lỗi mềm** là do một tác động ngẫu nhiên tác động đến một hoặc nhiều ô nhớ chỉ làm thay đổi nội dung mà không phá hủy chúng. Lỗi mềm có thể xảy ra do vấn đề về nguồn điện cung cấp hoặc do các hạt alpha. Những hạt này phát ra từ sự phân rã phóng xạ rất phổ biến trong máy tính vì các hạt phóng xạ được tìm thấy với số lượng nhỏ trong hầu hết các vật liệu. Cả lỗi cứng và lỗi mềm rõ ràng là các lỗi không mong muốn và hầu hết các hệ thống bộ nhớ hiện đại đều có một bộ phận thực hiện chức năng phát hiện và sửa lỗi.

Cơ chế phát hiện và sửa lỗi được minh họa trong Hình 5.7. Khi dữ liệu được ghi vào bộ nhớ, đồng thời một hàm f cũng được tính toán trên dữ liệu để sinh ra một mã. Cả mã và dữ liệu sẽ cùng được lưu trữ. Do đó, nếu lưu trữ một từ dữ liệu M-bit và mã có độ dài K bit thì kích thước thực tế cần được lưu trữ là $M + K$ bit.



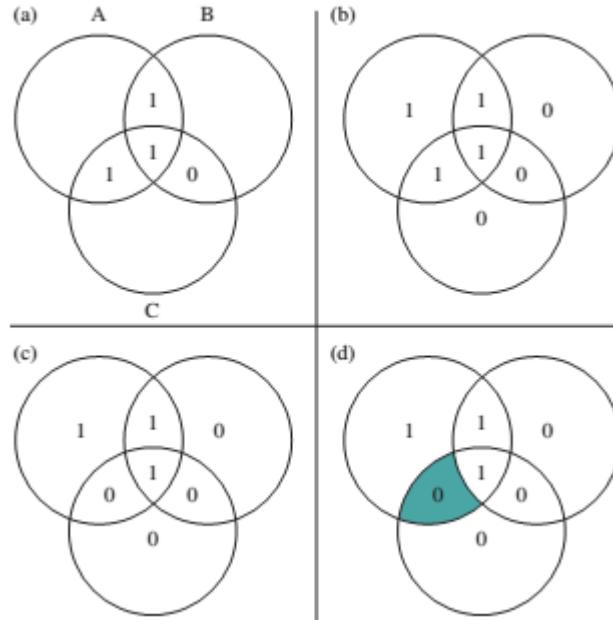
Hình 5.7 Cơ chế sửa lỗi

Khi đọc dữ liệu ra, mã được sử dụng để phát hiện và trong một số trường hợp có thể giúp sửa lỗi: dựa vào M bit dữ liệu đọc ra, người ta tính toán K bit kiểm tra mới và so sánh với K bit kiểm tra lưu trữ trong bộ nhớ, có một trong ba kết quả như sau:

- Không phát hiện lỗi. Các bit dữ liệu được truy xuất sẽ được gửi đi.

- Phát hiện lỗi và sửa lỗi (nếu có thể). Các bit dữ liệu và các **bit sửa lỗi** được đưa vào bộ sửa lỗi và đưa ra ở đầu ra một tập M bit chính xác để gửi đi.
- Phát hiện lỗi nhưng không thể sửa lỗi: Lỗi này sẽ được gửi báo cáo đến bộ xử lý.

Mã trên ta gọi là **mã sửa lỗi** đặc trưng bởi số bit lỗi mà nó có thể phát hiện và sửa.



Hình 5.8 Mã sửa lỗi Hamming

Mã sửa lỗi đơn giản nhất là **mã Hamming** do Richard Hamming đưa ra tại Phòng thí nghiệm Bell. Hình 5.8 sử dụng sơ đồ Venn để minh họa việc sử dụng mã này trên một từ nhớ 4 bit ($M = 4$). Với ba vòng tròn giao nhau ta có bảy ngăn, 4 bit dữ liệu được điền vào các ngăn bên trong (hình 5.8a). Các ngăn còn lại ta điền vào các *bit chẵn lẻ*. Mỗi bit chẵn lẻ được chọn sao cho tổng số các bit 1 trong một vòng tròn là chẵn (Hình 5.8b). Vì vậy, vòng tròn A có ba bit 1, bit chẵn lẻ trong vòng tròn đó được đặt là 1, tương tự với các vòng còn lại. Vậy, trong trường hợp có lỗi xảy ra với một bit dữ liệu (Hình 5.8c), ta có thể xác định bằng cách kiểm tra các bit chẵn lẻ, vấn đề được phát hiện trong vòng A và vòng C nhưng không trong vòng tròn B. Vậy lỗi bit chỉ có thể là một trong các ngăn của cả vòng A và C mà không nằm trong vòng B: vùng tô màu trong Hình 5.8d. Vậy, lỗi được sửa bằng cách đổi bit đó từ 0 sang 1.

Vậy trong trường hợp một từ nhớ 8-bit, ta sẽ tìm hiểu cách thực hiện một mã phát hiện và sửa một lỗi bit như dưới đây.

Đầu tiên, chúng ta phải xác định độ dài mã là bao nhiêu. Như trong Hình 5.7, đầu vào của bộ so sánh là hai giá trị có độ dài K-bit. Việc so sánh được thực hiện với từng bit tương ứng của hai đầu vào bằng cách sử dụng hàm XOR. Kết quả được gọi là *syndrome*. Mỗi bit của *syndrome* là 0 hoặc 1 tùy theo việc hai bit cùng một vị trí của hai đầu vào giống hoặc không giống.

Vì vậy, *syndrome* cũng sẽ có kích thước là K-bit và có giá trị từ 0 đến $2^K - 1$. Với giá trị 0 nghĩa là không phát hiện lỗi nào và $2^K - 1$ giá trị còn lại chỉ ra rằng dữ liệu có lỗi và

vị trí bit lỗi. Vậy, để chỉ ra được bit nào trong M-bit dữ liệu hoặc K-bit kiểm tra bị lỗi, ta phải có

$$2^K - 1 \geq M + K$$

Từ bất đẳng thức này ta có thể xác định được số bit cần thiết để sửa một lỗi với số lượng M bit dữ liệu bất kỳ. Ví dụ, trong trường hợp $M = 8$, ta có

- $K = 3: 2^3 - 1 < 8 + 3$
- $K = 4: 2^4 - 1 > 8 + 4$

Do đó, tám bit dữ liệu ta cần bốn bit kiểm tra. Bảng 5.2 liệt kê số lượng các bit kiểm tra cần thiết cho các từ nhớ có kích thước khác nhau.

Bảng 5.2 Kích thước từ tăng do việc sử dụng Mã sửa lỗi

Số bit dữ liệu	Mã SEC (Sửa một lỗi)		Mã SEC-DEC (Phát hiện hai lỗi-Sửa một lỗi)	
	Số bit kiểm tra	% Tăng	Số bit kiểm tra	% Tăng
8	4	50	5	62.5
16	5	31.25	6	37.5
32	6	18.75	7	21.875
64	7	10.94	8	12.5
128	8	6.25	9	7.03
256	9	3.52	10	3.91

Để thực hiện được việc phát hiện và sửa lỗi, một syndrome 4-bit của một từ dữ liệu 8-bit sẽ có các đặc điểm như sau:

- Nếu tất cả các bit của syndrome là 0, không có lỗi được phát hiện.
- Nếu syndrome chỉ có một bit 1 thì lỗi xảy ra ở một trong 4 bit kiểm tra. Không cần sửa lỗi.
- Nếu syndrome có nhiều hơn một bit 1 thì giá trị của syndrome sẽ cho ta biết vị trí của bit dữ liệu bị lỗi. Việc sửa lỗi sẽ được thực hiện bằng cách đảo giá trị của bit lỗi.

Vị trí bit	12	11	10	9	8	7	6	5	4	3	2	1
Số vị trí	1100	1011	1010	1001	1000	0111	0110	0101	0100	0011	0010	0001
Bit dữ liệu	D8	D7	D6	D5		D4	D3	D2		D1		
Bit kiểm tra					C8				C4		C2	C1

Hình 5.9 Vị trí các bit dữ liệu và bit kiểm tra

Để thực hiện được những điều này, các bit dữ liệu và bit kiểm tra được sắp xếp thành một từ 12-bit như trong Hình 5.9. Vị trí bit được đánh số từ 1 đến 12. Những bit có vị trí là lũy thừa của 2 là những bit kiểm tra. Các bit kiểm tra được tính như sau (trong đó toán tử \oplus đại diện cho phép toán XOR):

$$\begin{aligned} C1 &= D1 \oplus D2 \oplus D4 \oplus D5 \oplus D7 \\ C2 &= D1 \oplus D3 \oplus D4 \oplus D6 \oplus D7 \\ C4 &= D2 \oplus D3 \oplus D4 \oplus D8 \\ C8 &= D5 \oplus D6 \oplus D7 \oplus D8 \end{aligned}$$

Ta thấy trong Hình 5.9, *số vị trí* của mỗi bit kiểm tra đều có một bit 1 với trọng số khác nhau. Các bit dữ liệu có vị trí 3, 5, 7, 9, 11 (D1, D2, D4, D5, D7): bit có trọng số nhỏ nhất trong *số vị trí* của chúng đều là 1, tương ứng với C1. Các bit vị trí 3, 6, 7, 10, và 11 chứa 1 ở bit thứ hai trong *số vị trí*, giống như C2. Tương tự với C4: 5, 6, 7, 12 và C8: 9, 10, 11, 12. Vậy bit ở vị trí n được kiểm tra bởi các bit Ci sao cho $\sum i = n$. Ví dụ, vị trí 7 được kiểm tra bởi các bit ở vị trí 4, 2, và 1 ($7 = 4 + 2 + 1$), hay vị trí 11 sẽ được kiểm tra bởi các bit vị trí: 8, 2, 1

Để rõ hơn chúng ta theo dõi ví dụ sau: giả sử ta có 8 bit dữ liệu là 00111001, với bit dữ liệu D1 ở vị trí bên phải. Ta tính được các bit kiểm tra như sau:

$$\begin{aligned} C1 &= 1 \oplus 0 \oplus 1 \oplus 1 \oplus 0 = 1 \\ C2 &= 1 \oplus 0 \oplus 1 \oplus 1 \oplus 0 = 1 \\ C4 &= 0 \oplus 0 \oplus 1 \oplus 0 = 1 \\ C8 &= 1 \oplus 1 \oplus 0 \oplus 0 = 0 \end{aligned}$$

Giả sử bit dữ liệu D3 bị lỗi và đổi từ 0 thành 1. Khi các bit kiểm tra được tính lại, chúng ta có

$$\begin{aligned} C1 &= 1 \oplus 0 \oplus 1 \oplus 1 \oplus 0 = 1 \\ C2 &= 1 \oplus 1 \oplus 1 \oplus 1 \oplus 0 = 0 \\ C4 &= 0 \oplus 1 \oplus 1 \oplus 0 = 1 \\ C8 &= 1 \oplus 1 \oplus 0 \oplus 0 = 0 \end{aligned}$$

Khi cho các bit kiểm tra mới và cũ qua bộ so sánh, ta được syndrome như sau:

$$\begin{array}{rcccc} & C8 & C4 & C2 & C1 \\ & 0 & 1 & 1 & 1 \\ \oplus & 0 & 0 & 0 & 1 \\ \hline & 0 & 1 & 1 & 0 \end{array}$$

Kết quả là 0110 chỉ ra rằng bit vị trí 6 tức bit dữ liệu D3 là bị lỗi. Việc tính toán trên được tổng kết trong Hình 5.10. Các bit dữ liệu và bit kiểm tra được điền vào 12 vị trí. 4 bit dữ liệu có giá trị 1 được tô màu, ta thực hiện phép toán XOR với số vị trí của các bit đó ta được mã Hamming là 0111. Dữ liệu được lưu trữ là 001101001111. Giả sử, có lỗi ở D3, bit vị trí 6: thay đổi từ 0 thành 1. Khối dữ liệu được đọc ra từ bộ nhớ là 001101101111, với mã Hamming là 0111. Tính XOR của mã Hamming với tất cả các giá trị vị trí của các bit 1 ta được: 0110. Kết quả này khác không nghĩa là dữ liệu này đã bị lỗi và lỗi ở bit vị trí 6.

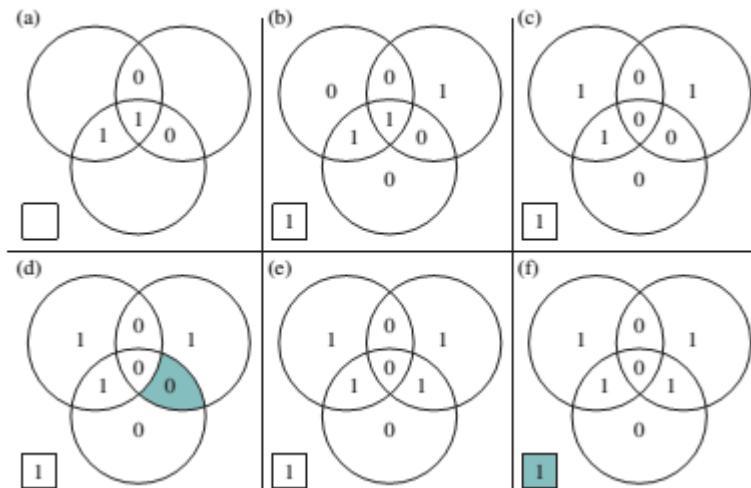
Vị trí bit	12	11	10	9	8	7	6	5	4	3	2	1
Số vị trí	1100	1011	1010	1001	1000	0111	0110	0101	0100	0011	0010	0001
Bit dữ liệu	D8	D7	D6	D5		D4	D3	D2		D1		
Bit kiểm tra					C8				C4		C2	C1
Từ được lưu trữ	0	0	1	1	0	1	0	0	1	1	1	1
Từ được lấy ra	0	0	1	1	0	1	1	0	1	1	1	1
Số vị trí	1100	1011	1010	1001	1000	0111	0110	0101	0100	0011	0010	0001
Mã kiểm tra					0				0		0	1

Hình 5.10 Tính toán mã kiểm tra

Mã chúng ta vừa tìm hiểu là **mã SEC – mã sửa một lỗi duy nhất**. Tuy nhiên, các bộ nhớ bán dẫn thường sử dụng **mã SEC-DED – mã phát hiện hai lỗi, sửa một lỗi**. Như trong bảng 5.2, mã SEC-DED nhiều hơn so với mã SEC một bit.

Hình 5.11 minh họa hoạt động của mã SEC-DEC với một từ dữ liệu 4-bit được điền vào các ngăn. Một mã Hamming SEC được tính toán tương tự như đã nói ở trên (Hình 5.11b). Giả sử có hai lỗi xảy ra như trong Hình 5.11c, việc kiểm tra và sửa lỗi bị sai (d) và làm trầm trọng thêm vấn đề vì đã tạo ra lỗi thứ ba (e). Để khắc phục vấn đề này, ta sẽ thêm vào bit thứ tám bằng cách thiết lập giá trị của nó sao cho số bit 1 trong sơ đồ là chẵn. Vậy như trong hình f, việc sửa lỗi ở trên đã sai vì tổng số bit 1 là lẻ, như vậy dữ liệu có nhiều hơn một bit bị lỗi và không thể sửa được.

Mã sửa lỗi làm tăng độ tin cậy của bộ nhớ nhưng lại làm tăng chi phí. Một số ví dụ thực tế với các chip nhớ như IBM 30xx sử dụng mã SEC-DED 8-bit cho 64 bit dữ liệu, kích thước của bộ nhớ tăng khoảng 12% so với kích thước người dùng có thể sử dụng được. Máy tính VAX sử dụng mã SEC-DED 7-bit cho bộ nhớ 32 bit, tăng 22%. Một số DRAM hiện đại sử dụng 9 bit kiểm tra cho 128 bit dữ liệu làm tăng kích thước 7%.



Hình 5.11 Mã Hamming SEC-DEC

5.3. TỔ CHỨC DRAM MỞ RỘNG

Như Chương 2 đã đề cập, một trong những yếu tố ảnh hưởng đến hiệu năng của một hệ thống máy tính đó là tình trạng nghẽn do việc giao tiếp giữa bộ xử lý và bộ nhớ chính. Bộ xử lý có tốc độ và hiệu năng ngày càng cao, nếu bộ nhớ không đáp ứng được nó sẽ gây tình trạng trên. Trong nhiều thập kỷ, khái niệm cơ bản của bộ nhớ chính vẫn là chip DRAM và kiến trúc này hầu như có rất ít sự thay đổi kể từ đầu những năm 1970. Do đó, việc cải thiện chip DRAM truyền thống là một trong những yêu cầu quan trọng trong quá trình phát triển của các hệ thống máy tính.

Bảng 5.3 So sánh hiệu suất của các chip DRAM thay thế

	Tần số đồng hồ	Tốc độ truy cập	Thời gian truy cập	Số lượng chân
SDRAM	166	1.3	18	168
DDR	200	3.2	12.5	184
RDRAM	600	4.8	12	162

Ở chương trước chúng ta đã có một kiến trúc giúp cải thiện hiệu suất của bộ nhớ chính DRAM đó là việc sử dụng một hoặc nhiều cache SRAM tốc độ cao. Tuy nhiên, do SRAM đắt hơn DRAM nên việc tăng kích thước SRAM quá lớn lại làm giảm lợi nhuận hệ thống.

Song song với nó, trong những năm gần đây, người ta cũng đưa ra một số cải tiến trong kiến trúc DRAM, trong đó một số sản phẩm được sử dụng rộng rãi trên thị trường là SDRAM, DDR-DRAM và RDRAM. Bảng 5.3 so sánh hiệu năng của các kiến trúc này.

5.3.1. SDRAM – DRAM đồng bộ

Một trong những loại DRAM được sử dụng rộng rãi nhất là SDRAM (DRAM đồng bộ). Không giống như DRAM truyền thống, việc trao đổi dữ liệu của SDRAM với bộ xử lý

được đồng bộ với tín hiệu đồng hồ và chạy với tốc độ cao của bộ xử lý/bus bộ nhớ mà không cần chịu trạng thái chờ đợi.

Với DRAM thông thường, bộ xử lý gửi địa chỉ và tín hiệu điều khiển đến bộ nhớ, chỉ ra vị trí cụ thể của dữ liệu trong cần được đọc ra hoặc ghi vào DRAM. Sau một khoảng trễ nhất định (thời gian truy cập) DRAM thực hiện việc ghi hoặc đọc dữ liệu. Khoảng thời gian trễ truy cập này là khoảng thời gian để DRAM thực hiện một số chức năng bên trong, ví dụ như kích hoạt các đường địa chỉ hàng và cột, cảm nhận dữ liệu hoặc đưa dữ liệu ra bộ đệm đầu ra. Bộ vi xử lý phải chờ đợi khoảng thời gian trễ này do đó làm chậm hiệu suất hệ thống.

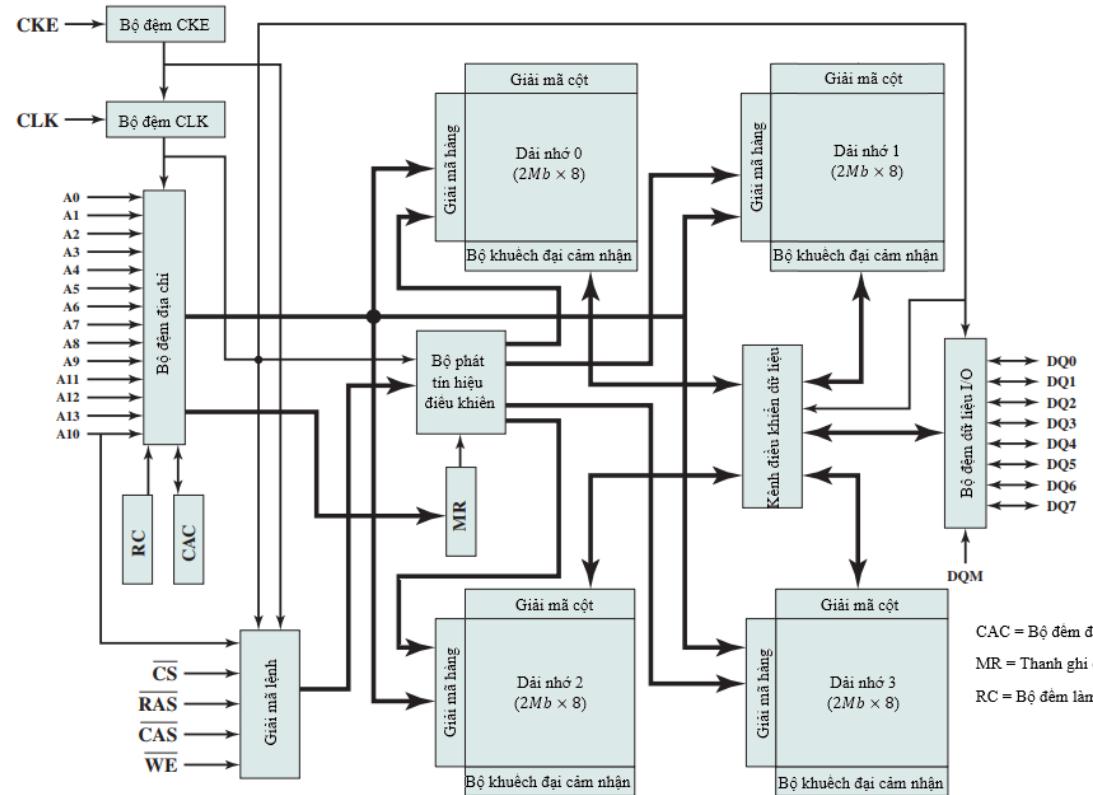
Với truy cập đồng bộ, SDRAM đưa dữ liệu vào hoặc ra dưới sự điều khiển của đồng hồ hệ thống. Bộ vi xử lý thực thi lệnh và thông tin địa chỉ. SDRAM sẽ trả lời sau một số chu kỳ đồng hồ nhất định. Như vậy, bộ xử lý có thể thực hiện các tác vụ khác trong khi SDRAM xử lý yêu cầu.

Hình 5.12 là tổ chức bộ nhớ SDRAM IBM 64-MB, Bảng 5.4 liệt kê các chân của chip. SDRAM sử dụng một chế độ truyền nhóm (burst mode) để giảm bớt thời gian thiết lập địa chỉ và thời gian nạp địa chỉ hàng và cột. Với chế độ này, một chuỗi bit có thể được truy cập một cách nhanh chóng sau lần truy cập đầu tiên.

Thanh ghi chế độ MR và bộ điều khiển là một điểm khác biệt của SDRAM so với DRAM thông thường. Nó cung cấp một cơ chế tùy chỉnh SDRAM cho phù hợp với các yêu cầu cụ thể của hệ thống. Thanh ghi MR xác định độ dài nhóm (burst): số đơn vị dữ liệu được đặt đồng bộ vào bus. Thanh ghi này cũng cho phép người lập trình có thể điều chỉnh độ trễ giữa việc nhận yêu cầu đọc và việc bắt đầu truyền dữ liệu.

SDRAM hoạt động tốt nhất khi truyền một khối dữ liệu lớn ví dụ như dữ liệu của các ứng dụng xử lý văn bản, bảng tính và các ứng dụng đa phương tiện.

Hình 5.13 là ví dụ về hoạt động của SDRAM với kích thước burst là 4 và độ trễ là 2. Lệnh Đọc nhóm bắt đầu khi tín hiệu ở các chân \overline{CS} và \overline{CAS} ở mức thấp trong khi \overline{RAS} và \overline{WE} ở mức cao tại sườn lên của xung đồng hồ. Tín hiệu địa chỉ xác định địa chỉ cột bắt đầu của burst, thanh ghi MR thiết lập kiểu burst (tuần tự hoặc đan xen) và độ dài burst (1, 2, 4, 8 hoặc toàn bộ page). Độ trễ từ lúc bắt đầu lệnh cho đến khi dữ liệu từ ô nhớ đầu tiên xuất hiện ở đầu ra bằng giá trị độ trễ \overline{CAS} đặt trong thanh ghi MR.

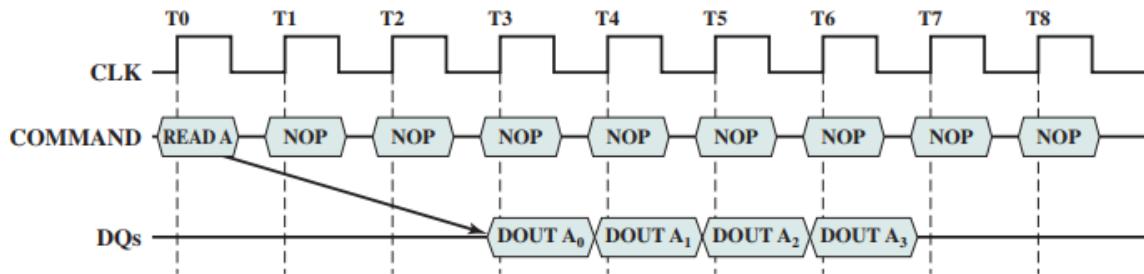


Hình 5.12 DRAM đồng bộ (SDRAM)

Bảng 5.4 Các chân SDRAM

Từ A0 đến A13	Các đầu vào địa chỉ
CLK	Đầu vào tín hiệu đồng hồ
CKE	Cho phép đồng hồ
CS	Chọn Chip
RAS	Chuyển địa chỉ hàng
CAS	Chuyển địa chỉ cột
WE	Cho phép ghi
DQ0 to DQ7	Vào/ra dữ liệu
DQM	Mặt nạ dữ liệu

CAC = Bộ đệm địa chỉ cột
MR = Thanh ghi chế độ
RC = Bộ đệm làm tươi



Hình 5.13 Tiết trình thời gian hoạt động Đọc SDRAM (kích thước burst = 4, độ trễ $\overline{CSA} = 2$)

Một phiên bản nâng cấp của SDRAM là DDR-SDRAM (SDRAM tốc độ dữ liệu gấp đôi). Với SDRAM, ta thấy mỗi chu kỳ xung nhịp một dữ liệu được gửi đi, còn với DDR SDRAM dữ liệu được gửi hai lần trong mỗi chu kỳ đồng hồ.

5.3.2. Rambus DRAM

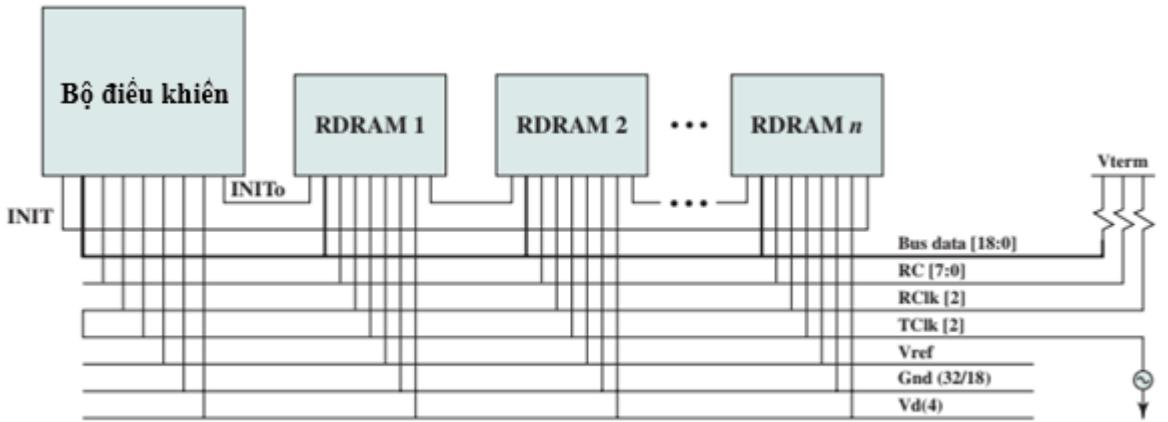
RDRAM là loại RAM được Intel sử dụng cho các bộ vi xử lý Pentium và Itanium và trở thành đối thủ cạnh tranh chính của SDRAM. Chip RDRAM đóng gói theo chiều dọc, với tất cả các chân ở một bên. Chip trao đổi dữ liệu với bộ vi xử lý thông qua 28 dây, chiều dài chip ngắn hơn 12 cm. Bus có thể định địa chỉ được đến 320 chip RDRAM với tốc độ lên đến 1.6 GBps.

Bus RDRAM truyền thông tin địa chỉ và điều khiển bằng cách sử dụng giao thức hướng khối không đồng bộ. Sau khoảng thời gian 480 ns truy cập ban đầu, nó cho tốc độ truyền dữ liệu là 1.6 GBps. Thay vì bị điều khiển bởi các tín hiệu RAS, CAS, R/W, CE như với DRAM thông thường, RDRAM nhận các yêu cầu bộ nhớ qua một bus tốc độ cao. Yêu cầu này chứa thông tin địa chỉ, loại hoạt động (đọc hoặc ghi) và số byte hoạt động yêu cầu.

Hình 5.14 minh họa cấu trúc RDRAM. Cấu trúc bao gồm một bộ điều khiển và một số module RDRAM được kết nối với nhau thông qua một bus chung. Bộ điều khiển được đặt ở một đầu của chip, các đường bus được nối song song. Bus gồm có 18 đường dữ liệu (16 đường dữ liệu thực, hai đường chẵn lẻ) hoạt động với tốc độ gấp đôi tín hiệu đồng hồ; nghĩa là 1 bit được gửi ở sườn lên và 1 bit được gửi ở sườn xuống của xung đồng hồ. Nhờ đó, tốc độ tín hiệu trên mỗi đường là 800 Mbps. Một nhóm gồm 8 đường RC được sử dụng để truyền các tín hiệu địa chỉ và điều khiển. Tín hiệu đồng hồ được truyền từ đầu bên kia đến bộ điều khiển sau đó vòng trở lại. Một module RDRAM gửi dữ liệu đến bộ điều khiển đồng bộ với tín hiệu đồng hồ, bộ điều khiển gửi dữ liệu đến một RDRAM đồng bộ với tín hiệu đồng hồ theo hướng ngược lại. Các đường bus còn lại bao gồm đường điện áp tham chiếu, đường nối đất và đường nối nguồn.

5.3.1. DDR SDRAM

Như phần trên đã đề cập, tốc độ SDRAM bị hạn chế vì nó chỉ có thể gửi dữ liệu đến bộ xử lý một lần mỗi chu kỳ đồng hồ bus. DDR SDRAM cải tiến hơn SDRAM ở chỗ nó cho phép gửi dữ liệu hai lần mỗi chu kỳ đồng hồ, một ở sườn lên và một ở sườn xuống của xung đồng hồ.



Hình 5.14 Cấu trúc RDRAM

DDR DRAM được đưa ra bởi Hiệp hội Công nghệ Solid State JEDEC và đã được nhiều công ty sản xuất cũng như được sử dụng rộng rãi trong các máy tính để bàn và máy chủ.

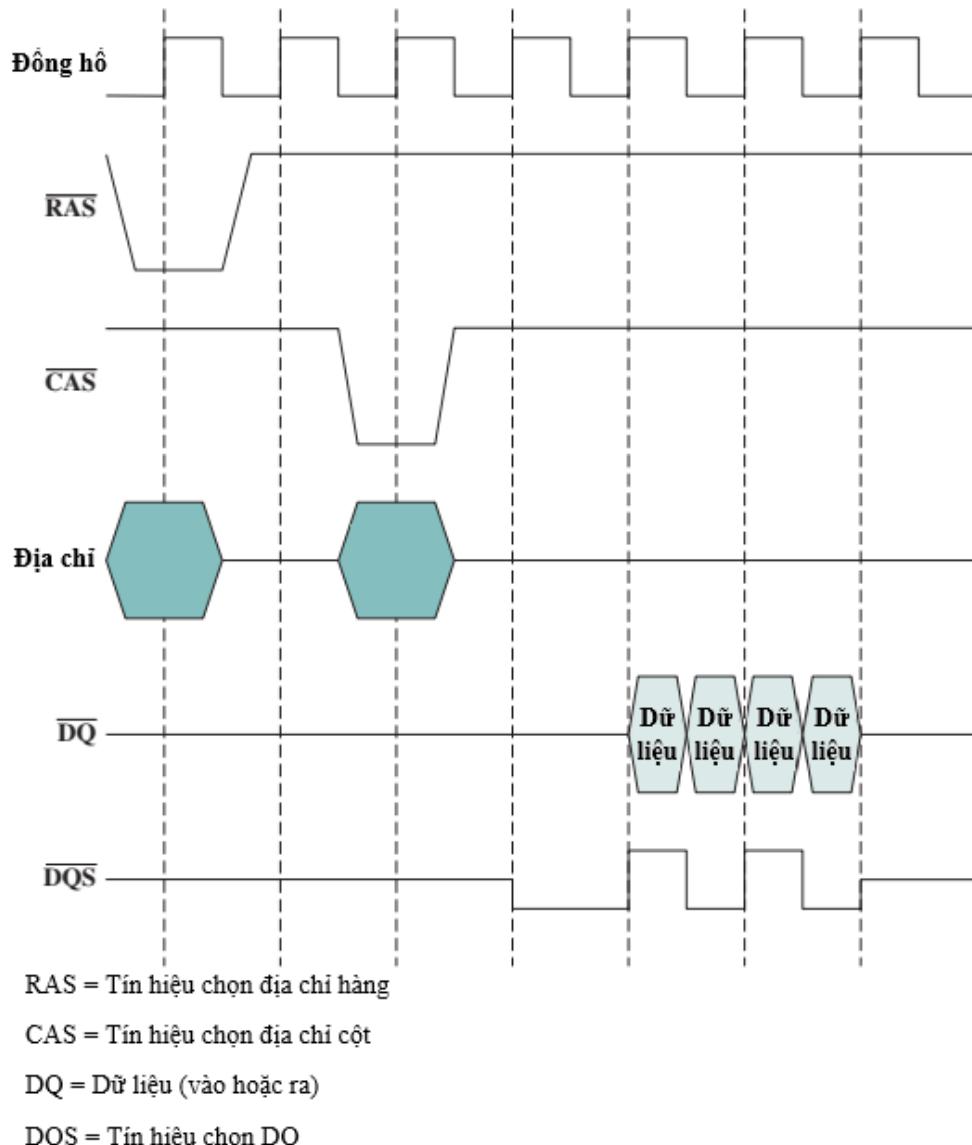
Hình 5.15 là sơ đồ thời gian hoạt động Đọc DDR. Việc truyền dữ liệu được đồng bộ với sườn lên và xuống của xung đồng hồ. Nó cũng được đồng bộ với tín hiệu dữ liệu hai chiều (DQS) do bộ điều khiển bộ nhớ phát ra khi thực hiện hoạt động Đọc và do DRAM phát ra trong quá trình Ghi. Trong quá trình thực hiện lệnh Đọc. DQS bị bỏ qua, còn giải thích về việc sử dụng DQS đối với lệnh Ghi vượt quá phạm vi của chúng tôi

Đến nay, đã có hai phiên bản cải tiến của công nghệ DDR: DDR2 và DDR3. DDR2 tăng tốc độ truyền dữ liệu bằng cách tăng tần số hoạt động của chip RAM và tăng bộ đệm tiền truy xuất từ 2 bit lên thành 4 bit cho mỗi chip. Bộ đệm tiền truy xuất một bộ nhớ cache nằm trên chip RAM, nó cho phép chip RAM chuẩn bị trước vị trí các bit để đặt chúng vào bus dữ liệu càng nhanh càng tốt. DDR3 được giới thiệu trong năm 2007, trong đó kích thước bộ đệm tăng lên tới 8 bit.

Về mặt lý thuyết, một module DDR có thể truyền dữ liệu với tốc độ khoảng từ 200 đến 600 MHz; một module DDR2 truyền với tốc độ từ 400 đến 1066 MHz; và tốc độ của DDR3 lên từ 800 đến 1600 MHz. Tuy nhiên trong thực tế, tốc độ này có thể nhỏ hơn.

5.3.1. Cache DRAM

CDRAM - Cache DRAM được phát triển bởi hãng Mitsubishi. Bộ nhớ này tích hợp một cache SRAM nhỏ (16 Kb) vào chip DRAM. Bộ nhớ SRAM này được sử dụng theo hai cách: như một bộ nhớ cache thực sự với các line 64-bit hoặc như một bộ đệm hỗ trợ việc truy cập liên tiếp một khối dữ liệu.



Hình 5.15 Đồ thị thời gian DDR SDRAM

5.4. CÂU HỎI

1. Các tính chất chính của bộ nhớ bán dẫn là gì?
2. Hai cách diễn đạt của từ khóa *bộ nhớ truy cập ngẫu nhiên* là gì?
3. Khác biệt giữa DRAM và SRAM về mặt ứng dụng là gì?
4. Sự khác biệt giữa DRAM và SRAM về các đặc điểm như tốc độ, kích thước và chi phí là gì?
5. Giải thích lý do tại sao một loại RAM được coi là linh kiện tương tự và còn một loại khác là linh kiện số.
6. Một số ứng dụng cho ROM là gì?
7. Sự khác nhau giữa EPROM, EEPROM và bộ nhớ flash là gì?
8. Giải thích chức năng của mỗi chân trong Hình 5.4b.
9. Bit chẵn lẻ là gì?
10. Làm thế nào để xác định được syndrome của mã Hamming?
11. SDRAM khác với DRAM thông thường như thế nào?

Chương 6. BỘ NHỚ NGOÀI

Chương này xem xét một loạt các thiết bị và hệ thống bộ nhớ ngoài. Chúng ta bắt đầu với thiết bị quan trọng nhất, đĩa từ. Đĩa từ là nền tảng của bộ nhớ ngoài trong hầu hết các hệ thống máy tính. Phân tiếp theo sẽ tìm hiểu việc sử dụng mảng đĩa để đạt được hiệu quả cao hơn, đặc biệt là hệ thống RAID (Mảng dư thừa các đĩa độc lập). Một thành phần ngày càng trở nên quan trọng trong nhiều hệ thống máy tính là đĩa bán dẫn, sẽ được thảo luận tiếp theo. Sau đó, bộ nhớ quang học bên ngoài và băng từ cũng được trình bày.

6.1. ĐĨA TỪ

Đĩa từ là một tấm platter tròn chế tạo bằng vật liệu không từ tính, được gọi là chất nền (substrate), được phủ bởi một lớp vật liệu có từ tính. Thông thường, chất nền thường là vật liệu nhôm hoặc hợp kim nhôm. Gần đây, chất nền thủy tinh đã được giới thiệu. Ưu điểm của chất nền thủy tinh là:

- Cải thiện tính đồng nhất của bề mặt phim từ để tăng độ tin cậy của đĩa
- Giảm đáng kể các khiếm khuyết bề mặt để giúp giảm thiểu lỗi đọc-ghi
- Hỗ trợ khoảng cách giữa đầu đọc/ghi với bề mặt đĩa rất nhỏ (giải thích ở phần sau)
- Độ cứng tốt hơn nên giảm động lực đĩa
- Khả năng chống sốc và hư hỏng lớn hơn

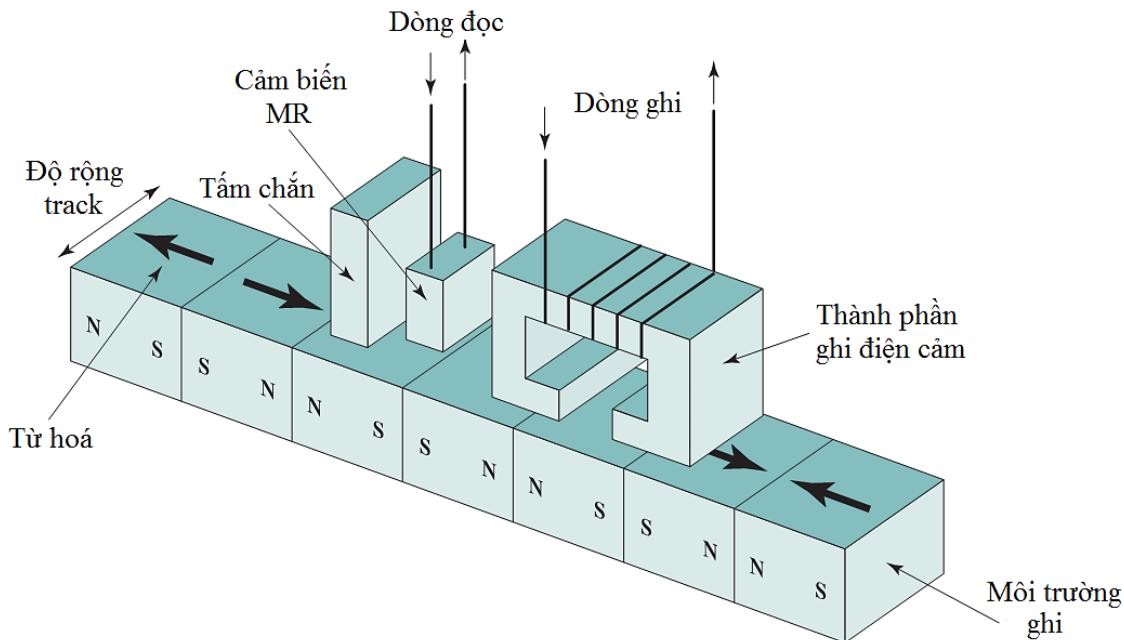
6.1.1. Cơ chế đọc và ghi từ

Dữ liệu được ghi vào đĩa và lấy ra từ đĩa thông qua một cuộn dây dẫn được gọi là **đầu**; các hệ thống thường có 2 đầu: đầu đọc và đầu ghi. Trong quá trình đọc hoặc ghi, đầu đứng yên trong khi đĩa xoay tròn bên dưới.

Cơ chế ghi lợi dụng tính chất: dòng điện chạy qua cuộn dây tạo ra từ trường. Các xung điện được gửi đến đầu ghi và các mẫu từ trường sinh ra được ghi vào bề mặt bên dưới. Các mẫu từ khác nhau thể hiện dòng điện dương và âm khác nhau. Đầu ghi được làm bằng vật liệu từ hoá, dạng hình chữ nhật rỗng với khe hở đọc một cạnh và vài vòng dây dẫn cuốn đọc cạnh đối diện (Hình 6.1). Dòng điện chạy trong dây cảm ứng một từ trường qua khe, làm từ hoá một vùng nhỏ của môi trường ghi. Đảo chiều dòng điện sẽ làm đảo chiều từ hóa trên môi trường ghi.

Cơ chế đọc truyền thống lợi dụng tính chất: từ trường chuyển động quanh cuộn dây tạo ra dòng điện trong cuộn dây. Khi bề mặt đĩa di chuyển qua đầu, nó tạo ra một dòng điện cùng cực với dòng đã ghi trên đĩa. Cấu trúc của đầu đọc cơ bản giống đầu ghi, do đó cùng một đầu có thể được sử dụng cho cả đọc và ghi. Kiểu đầu đơn này được sử dụng trong hệ thống đĩa mềm và trong các hệ thống đĩa cứng trước đây.

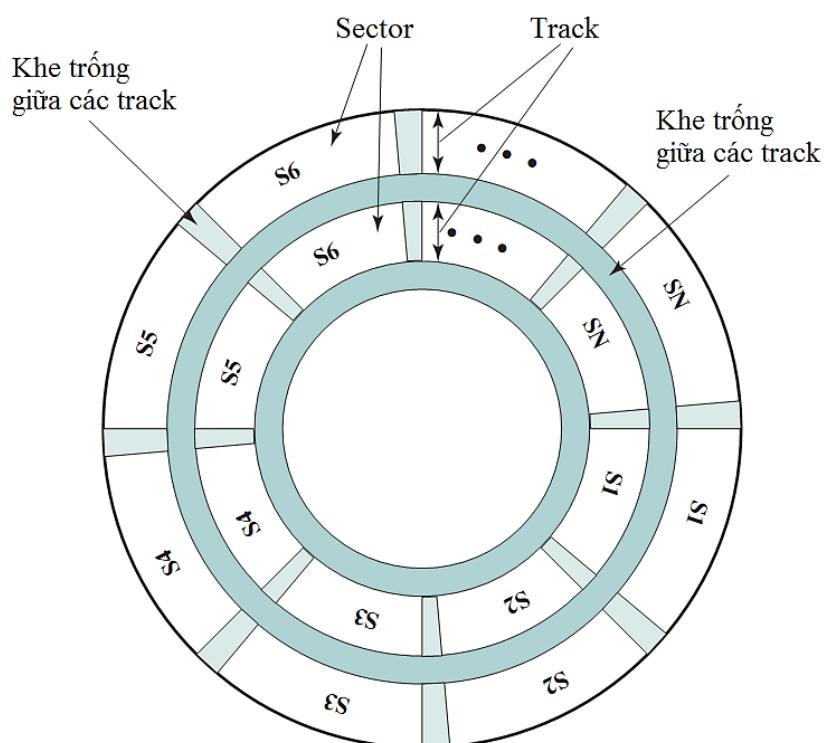
Các hệ thống đĩa cứng hiện đại sử dụng cơ chế đọc khác, đòi hỏi phải có đầu đọc, đầu ghi riêng biệt. Đầu đọc gồm một bộ cảm biến điện từ (MR – magnetoresistive) được che một phần. Vật liệu MR có điện trở phụ thuộc vào hướng từ hóa của môi trường di chuyển dưới nó. Khi cho một dòng điện chạy qua cảm biến MR, sự thay đổi điện trở được nhận biết dưới dạng tín hiệu điện áp. Thiết kế của MR cho phép mật độ lưu trữ lớn hơn và tốc độ vận hành cao hơn.



Hình 6.1 Đầu đọc điện từ/ghi điện cảm

6.1.2. Tổ chức dữ liệu

Đầu là một thiết bị tương đối nhỏ có khả năng đọc hoặc ghi từng phần nhỏ của đĩa xoay bên dưới nó. Điều này dẫn đến việc dữ liệu trên đĩa được tổ chức thành một tập hợp các vòng tròn, gọi là **track**. Mỗi track có độ rộng bằng độ rộng đầu. Một bề mặt có hàng ngàn track.

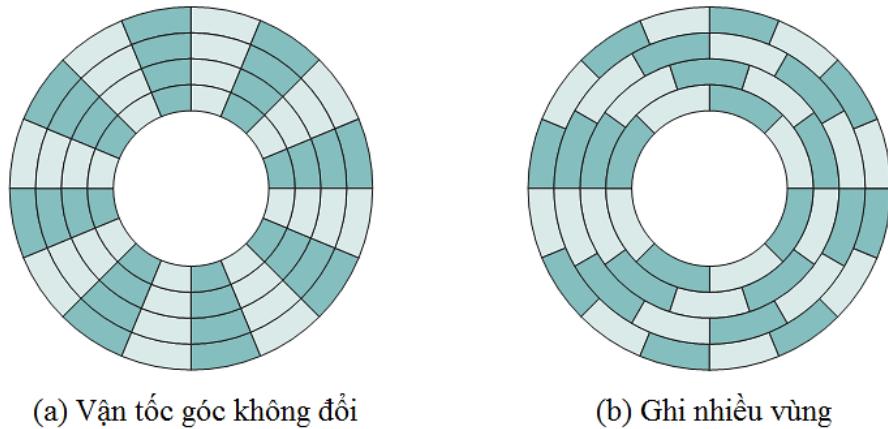


Hình 6.2 Bố trí dữ liệu trên đĩa

Hình 6.2 mô tả bộ cục dữ liệu này. Các track liền kề nhau được ngăn cách nhau bằng **khe trống** (gap). Khe trống giúp giảm thiểu sai sót do sai lệch đầu hoặc nhiễu từ trường.

Dữ liệu được truyền ra khỏi đĩa/vào đĩa theo từng sector (Hình 6.2). Có hàng trăm sector trong mỗi track và các sector có thể có độ dài cố định hoặc biến đổi. Trong hầu hết các hệ thống hiện đại, sector có độ dài cố định, phổ biến là 512 byte. Tương tự, các sector liền kề được ngăn cách bởi khe trống.

Khi đĩa quay tròn, một bit gần tâm đĩa di chuyển qua một điểm xác định (dưới đầu đọc/ghi) chậm hơn một bit phía ngoài vành đĩa. Do đó, cần có phương pháp bù lại độ biến động tốc độ sao cho đầu có thể đọc các bit ở cùng tốc độ. Điều này có thể được thực hiện bằng cách tăng khoảng cách giữa các bit thông tin trong các đoạn khác nhau trên đĩa. Khi đó thông tin được quét ở cùng tốc độ khi đĩa quay, được gọi là **vận tốc góc không đổi** (CAV – constant angular velocity). Hình 6.3a cho thấy bộ cục của một đĩa sử dụng CAV. Đĩa chia thành một số sector hình bánh pie và một chuỗi track đồng tâm. Ưu điểm của CAV là có thể đánh địa chỉ trực tiếp cho từng khối dữ liệu theo track và sector. Để di chuyển đầu từ vị trí hiện tại đến một địa chỉ cụ thể, đầu chỉ cần di chuyển một đoạn ngắn đến một track cụ thể và chờ một khoảng thời gian ngắn để sector thích hợp quay đến bên dưới đầu. Nhược điểm của CAV là số lượng dữ liệu có thể lưu trữ trên track dài hơn phía ngoài chỉ bằng lượng dữ liệu lưu trữ trên các track ngắn gần trong tâm đĩa.



Hình 6.3 So sánh các phương pháp bố trí đĩa

Do mật độ, số bit trên một đơn vị chiều dài, tăng dần nếu tính từ track ngoài cùng tới track trong cùng, nên dung lượng lưu trữ của đĩa trong trường hợp CAV bị hạn chế bởi mật độ ghi tối đa của track trong cùng. Để tăng mật độ, hệ thống đĩa cứng hiện đại sử dụng kỹ thuật **ghi nhiều vùng**, theo đó bề mặt được chia thành nhiều vùng đồng tâm (thường là 16 vùng). Trong cùng một vùng, các track có số lượng bit như nhau. Các vùng xa trung tâm đĩa hơn chứa nhiều bit hơn (nhiều sector hơn) so với vùng gần trung tâm. Điều này cho phép tổng dung lượng lưu trữ lớn hơn. Tuy nhiên, mạch điện phức tạp hơn. Khi đầu đĩa di chuyển từ vùng này sang vùng khác, chiều dài (đọc track) của từng bit sẽ thay đổi, làm thay đổi thời gian cho việc đọc và ghi. Hình 6.3b cho thấy bản chất của việc ghi nhiều vùng; trong hình minh họa này, mỗi vùng chỉ rộng một track.

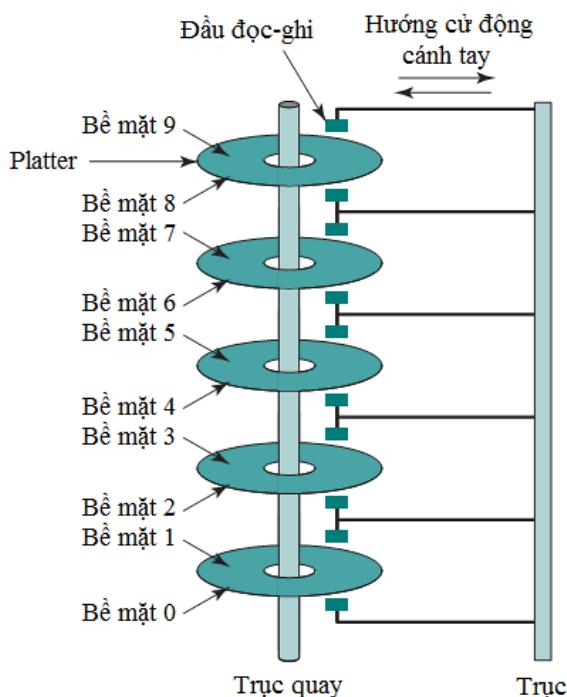
6.1.3. Đặc tính vật lý

Bảng 6.1 liệt kê các đặc tính chính phân biệt các loại đĩa từ. Thứ nhất, đầu có thể cố định hoặc di chuyển được theo chiều hướng tâm của đĩa. Trong ổ đĩa có **đầu cố định**, có một đầu đọc-ghi dành riêng cho mỗi track. Tất cả các đầu được gắn trên một cánh tay cố định kéo dài trên toàn bộ track; tuy nhiên ngày nay rất hiếm các hệ thống như vậy. Trong ổ đĩa có **đầu di chuyển**, chỉ có một đầu đọc-ghi. Đầu này cũng được gắn trên một cánh tay. Để đầu có thể được đặt trên bất kỳ track nào, cánh tay phải có thể kéo dài hoặc rút ngắn được.

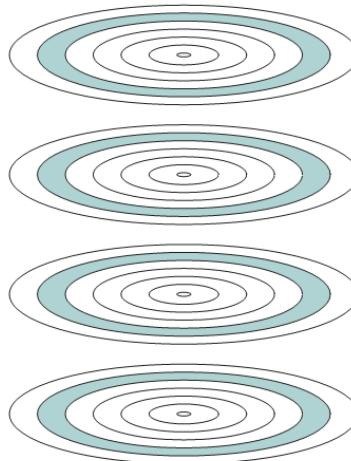
Một ổ đĩa bao gồm cánh tay, trục quay đĩa, và các thiết bị điện tử cần thiết để nhập và xuất dữ liệu nhị phân. **Đĩa không tháo được** được gắn vĩnh viễn vào ổ đĩa; đĩa cứng trong máy tính cá nhân là một ví dụ về đĩa không tháo được. **Đĩa tháo được** có thể được gỡ ra và thay thế bằng một đĩa khác. Ưu điểm của loại thứ hai là hệ thống đĩa hữu hạn nhưng sẵn sàng chứa lượng dữ liệu vô hạn. Hơn nữa, đĩa kiểu này có thể di chuyển được từ máy tính này sang máy tính khác. Đĩa mềm và đĩa cartridge ZIP là những ví dụ về đĩa tháo được.

Lớp phủ từ tính thường được phủ lên cả hai mặt của đĩa, chúng được gọi là **đĩa hai mặt**. Một số hệ thống đĩa rẻ hơn sử dụng **đĩa một mặt**.

Một số ổ đĩa chứa nhiều tám đĩa xếp chồng lên nhau theo chiều dọc cách nhau vài milimet. Ổ đĩa như vậy sử dụng nhiều cánh tay (Hình 6.4). Ổ đĩa **đa tám** sử dụng đầu di chuyển, với một đầu đọc-ghi cho mỗi bề mặt đĩa. Tất cả các đầu được gắn cố định sao cho tất cả đều cách tâm đĩa một khoảng như nhau và di chuyển cùng nhau. Do đó, bất cứ lúc nào, tất cả các đầu đều được đặt trên các track có khoảng cách tới tâm đĩa bằng nhau. Tập hợp tất cả các track ở cùng vị trí tương đối trên đĩa được gọi là **cylinder**. Ví dụ, tất cả các track được tô đậm trong Hình 6.5 là một phần của cylinder.



Hình 6.4 Các thành phần của ổ đĩa cứng



Hình 6.5 Track và cylinder

Bảng 6.1 Đặc tính vật lý của hệ thống đĩa

Chuyển động đầu	Tấm platter
Đầu cố định	Đơn tấm
Đầu di chuyển	Đa tấm
Tính di động của đĩa	Cơ chế của đầu
Đĩa không tháo được	Tiếp xúc (đĩa mềm)
Đĩa tháo được	Khe cố định
Mặt	Cơ chế
Một mặt	Khe khí động học (Winchester)
Hai mặt	

Cuối cùng, dựa theo cơ chế của đầu, có thể phân loại đĩa thành ba loại. Loại truyền thống, đầu đọc-ghi được đặt cách tấm đĩa một khoảng cố định, ở giữa là một khe không khí. Một loại khác hoàn toàn là cơ chế đầu thực sự tiếp xúc vật lý với môi trường đĩa trong quá trình đọc hoặc ghi. Cơ chế này được áp dụng với **đĩa mềm** (một loại đĩa nhỏ, mềm dẻo) và loại đĩa cứng rẻ nhất.

Trước khi tìm hiểu loại đĩa thứ ba, ta cần phải bàn về mối quan hệ giữa mật độ dữ liệu và độ rộng khe không khí. Đầu phải tạo ra hoặc cảm nhận một trường điện từ đủ lớn để ghi và đọc đúng. Đầu càng hẹp thì càng cần được đặt gần bề mặt đĩa. Đầu hẹp hơn nghĩa là các track hẹp hơn và do đó mật độ dữ liệu lớn hơn, đây là điều ta mong muốn. Tuy nhiên, đầu càng gần đĩa thì nguy cơ lỗi do tạp chất hoặc do không hoàn hảo càng cao. Xét ví dụ về đĩa Winchester. Đầu Winchester được sử dụng trong các cụm ổ kín mà hầu như không có tạp chất. Nó được thiết kế để hoạt động gần bề mặt đĩa hơn so với các đầu đĩa cứng thông thường, do đó cho phép mật độ dữ liệu lớn hơn. Đầu là một tấm giấy foil khí động học đặt nhẹ trên bề mặt đĩa khi đĩa không di chuyển. Áp suất không khí tạo ra khi đĩa quay là đủ lớn để làm cho tấm foil bay lên khỏi bề mặt. Hệ thống không tiếp xúc này có thể được thiết

ké để cho phép đầu đọc/ghi hép hoạt động gần bề mặt đĩa hơn so với các loại đầu đĩa cứng cùng thời.

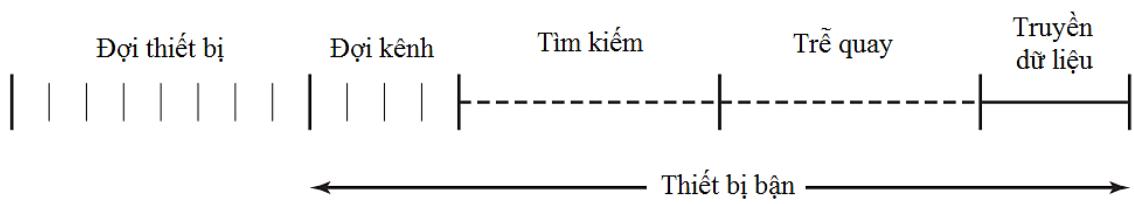
Bảng 6.2 cung cấp các thông số của các loại ô đĩa hiệu suất cao điển hình.

6.1.4. Thông số hiệu suất đĩa

Bảng 6.2 Các thông số ô đĩa điển hình

Đặc tính	Constellation ES.2	Seagate Barracuda XT	Cheetah NS	Momentus
Ứng dụng	Doanh nghiệp	Máy tính để bàn	Server ứng dụng	Máy tính xách tay
Dung lượng	3 TB	3 TB	400 GB	640 GB
Thời gian tìm kiếm trung bình	8.5 ms đọc 9.5 ms ghi	N/A	3.9 ms đọc 4.2 ms ghi	13 ms
Tốc độ quay trục	7200 rpm	7200 rpm	10,075 rpm	5400 rpm
Độ trễ trung bình	4.16 ms	4.16 ms	2.98	5.6 ms
Tốc độ truyền tối đa	155 MB/s	149 MB/s	97 MB/s	300 MB/s
Byte/sector	512	512	512	4096
Track/cylinder (số lượng bệ mặt platter)	8	10	8	4
Cache	64 MB	64 MB	16 MB	8 MB

Chi tiết thực tế về hoạt động vào/ra của đĩa phụ thuộc vào hệ thống máy tính, hệ điều hành, tính chất của kênh vào/ra và phần cứng bộ điều khiển đĩa. Một sơ đồ thời gian tổng quát của quá trình truyền vào/ra của đĩa được thể hiện trong hình 6.6.



Hình 6.6 Thời gian truyền vào/ra của đĩa

Khi ô đĩa đang hoạt động, đĩa quay với tốc độ không đổi. Để đọc hoặc ghi, đầu phải được đặt ở track mong muốn và ở sector đầu tiên của track đó. Việc chọn track bao gồm việc di chuyển đầu (trong hệ thống đầu di chuyển) hoặc lựa chọn một đầu (trong hệ thống đầu cố định). Trong hệ thống đầu di chuyển, thời gian cần để đặt đầu vào track mong muốn được gọi là **thời gian tìm kiếm**. Khi đã chọn được track, bộ điều khiển đĩa sẽ đợi cho đến

khi sector thích hợp quay tới thẳng hàng với đầu. Thời gian cần thiết để điểm bắt đầu của sector đến dưới đầu được gọi là **trễ quay**. Tổng cộng thời gian tìm kiếm và trễ quay là **thời gian truy cập**, đó là thời gian cần thiết để đầu vào vị trí để đọc hoặc ghi. Một khi đầu vào vị trí, hoạt động đọc hoặc ghi được thực hiện khi sector di chuyển dưới đầu; đây là giai đoạn truyền dữ liệu; thời gian cần thiết để truyền là **thời gian truyền**.

Ngoài thời gian truy cập và thời gian truyền, hoạt động vào/ra của đĩa còn có một vài khoảng trễ hàng đợi khác. Khi tiến trình đưa ra một yêu cầu vào/ra, trước hết nó phải đợi trong hàng đợi cho tới khi thiết bị sẵn sàng. Lúc này, thiết bị đó được gán cho tiến trình này. Nếu thiết bị chia sẻ một kênh vào/ra hoặc một tập hợp các kênh vào/ra với các ổ đĩa khác, thì có thể nó phải chờ đợi thêm cho tới khi kênh này sẵn sàng. Tại thời điểm này, việc tìm kiếm được thực hiện để bắt đầu truy cập đĩa.

Trong một số hệ thống máy chủ cao cấp, kỹ thuật RPS (Rotational Positional Sensing) được sử dụng. RPS hoạt động như sau: Khi lệnh tìm kiếm đã được ban hành, kênh được giải phóng để xử lý các hoạt động vào/ra khác. Khi hoàn thành tìm kiếm, thiết bị sẽ xác định khi nào thì dữ liệu xoay tới dưới đầu. Khi sector đến dưới đầu, thiết bị thiết lập lại đường truyền nối tới host. Nếu khỏi điều khiển hoặc kênh đang bận với một hoạt động vào/ra khác, thì lần kết nối lại này không thành công và thiết bị phải quay trọn một vòng trước khi nó có thể kết nối lại. Đây là một RPS miss. Nó là một khoảng trễ khác phải được bổ sung vào đồ thị thời gian ở hình 6.6.

6.2. RAID

Như đã thảo luận trước đây, việc cải tiến hiệu suất bộ xử lý và bộ nhớ chính diễn ra với tốc độ rất nhanh, trong khi đó, tốc độ cải thiện hiệu quả của bộ lưu trữ thứ cấp rất chậm. Sự chênh lệch này làm cho hệ thống lưu trữ trên đĩa trở thành một mối quan tâm chính trong việc cải thiện hiệu năng tổng thể hệ thống máy tính.

Một giải pháp làm tăng hiệu suất của hệ thống lưu trữ là sử dụng song song nhiều thành phần, hay là sử dụng mảng nhiều đĩa độc lập hoạt động song song. Khi dùng nhiều đĩa, có thể xử lý song song các yêu cầu vào/ra riêng biệt, miễn là dữ liệu mong muốn nằm trên các đĩa riêng biệt. Hơn nữa, cũng có thể thực hiện song song một yêu cầu vào/ra duy nhất nếu khỏi dữ liệu cần truy cập được phân tán trên nhiều đĩa.

Khi sử dụng nhiều đĩa, có rất nhiều cách để tổ chức dữ liệu và có thể thêm độ dư thừa để nâng cao độ tin cậy. Chính vì thế, việc phát triển cơ chế cơ sở dữ liệu dùng được trên nhiều nền tảng và hệ điều hành khác nhau trở nên khó khăn hơn. Trong bối cảnh đó, một cơ chế chuẩn hóa về thiết kế cơ sở dữ liệu trên nhiều đĩa đã được công nhận là RAID (Mảng dư thừa nhiều đĩa độc lập – Redundant Array of Independent Disks). RAID bao gồm bảy cấp, từ 0 đến 6.

Các cấp độ này không thể hiện mối quan hệ thứ bậc mà là các kiến trúc thiết kế khác nhau có chung ba đặc điểm:

1. RAID là tập hợp nhiều ổ đĩa vật lý được hệ điều hành coi như một ổ đĩa ảo đơn.
2. Dữ liệu được phân tán trên các ổ đĩa vật lý trong mảng theo cơ chế striping (phân dài).
3. Dung lượng đĩa dư thừa được sử dụng để lưu trữ thông tin parity, đảm bảo khả năng phục hồi dữ liệu trong trường hợp đĩa bị hỏng.

Đặc điểm thứ 2 và thứ 3 thể hiện khác nhau ở mỗi cấp RAID. RAID 0, RAID 1 không hỗ trợ đặc điểm thứ 3.

Bây giờ ta tìm hiểu từng cấp RAID. Bảng 6.3 tóm tắt các đặc điểm tổng quát của bảy cấp RAID. Trong bảng, hiệu quả vào/ra được thể hiện qua khả năng truyền dữ liệu và tốc độ yêu cầu vào/ra, hay khả năng đáp ứng các yêu cầu vào/ra, do các mức RAID chủ yếu khác nhau ở hai chỉ số này. Điểm mạnh của từng cấp RAID được tô đậm hơn. Hình 6.7 mô tả việc sử dụng bảy cấp RAID để lưu trữ dữ liệu có dung lượng bằng bốn đĩa chỉ chứa dữ liệu, không có độ dư thừa. Hình vẽ này tập trung thể hiện cách bố trí dữ liệu người dùng và dữ liệu dự phòng và cho thấy yêu cầu lưu trữ tương đối ở từng cấp độ khác nhau.

6.2.1. RAID cấp 0

RAID cấp 0 không phải là thành viên thực sự của họ RAID vì nó không có độ dư thừa để cải thiện hiệu suất. Tuy nhiên, có một vài ứng dụng, chẳng hạn như một số ứng dụng trên siêu máy tính trong đó hiệu suất và dung lượng là mối quan tâm chính và chi phí thấp quan trọng hơn độ tin cậy cao.

Đối với RAID 0, dữ liệu người dùng và dữ liệu hệ thống được phân phối trên tất cả các đĩa trong mảng. Việc này có ưu điểm đáng kể so với khi sử dụng một đĩa lớn duy nhất: Nếu hai yêu cầu vào/ra khác nhau đang chờ hai khối dữ liệu khác nhau, thì có khả năng hai khối được yêu cầu nằm trên các đĩa khác nhau. Do đó, hai yêu cầu này có thể được ban hành song song, làm giảm thời gian hàng đợi vào/ra.

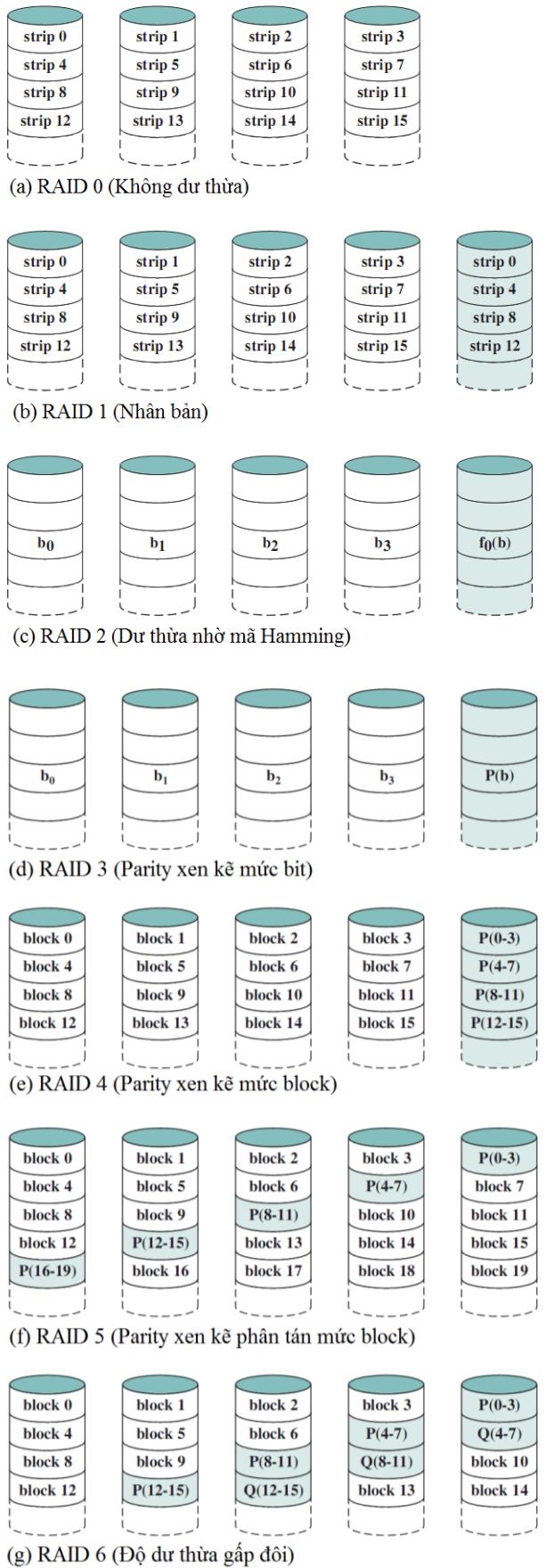
Nhưng RAID 0, cũng như tất cả các cấp RAID khác, làm nhiều hơn so với việc phân phối dữ liệu đơn giản trên một mảng đĩa: Dữ liệu được phân dài (*strip*) trên các đĩa có sẵn. Điều này được mô tả trong Hình 6.8. Tất cả dữ liệu người dùng và dữ liệu hệ thống được xem như đang được lưu trữ trên một đĩa ảo (đĩa logic). Đĩa ảo được chia thành các dải (*strip*); các dải này có thể là các khôi, sector, hoặc một số đơn vị khác. Các dải này được ánh xạ lần lượt tới các đĩa vật lý liên tiếp trong mảng RAID. Một nhóm các dải ảo liên tiếp mà mỗi dải trong đó ánh xạ vào một đĩa trong mảng được gọi là **stripe**. Trong mảng n đĩa, n dải ảo đầu tiên được lưu trữ thực tế thành dải đầu tiên trên mỗi đĩa, tạo thành stripe thứ nhất; nhóm n dải thứ hai được phân phối thành các dải thứ hai trên mỗi đĩa; v.v... Ưu điểm của cách bố trí này là nếu một yêu cầu vào/ra bao gồm nhiều dải ảo liên tiếp, thì có thể xử lý song song cùng lúc n dải của yêu cầu đó, làm giảm đáng kể thời gian truyền vào/ra.

Hình 6.8 cho thấy việc sử dụng phần mềm quản lý mảng để ánh xạ giữa không gian đĩa ảo và không gian đĩa vật lý. Phần mềm này có thể thực thi trong hệ thống đĩa hoặc trong một máy host.

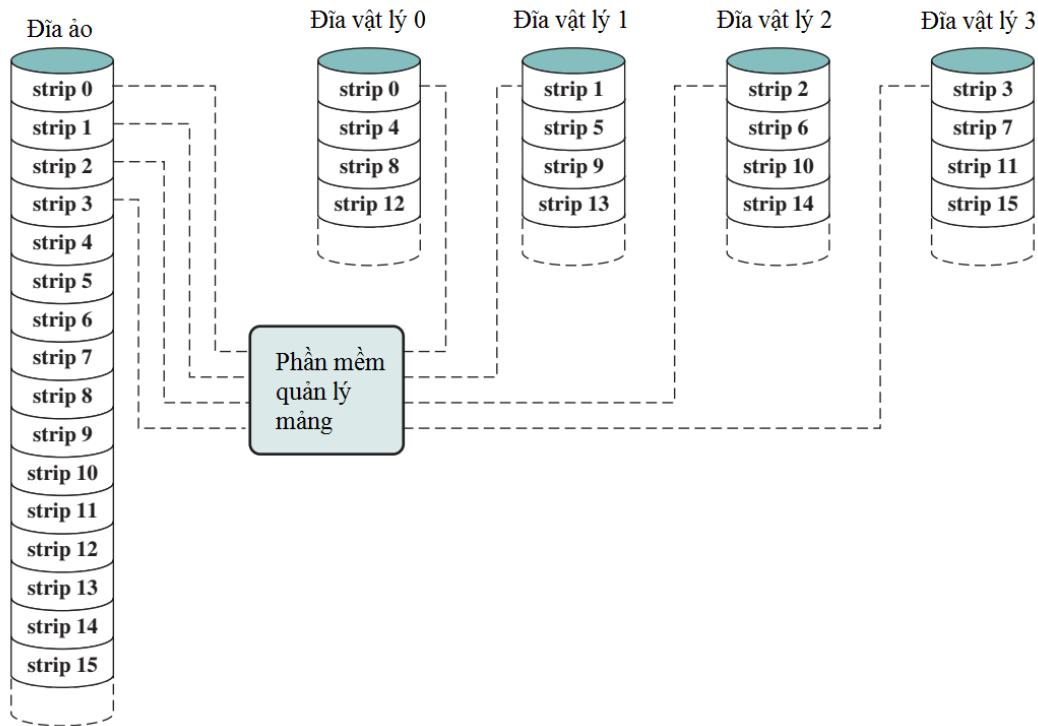
Bảng 6.3 Các cấp RAID

Loại	Cấp	Mô tả	Số đĩa cần	Độ sẵn sàng dữ liệu	Khả năng truyền dữ liệu vào/ra cỡ lớn	Tốc độ yêu cầu vào/ra cỡ nhỏ
Phân dải	0	Không dư thừa	N	Thấp hơn đĩa đơn	Rất cao	Rất cao đối với cả đọc và ghi
Nhân đôi	1	Nhân đôi	$2N$	Cao hơn RAID 2, 3, 4, hoặc 5; thấp hơn RAID 6	Cao hơn đĩa đơn đối với đọc; tương tự đĩa đơn đối với ghi	Gấp đôi đĩa đơn đối với đọc; tương tự đĩa đơn đối với ghi
Truy cập song song	2	Dư thừa nhờ mã Hamming	$N+m$	Cao hơn đĩa đơn; tương đương RAID 3, 4, hoặc 5	Cao nhất trong các cấp được nêu	Xáp xỉ gấp đôi đĩa đơn
	3	Parity xen kẽ mức bit	$N+1$	Cao hơn đĩa đơn; tương đương RAID 2, 4, hoặc 5	Cao nhất trong các cấp được nêu	Xáp xỉ gấp đôi đĩa đơn
Truy cập độc lập	4	Parity xen kẽ mức block	$N+1$	Cao hơn đĩa đơn; tương đương RAID 2, 3, hoặc 5	Tương tự RAID 0 đối với đọc; thấp hơn đáng kể so với đĩa đơn đối với ghi	Tương tự RAID 0 đối với đọc; thấp hơn đáng kể so với đĩa đơn đối với ghi
	5	Parity phân tán xen kẽ mức block	$N+1$	Cao hơn đĩa đơn; tương đương RAID 2, 3, hoặc 4	Tương tự RAID 0 đối với đọc; thấp hơn đĩa đơn đối với ghi	Tương tự RAID 0 đối với đọc; thấp hơn đĩa đơn đối với ghi
	6	Parity nhân đôi phân tán xen kẽ mức block	$N+2$	Cao nhất trong các cấp được nêu	Tương tự RAID 0 đối với đọc; thấp hơn RAID 5 đối với ghi	Tương tự RAID 0 đối với đọc; thấp hơn RAID 5 đối với ghi

Lưu ý: N = số lượng đĩa dữ liệu; m tỷ lệ thuận với $\log N$



Hình 6.7 Các cấp RAID



Hình 6.8 Ánh xạ dữ liệu trong mảng RAID cấp 0

6.2.1.1. RAID 0 đối với khả năng truyền dữ liệu cao

Hiệu suất của một cấp RAID bất kỳ phụ thuộc chủ yếu vào mẫu yêu cầu của hệ thống host và cách bố trí dữ liệu. Những vấn đề này có thể được phân tích rõ ràng nhất trong RAID 0, bởi việc phân tích không chịu tác động của độ dư thừa. Trước tiên, hãy xem xét việc sử dụng RAID 0 để đạt được tốc độ truyền dữ liệu cao. Các ứng dụng muốn có được tốc độ truyền cao phải đáp ứng được hai yêu cầu. Thứ nhất, phải có dung lượng truyền tải cao trên toàn bộ đường dẫn giữa bộ nhớ host và các ổ đĩa riêng lẻ. Điều này bao gồm các bus điều khiển bên trong, bus vào/ra hệ thống vào/ra, các bộ chuyển đổi vào/ra và bus bộ nhớ host.

Yêu cầu thứ hai là ứng dụng đó phải tạo ra các yêu cầu vào/ra để điều khiển mảng đĩa một cách hiệu quả. Yêu cầu này được thỏa mãn nếu các bản tin yêu cầu đòi hỏi lượng lớn dữ liệu ảo liên tục, lớn hơn so với kích thước dài. Trong trường hợp này, một bản tin yêu cầu vào/ra sẽ được đáp lại bằng việc truyền song song dữ liệu từ nhiều đĩa, làm tăng tốc truyền dữ liệu hơn truyền từng đĩa một.

6.2.1.2. RAID 0 đối với tốc độ yêu cầu vào/ra cao

Trong môi trường hướng giao dịch, người dùng thường quan tâm đến thời gian phản hồi hơn là tốc độ truyền. Đối với một yêu cầu vào/ra cho một lượng nhỏ dữ liệu, chiếm đa phần thời gian vào/ra là thời gian di chuyển đầu (tìm kiếm) và di chuyển đĩa (trẽ quay).

Trong một môi trường giao dịch, có hàng trăm yêu cầu vào/ra/giây. Mảng đĩa có thể đem lại tốc độ thực thi lệnh vào/ra cao nhờ sự cân bằng tải vào/ra trên nhiều đĩa. Sự cân bằng tải chỉ hiệu quả khi có nhiều yêu cầu vào/ra. Điều này nghĩa là có nhiều ứng dụng độc lập hoặc một ứng dụng định hướng giao dịch có khả năng tạo ra nhiều yêu cầu I / O không

đồng bộ. Hiệu suất còn bị ảnh hưởng bởi kích thước dải. Nếu kích thước dải tương đối lớn, sao cho một yêu cầu I / O chỉ yêu cầu dữ liệu trong một đĩa duy nhất, khi đó nhiều yêu cầu vào/ra có thể được xử lý song song, giảm thời gian hàng đợi cho mỗi yêu cầu.

6.2.2. RAID cấp 1

RAID 1 khác với các cấp RAID từ 2 đến 6 ở cách tạo độ dư thừa. Trong các cấp RAID khác, độ dư thừa được tạo ra bằng một số cách tính bít chẵn lẻ, trong khi độ dư thừa ở RAID 1 được tạo ra bằng cách sao chép tất cả dữ liệu. Hình 6.7b cho thấy, RAID 1 cũng phân chia dữ liệu như trong RAID 0. Nhưng trong trường hợp này, mỗi dải ảo được ánh xạ tới hai đĩa vật lý riêng biệt để mọi đĩa trong mảng có một đĩa nhân bản chứa dữ liệu giống hệt đĩa gốc.

Có một số khía cạnh tích cực đối với tổ chức RAID 1:

1. Một yêu cầu đọc có thể được phục vụ bởi một trong hai đĩa có chứa dữ liệu được yêu cầu, sao cho thời gian tìm kiếm cộng với trễ quay là tối thiểu.
2. Một yêu cầu ghi đòi hỏi phải cập nhật cả hai dải tương ứng một cách đồng thời. Do đó, hiệu suất ghi được quyết định bởi lần ghi chậm hơn (tức là lần ghi có thời gian tìm kiếm cộng với trễ quay lớn hơn). Tuy nhiên, không có "write penalty" với RAID 1. RAID cấp 2 đến cấp 6 liên quan đến việc sử dụng các bit chẵn lẻ. Vì vậy, khi một dải được cập nhật, phần mềm quản lý mảng ngoài việc cập nhật dải thực tế được đề cập thì còn phải tính toán và cập nhật các bit chẵn lẻ.
3. Việc khôi phục dữ liệu bị lỗi, hỏng là rất đơn giản. Khi một ổ bị lỗi, dữ liệu vẫn có thể được truy cập từ ổ đĩa thứ hai.

Nhược điểm chính của RAID 1 là chi phí; nó đòi hỏi không gian đĩa gấp đôi đĩa ảo mà nó hỗ trợ. Do đó, cấu hình RAID 1 chỉ được ứng dụng trong các ổ đĩa lưu trữ phần mềm hệ thống, dữ liệu và các tệp tin quan trọng khác. Trong những trường hợp này, RAID 1 tạo ra bản sao thời gian thực của tất cả dữ liệu sao cho trong trường hợp đĩa bị lỗi, tất cả các dữ liệu quan trọng vẫn sẵn sàng.

Trong môi trường hướng giao dịch, RAID 1 có thể đạt được tốc độ yêu cầu I/O cao nếu phần lớn là các yêu cầu đọc. Trong trường hợp này, hiệu năng của RAID 1 có thể tăng gấp đôi so với RAID 0. Tuy nhiên, nếu đa phần các yêu cầu I / O là yêu cầu ghi, thì mức tăng hiệu suất là không đáng kể so với RAID 0.

So sánh các cấp RAID

Cấp	Ưu điểm	Nhược điểm	Ứng dụng
0	Hiệu suất truyền vào/ra cải thiện nhiều nhờ dàn tải I/O trên nhiều kênh và ổ đĩa Không cần tính bit chẵn lẻ Thiết kế rất đơn giản Dễ triển khai	Hỗntrong 1 ổ đĩa thì mất tất cả dữ liệu trong mảng	Sản xuất, biên tập video Biên tập ảnh Ứng dụng yêu cầu băng thông cao

1	Độ dư thừa 100% Không cần tái tạo lại dữ liệu nếu có một ổ đĩa bị lỗi Thiết kế lưu trữ RAID đơn giản nhất	Không hiệu quả Chi phí ổ đĩa lớn nhất (100%)	Kế toán Tài chính Ứng dụng đòi hỏi độ tin cậy cao
2	Tốc độ truyền dữ liệu cực cao Thiết kế điều khiển đơn giản hơn RAID 3, 4, 5	Tốc độ truyền dữ liệu càng cao, tỉ lệ ổ mã sửa lỗi/ổ dữ liệu rất cao nếu kích thước từ nhỏ - kém hiệu quả Đắt	Không sử dụng trong thực tế
3	Tốc độ đọc/ ghi dữ liệu rất cao Đĩa hỏng không ảnh hưởng nhiều đến thông lượng Tỉ lệ ổ mã sửa lỗi/ổ dữ liệu thấp - hiệu quả cao	Tốc độ giao dịch bằng tốc độ của 1 ổ đĩa đơn Thiết kế điều khiển khá phức tạp	Sản xuất video Live streaming Biên tập ảnh Ứng dụng yêu cầu thông lượng cao
4	Tốc độ giao dịch dữ liệu đọc cao Tỉ lệ ổ mã sửa lỗi/ổ dữ liệu thấp - hiệu quả cao	Thiết kế điều khiển khá phức tạp Tốc độ giao dịch ghi thấp nhất Tái tạo dữ liệu khó khăn, kém hiệu quả	Không sử dụng trong thực tế
5	Tốc độ giao dịch dữ liệu đọc cao nhất Tỉ lệ ổ mã sửa lỗi/ổ dữ liệu thấp - hiệu quả cao	Thiết kế điều khiển phức tạp nhất Tái tạo dữ liệu khó khăn, kém hiệu quả	Server ứng dụng, Server database Server web, e-mail Server intranet
6	Chịu được lỗi dữ liệu cực lớn Chịu được lỗi ổ đĩa đồng thời	Thiết kế điều khiển phức tạp hơn Overhead để tính địa chỉ parity cực cao	Giải pháp hoàn hảo cho ứng dụng quan trọng

6.2.3. RAID cấp 5

RAID 5 được tổ chức tương tự như RAID 4. Tuy nhiên, RAID 5 phân bố các dải chẵn lẻ trên tất cả các đĩa. Cách phân bố điển hình là tuân theo cơ chế round-robin, như ví dụ trong hình 6.8f. Trong mảng n đĩa, mỗi dải chẵn lẻ nằm trên một đĩa khác nhau đối với n stripe đầu tiên và mẫu này tiếp tục được lặp lại.

Việc phân bố dải chẵn lẻ trên tất cả các ổ đĩa tránh được khả năng nghẽn cổ chai của RAID 4.

6.2.4. RAID cấp 6

Trong sơ đồ RAID 6, hai phép tính chẵn lẻ khác nhau được thực hiện và được lưu trữ trong các block riêng biệt trên các đĩa khác nhau. Do đó, nếu cần lưu dữ liệu người dùng trên N đĩa thì mảng RAID 6 sẽ bao gồm $N + 2$ đĩa.

Hình 6.8g cho cơ chế này. P và Q là hai thuật toán kiểm tra dữ liệu khác nhau. Một trong hai phép tính đó là phép XOR được sử dụng trong RAID 4 và 5. Phép tính còn lại là thuật toán kiểm tra dữ liệu độc lập. Do đó, ta vẫn có thể khôi phục lại dữ liệu ngay cả khi hai đĩa chứa dữ liệu người dùng bị hỏng.

Ưu điểm của RAID 6 là tính sẵn sàng dữ liệu cực kỳ cao. Dữ liệu chỉ bị mất chừng nào cả ba ổ đĩa bị hỏng trong khoảng thời gian MTTR (mean time to repair – thời gian trung bình để sửa chữa). Mặt khác, RAID 6 phải chịu write penalty đáng kể, bởi vì lần ghi liên quan đến cả hai khối chẵn lẻ. Hiệu suất ghi tổng thể của RAID 6 kém hơn 30% so với RAID 5; còn hiệu suất đọc là tương đương.

Bảng 6.4 là bảng so sánh bảy cấp RAID.

6.3. Ô ĐĨA BÁN DẪN SSD

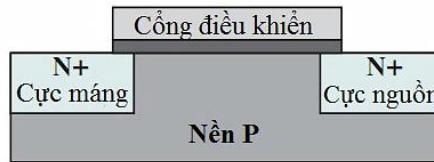
Một phát triển rất quan trọng trong kiến trúc máy tính những năm gần đây là việc sử dụng ngày càng phổ biến các ổ đĩa bán dẫn (SSD – solid state drive) để bổ sung hoặc thậm chí thay thế cho ổ đĩa cứng, cả ở bộ nhớ bên trong lẫn bên ngoài. Thuật ngữ “solid state” chỉ mạch điện tử được chế tạo bằng chất bán dẫn. Ô đĩa bán dẫn là thiết bị nhớ được chế tạo bằng các linh kiện bán dẫn có thể được sử dụng thay thế cho ổ đĩa cứng. Các ổ SSD hiện đang có mặt trên thị trường sử dụng một loại bộ nhớ bán dẫn được gọi là bộ nhớ flash. Trong mục này, trước tiên ta tìm hiểu về bộ nhớ flash, rồi sau đó xem xét việc sử dụng nó trong SSD.

6.3.1. Bộ nhớ flash

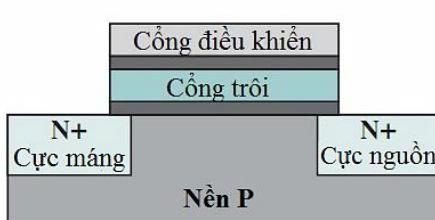
Bộ nhớ flash là một loại bộ nhớ bán dẫn đã xuất hiện khá lâu và được sử dụng trong nhiều sản phẩm điện tử tiêu dùng, bao gồm điện thoại thông minh, thiết bị GPS, máy nghe nhạc MP3, máy ảnh kỹ thuật số và USB. Trong những năm gần đây, chi phí và hiệu suất của bộ nhớ flash đã phát triển đến mức ổ đĩa với bộ nhớ flash hoàn toàn có thể được sử dụng thay thế ổ cứng.

Hình 6.9 mô tả hoạt động cơ bản của bộ nhớ flash. Hình 6.9a mô tả hoạt động của transistor. Transistor lợi dụng tính chất của chất bán dẫn là đặt một điện áp nhỏ vào cực cổng có thể điều khiển một dòng điện lớn chạy giữa cực nguồn và cực công.

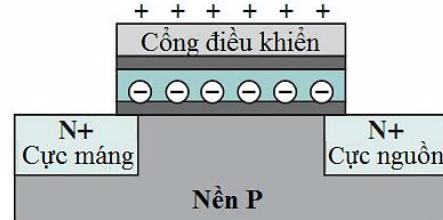
Trong một ô nhớ flash, một cực cổng thứ hai được thêm vào transistor. Nó được gọi là cổng trôi do được cách điện bởi lớp oxit mỏng. Ban đầu, cổng trôi không can thiệp vào hoạt động của transistor (Hình 6.9b). Trong trường hợp này, ô nhớ được xem là đang biểu diễn nhị phân 1. Đặt một điện áp lớn trên lớp ôxít làm cho các electron đi qua nó và bị mắc kẹt lại ở cổng trôi, và chúng vẫn nằm nguyên ở đó ngay cả khi đã ngắt điện (Hình 6.9c). Trong trạng thái này, ô nhớ được xem là đang biểu diễn nhị phân 0. Ta có thể đọc trạng thái của ô nhớ bằng cách sử dụng các mạch bên ngoài để kiểm tra xem transistor có đang hoạt động hay không. Đặt một điện áp lớn theo chiều ngược lại sẽ làm mất đi các electron ở cổng trôi, ô nhớ quay trở lại trạng thái nhị phân 1.



(a) Cấu trúc transistor



(b) Ô nhớ flash ở trạng thái 1



(c) Ô nhớ flash ở trạng thái 0

Hình 6.9 Hoạt động của bộ nhớ flash

Có hai loại bộ nhớ flash đặc biệt là NOR và NAND. Trong bộ nhớ flash NOR, đơn vị truy cập cơ bản là 1 bit, và tổ chức logic giống với thiết bị logic NOR. Đối với bộ nhớ flash NAND, đơn vị cơ bản là 16 hoặc 32 bit, và tổ chức logic giống với thiết bị NAND.

Bộ nhớ flash NOR cung cấp khả năng truy cập ngẫu nhiên tốc độ cao. Nó có thể đọc và ghi dữ liệu ở các vị trí cụ thể, có thể tham chiếu và lấy ra một byte đơn. Bộ nhớ flash NOR được sử dụng để lưu trữ mã hệ điều hành cho điện thoại di động và chương trình BIOS trên máy tính Windows. Bộ nhớ flash NAND đọc và ghi trong các block nhỏ. Nó được sử dụng trong ổ đĩa flash USB, thẻ nhớ (trong máy ảnh kỹ thuật số, máy nghe nhạc MP3, vv), và trong ổ SSD. NAND cung cấp mật độ bit cao hơn NOR và tốc độ ghi lớn hơn. Bộ nhớ flash NAND không cung cấp bus địa chỉ bên ngoài cho việc truy cập ngẫu nhiên nên dữ liệu phải được đọc theo block, trong đó mỗi block chứa hàng trăm đến hàng nghìn bit.

6.3.2. SSD so với HDD

Khi giá thành các ổ SSD dựa trên bộ nhớ flash giảm còn hiệu suất và mật độ bit tăng lên thì SSD ngày càng trở nên cạnh tranh với HDD. Bảng 6.5 đưa ra những so sánh điển hình giữa SSD và HDD.

SSD có những điểm vượt trội hơn HDD như sau:

- Số lần xử lý vào/ra trên giây (IOPS) cao:** Tăng đáng kể hiệu suất các hệ thống con vào/ra.
- Độ bền:** Ít bị ảnh hưởng bởi xóc và rung động.
- Tuổi thọ dài hơn:** SSD không dễ bị mài mòn cơ học.
- Tiêu thụ điện năng ít hơn:** Mỗi ổ SSD chỉ sử dụng 2.1W, ít hơn đáng kể so với các ổ cứng dung lượng tương đương.
- Khả năng chạy êm hơn và mát hơn:** Yêu cầu ít điện tích sàn, chi phí năng lượng thấp hơn, và thân thiện với môi trường hơn.

- **Thời gian truy cập và trễ ngắn hơn:** Nhanh gấp hơn 10 lần so với đĩa quay trong HDD.

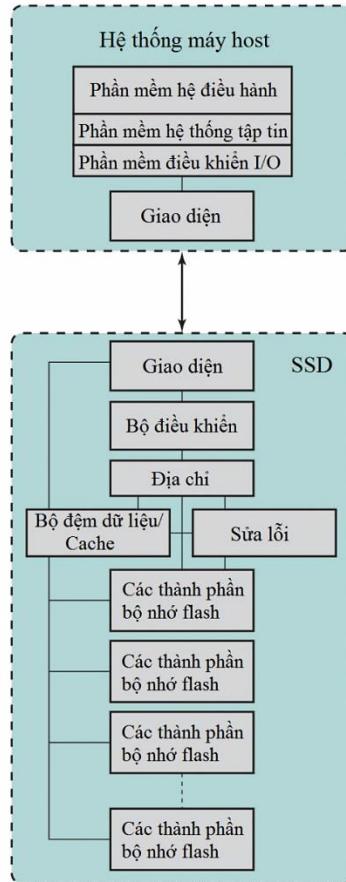
Hiện tại, HDD có ưu điểm về chi phí cho mỗi bit tốt hơn và dung lượng lớn hơn, nhưng những khác biệt này đang dần thu hẹp lại.

Bảng 6.4 So sánh ổ đĩa bán dẫn SSD và ổ cứng HDD

	Ổ flash NAND	Ổ đĩa cứng
Xử lý vào/ra trên giây	Đọc: 45000 Ghi: 15000	300
Thông lượng (MB/s)	Đọc: 200+ Ghi: 100+	Lên tới 80
Thời gian truy cập ngẫu nhiên (ms)	0.1	4 – 10
Dung lượng	Lên tới 256 GB	Lên tới 4 TB

6.3.3. Tổ chức SSD

Hình 6.10 là hình vẽ mô tả các thành phần trong hệ thống kiến trúc tổng quát gắn với bất kỳ hệ thống SSD nào. Trên hệ thống máy host, hệ điều hành yêu cầu phần mềm hệ thống tập tin truy cập dữ liệu trên đĩa. Sau đó, hệ thống tập tin gọi phần mềm điều khiển I/O. Phần mềm điều khiển I/O cung cấp truy cập máy host cho sản phẩm SSD cụ thể. Thành phần giao diện trong hình 6.11 là giao diện vật lý (giao diện điện) giữa bộ xử lý của máy host và thiết bị ngoại vi SSD. Nếu thiết bị này là một ổ cứng gắn bên trong, giao diện thường gặp là PCIe. Đối với thiết bị bên ngoài, giao diện thường gặp là USB.



Hình 6.10 Kiến trúc ổ đĩa SSD

Ngoài thành phần giao diện với hệ thống máy host, SSD còn có các thành phần sau:

- **Bộ điều khiển:** Cung cấp giao diện và sự thi hành phần sụn (firmware) ở cấp thiết bị SSD.
- **Địa chỉ:** Logic thực hiện chức năng lựa chọn trên các thành phần bộ nhớ flash.
- **Bộ đệm dữ liệu/bộ nhớ cache:** Các thành phần bộ nhớ RAM tốc độ cao được sử dụng để phối hợp tốc độ và tăng thông lượng dữ liệu.
- **Sửa lỗi:** Logic để phát hiện và sửa lỗi.
- **Các thành phần bộ nhớ flash:** Các chip flash NAND riêng lẻ.

6.3.4. Vấn đề thực tế

Có hai vấn đề thực tế gặp phải ở SSD mà không xảy ra với HDD. Thứ nhất, hiệu suất SSD có khuynh hướng giảm dần trong quá trình thiết bị được sử dụng. Để hiểu điều này, ta cần biết rằng các tệp tin được lưu trữ trên đĩa thành tập hợp nhiều page, mỗi page thường có kích thước 4 KB. Các page này không nhất thiết phải được lưu trữ thành tập hợp các page liên tục trên đĩa. Tuy nhiên, bộ nhớ flash được truy cập theo block, với kích thước block là 512 KB, do đó mỗi block thường có 128 page. Vậy giờ ta xét xem cần phải thực hiện những gì để ghi được một page vào bộ nhớ flash.

1. Toàn bộ block phải được đọc từ bộ nhớ flash và đặt vào một bộ đệm RAM. Sau đó page thích hợp trong bộ đệm RAM được cập nhật.

2. Trước khi block có thể được ghi lại vào bộ nhớ flash, toàn bộ block của bộ nhớ flash phải được xoá hoàn toàn - không thể chỉ xóa một page trong bộ nhớ flash.
3. Toàn bộ block từ bộ đệm lúc này được ghi lại vào bộ nhớ flash.

Lúc này, khi một ổ đĩa flash tương đối rỗng và một tập tin mới được tạo ra, các page của tập tin đó sẽ được ghi liên tiếp vào ổ đĩa, do đó chỉ một hoặc một vài block bị ảnh hưởng. Tuy nhiên, theo thời gian, do cách bộ nhớ ảo hoạt động, các tập tin dần bị phân mảnh, với các page được phân tán trên nhiều block. Khi ổ đĩa được lắp đầy hơn, sự phân mảnh càng nhiều hơn, vì vậy việc ghi một tập tin mới có thể ảnh hưởng đến nhiều block. Do đó, khi đĩa càng bị lắp đầy, việc ghi nhiều page từ một block càng chậm hơn. Các nhà sản xuất đã phát triển một loạt các kỹ thuật để bù đắp cho đặc tính này của bộ nhớ flash, chẳng hạn như dành ra một phần đáng kể của SSD làm không gian bổ sung cho các hành động ghi (gọi là over-provisioning), sau đó xóa các page không hoạt động trong thời gian nhàn rỗi để giảm phân mảnh đĩa.

Vấn đề thứ hai của ổ đĩa bộ nhớ flash là bộ nhớ flash trở nên không sử dụng được nữa sau một số lần ghi nhất định. Khi các ô nhớ flash bị nén, chúng sẽ mất khả năng ghi và giữ các giá trị. Giới hạn thông thường là 100000 lần ghi. Các kỹ thuật kéo dài tuổi thọ của ổ đĩa SSD bao gồm sử dụng bộ nhớ cache để trì hoãn và gộp nhóm các thao tác ghi, sử dụng các thuật toán san bằng nhằm phân bố đều các lượt ghi trên khối của các ô nhớ và các kỹ thuật phức tạp để quản lý bad-block. Ngoài ra, các nhà cung cấp đang triển khai SSD ở các cấu hình RAID để giảm nguy cơ mất dữ liệu. Hầu hết các thiết bị flash cũng có khả năng ước lượng thời gian sống còn lại của mình để hệ thống có thể dự đoán trước hỏng hóc và có hành động dự phòng.

6.4. BỘ NHỚ QUANG

Năm 1983, một trong những sản phẩm tiêu dùng thành công nhất mọi thời đại đã được giới thiệu: hệ thống âm thanh số chạy đĩa CD. CD là đĩa không xoá được có thể lưu trữ hơn 60 phút thông tin âm thanh trên một mặt đĩa. Thành công thương mại khổng lồ của đĩa CD đã thúc đẩy sự phát triển của công nghệ lưu trữ trên đĩa quang chi phí thấp và cách mạng hóa sự lưu trữ dữ liệu máy tính. Một loạt các hệ thống đĩa quang đã được giới thiệu (Bảng 6.6).

Bảng 6.5 Các sản phẩm đĩa quang

CD (Compact Disk) Đĩa không xoá được, lưu trữ thông tin âm thanh số hóa. Hệ thống tiêu chuẩn sử dụng đĩa đường kính 12 cm và có thể ghi lại hơn 60 phút không gián đoạn.
CD-ROM (Compact Disk Read-Only Memory) Đĩa không xoá được được sử dụng để lưu trữ dữ liệu máy tính. Hệ thống tiêu chuẩn sử dụng đĩa đường kính 12 cm và có thể chứa hơn 650 Mbyte.
CD-R (CD Recordable) Tương tự CD-ROM. Người dùng có thể ghi đĩa một lần duy nhất.

CD-RW

(CD Rewritable) Tương tự CD-ROM. Người dùng có thể xoá và ghi lại vào đĩa nhiều lần.

DVD

(Digital Versatile Disk) Công nghệ tạo ra dạng biểu diễn nén và số hoá của thông tin video, cũng như lượng lớn các dữ liệu số khác. Có hai loại đường kính 8 và 12 cm được sử dụng, với dung lượng hai mặt đĩa lên đến 17 Gbyte. DVD cơ bản là chỉ đọc (DVD-ROM).

DVD-R

(DVD Recordable) Tương tự DVD-ROM. Người dùng có thể ghi đĩa một lần duy nhất. Chỉ sử dụng đĩa một mặt.

DVD-RW

(DVD Rewritable) Tương tự DVD-ROM. Người dùng có thể xoá và ghi lại vào đĩa nhiều lần. Chỉ sử dụng đĩa một mặt.

Blu-ray DVD

Đĩa video độ phân giải cao. Cung cấp mật độ lưu trữ dữ liệu lớn hơn nhiều so với đĩa DVD, sử dụng tia laser 405 nm (tia xanh tím). Một lớp trên một mặt có thể lưu trữ 25 Gbyte.

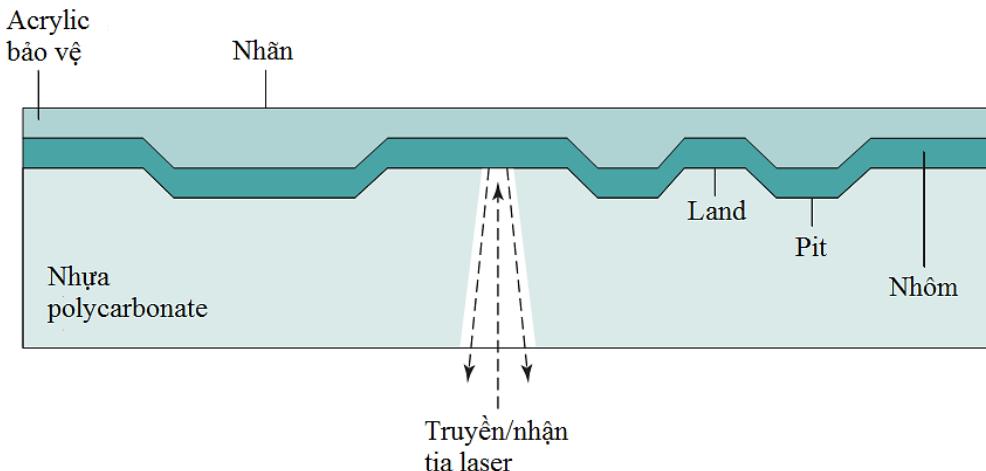
6.4.1. Đĩa CD

Đĩa CD âm thanh và CD-ROM (đĩa CD chỉ đọc) cùng sử dụng công nghệ tương tự nhau. Điểm khác biệt chính là máy chạy đĩa CD-ROM có độ gồ ghề lớn hơn và có thiết bị sửa lỗi để đảm bảo rằng dữ liệu được truyền đúng từ đĩa vào máy tính. Cả hai loại đĩa này đều được sản xuất theo cùng một cách. Đĩa được chế tạo từ nhựa polycarbonate. Thông tin số (âm nhạc hoặc dữ liệu máy tính) được in thành một chuỗi các lỗ cực nhỏ (pit) trên bề mặt polycarbonate. Việc này được thực hiện bằng tia laser tập trung cường độ cao, sẽ tạo ra một đĩa master. Đĩa master được dùng làm khuôn góc để tạo ra các bản sao trên polycarbonate. Bề mặt sau đó được phủ một lớp có độ phản xạ tốt, thường là nhôm hoặc vàng. Bề mặt này tiếp tục được phủ lên một lớp sơn acrylic trong suốt để chống bụi bẩn và trầy xước. Cuối cùng, có thể dùng kỹ thuật in lụa để in nhãn hiệu lên bề mặt acrylic.

Thông tin được lấy ra từ đĩa CD hoặc CD-ROM bằng một tia laser công suất thấp bên trong máy chạy đĩa quang hoặc ổ đĩa. Tia laser chiếu qua lớp polycarbonate trong khi động cơ quay đĩa quét qua nó (Hình 6.11). Cường độ ánh sáng phản xạ của tia laser thay đổi khi nó gặp lỗ **pit**. Cụ thể, nếu chùm tia laser lọt vào lỗ pit có bề mặt hơi gồ ghề, ánh sáng bị phân tán và cường độ ánh sáng phản xạ lại nguồn sẽ thấp. Vùng nằm giữa các pit được gọi là **land**. Land là bề mặt nhẵn, phản chiếu ánh sáng lại với cường độ cao hơn. Sự thay đổi giữa pit và land được nhận biết bởi một bộ cảm biến quang học và chuyển đổi thành tín hiệu số. Bộ cảm biến đều đặn kiểm tra bề mặt. Sự bắt đầu hoặc kết thúc của pit biểu diễn cho logic 1; khi không có sự thay đổi độ cao thì logic 0 được ghi lại.

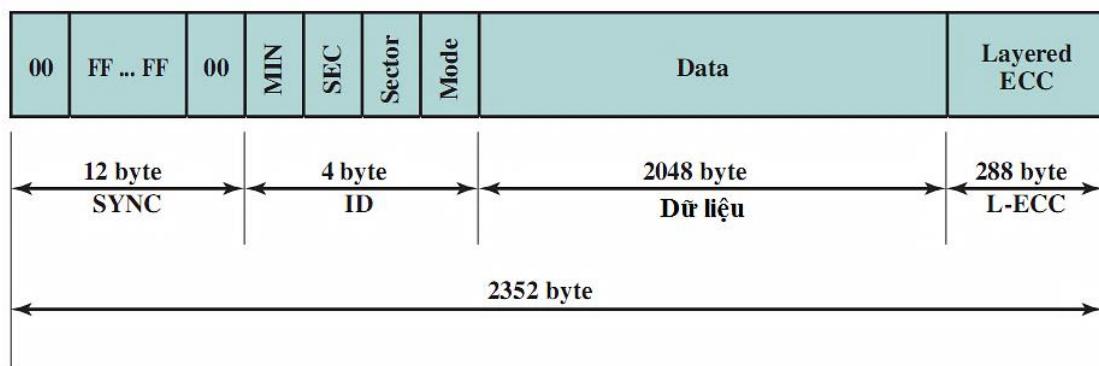
Nhớ lại về đĩa từ, thông tin được ghi trong các track đồng tâm. Với phương pháp vận tốc góc không đổi CAV, số bit trong các track là bằng nhau. Sự gia tăng mật độ bit đạt được khi sử dụng phương pháp ghi nhiều vùng, trong đó bề mặt được chia thành nhiều

vùng, với các vùng xa trung tâm chứa nhiều bit hơn các vùng gần trung tâm hơn. Mặc dù kỹ thuật này làm tăng dung lượng, nhưng nó vẫn không tối ưu.



Hình 6.11 Hoạt động của CD

Để đạt được dung lượng lớn hơn, đĩa CD và CD-ROM không tổ chức thông tin thành các track đồng tâm. Thay vào đó, đĩa có một track dạng xoáy ốc duy nhất, bắt đầu ở gần trung tâm và xoáy ốc ra phía ngoài vành đĩa. Các sector gần vành ngoài của đĩa có chiều dài tương tự như sector bên trong của đĩa. Do đó, thông tin được đóng gói đều trên đĩa trong các phân đoạn có cùng kích thước và chúng được quét cùng tốc độ bằng cách quay đĩa ở tốc độ khác nhau. Lúc này các pit được đọc bằng tia laser ở **tốc độ tuyến tính không đổi** (CLV – constant linear velocity). Đĩa quay chậm hơn để truy cập gần vành ngoài, quay nhanh hơn để truy cập ở gần trung tâm đĩa. Do đó, cả dung lượng của một track và trễ quay đều tăng lên đôi với các vị trí gần vành ngoài của đĩa. Dung lượng dữ liệu của đĩa CD-ROM là khoảng 680 MB.



Hình 6.12 Định dạng block CD-ROM

Dữ liệu trên CD-ROM được tổ chức thành một chuỗi các block. Định dạng điển hình của một block được thể hiện trong hình 6.12. Nó gồm các trường sau:

- **Đồng bộ (Sync):** Trường đồng bộ xác định điểm bắt đầu của một block. Nó bao gồm một byte toàn bit 0, 10 byte toàn bit 1, và một byte toàn bit 0.

- **Header:** Header chứa địa chỉ block và byte chế độ (mode). Chế độ 0 xác định trường dữ liệu trống; chế độ 1 xác định việc sử dụng mã sửa lỗi và 2048 byte dữ liệu; chế độ 2 xác định 2336 byte dữ liệu người dùng và không sử dụng mã sửa lỗi.
- **Dữ liệu:** Dữ liệu người dùng.
- **Phụ trợ:** Dữ liệu người dùng bổ sung ở chế độ 2. Trong chế độ 1, đây là 288 byte mã sửa lỗi.

Khi sử dụng CLV, việc truy cập ngẫu nhiên trở nên khó khăn hơn. Việc định vị một địa chỉ cụ thể liên quan đến việc di chuyển đầu đọc/ghi đến một khu vực, điều chỉnh tốc độ quay và đọc địa chỉ, rồi sau đó vi chỉnh để tìm và truy cập vào sector cụ thể.

CD-ROM thích hợp cho việc phân phối lượng lớn dữ liệu cho một số lượng lớn người dùng. So với đĩa từ truyền thống, CD-ROM có hai ưu điểm:

- Đĩa quang cùng với các thông tin được lưu trữ trên đó có thể được nhân bản rộng rãi một cách không tốn kém – không giống với đĩa từ. Muốn nhân rộng cơ sở dữ liệu trên một đĩa từ thì phải sử dụng hai ổ đĩa để sao chép từng đĩa một.
- Đĩa quang là đĩa tháo được, cho phép sự lưu trữ trên bản thân đĩa. Hầu hết các đĩa từ là không tháo ra được. Thông tin trên đĩa từ không tháo được trước tiên phải được sao chép vào một thiết bị lưu trữ khác trước khi đĩa/ổ đĩa có thể được sử dụng để lưu trữ thông tin mới.

Nhược điểm của CD-ROM như sau:

- Là đĩa chỉ đọc và không thể cập nhật.
- Thời gian truy cập lâu hơn nhiều so với ổ đĩa từ, khoảng nửa giây.

6.4.2. CD-R

Để chứa các ứng dụng trong đó chỉ cần một hoặc một số lượng ít bản sao của một bộ dữ liệu, đĩa CD ghi một lần đọc nhiều lần hay được gọi là đĩa **CD-R** (CD recordable) đã được phát triển. Đối với CD-R, đĩa có thể được ghi một lần bằng tia laser có cường độ vừa phải. Do đó, người dùng có thể ghi dữ liệu một lần và đọc đĩa đã ghi.

Môi trường đĩa CD-R tương tự nhưng không giống hoàn toàn đĩa CD hoặc CD-ROM. Đối với đĩa CD và CD-ROM, thông tin được ghi lại bởi các lỗ trên bề mặt của môi trường, làm thay đổi cường độ phản xạ. Đối với đĩa CD-R, môi trường chứa một lớp màu nhuộm (dye). Lớp dye được sử dụng để thay đổi độ phản xạ và được kích hoạt bởi laser cường độ cao. Đĩa có thể được đọc bởi ổ đĩa CD-R hoặc ổ đĩa CD-ROM.

Đĩa quang CD-R phù hợp đối với việc lưu trữ tài liệu và tập tin. Nó cung cấp bản ghi vĩnh viễn cho khối lượng lớn dữ liệu người dùng.

6.4.3. CD-RW

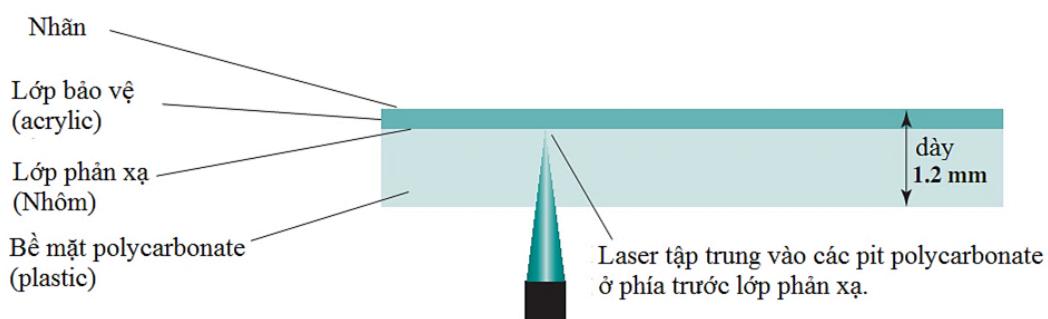
Đĩa quang **CD-RW** (CD Rewritable) có thể ghi và ghi đè nhiều lần, như với đĩa từ. Nó sử dụng phương pháp quang học thuận tuý là **thay đổi pha**. Đĩa thay đổi pha sử dụng vật liệu có hai mức độ phản xạ khác nhau đáng kể ở hai trạng thái pha khác nhau. Trạng thái thứ nhất là trạng thái vô định, trong đó các phân tử có hướng ngẫu nhiên và phản xạ ánh sáng kém. Trạng thái thứ hai là trạng thái tinh thể, có bề mặt nhẵn phản xạ ánh sáng tốt. Một chùm tia laser có thể thay đổi vật liệu này từ pha này sang pha kia. Nhược điểm chính

của đĩa quang thay đổi pha là cuối cùng vật liệu này sẽ vĩnh viễn mất đi tính chất mong muốn của nó. Các vật liệu hiện tại có thể được sử dụng trong khoảng 500.000 đến 1.000.000 chu kỳ xóa.

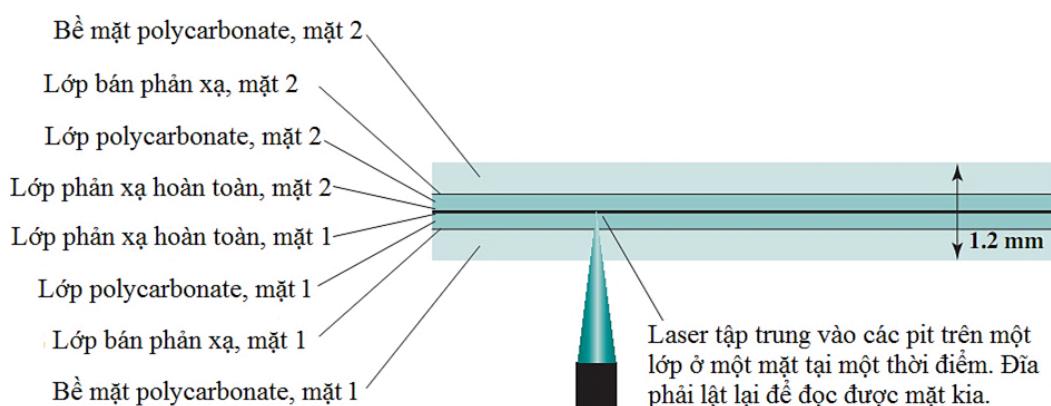
CD-RW có ưu điểm vượt trội rõ ràng so với đĩa CD-ROM và CD-R ở chỗ nó có thể ghi lại được và do đó được sử dụng làm bộ lưu trữ thứ cấp thực sự. Như vậy, nó cạnh tranh với đĩa từ. Một ưu điểm quan trọng của đĩa quang là dung sai kỹ thuật của đĩa quang nhỏ hơn nhiều so với đĩa từ có dung lượng cao. Do vậy, chúng thể hiện độ tin cậy cao hơn và tuổi thọ dài hơn.

6.4.4. DVD

Đĩa **DVD** (digital versatile disk) được chấp nhận là sự thay thế cho băng video analog VHS. Đĩa DVD thay thế cho băng video sử dụng trong đầu đọc băng video VCR và quan trọng hơn nữa là thay thế đĩa CD-ROM trong máy tính cá nhân và máy chủ. DVD đưa video vào thời đại kỹ thuật số. Nó lưu trữ phim với chất lượng hình ảnh ánh sáng, và có thể được truy cập ngẫu nhiên như đĩa CD âm thanh. Máy đọc DVD cũng có thể đọc đĩa CD. Đĩa có thể chứa lượng rất lớn dữ liệu, tính đến hiện tại là gấp bảy lần đĩa CD-ROM. Với dung lượng lưu trữ khổng lồ và chất lượng sóng động của DVD, các trò chơi trên PC trở nên thực hơn và phần mềm giáo dục kết hợp thêm nhiều video hơn. Tiếp đó, lưu lượng truy cập qua Internet và mạng nội bộ công ty cũng đạt đến đỉnh giá trị mới, khi video cũng được kết hợp vào các trang Web.



(a) CD-ROM dung lượng 682 MB



(b) DVD-ROM, hai mặt, hai lớp - dung lượng 17 GB

Hình 6.13 CD-ROM và DVD-ROM

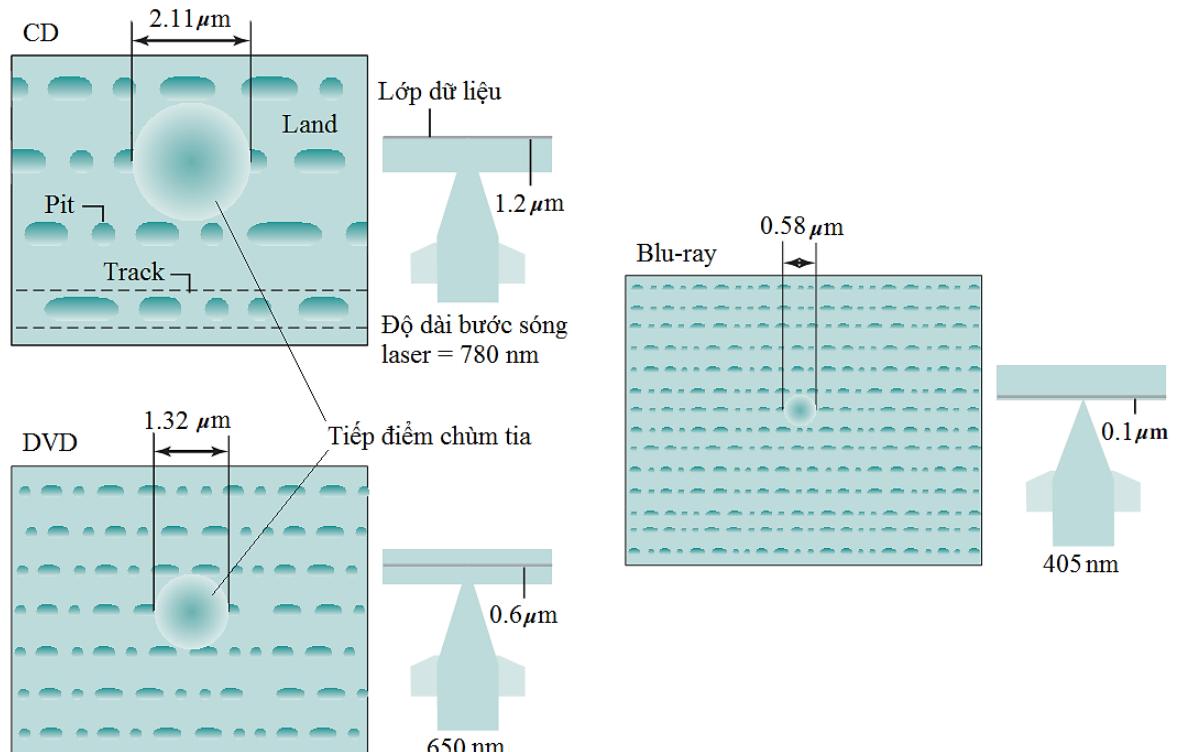
Dung lượng lớn hơn của đĩa DVD đạt được do ba sự khác biệt so với đĩa CD (Hình 6.13):

1. Các bit trên đĩa DVD được sắp xếp sát nhau hơn. Khoảng cách giữa các vòng xoáy ốc trên đĩa CD là $1,6 \mu\text{m}$ và khoảng cách tối thiểu giữa các pit trên đường xoáy ốc là $0,834 \mu\text{m}$. DVD sử dụng tia laser với bước sóng ngắn hơn và đạt được khoảng cách các vòng là $0,74 \mu\text{m}$ và khoảng cách tối thiểu giữa các pit là $0,4 \mu\text{m}$. Hai cải tiến này làm cho dung lượng tăng gấp bảy lần, khoảng $4,7 \text{ GB}$.
2. DVD sử dụng thêm một lớp pit và land thứ hai phía trên lớp đầu tiên. Đĩa DVD hai lớp có một lớp bán phản xạ phía trên lớp phản xạ, và bằng cách điều chỉnh tiêu điểm, tia laser trong ổ đĩa DVD có thể đọc từng lớp riêng biệt. Kỹ thuật này làm dung lượng của đĩa tăng gần gấp đôi, khoảng $8,5 \text{ GB}$. Độ phản xạ của lớp thứ hai thấp hơn làm hạn chế khả năng lưu trữ của nó, do đó không đạt được sự tăng gấp đôi dung lượng.
3. Đĩa **DVD-ROM** có thể có hai mặt, trong khi dữ liệu chỉ được ghi trên một mặt của đĩa CD. Điều này khiến cho tổng dung lượng đạt đến 17 GB .

Giống như CD, DVD cũng có các phiên bản chỉ đọc và ghi được (Bảng 6.6).

6.4.5. Đĩa quang độ phân giải cao

Đĩa quang độ phân giải cao được thiết kế để lưu trữ video độ phân giải cao và cung cấp dung lượng lưu trữ lớn hơn đáng kể so với đĩa DVD. Mật độ bit cao hơn đạt được do sử dụng tia laser với bước sóng ngắn hơn, trong dải xanh tím. Các pit dữ liệu trên đĩa quang độ phân giải cao nhỏ hơn so với trên đĩa DVD vì bước sóng laser ngắn hơn.



Hình 6.14 Đặc tính bộ nhớ quang

Ban đầu, trên thị trường, có hai định dạng và công nghệ đĩa cạnh tranh lẫn nhau là: HD DVD và **Blu-ray** DVD. Cuối cùng, công nghệ Blu-ray đạt được sự thống trị thị trường. HD DVD có thể lưu trữ 15 GB trên một lớp duy nhất trên một mặt. Blu-ray đặt lớp dữ liệu trên đĩa dày với tia laser hơn (Hình 6.14). Điều này cho phép độ tập trung cao hơn và ít biến dạng hơn, do đó các pit và track nhỏ hơn. Blu-ray có thể lưu trữ 25 GB trên một lớp. Có ba phiên bản đĩa Blu-ray: chỉ đọc (BD-ROM), có thể ghi một lần (BD-R), và có thể ghi lại nhiều lần (BD-RE).

6.5. BĂNG TỪ

Hệ thống băng từ sử dụng kỹ thuật đọc và ghi giống như hệ thống đĩa. Môi trường đọc/ghi là băng polyester mềm dẻo được phủ bởi vật liệu có từ tính. Lớp phủ này có thể là các hạt kim loại tinh khiết trong chất liên kết đặc biệt hoặc màng kim loại mạ hơi. Băng từ và đầu đọc băng từ tương tự như hệ thống máy ghi băng gia đình. Băng có độ rộng từ 0,38 cm đến 1,27 cm. Băng được đóng gói thành cuộn, phải cuộn qua hai trực quay và được đặt trong hộp cartridge.

Dữ liệu trên băng được tổ chức thành một số track song song chạy dài theo chiều đọc. Các hệ thống băng trước đây thường sử dụng 9 track. Với 9 track, hệ thống có thể lưu trữ một byte tại một thời điểm và một bit chẵn lẻ bổ sung ở track thứ chín. Sau này, hệ thống băng từ có thể sử dụng 18 hoặc 36 track, tương ứng với một word hoặc double word. Việc ghi dữ liệu theo dạng này được gọi là ghi song song. Hầu hết các hệ thống hiện đại sử dụng kỹ thuật ghi tuần tự, trong đó dữ liệu được đặt trong một chuỗi bit đọc theo mỗi track, như được thực hiện ở đĩa từ. Giống với đĩa từ, dữ liệu trên băng được đọc và ghi theo từng block liên tiếp (được gọi là bản ghi vật lý). Các block trên băng được ngăn cách bởi khe trống (gap hoặc interrecord gap). Giống đĩa từ, băng từ cũng được định dạng để hỗ trợ cho việc xác định vị trí bản ghi vật lý.

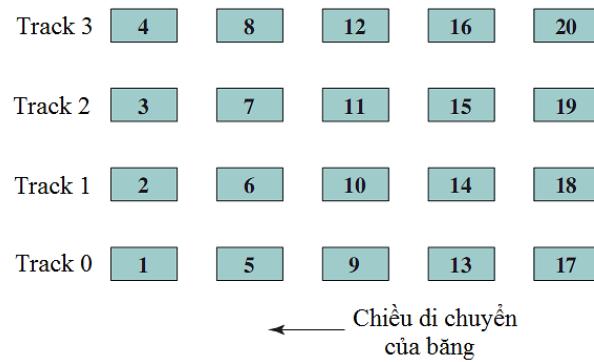
Kỹ thuật ghi diễn hình được sử dụng trong băng tuần tự được gọi là ghi serpentine (dạng lượn sóng). Trong kỹ thuật này, khi dữ liệu được ghi, nhóm bit đầu tiên được ghi đọc trên toàn bộ chiều dài của băng. Khi đến điểm kết thúc băng, đầu ghi được định vị lại để bắt đầu ghi vào một track mới và các bit lại được ghi đọc toàn bộ chiều dài băng, nhưng lần này theo chiều ngược lại. Quá trình đó tiếp tục, ghi xuôi rồi ghi ngược, cho đến khi băng đầy (Hình 6.15a). Để tăng tốc độ, đầu đọc-ghi có khả năng đọc và ghi một số track liền kề cùng một lúc (thường là từ hai đến tám track). Dữ liệu vẫn được ghi tuần tự vào từng track, nhưng các block liên tiếp được lưu trữ trên các track liền kề, như ví dụ trong Hình 6.15b.

Hệ thống máy chạy băng từ là một thiết bị truy cập tuần tự. Nếu đầu đọc đang được định vị ở bản ghi 1, sau đó để đọc bản ghi N , nó cần phải đọc lần lượt từng bản ghi vật lý từ 1 đến $N - 1$. Nếu đầu đọc hiện đang đặt ở vị trí vượt quá bản ghi mong muốn, cần phải tua băng ngược lại một khoảng nhất định và bắt đầu đọc về phía trước. Không giống như đĩa, băng chỉ chuyển động trong quá trình đọc hoặc ghi.

Ngược lại, ổ đĩa được gọi là thiết bị truy cập trực tiếp. Ổ đĩa không cần phải lần lượt đọc tất cả các sector trên đĩa để đến được một vị trí sector mong muốn. Nó chỉ cần chờ các sector trong một track đến và có thể truy cập liên tiếp vào bất kỳ track nào.



(a) Đọc và ghi kiểu lượn sóng



(b) Bố trí block trong hệ thống đọc-ghi 4 track đồng thời

Hình 6.15 Đặc tính điển hình của băng từ

Băng từ là loại bộ nhớ thứ cấp đầu tiên. Nó vẫn được sử dụng rộng rãi như một dạng lưu trữ có tốc độ chậm nhất, rẻ nhất trong hệ thống phân cấp bộ nhớ.

6.6. CÂU HỎI

1. Ưu điểm của việc sử dụng chất nền thủy tinh trong sản xuất đĩa từ là gì?
2. Dữ liệu được ghi lên đĩa từ như thế nào?
3. Dữ liệu được đọc ra từ đĩa từ như thế nào?
4. So sánh hai kỹ thuật CAV vận tốc góc không đổi và ghi nhiều vùng?
5. Định nghĩa *track*, *cylinder* và *sector*?
6. Kích thước điển hình của sector là bao nhiêu?
7. Định nghĩa các thuật ngữ: *thời gian tìm kiếm*, *trễ quay*, *thời gian truy cập*, và *thời gian truyền*?
8. Những đặc điểm chung của các cấp độ RAID là gì?
9. Phân biệt các cấp RAID?
10. Giải thích thuật ngữ *phân dài dữ liệu*?
11. Độ dư thừa trong một hệ thống RAID được tạo ra như thế nào?
12. Phân biệt hai kỹ thuật truy cập song song và truy cập độc lập trong RAID?
13. Phân biệt CAV và CLV?
14. Trình bày điểm khác biệt giữa đĩa CD và DVD?

Trình bày phương pháp ghi lượn sóng trong lưu trữ băng từ?

Chương 7. MODULE VÀO/RA (I/O)

Ngoài bộ vi xử lý và bộ nhớ, thành phần quan trọng thứ ba của một hệ thống máy tính là các module I/O. Mỗi module giao tiếp với bus hệ thống hoặc bộ chuyển mạch trung tâm và điều khiển một hoặc nhiều thiết bị ngoại vi. Một module I/O không đơn giản chỉ kết nối cơ học các thiết bị với bus hệ thống mà còn chứa các logic thực hiện chức năng truyền thông tin giữa thiết bị ngoại vi và bus.

Vậy tại sao ta không kết nối trực tiếp thiết bị ngoại vi bus hệ thống. Có một số lý do như sau:

- Có rất nhiều thiết bị ngoại vi với nhiều phương thức hoạt động khác nhau. Nếu bộ xử lý kết nối trực tiếp với thiết bị ngoại vi nó sẽ cần phải được trang bị các logic để điều khiển các thiết bị đó, điều này không cần thiết và gây lãng phí tài nguyên của bộ xử lý.
- Tốc độ truyền dữ liệu của thiết bị ngoại vi thường chậm hơn nhiều so với bộ nhớ hoặc bộ vi xử lý. Do đó, sẽ là không thực tế nếu sử dụng bus tốc độ cao để giao tiếp trực tiếp với thiết bị ngoại vi.
- Mặt khác, tốc độ truyền dữ liệu của một số thiết bị ngoại vi nhanh hơn bộ nhớ hoặc bộ vi xử lý. Một lần nữa, sự chênh lệch này sẽ dẫn đến sự không hiệu quả trong hiệu năng hệ thống nếu ta không có cơ chế quản lý phù hợp.
- Thiết bị ngoại vi thường sử dụng các định dạng dữ liệu và kích thước từ khác so với máy tính mà chúng được gắn vào.

Do đó, việc sử dụng một module I/O là cần thiết. Module này có hai chức năng chính (Hình 7.1):

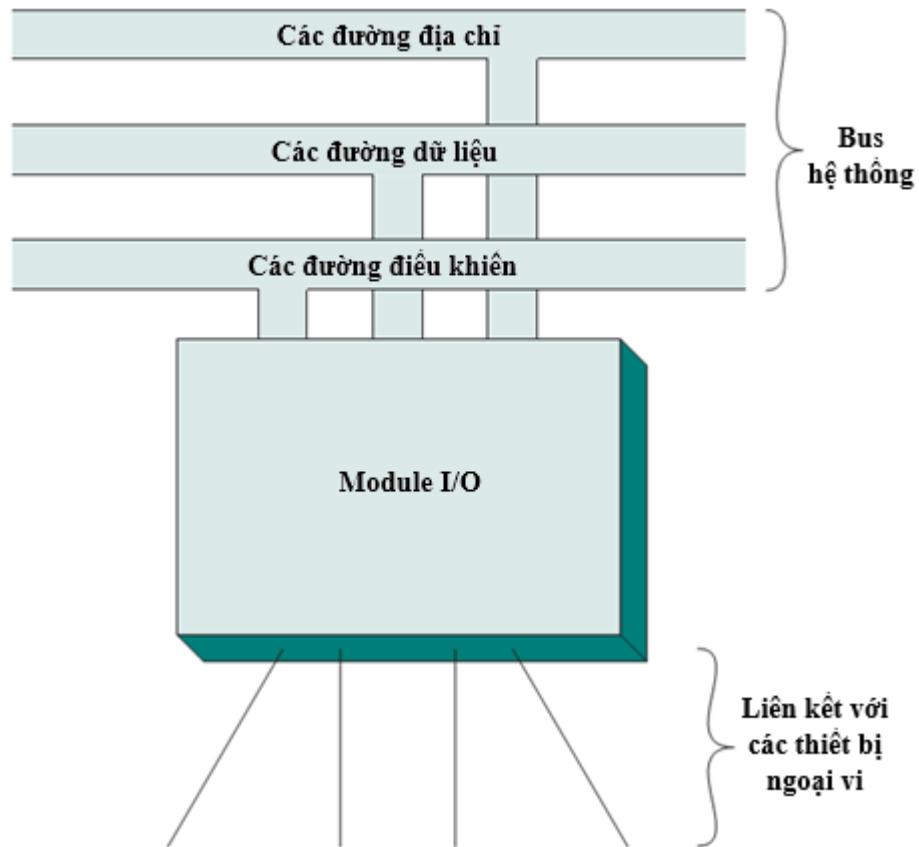
- Giao tiếp với bộ xử lý và bộ nhớ thông qua bus hệ thống hoặc một chuyển mạch trung tâm
- Giao tiếp với một hoặc nhiều thiết bị ngoại vi bằng các liên kết dữ liệu phù hợp

7.1. THIẾT BỊ NGOẠI VI

Các hoạt động I/O được thực hiện thông qua một loạt các thiết bị ngoại cung cấp một phương tiện trao đổi dữ liệu giữa môi trường bên ngoài và máy tính. Thiết bị ngoại kết nối với máy tính thông qua một liên kết tới module I/O (Hình 7.1). Liên kết này được sử dụng để chuyển tiếp điều khiển, trạng thái và dữ liệu giữa module I/O và thiết bị ngoại vi.

Chúng ta có thể phân loại các thiết bị ngoại vi thành ba loại như sau:

- Con người có thể đọc được: Thích hợp để giao tiếp với người sử dụng máy tính
- Máy có thể đọc được: Thích hợp để giao tiếp với thiết bị
- Truyền thông tin: Thích hợp để giao tiếp với các thiết bị từ xa



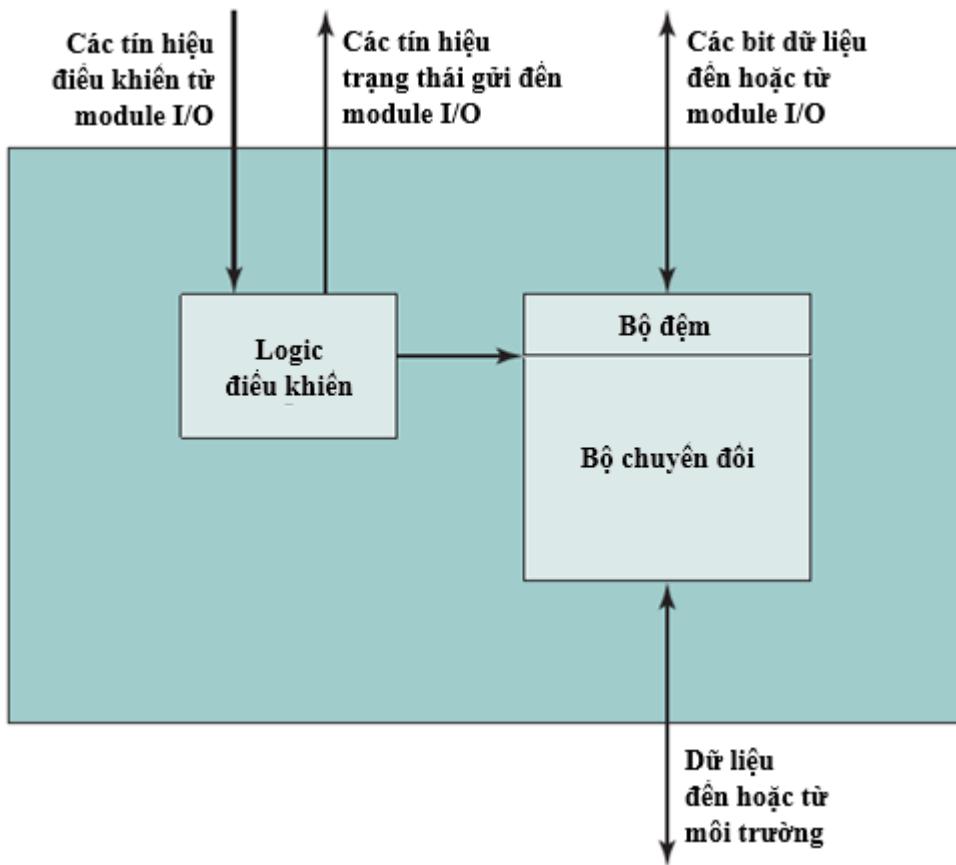
Hình 7.1 Mô hình cơ bản của các module vào/ra

Ví dụ về thiết bị con người có thể đọc được đó là màn hình và máy in. Các thiết bị máy có thể đọc được là các hệ thống đĩa và băng từ, các cảm biến và bộ truyền động được sử dụng trong ứng dụng robot. Có một điểm cần chú ý là ta đang xem các hệ thống đĩa và băng từ là các thiết bị I/O trong khi ở chương 6, ta xem chúng như các thiết bị nhớ. Về mặt chức năng, các thiết bị này là một phần của hệ thống phân cấp bộ nhớ, tuy nhiên về mặt cấu trúc, các thiết bị này được điều khiển bởi các module I/O và do đó phải được xét đến trong chương này. Thiết bị truyền thông tin cho phép máy tính trao đổi dữ liệu với thiết bị từ xa.

Hình 7.2 là sơ đồ khái chung của các thiết bị ngoại vi. Giao diện với module I/O thông qua các tín hiệu điều khiển, dữ liệu và trạng thái. Các tín hiệu điều khiển xác định chức năng mà thiết bị sẽ thực hiện, chẳng hạn như gửi dữ liệu đến module I/O (INPUT hoặc ĐỌC), tiếp nhận dữ liệu từ module I/O (OUTPUT hoặc GHI), báo cáo trạng thái hoặc thực hiện một số chức năng điều khiển đặc biệt đối với thiết bị (ví dụ: đặt đầu đọc/ghi của đĩa từ vào một vị trí nào đó). *Dữ liệu* ở dạng một tập bit được gửi tới hoặc nhận từ module I/O. Các tín hiệu trạng thái cho biết trạng thái của thiết bị. Ví dụ là READY/NOT-READY (sẵn sàng hay chưa sẵn sàng) để thông báo thiết bị có sẵn sàng cho việc truyền dữ liệu hay không.

Logic điều khiển tiếp nhận tín hiệu điều khiển từ module I/O và thực hiện việc điều khiển hoạt động thiết bị. Bộ chuyển đổi thực hiện việc chuyển đổi dữ liệu từ dạng tín hiệu điện sang các dạng biểu diễn khác đối với các thiết bị ra và từ các dạng tín hiệu khác nhau thành dữ liệu dạng điện với các thiết bị vào. Bộ chuyển đổi thường đi kèm với một bộ nhớ

đệm để lưu trữ dữ liệu tạm thời trong quá trình trao đổi dữ liệu giữa module I/O và môi trường bên ngoài; một kích thước bộ đệm phổ biến từ 8 đến 16 bit.



Hình 7.2 Sơ đồ khái niệm của các thiết bị ngoại vi

Giao diện giữa module I/O và thiết bị bên ngoài sẽ được trình bày trong Phần 7.7. Giao diện giữa thiết bị ngoại vi và môi trường không nằm trong phạm vi kiến thức của cuốn giáo trình này nhưng chúng tôi vẫn đưa ra một số ví dụ ngắn sau đây.

7.1.1. Bàn phím/Màn hình

Các thiết bị ngoại vi phổ biến nhất tương tác giữa con người và máy tính là bàn phím và màn hình. Người dùng đưa thông tin vào thông qua bàn phím. Dữ liệu vào này sau đó được truyền đến máy tính và cũng có thể được hiển thị trên màn hình. Ngoài ra, màn hình hiển thị dữ liệu được máy tính đưa ra.

Đơn vị dữ liệu cơ bản được trao đổi là các ký tự. Mỗi ký tự được liên kết với một mã, thường có chiều dài 7 đến 8 bit. Bảng mã ký tự được sử dụng phổ biến nhất là bảng mã International Reference Alphabet (IRA). Mỗi ký tự trong bảng mã này được biểu diễn bởi một mã nhị phân 7-bit; do đó có 128 ký tự khác nhau có thể được mã hóa. Ký tự có hai loại: ký tự có thể in được và ký tự điều khiển. Các ký tự in được là chữ cái, chữ số và một số ký tự đặc biệt có thể được in trên giấy hoặc hiển thị trên màn hình. Các ký tự điều khiển liên quan đến điều khiển việc in hoặc hiển thị ký tự. Một số ký tự điều khiển khác liên quan đến các thủ tục truyền thông.

Với bàn phím, khi người dùng nhấn một phím, bàn phím sẽ phát ra một tín hiệu điện, tín hiệu này được bộ chuyển đổi dịch sang mẫu bit nhị phân dưới dạng mã IRA tương ứng. Mẫu bit này sau đó được truyền đến module I/O trong máy tính. Với đầu ra, các ký tự biểu diễn bằng mã IRA được truyền đến thiết bị ngoại vi từ module I/O. Bộ chuyển đổi của thiết bị này giải mã và gửi thông tin đến thiết bị đầu ra để hiển thị ký tự hoặc thực hiện chức năng điều khiển.

7.1.2. Ô cứng

Một ô đĩa chứa một số bộ phận điện tử để trao đổi các tín hiệu dữ liệu, điều khiển và trạng thái với module I/O cộng với các bộ phận khác để điều khiển cơ chế đọc/ghi đĩa. Với đĩa đầu cố định, bộ chuyển đổi sẽ thực hiện việc chuyển đổi giữa các mẫu từ tính trên bề mặt đĩa thành các bit ghi vào bộ đệm và ngược lại (Hình 7.2). Với đĩa cứng đầu di chuyển, cũng cần phải có một bộ phận thực hiện chức năng điều khiển việc di chuyển đầu đọc/ghi đến vị trí nhất định trên bề mặt đĩa.

7.2. CÁC MODULE I/O

7.2.1. Chức năng của module

Các chức năng chính module I/O như sau:

- Điều khiển và định thời
- Giao tiếp với bộ vi xử lý
- Giao tiếp với thiết bị
- Đệm dữ liệu
- Phát hiện lỗi

Trong bất kỳ thời điểm nào, bộ xử lý có thể giao tiếp với một hoặc nhiều thiết bị ngoài theo nhiều cách khác nhau tùy thuộc vào yêu cầu vào/ra dữ liệu của chương trình. Các tài nguyên bên trong, ví dụ như bộ nhớ chính và hệ thống bus sẽ bị chia sẻ với nhiều hoạt động, trong đó gồm cả các hoạt động I/O. Vì vậy, chức năng I/O bao gồm cả các yêu cầu **điều khiển và định thời** để phối hợp luồng lưu lượng giữa các tài nguyên bên trong và các thiết bị ngoài. Ví dụ quá trình điều khiển truyền dữ liệu từ thiết bị ngoại vi đến bộ xử lý gồm các bước sau:

1. Bộ xử lý yêu cầu module I/O kiểm tra trạng thái của thiết bị.
2. Module I/O trả về trạng thái thiết bị.
3. Nếu thiết bị đang hoạt động và sẵn sàng truyền, bộ xử lý yêu cầu truyền dữ liệu bằng cách gửi lệnh cho module I/O.
4. Module I/O nhận một đơn vị dữ liệu (ví dụ, 8 hoặc 16 bit) từ thiết bị ngoài.
5. Dữ liệu được module I/O chuyển sang cho bộ xử lý.

Nếu hệ thống chỉ sử dụng một bus thì tương tác giữa bộ vi xử lý và module I/O sẽ phải sử dụng các cơ chế phân xử bus.

Trong đó, module I/O phải thực hiện việc giao tiếp với bộ vi xử lý và thiết bị ngoại vi. **Giao tiếp với bộ xử lý** như sau:

- **Giải mã lệnh:** Module I/O nhận lệnh từ bộ xử lý (thường được gửi dưới dạng các tín hiệu trên bus điều khiển). Ví dụ, một module I/O cho ô đĩa có thể nhận các lệnh

sau: READ SECTOR (đọc một sector), WRITE SECTOR (ghi một sector), SEEK track number (tìm kiếm một track), và SCAN record ID (thăm dò một bản ghi). Hai lệnh sau được gửi kèm với một tham số qua bus dữ liệu.

- **Dữ liệu:** Dữ liệu được trao đổi giữa bộ xử lý và module I/O qua bus dữ liệu.
- **Báo cáo trạng thái:** Vì các thiết bị ngoại vi thường rất chậm nên bộ xử lý thường phải biết tình trạng của module I/O để yêu cầu thực hiện các hoạt động tiếp theo. Ví dụ, nếu một module I/O được yêu cầu gửi dữ liệu đến bộ xử lý (READ), có thể module này chưa sẵn sàng để thực hiện yêu cầu vì nó vẫn đang thực hiện yêu cầu I/O trước. Điều này cần được báo cáo với bộ xử lý thông qua một tín hiệu trạng thái. Tín hiệu trạng thái thường sử dụng BUSY và READY. Ngoài ra, cũng có thể có một số tín hiệu báo cáo khác với các điều kiện lỗi khác nhau.
- **Nhận dạng địa chỉ:** Giống như mỗi từ nhớ có một địa chỉ, các thiết bị I/O cũng vậy. Một module I/O phải nhận ra một địa chỉ duy nhất cho mỗi thiết bị ngoại vi mà nó điều khiển.

Mặt khác, module I/O phải thực hiện việc **giao tiếp với các thiết bị**. Giao tiếp này gồm có các lệnh, thông tin trạng thái và dữ liệu (Hình 7.2).

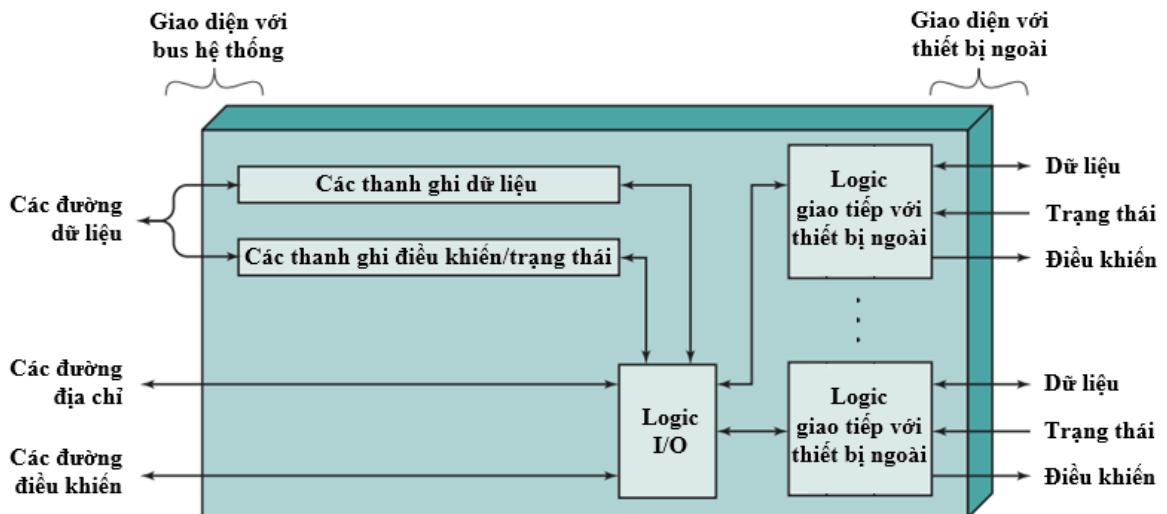
Một nhiệm vụ yêu nữa của module I/O là **đếm dữ liệu**. Chức năng này hết sức cần thiết do tốc độ trao đổi dữ liệu của bộ nhớ chính và bộ xử lý khá cao so với hầu hết các thiết bị ngoại vi. Dữ liệu từ bộ nhớ chính được gửi đến một module I/O với tốc độ nhanh và cần được đếm trong module I/O sau đó gửi đến thiết bị ngoại vi với tốc độ của thiết bị. Ngược lại, dữ liệu được đếm lại để không làm mất thời gian của bộ nhớ nhận truyền chậm. Do đó, module I/O phải hoạt động ở cả tốc độ của bộ nhớ và thiết bị ngoại vi. Tương tự trong trường hợp nếu thiết bị I/O có tốc độ cao hơn tốc độ truy cập bộ nhớ, module I/O sẽ phải thực hiện hoạt động đếm dữ liệu cần thiết.

Cuối cùng, một module I/O thường chịu trách nhiệm **phát hiện lỗi** và sau đó báo lỗi cho bộ vi xử lý. Một số loại bao gồm: hỏng về điện hoặc cơ học (ví dụ: kẹt giấy, một track trong ổ cứng). Một loại lỗi khác có thể do mẫu dữ liệu truyền từ thiết bị đến module I/O vô tình bị thay đổi. Người ta sử dụng một loại mã phát hiện lỗi để phát hiện lỗi truyền tải. Ví dụ như việc truyền mã ký tự IRA, mã này chiếm 7 bit của một byte, vậy bit thứ tám được thiết lập để tổng các bit 1 trong byte là chẵn (parity chẵn) hoặc lẻ (parity lẻ). Khi module I/O nhận được byte, nó sẽ kiểm tra bit chẵn lẻ này để xác định xem có lỗi xảy ra hay không.

7.2.2. Cấu trúc Module I/O

Các module I/O khác nhau đáng kể về độ phức tạp và số lượng thiết bị bên ngoài mà chúng điều khiển. Tại đây ta sẽ cố gắng chỉ mô tả một cấu trúc chung nhất của các module này. (Một thiết bị cụ thể là Intel 82C55A sẽ được mô tả chi tiết hơn trong Phần 7.4). Sơ đồ khái quát của module I/O được minh họa trong Hình 7.3. Module này kết nối với phần còn lại của máy tính thông qua một tập hợp các đường tín hiệu (ví dụ: bus hệ thống). Dữ liệu được chuyển đến hoặc từ module được lưu trữ trong các thanh ghi dữ liệu. Các thanh ghi trạng thái cung cấp thông tin về trạng thái hiện tại của thiết bị. Thanh ghi trạng thái cũng có chức năng như một thanh ghi điều khiển: nhận các thông tin điều khiển từ bộ vi xử lý. Logic I/O tương tác với bộ vi xử lý thông qua một tập các đường điều khiển. Bộ vi xử lý sử dụng các đường điều khiển để chuyển các lệnh cho module I/O. Một số đường điều

khiến có thể được sử dụng bởi module I/O (ví dụ các tín hiệu phân xử và trạng thái bus). Module này cũng phải có khả năng nhận dạng và sinh ra các địa chỉ liên kết với các thiết bị mà nó điều khiển. Mỗi module I/O có một (nếu chỉ nối với một TBNV) hoặc một tập địa chỉ (nếu module nối với nhiều TBNV). Cuối cùng, module I/O chứa các logic giao tiếp với từng thiết bị nối vào nó.



Hình 7.3 Sơ đồ khái niệm về Module I/O

Với hoạt động của module I/O, bộ xử lý có thể nhìn nhận một loạt các thiết bị theo một cách đơn giản. Module I/O có thể ẩn các thông tin về định thời, định dạng, các vấn đề về điện-cơ của thiết bị ngoại vi, bộ xử lý chỉ cần điều khiển hoạt động thông qua các lệnh đọc và ghi đơn giản hoặc có thể là các lệnh đóng, mở tập tin. Tuy nhiên, cũng có những module I/O hoạt động khá đơn giản và vì vậy để lại phần lớn công việc điều khiển thiết bị cho bộ xử lý.

Những module I/O có khả năng xử lý cao, hỗ trợ nhiều cho bộ xử lý thường được gọi là *kênh I/O (I/O channel)* hoặc *bộ xử lý I/O (I/O processor)*. Những module I/O đơn giản hơn và cần có sự điều khiển chi tiết hơn từ bộ xử lý thường được gọi là *bộ điều khiển I/O (I/O controller)* hoặc *bộ điều khiển thiết bị (device controller)*. Bộ điều khiển I/O thường được sử dụng trong các máy vi tính, còn kênh I/O thường được sử dụng trên các dòng máy tính lớn (mainframe).

Phần dưới đây, chúng tôi sẽ sử dụng khái niệm module I/O chung cho cả hai loại trên, nếu có điểm khác biệt thì chúng tôi sẽ giải thích cụ thể trong từng trường hợp.

7.3. CÁC KỸ THUẬT VÀO/RA

Có ba kỹ thuật để thực hiện các hoạt động vào/ra (I/O).

- I/O chương trình:** Dữ liệu được trao đổi giữa bộ vi xử lý và module I/O. Bộ xử lý thực thi một chương trình cho phép nó trực tiếp điều khiển hoạt động vào/ra, bao gồm cảm nhận tình trạng thiết bị, gửi lệnh đọc hoặc ghi, truyền dữ liệu. Khi bộ vi xử lý ra lệnh cho module I/O, nó phải đợi cho đến khi hoạt động I/O hoàn thành. Nếu bộ xử lý nhanh hơn module I/O thì việc chờ đợi này gây lãng phí thời gian của bộ xử lý.

- **I/O điều khiển ngắn:** bộ xử lý đưa ra một *lệnh I/O* (I/O command) sau đó tiếp tục thực hiện các lệnh (instruction) khác trong chương trình, khi nào module I/O hoàn thành công việc của mình nó sẽ gửi yêu cầu ngắn tới bộ xử lý để bộ xử lý tiếp tục điều khiển hoạt động vào/ra.
- Với cả I/O chương trình và I/O điều khiển ngắn, bộ vi xử lý có trách nhiệm lấy dữ liệu từ bộ nhớ chính rồi chuyển cho thiết bị ngoại vi (hoạt động Ghi) hoặc lưu dữ liệu từ thiết bị ngoại vi vào bộ nhớ chính (hoạt động Đọc). Vì vậy, một cơ chế thay thế được đưa ra là **cơ chế DMA (direct memory access – truy cập bộ nhớ trực tiếp)** để giảm thiểu sự tham gia của bộ xử lý vào quá trình trao đổi dữ liệu giữa bộ nhớ và thiết bị ngoại vi, tăng hiệu năng của hệ thống. Bảng 7.1 chỉ ra mối quan hệ giữa ba kỹ thuật này.

Bảng 7.1 Các kỹ thuật vào/ra

	Không có ngắn	Sử dụng ngắn
Truyền dữ liệu giữa bộ nhớ và thiết bị ngoại vi thông qua bộ xử lý	I/O chương trình	I/O điều khiển ngắn
Truyền dữ liệu trực tiếp giữa bộ nhớ và thiết bị ngoại vi		Truy cập bộ nhớ trực tiếp (DMA)

7.3.1. I/O chương trình

7.3.1.1. Tổng quan

Khi bộ vi xử lý đang thực hiện một chương trình và gặp một chỉ thị (instruction) liên quan đến vào/ra dữ liệu, nó thực hiện chỉ thị đó bằng cách phát ra *lệnh (command)* cho module I/O thích hợp. Với I/O chương trình, module I/O thực hiện hành động yêu cầu và sau đó thiết lập các bit thích hợp trong thanh ghi trạng thái I/O (Hình 7.3). Module I/O không có tác vụ nào khác để báo cáo về bộ vi xử lý. Do đó, để biết được công việc đã hoàn thành hay chưa, bộ xử lý phải định kỳ để kiểm tra trạng thái của module I/O cho đến khi hoạt động đã hoàn thành.

Để giải thích cho kỹ thuật I/O chương trình, chúng ta xem xét nó từ cách nhìn của các *lệnh I/O* do bộ vi xử lý gửi đến module I/O và sau đó từ cách nhìn của các chỉ thị được thực hiện bởi bộ vi xử lý.

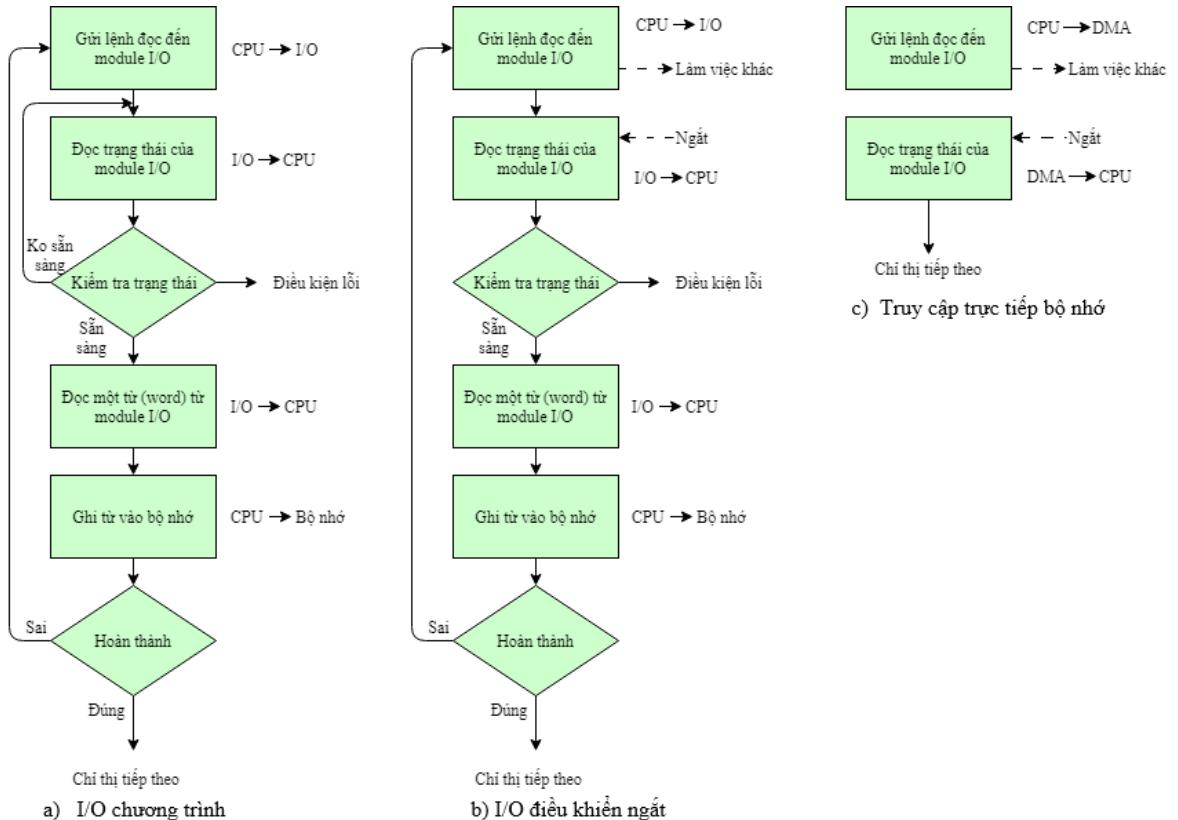
7.3.1.2. Các lệnh I/O

Để thực hiện chỉ thị liên quan đến vào/ra, bộ xử lý sẽ đưa ra một địa chỉ xác định module I/O và thiết bị bên ngoài cụ thể và và đưa ra một lệnh I/O. Có bốn loại lệnh I/O như sau:

- **Điều khiển:** Được sử dụng để kích hoạt một thiết bị ngoại vi và ra lệnh cho nó phải làm gì. Ví dụ, một ổ đĩa được điều khiển để quay lại hoặc tới trước một bản ghi. Các lệnh điều khiển được thiết kế riêng cho từng loại thiết bị ngoại vi.
- **Kiểm tra:** Được sử dụng để kiểm tra các điều kiện trạng thái của module I/O và thiết bị ngoại vi. Bộ xử lý cần biết liệu các thiết bị ngoại vi có đang bật nguồn

và sẵn sàng sử dụng. Nó cũng cần phải biết hoạt động I/O gần nhất đã hoàn thành chưa và có lỗi nào xảy ra không.

- **Đọc:** Yêu cầu module I/O lấy dữ liệu từ thiết bị ngoại vi và đặt nó vào bộ đệm (thanh ghi dữ liệu trong Hình 7.3). Bộ xử lý sau đó có thể lấy dữ liệu bằng cách yêu cầu module I/O đặt dữ liệu lên bus dữ liệu.
- **Ghi:** Yêu cầu module I/O chuyển dữ liệu (1 byte hoặc 1 từ) từ bus dữ liệu đến thiết bị ngoại vi.



Hình 7.4 Ba kỹ thuật Đọc một khối dữ liệu vào

Hình 7.4a minh họa một ví dụ về việc sử dụng I/O chương trình để đọc một khối dữ liệu từ thiết bị ngoại vi (ví dụ: một bản ghi từ băng từ) vào bộ nhớ. Dữ liệu được đọc mỗi lần một từ (16 bit). Đối với mỗi từ được đọc, bộ xử lý phải kiểm tra trạng thái cho đến khi nó xác định được từ đó đã được đọc vào thanh ghi dữ liệu của module I/O. Từ sơ đồ này ta thấy rõ nhược điểm chính của kỹ thuật này là một quá trình lặp đi lặp lại việc kiểm tra trạng thái làm cho bộ xử lý thêm bận rộn.

7.3.1.3. Chỉ thị vào/ra

Với I/O chương trình, có sự tương ứng chặt chẽ giữa các chỉ thị (lệnh) liên quan đến vào/ra mà bộ vi xử lý truy xuất từ bộ nhớ và các lệnh I/O ((command) mà bộ xử lý gửi đến module I/O để thực hiện chỉ thị đó. Do đó, các chỉ thị cần được ánh xạ thành các lệnh I/O và chúng thường có quan hệ một-một đơn giản. Dạng của chỉ thị phụ thuộc vào cách các thiết bị ngoại vi được định địa chỉ.

Thông thường, có nhiều thiết bị I/O được kết nối với hệ thống thông qua các module I/O. Mỗi thiết bị có một số nhận dạng hoặc địa chỉ duy nhất. Khi bộ xử lý đưa lệnh I/O đến

thiết bị thì trong lệnh phải chứa thông tin địa chỉ của thiết bị. Sau đó, các module I/O phải dịch các dòng địa chỉ để xác định xem liệu lệnh này có phải gửi cho nó.

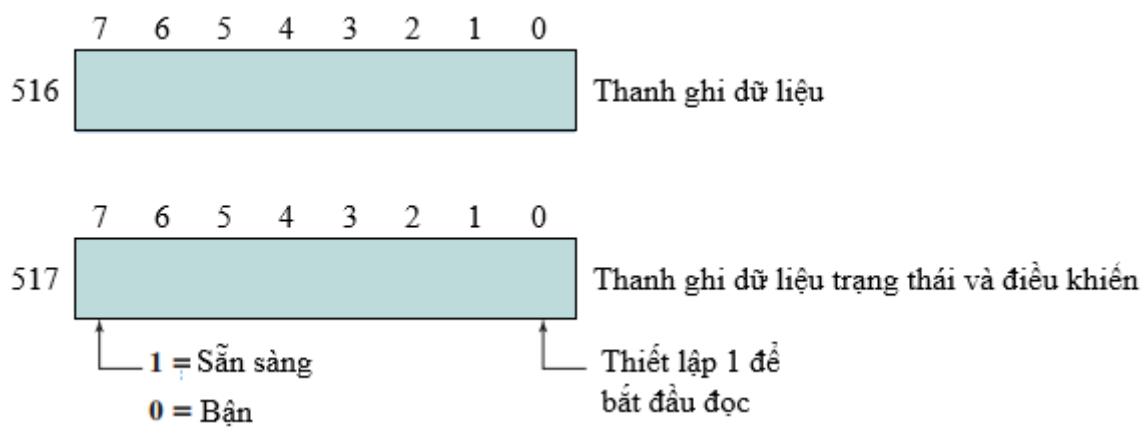
Khi bộ xử lý, bộ nhớ và module I/O chia sẻ một bus chung, hai chế độ định địa chỉ có thể được thực hiện như sau: **I/O ánh xạ bộ nhớ (memory-mapped I/O)** và **I/O riêng biệt (isolated I/O)**. I/O ánh xạ bộ nhớ sử dụng một không gian địa chỉ chung cho bộ nhớ và các thiết bị ngoại vi. Bộ xử lý coi thanh ghi trạng thái và dữ liệu của các module I/O giống như vị trí bộ nhớ và sử dụng cùng các lệnh máy để truy cập cả bộ nhớ và các thiết bị ngoại vi. Ví dụ, với 10 đường địa chỉ ta có một không gian địa chỉ gồm $2^{10} = 1024$ địa chỉ vị trí bộ nhớ.

Với kỹ thuật I/O riêng biệt, bus được trang bị thêm một đường command line, đường này sẽ cho biết địa chỉ trên bus là địa chỉ của vị trí bộ nhớ hay thiết bị I/O. Như vậy, ta có thể sử dụng toàn bộ không gian địa chỉ cho cả bộ nhớ và thiết bị ngoại vi. Như vậy, với 10 đường địa chỉ, hệ thống có thể hỗ trợ 1024 vị trí bộ nhớ và 1024 địa chỉ I/O.

Hình 7.5 so sánh hai kỹ thuật I/O chương trình. Với Hình 7.5a, bàn phím được kết nối với hệ thống theo kỹ thuật I/O ánh xạ bộ nhớ. Giả sử hệ thống có 10-bit địa chỉ, trong đó 512 giá trị dành cho bộ nhớ (từ 0-511) và 512 dành cho I/O (từ 512-1023). Hai địa chỉ được dành riêng cho việc giao tiếp với bàn phím. Địa chỉ 516 được gán cho thanh ghi dữ liệu và 517 được gán cho thanh ghi trạng thái. Thanh ghi trạng thái hoạt động như một thanh ghi điều khiển để nhận các lệnh của bộ vi xử lý. Chương trình trong hình có mục đích đọc 1 byte dữ liệu từ bàn phím vào bộ thanh ghi AC trong bộ xử lý.

Với I/O riêng biệt (Hình 7.5b), các cổng vào/ra chỉ có thể truy cập bằng các lệnh I/O đặc biệt. Các lệnh này sẽ kích hoạt các đường command line trên bus.

Hầu hết các bộ xử lý đều có một tập khá lớn các chỉ thị khác nhau để tham chiếu bộ nhớ. Nếu sử dụng I/O riêng biệt, số lượng chỉ thị vào/ra sẽ khá ít. Do đó, ưu điểm của I/O ánh xạ bộ nhớ là số lượng các chỉ thị tham chiếu bộ nhớ cũng có thể được sử dụng trong tham chiếu thiết bị ngoại vi, vì vậy cho phép việc lập trình hiệu quả hơn. Tuy nhiên, nhược điểm của nó là không gian địa chỉ phải chia sẻ với thiết bị ngoại vi. Cả hai kỹ thuật I/O này đều được sử dụng phổ biến.



ĐỊA CHỈ	CHỈ THỊ	TOÁN HẠNG	CHÚ THÍCH
200	Load AC Store AC	"1" 517	Đọc "1" vào AC Ghi AC vào 517

202	Load AC	517	Khởi tạo việc đọc bàn phím
	Branch if Sign = 0	202	Đọc 517 vào AC (đọc trạng thái tb)
	Load AC	516	Lặp lại 202 đến khi sẵn sàng

a) I/O ánh xạ bộ nhớ

ĐỊA CHỈ	CHỈ THỊ	TOÁN HẠNG	CHÚ THÍCH
200	Load I/O	5	Khởi tạo việc đọc bàn phím
201	Test I/O	5	Kiểm tra sự sẵn sàng
	Branch If Not Ready	201	Lặp lại 201 đến khi tb sẵn sàng
	In	5	Đọc dữ liệu

b) I/O riêng biệt

Hình 7.5 So sánh I/O ánh xạ bộ nhớ và I/O riêng biệt

7.3.2. I/O điều khiển ngắt

Một vấn đề của I/O chương trình là bộ xử lý phải đợi một thời gian dài cho đến khi module I/O sẵn sàng tiếp nhận hoặc truyền dữ liệu. Trong khoảng thời gian đó, bộ xử lý phải liên tục kiểm tra trạng thái của module I/O. Kết quả là hiệu năng của toàn bộ hệ thống suy giảm nghiêm trọng.

Để giải quyết vấn đề trên, người ta đưa ra một giải pháp như sau: bộ vi xử lý gửi lệnh I/O đến module và sau đó thực hiện các hoạt động khác. Khi nào module I/O đã sẵn sàng để trao đổi dữ liệu nó sẽ ngắt bộ xử lý để yêu cầu phục vụ. Bộ xử lý thực thi việc truyền dữ liệu rồi quay trở lại công việc của nó.

Chúng ta sẽ xem xét cách hoạt động này từ hai phía: bộ xử lý và module I/O.

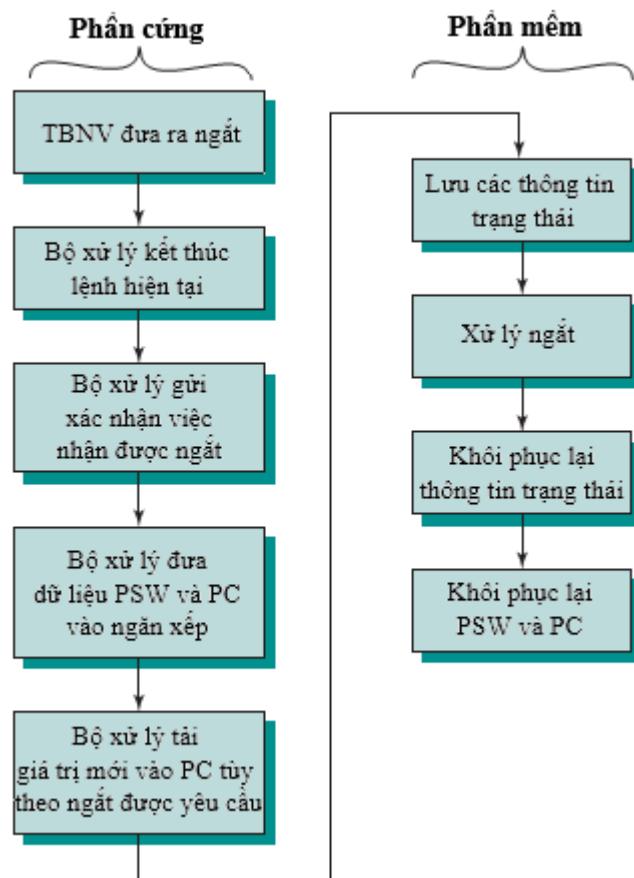
Từ phía module I/O, việc đọc dữ liệu vào như sau: module I/O nhận lệnh READ từ bộ xử lý. Sau đó, nó tiến hành đọc dữ liệu từ một thiết bị ngoại vi được yêu cầu. Khi dữ liệu đã được đọc vào thanh ghi dữ liệu của module, module gửi một báo hiệu ngắt tới bộ xử lý qua một đường điều khiển. Module chờ đợi cho đến khi bộ xử lý gửi yêu cầu dữ liệu. Khi có yêu cầu, module đặt dữ liệu vào bus dữ liệu và được giải phóng, sẵn sàng cho một hoạt động vào/ra khác.

Từ phía bộ xử lý, hoạt động Đọc dữ liệu vào như sau: bộ xử lý ra lệnh READ, sau đó nó sẽ thực hiện một công việc khác (ví dụ: bộ xử lý có thể đang chạy trên nhiều chương trình cùng một lúc). Vào cuối mỗi chu kỳ lệnh, bộ xử lý sẽ kiểm tra các ngắt (Hình 3.9). Khi có ngắt gửi từ module I/O, bộ xử lý lưu lại ngữ cảnh (ví dụ: thanh ghi PC và các thanh ghi khác của bộ xử lý) của chương trình hiện tại và thực hiện việc xử lý ngắt. Khi xử lý ngắt, bộ xử lý đọc dữ liệu từ module I/O và lưu trữ nó vào bộ nhớ. Sau đó nó khôi phục lại ngữ cảnh trước và tiếp tục thực thi công việc.

Hình 7.4b minh họa việc sử dụng I/O điều khiển ngắt để đọc trong một khối dữ liệu. So với hình 7.4a, rõ ràng việc sử dụng ngắt hiệu quả hơn vì nó giúp loại bỏ khoảng thời gian chờ không cần thiết của bộ xử lý. Tuy nhiên, việc truyền dữ liệu giữa bộ nhớ và module I/O vẫn phải có sự tham gia của bộ xử lý.

7.3.2.1. Xử lý ngắt

Ta hãy xem xét chi tiết hơn vai trò của bộ xử lý trong I/O điều khiển ngắt. Sự xuất hiện của một ngắt gây ra một số sự kiện của cả phần cứng và phần mềm (Hình 7.6). Khi một thiết bị vào/ra thực hiện một hoạt động vào/ra, một chuỗi các hoạt động phần cứng sau đây sẽ xảy ra



Hình 7.6 Quá trình xử lý ngắt đơn giản

- Thiết bị phát tín hiệu ngắt cho bộ xử lý.
- Bộ xử lý hoàn thành lệnh hiện tại trước khi trả lời ngắt (Hình 3.9).
- Bộ xử lý kiểm tra xem có ngắt hay không, nếu có một ngắt, và gửi một tín hiệu báo đã nhận (tín hiệu ACK) đến thiết bị đã gửi ngắt. Khi nhận được ACK, thiết bị loại bỏ tín hiệu ngắt.
- Bộ xử lý cần phải chuyển đổi sang chế độ ngắt. Đầu tiên, nó cần phải lưu lại các thông tin của chương trình hiện tại để có thể khôi phục lại công việc sau khi hoàn thành xong việc xử lý ngắt. Các thông tin tối thiểu bắt buộc là (a) trạng thái của bộ xử lý được lưu trữ trong một thanh ghi PSW (thanh ghi trạng thái chương trình), và (b) địa chỉ lệnh tiếp theo sẽ được thực hiện (nội dung thanh ghi PC). Chúng được đẩy lên vùng bộ nhớ ngăn xếp của hệ thống.
- Sau đó, bộ xử lý sẽ tải vị trí đầu tiên của chương trình xử lý ngắt vào thanh ghi PC. Tùy thuộc vào kiến trúc hệ thống và thiết kế hệ điều hành, trình xử lý ngắt có thể là một chương trình; một chương trình cho mỗi loại ngắt hoặc một chương trình cho mỗi thiết bị và mỗi loại ngắt. Nếu có nhiều hơn một trình xử lý ngắt, bộ xử lý phải

xác định xem ngắt được gửi đến từ đâu. Thông tin đó có thể đã được chứa trong tín hiệu ngắt gửi đến hoặc bộ xử lý phải gửi đến thiết bị gửi ngắt để yêu cầu phản hồi các thông tin cần thiết. Sau khi thanh ghi PC đã được nạp, bộ xử lý bắt đầu thực thi các lệnh trong chương trình xử lý ngắt. Việc thực hiện chương trình này dẫn đến các hoạt động sau:

6. Tại thời điểm này, nội dung các thanh ghi PC và PSW của chương trình bị ngắt đã được lưu trên vùng nhớ ngắn xếp của hệ thống. Tuy nhiên, vẫn còn một số thông tin khác liên quan đến trạng thái chương trình cũng cần phải được lưu lại, đặc biệt là nội dung các thanh ghi trong bộ xử lý vì các thanh ghi này có thể được sử dụng bởi trình xử lý ngắt. Do đó, thông thường, trình xử lý ngắt sẽ bắt đầu bằng việc lưu nội dung của tất cả các thanh ghi vào ngắn xếp. Một ví dụ đơn giản được minh họa trong Hình 7.7a . Trong trường hợp này, một chương trình bị ngắt tại vị trí N . Nội dung của tất cả các thanh ghi cộng với địa chỉ của lệnh tiếp theo ($N + 1$) được đẩy lên ngắn xếp. Thanh ghi SP (con trỏ ngắn xếp) (đọc thêm chương 12) trỏ đến đỉnh mới của ngắn xếp, và thanh ghi PC trỏ đến lệnh đầu tiên của trình phục vụ ngắt.
7. Lúc này, ngắt được xử lý bao gồm việc kiểm tra thông tin trạng thái liên quan đến hoạt động I/O hoặc các sự kiện khác gây ra ngắt. Nó cũng có thể thực hiện việc gửi thêm lệnh tiếp theo của một hoạt động I/O đang được thực hiện hoặc gửi tín hiệu ACK đến thiết bị ngoại vi.
8. Khi quá trình xử lý ngắt hoàn tất, giá trị các thanh ghi đã lưu trên ngắn xếp sẽ được lấy ra và khôi phục lại vào các thanh ghi (ví dụ Hình 7.7b).
9. Hoạt động cuối cùng là khôi phục giá trị PSW và PC từ ngắn xếp. Nhờ đó, chương trình bị ngắt sẽ được khôi phục lại tiếp tục được thực thi.

7.3.2.2. Vấn đề thiết kế

Thực hiện ngắt I/O phát sinh hai vấn đề như sau. Thứ nhất, bởi vì trong hệ thống thường có nhiều module I/O, vậy bộ xử lý xác định thiết bị gửi tín hiệu ngắt như thế nào? Thứ hai, nếu nhiều yêu cầu ngắt được cùng gửi đến, bộ xử lý sẽ quyết định phục vụ ngắt nào trước ngắt nào sau?

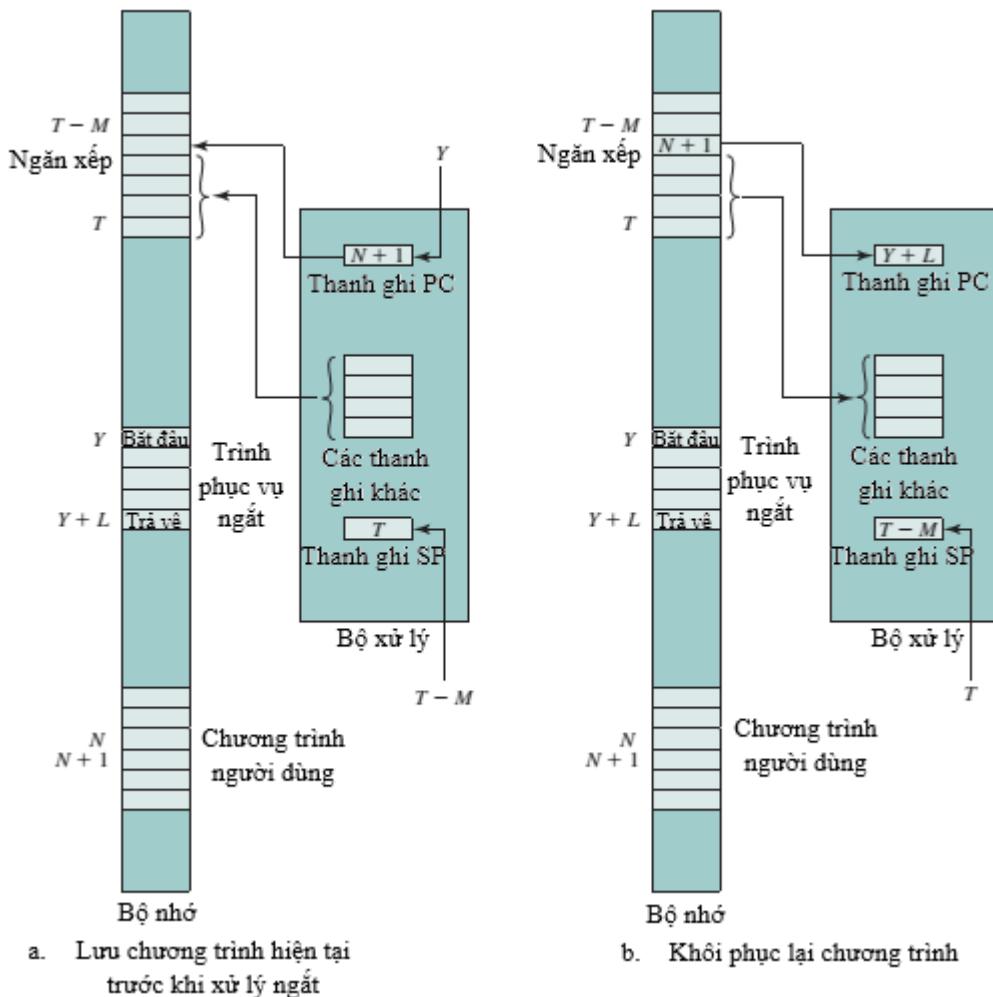
Với vấn đề đầu tiên, có bốn kỹ thuật thường được sử dụng để xác định thiết bị gửi ngắt như sau:

- Sử dụng nhiều đường ngắt
- Thăm dò phần mềm
- Chuỗi Daisy (thăm dò phần cứng, vector)
- Phân xử bus (vector)

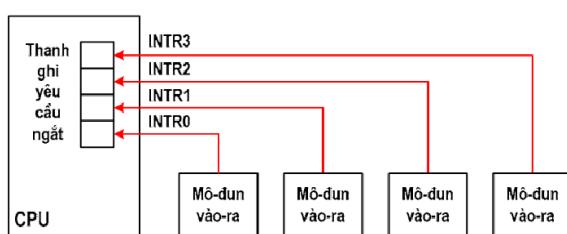
Phương pháp đơn giản nhất để giải quyết vấn đề này là ta có thể sử dụng **nhiều đường truyền yêu cầu ngắt** giữa bộ xử lý và các module I/O. Tuy nhiên, việc sử dụng các đường bus và các chân của bộ xử lý để làm các đường ngắt là không thực tế. Do đó, ngay cả khi sử dụng kỹ thuật này, có thể mỗi đường ngắt vẫn sẽ nối với nhiều module I/O và vẫn phải sử dụng một trong ba kỹ thuật còn lại cho mỗi đường.

Với kỹ thuật **thăm dò phần mềm**: khi bộ xử lý phát hiện ra một ngắt, nó rẽ nhánh sang một *trình phục vụ ngắt chung* có nhiệm vụ thăm dò từng module I/O để xác định module nào phát ra ngắt. Tín hiệu thăm dò có thể dưới dạng một đường lệnh riêng (ví dụ:

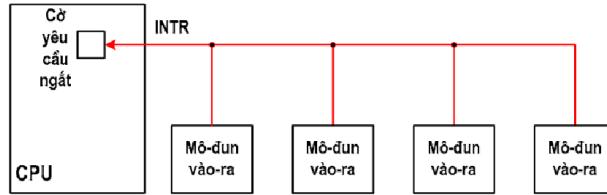
TESTI/O). Bộ xử lý phát ra tín hiệu TEST I/O và đặt địa chỉ của từng module vào các đường địa chỉ. Module I/O gửi yêu cầu ngắn sẽ trả lời khi nhận được thăm dò. Một cách thăm dò khác là bộ xử lý sẽ đọc các thanh ghi trạng thái của từng module để xác định module nào gửi ngắn. Sau khi xác định được module ngắn, bộ xử lý sẽ chuyển điều khiển sang trình phục vụ ngắn dành riêng cho thiết bị đó. Tuy nhiên, nhược điểm của phương pháp này là tốn thời gian.



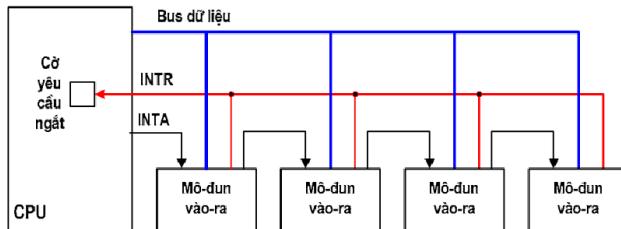
Hình 7.7 Sự thay đổi Bộ nhớ và Thanh ghi khi có Ngắt



Hình 7.8 Kỹ thuật nhiều đường ngắn



Hình 7.9 Kỹ thuật thăm dò phần mềm



Hình 7.10 Kỹ thuật chuỗi Daisy

Kỹ thuật chuỗi Daisy thực hiện thăm dò bằng phần cứng. Kỹ thuật này sử dụng một đường ngắt chung (INTR) cho tất cả các module. Khi nhận được yêu cầu ngắt, bộ xử lý gửi một tín hiệu ACK thửa nhận ngắt, tín hiệu này được truyền trên một đường INTA đi qua tất cả các module (nối chuỗi) (Hình 7.9) cho đến khi gặp được module gửi ngắt, module này sẽ gửi lại một tín hiệu trả lời bằng cách đặt một từ lên bus dữ liệu, từ này là có thể là địa chỉ hoặc thông tin nhận diện của module đó, được gọi là *vector*. Khi đó, bộ xử lý sẽ sử dụng vector để trỏ tới trình phục vụ ngắt của module đó. Với kỹ thuật này, bộ xử lý không cần phải rẽ nhánh đến trình phục vụ ngắt chung như trường hợp trên.

Một kỹ thuật khác cũng sử dụng vector ngắt là **kỹ thuật phân xử bus**. Một module I/O muốn gửi yêu cầu ngắt thì cần phải có quyền chiếm bus trước. Vì vậy, tại một thời điểm chỉ có một module có thể sử dụng bus. Khi bộ xử lý phát hiện ra ngắt, nó trả lời bằng đường ACK. Khi nhận được ACK, module yêu cầu đặt các vector của nó vào các đường dữ liệu và bộ xử lý cũng thực hiện việc xử lý ngắt tương tự như với kỹ thuật chuỗi Daisy.

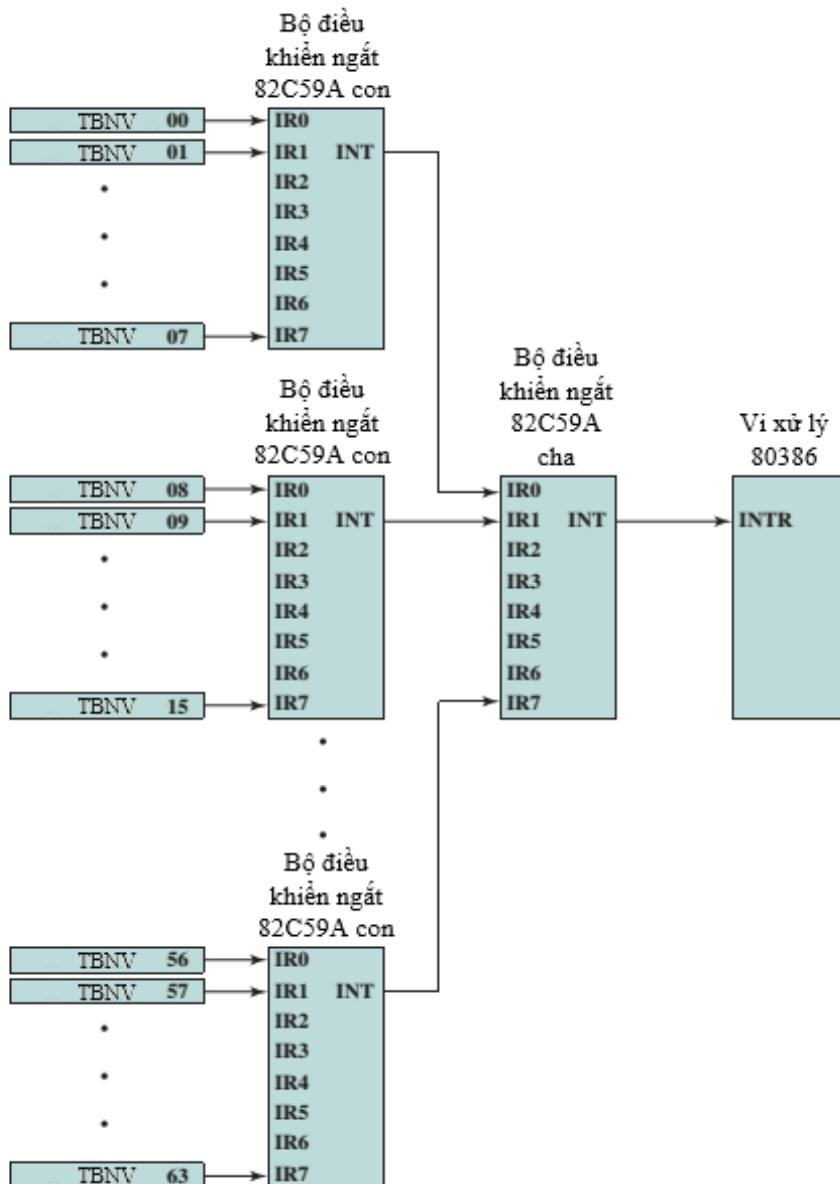
Với các kỹ thuật trên, ta cũng có thể giải quyết luôn vấn đề xác định thứ tự ưu tiên trong trường hợp có nhiều ngắt gửi đến bộ xử lý cùng một thời điểm. Với kỹ thuật nhiều đường ngắt, mỗi đường ngắt sẽ được gán một độ ưu tiên nhất định, khi đó, bộ xử lý chỉ cần chọn đường ngắt có độ ưu tiên cao nhất. Với thăm dò phần mềm, bộ xử lý sẽ thực hiện việc thăm dò các module lần lượt theo thứ tự ưu tiên nó. Tương tự, thứ tự các module được nối chuỗi daisy sẽ được xác định theo độ ưu tiên của chúng. Cuối cùng, trong kỹ thuật phân xử bus, module có độ ưu tiên cao hơn sẽ được cho phép chiếm bus trước, vì vậy cũng đã giải quyết được vấn đề này (xem thêm Phần 3.4).

Chúng tôi giới thiệu hai cấu trúc ngắt trong thực tế là: Bộ điều khiển ngắt Intel 82C59A và Giao diện thiết bị ngoại vi lập trình được Intel 82C55A

7.3.2.3. *Bộ điều khiển ngắt Intel 82C59A*

Chip Intel 80386 có một đường INTR (Yêu cầu Ngắt) và một đường INTA (Chấp nhận Ngắt). Để 80386 có thể giao tiếp với nhiều loại thiết bị ngoại vi và phân xử ưu tiên, nó thường được cấu hình với *bộ điều khiển ngắt 82C59A*. Nghĩa là, 80386 giao tiếp với các thiết bị ngoại vi thông qua 82C59A (Hình 7.11)

Một 82C59A có thể xử lý tối đa tám module. Nếu số lượng module cần điều khiển lớn hơn tám, người ta có thể thực hiện việc ghép nối tầng và có thể xử lý tới 64 module (như trong Hình 7.11). Nhiệm vụ duy nhất của 82C59A là quản lý các yêu cầu ngắt. Nó nhận các yêu cầu ngắt từ các module I/O, xác định ngắt nào có mức ưu tiên cao nhất, và sau đó báo hiệu bộ xử lý bằng cách thiết lập đường INTR. Bộ xử lý gửi tín hiệu thừa nhận ACK qua đường dây INTA. Khi nhận được ACK, 82C59A đặt thông tin vector thích hợp lên bus dữ liệu. Bộ xử lý tiến hành xử lý ngắt và liên lạc trực tiếp với module I/O để đọc hoặc ghi dữ liệu.



Hình 7.11 Bộ điều khiển Ngắt 82C59A

Chip 82C59A cho phép lập trình 80386 xác định sơ đồ ưu tiên rồi thiết lập nó vào từ điều khiển trong 82C59A. Sơ đồ ưu tiên này có thể thiết lập theo một trong ba chế độ ngắt như sau:

- **Lồng nhau hoàn toàn:** Các yêu cầu ngắt được sắp xếp theo thứ tự ưu tiên từ 0 (IR0) đến 7 (IR7).

- **Xoay vòng:** Một số thiết bị ngắt có mức ưu tiên ngang nhau. Sau khi một thiết bị được phục vụ ngắt nó sẽ nhận mức ưu tiên thấp nhất trong nhóm.
- **Mặt nạ đặc biệt:** Cho phép bộ xử lý chặn các ngắt từ một số thiết bị nhất định.

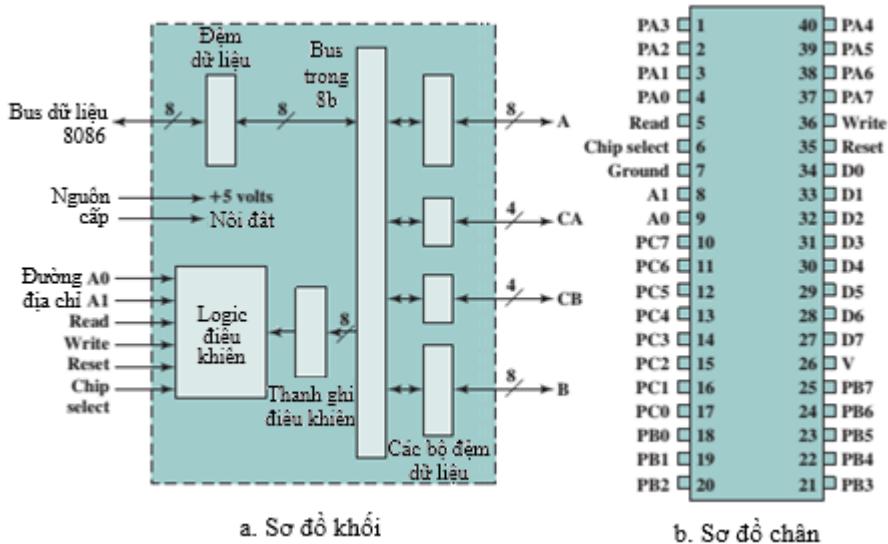
7.3.2.4. Giao diện ngoại vi có thể lập trình của Intel 82C55A

Chip Intel 82C55A sử dụng cho kỹ thuật I/O chương trình và I/O điều khiển ngắt. 82C55A là một module I/O đa nhiệm được thiết kế để sử dụng với vi xử lý Intel 80386. Hình 7.12 minh họa cấu trúc tổng quát và thông tin 40 chân của chip.

Phía bên phải của sơ đồ khố chip 82C55A gồm có: 24 đường vào/ra có thể lập trình được bằng cách 80386 thiết lập thanh ghi điều khiển để chỉ định đến một loạt các chế độ và cấu hình hoạt động. 24 đường được chia thành ba nhóm 8-bit (A, B, C). Mỗi nhóm có thể hoạt động như một cổng vào/ra 8-bit. Ngoài ra, nhóm C được chia thành hai nhóm 4-bit (C_A và C_B), hai nhóm này có thể được sử dụng kết hợp với các nhóm A và B. Với cách cấu hình này, các đường của nhóm C mang tín hiệu điều khiển và trạng thái.

Phía bên trái của sơ đồ khố là giao diện với bus 80386, gồm có: một bus dữ liệu hai chiều 8-bit (D0 đến D7), được sử dụng để truyền dữ liệu đến và đi tới các cổng vào/ra (A, B, C) hoặc để truyền thông tin điều khiển tới thanh ghi điều khiển; hai đường địa chỉ giúp xác định một trong ba cổng vào/ra hoặc thanh ghi điều khiển. Việc trao đổi dữ liệu sẽ xảy ra khi đường CHIP SELECT được thiết lập ở mức cao cùng với đường READ (Đọc) hoặc WRITE (Ghi). Đường RESET được sử dụng để khởi tạo lại module.

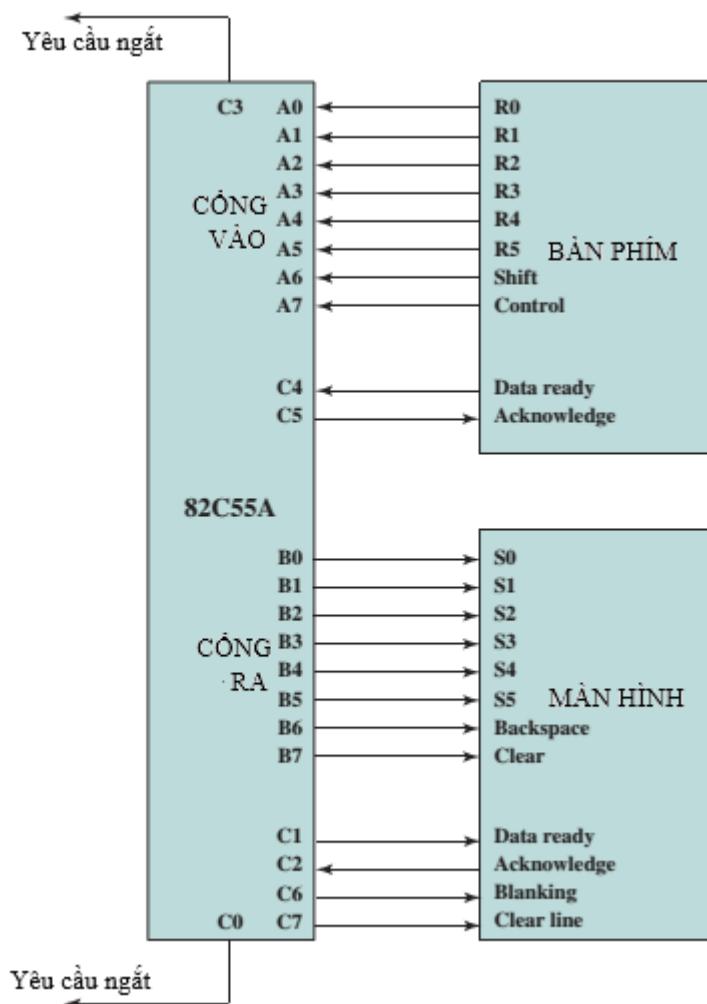
Như ta đã biết, bộ xử lý nạp các thông tin điều khiển vào thanh ghi điều khiển để định ra chế độ hoạt động hoặc định ra tín hiệu, nếu có. Trong chế độ 0, ba nhóm A, B, C hoạt động như ba cổng vào/ra 8-bit. Mỗi cổng có thể được chỉ định là đầu vào hoặc đầu ra. Ở một chế độ khác, các nhóm A và B được thiết lập là các cổng vào/ra còn các đường của nhóm C là các đường điều khiển cho A và B. Các tín hiệu điều khiển phục vụ cho hai mục đích chính: cơ chế "bắt tay" và yêu cầu ngắt. Cơ chế "bắt tay" là một cơ chế định thời đơn giản. Bên gửi sử dụng một đường điều khiển để gửi tín hiệu DATA READY báo cho bên nhận là dữ liệu đã có mặt trên đường dữ liệu. Bên nhận sử dụng một đường khác để gửi tín hiệu ACKNOWLEDGE (ACK) báo cho bên gửi biết rằng dữ liệu đã được đọc vào, bên gửi có thể loại bỏ các đường dữ liệu. Một đường được sử dụng làm đường Yêu cầu Ngắt.



a. Sơ đồ khối

b. Sơ đồ chân

Hình 7.12 Chip Intel 82C55A



Hình 7.13 Giao diện Bàn phím/Màn hình với 82C55A

Bởi vì 82C55A có thể lập trình được thông qua việc thiết lập thanh ghi điều khiển, nó còn được sử dụng để điều khiển một loạt các thiết bị ngoại vi đơn giản. Hình 7.10 minh

họa việc sử dụng 82C55A để điều khiển Bàn phím/Màn hình. Bàn phím cung cấp 8 bit đầu vào. Hai trong số các bit này là SHIFT và CONTROL, tương ứng với phím Shift và Ctrl trên bàn phím. Việc tiếp nhận dữ liệu từ bàn phím rồi chuyển đến bus hệ thống của 82C55A trong trường hợp này là hoàn toàn trong suốt, tức là dữ liệu được nhận vào rồi chuyển vào bus, 82C55A không phải thực hiện thao tác gì khác. Hai đường điều khiển được sử dụng để thực hiện cơ chế “bắt tay”.

Màn hình cũng được kết nối với một cổng dữ liệu 8-bit. Tương tự như với bàn phím, hai bit có ý nghĩa đặc biệt và trong suốt đối với 82C55A. Ngoài hai đường điều khiển thực hiện “bắt tay”, hai đường điều khiển được sử dụng để thực hiện các chức năng điều khiển bổ sung.

7.3.3. Cơ chế DMA – Truy cập bộ nhớ trực tiếp

7.3.3.1. Nhược điểm của các phương pháp I/O chương trình và I/O điều khiển ngắt

I/O điều khiển ngắt mặc dù đã hiệu quả hơn rất nhiều so với I/O chương trình, tuy nhiên nó vẫn cần có sự tham gia của bộ xử lý vào việc trao đổi dữ liệu giữa bộ nhớ và các module I/O, bất cứ dữ liệu nào cũng phải đi qua bộ xử lý. Do đó các phương pháp này có hai nhược điểm như sau:

1. Tốc độ trao đổi dữ liệu vào/ra bị hạn chế bởi tốc độ bộ xử lý thực hiện các thao tác kiểm tra và phục vụ thiết bị.
2. Bộ xử lý phải quản lý việc truyền I/O; với mỗi thao tác vào/ra dữ liệu, bộ xử lý phải thực thi một số chỉ thị (ví dụ, Hình 7.5).

Có một sự cân bằng giữa hai nhược điểm này. Khi xét đến việc trao đổi khối dữ liệu. Nếu sử dụng I/O chương trình, bộ xử lý được dành hoàn toàn cho việc vào/ra dữ liệu nên việc trao đổi này diễn ra với tốc độ khá cao, tuy nhiên bộ xử lý lại không làm được việc gì khác. Với I/O điều khiển ngắt, bộ xử lý được giải phóng nhưng thời gian thực hiện một hoạt động truyền dữ liệu lại nhiều hơn. Cả hai phương pháp đều có ảnh hưởng nhất định đến hoạt động của bộ xử lý và tốc độ truyền I/O.

Khi cần trao đổi một khối lượng lớn dữ liệu, người ta đưa ra một kỹ thuật hiệu quả hơn, đó là: truy cập bộ nhớ trực tiếp (DMA).

7.3.3.2. Chức năng DMA

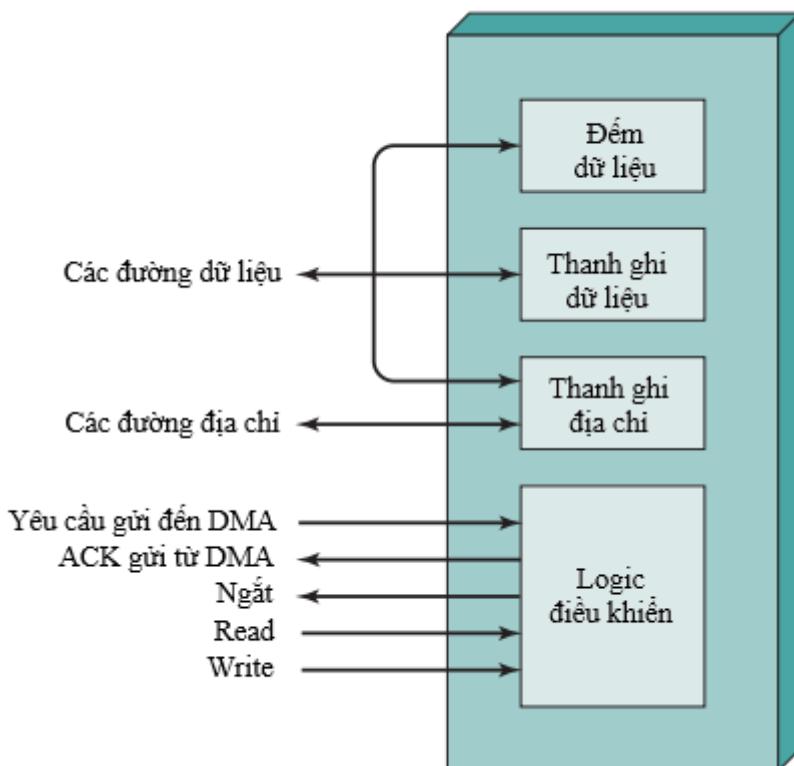
DMA thường là một module bổ sung trên hệ thống bus. Module DMA (Hình 7.14) có khả năng bắt chước bộ xử lý, tức là tiếp nhận việc điều khiển hệ thống từ bộ xử lý. Nó cần phải thực hiện điều này để trao đổi dữ liệu với bộ nhớ qua bus hệ thống. Với mục đích này, module DMA chỉ sử dụng bus khi bộ xử lý không cần đến nó hoặc DMA phải buộc bộ xử lý phải tạm ngừng hoạt động (gọi là *trộm chu kỳ*). Kỹ thuật trộm chu kỳ thường phổ biến hơn.

Khi bộ xử lý muốn đọc hoặc ghi một khối (block) dữ liệu, nó sẽ đưa ra một lệnh cho module DMA bằng cách gửi tới module DMA các thông tin sau:

- Yêu cầu Đọc hoặc Ghi bằng cách sử dụng các đường điều khiển READ hoặc WRITE bộ xử lý và module DMA

- Địa chỉ của thiết bị I/O liên quan, gửi qua các đường dữ liệu
- Địa chỉ vị trí đầu tiên trong bộ nhớ để đọc hoặc ghi lên, truyền qua các đường dữ liệu và được lưu trữ lại bởi module DMA trong Thanh ghi địa chỉ (xem hình)
- Số lượng từ cần đọc hoặc ghi, truyền qua các đường dữ liệu và được lưu trữ trong thanh ghi Đếm dữ liệu

Sau đó, bộ xử lý tiếp tục thực hiện các công việc khác. Hoạt động vào/ra được ủy quyền hoàn toàn cho module DMA. Module DMA chuyển toàn bộ khối dữ liệu giữa bộ nhớ và thiết bị ngoại vi mà không cần sự tham gia của bộ xử lý. Khi quá trình trao đổi hoàn tất, module DMA gửi một tín hiệu ngắn tới bộ xử lý. Do đó, bộ xử lý chỉ tham gia vào thời điểm bắt đầu và kết thúc quá trình trao đổi (Hình 7.4c).



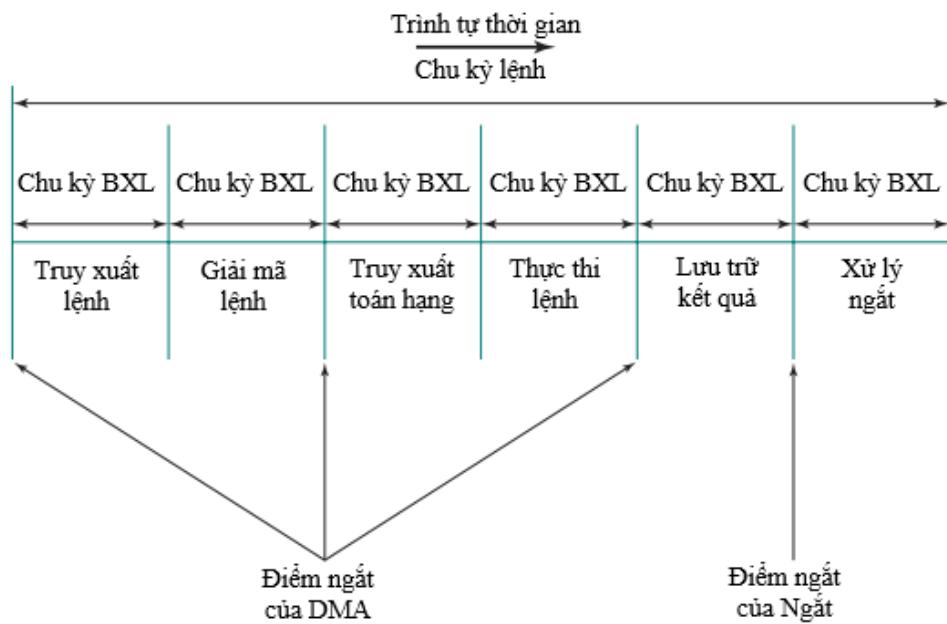
Hình 7.14 Sơ đồ khái niệm DMA

Như đã nói ở trên, khi DMA cần chiếm bus để truyền, nó sẽ trộm chu kỳ bus, Hình 7.15 chỉ ra các thời điểm trong chu kỳ lệnh mà bộ xử lý bị treo (trong lúc DMA sử dụng bus). Trong mỗi trường hợp, bộ xử lý bị treo ngay trước khi nó cần sử dụng bus. Module DMA chuyển một từ sau đó trả lại điều khiển cho bộ xử lý. Lưu ý rằng đây không phải là ngắn; bộ xử lý không cần lưu trữ ngữ cảnh hoặc bắt cứ thông tin gì. Đơn giản là bộ xử lý chỉ dừng lại một chu kỳ bus. Về mặt hiệu năng của bộ xử lý thì nó sẽ hoạt động chậm hơn. Tuy nhiên, với việc trao đổi vào/ra nhiều từ, cơ chế DMA hiệu quả hơn rất nhiều so với I/O điều khiển ngắn hoặc I/O chương trình.

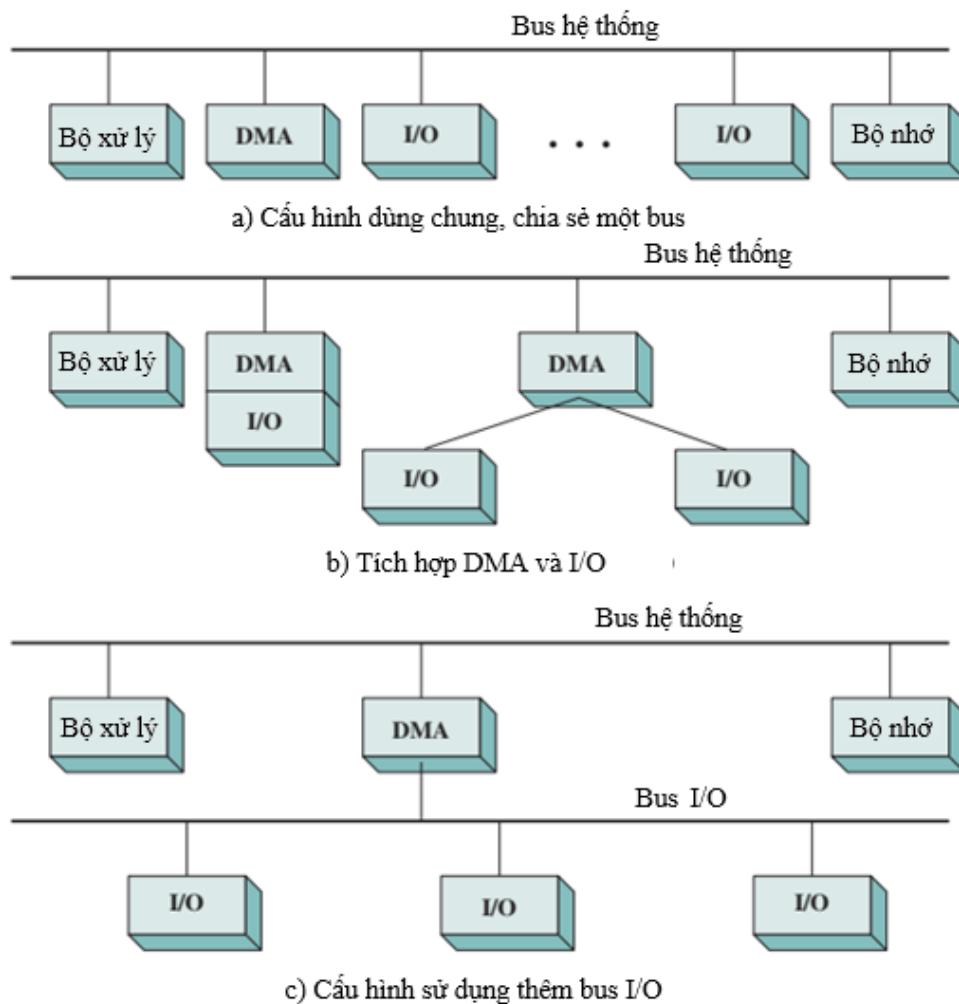
Cơ chế DMA có thể được cấu hình theo nhiều cách khác nhau. Hình 7.16 là sơ đồ một số cấu hình DMA. Hình 7.16a, tất cả các module chia sẻ chung hệ thống bus. Module DMA đại diện cho bộ xử lý sử dụng I/O chương trình để trao đổi dữ liệu giữa bộ nhớ và module I/O. Cấu hình này mặc dù không tốn kém nhưng rõ ràng là không hiệu quả. Giống

như I/O chương trình do bộ xử lý điều khiển, mỗi lần truyền một từ sẽ tiêu tốn hai chu kỳ bus.

Số lượng chu kỳ bus có thể được giảm bớt bằng cách tích hợp DMA và các module I/O. Như trong Hình 7.16b, module DMA kết nối trực tiếp với một hoặc nhiều module I/O mà không đi qua bus hệ thống. Logic DMA có thể là một phần của module I/O hoặc có thể là một module riêng biệt điều khiển một hoặc nhiều module I/O. Phương thức này có thể được cải tiến thêm một bước bằng cách kết nối các module I/O với module DMA qua một bus I/O (Hình 7.16c). Cấu hình này giảm số giao diện ngoại vi của module DMA thành một và cho phép khả năng mở rộng (tăng thêm số lượng module I/O thì chỉ cần kết nối với bus I/O). Trong cả hai trường hợp Hình 7.16b và c, module DMA chỉ chiếm bus hệ thống khi cần trao đổi dữ liệu với bộ nhớ, như vậy chỉ mất một chu kỳ bus cho mỗi từ được truyền.



Hình 7.15 Điểm ngắt của DMA và của Ngắt trong chu kỳ lệnh



Hình 7.16 Các cấu hình DMA

7.3.3.3. Bộ điều khiển Intel 8237A DMA

Bộ điều khiển Intel 8237A DMA giao tiếp với họ vi xử lý 80x86 và bộ nhớ DRAM, cung cấp cơ chế DMA (Hình 7.17). Khi module DMA cần sử dụng bus hệ thống (bus dữ liệu, địa chỉ và điều khiển) để truyền dữ liệu, nó sẽ gửi một tín hiệu HOLD cho bộ xử lý. Bộ xử lý trả lời bằng tín hiệu HLDA (chấp nhận Hold) để cho module DMA biết rằng nó có thể sử dụng bus. Xét ví dụ: nếu module DMA là truyền một khối dữ liệu từ bộ nhớ sang đĩa từ, nó sẽ làm như sau:

- Thiết bị ngoại vi (bộ điều khiển đĩa từ) sẽ yêu cầu dịch vụ của DMA bằng cách thiết lập đường DREQ (Yêu cầu DMA) ở mức cao.
- DMA sẽ đặt tín hiệu HOLD qua đường HRQ vào bộ xử lý để báo cho CPU biết rằng nó cần sử dụng bus.
- CPU sẽ kết thúc chu kỳ bus hiện tại (không nhất thiết là phải kết thúc chu kỳ lệnh hiện tại) và đáp ứng yêu cầu của DMA bằng cách thiết lập đường HDLA lên mức cao, khi đó DMA 8237 biết rằng nó có thể sử dụng bus để thực hiện nhiệm vụ. Tín hiệu HOLD sẽ luôn giữ ở mức cao trong cả quá trình DMA trao đổi dữ liệu.

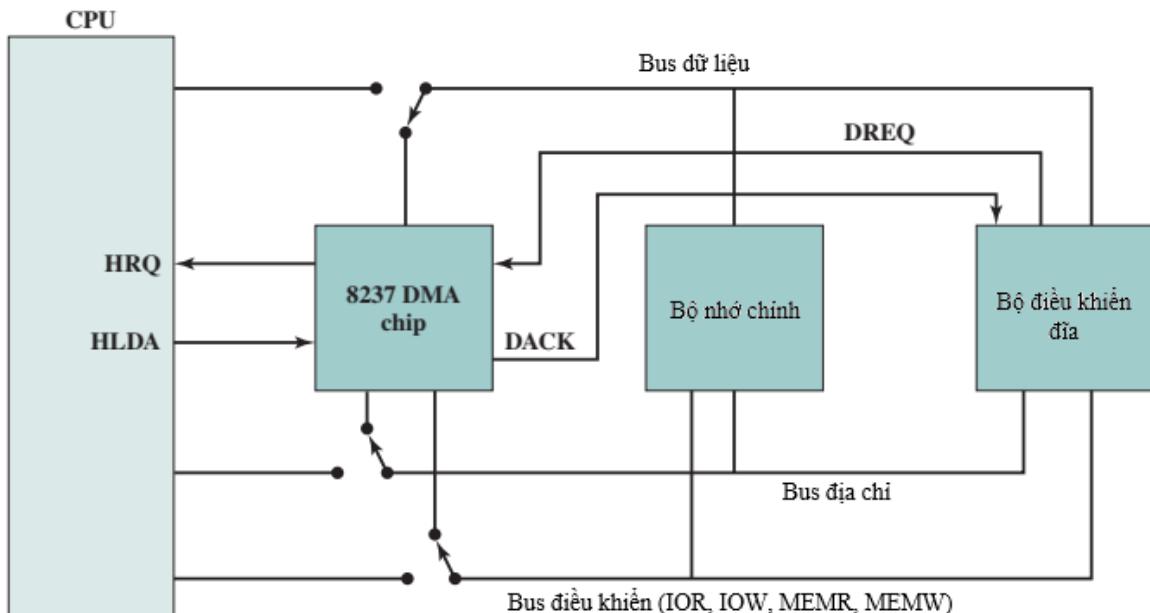
4. DMA thiết lập tín hiệu DACK (Chấp nhận DMA) để báo cho thiết bị ngoại vi rằng nó sẽ bắt đầu truyền dữ liệu.
5. DMA bắt đầu điều khiển việc truyền dữ liệu từ bộ nhớ sang thiết bị ngoại vi bằng cách đặt địa chỉ byte đầu tiên của khối dữ liệu lên bus địa chỉ và kích hoạt tín hiệu MEMR (tín hiệu điều khiển Đọc bộ nhớ), để một byte từ bộ nhớ vào bus dữ liệu; sau đó nó kích hoạt IOW (Ghi I/O) để ghi vào thiết bị ngoại vi. Tiếp đó, DMA giảm thanh ghi đếm dữ liệu và tăng thanh ghi địa chỉ để tiếp tục thao tác truyền các từ tiếp theo. Quá trình này lặp lại cho đến khi Thanh ghi đếm dữ liệu về không.
6. Sau khi DMA kết thúc công việc, nó sẽ huỷ bỏ HRQ để báo cho CPU biết rằng nó có thể lấy lại quyền điều khiển bus.

Trong khi DMA sử dụng bus để truyền dữ liệu, bộ xử lý không hoạt động (bị treo tạm thời). Tương tự như vậy, khi bộ xử lý đang sử dụng bus, DMA không hoạt động. 8237 DMA được biết đến như là một *bộ điều khiển fly-by DMA*, có nghĩa là dữ liệu được di chuyển từ vị trí này đến vị trí khác không đi qua chip DMA và không được lưu trữ trong chip DMA. Do đó, DMA chỉ có thể điều khiển việc truyền dữ liệu giữa cổng I/O và bộ nhớ, chứ không phải giữa hai cổng I/O hoặc hai vị trí bộ nhớ. Tuy nhiên, như được giải thích sau đây, chip DMA có thể thực hiện truyền dữ liệu từ bộ nhớ đến bộ nhớ thông qua thanh ghi.

8237 có bốn kênh DMA có thể được lập trình độc lập và một kênh có thể hoạt động bất kỳ lúc nào. Các kênh này được đánh số 0, 1, 2 và 3.

8237 có 5 thanh ghi điều khiển/lệnh để lập trình và điều khiển hoạt động DMA qua một trong các kênh của nó như sau:

- Lệnh: Bộ xử lý nạp thanh ghi này để điều khiển hoạt động của DMA. D0 cho phép hoặc không cho phép việc chuyển dữ liệu giữa hai vị trí bộ nhớ, trong đó kênh 0 được sử dụng để truyền một byte vào thanh ghi dữ liệu tạm thời của 8237, kênh 1 được sử dụng để truyền byte từ thanh ghi vào vị trí bộ nhớ (bên nhận). Khi bộ nhớ được kích hoạt, D1 có thể được thiết lập để vô hiệu hóa việc tăng/giảm trên kênh 0 để ghi một giá trị cố định vào một khối bộ nhớ. D2 có tác dụng bật hoặc tắt DMA.



DACK = DMA acknowledge = Chấp nhận DMA

DREQ = DMA request = Yêu cầu DMA

HLDA = HOLD acknowledge = Chấp nhận HOLD

HRQ = HOLD request = Yêu cầu HOLD

Hình 7.17 DMA 8237

- **Trạng thái:** Bộ xử lý đọc thanh ghi này để xác định trạng thái DMA. Các bit D0-D3 được sử dụng để cho biết liệu các kênh 0-3 đã đạt được TC (số cuối cùng). Các bit D4-D7 được bộ xử lý sử dụng để xác định xem liệu có kênh nào có yêu cầu DMA đang chờ được xử lý.
- **Chế độ:** Bộ xử lý thiết lập thanh ghi này để xác định chế độ hoạt động của DMA. Các bit D0 và D1 được sử dụng để chọn một kênh. Các bit khác chọn các chế độ hoạt động khác nhau cho kênh đó. Các bit D2 và D3 quy định việc truyền dữ liệu từ thiết bị ngoài sang bộ nhớ (Ghi) hoặc từ bộ nhớ sang I/O (Đọc) hoặc hoạt động kiểm tra. Nếu D4 được thiết lập, thanh ghi địa chỉ bộ nhớ và thanh ghi đếm được nạp lại với giá trị ban đầu của chúng sau khi kết thúc một hoạt động DMA. Các bit D6 và D7 quy định các chế độ (mode) của 8237. Trong chế độ đơn mode, duy nhất một byte dữ liệu được truyền. Chế độ truyền nhóm được sử dụng cho hoạt động truyền một khối dữ liệu, chế độ yêu cầu cho phép kết thúc sớm một hoạt động truyền. Chế độ nối tầng cho phép nhiều 8237 được nối tầng với nhau để mở rộng số kênh
- **Single Mask:** Bộ vi xử lý thiết lập thanh ghi này. Các bit D0 và D1 để chọn kênh. Bit D2 quy định việc thiết lập hoặc không thiết lập bit mặt nạ cho kênh đó. Thanh ghi Lệnh có thể được sử dụng để vô hiệu hóa toàn bộ chip DMA, còn thanh ghi Single Mask có thể được lập trình để cho phép hoặc cho không phép một kênh cụ thể.

- All Mask: thanh ghi này tương tự như thanh ghi Single Mask như nó thực hiện việc thiết lập hoặc không thiết lập mặt nạ với tất cả bốn kênh trong những hoạt động Ghi.

Ngoài ra, 8237A có tám thanh ghi dữ liệu: một thanh ghi địa chỉ bộ nhớ và một thanh ghi đếm dữ liệu cho mỗi kênh. Bộ vi xử lý thiết lập những thanh ghi này để chỉ định đến một vị trí của bộ nhớ chính bị ảnh hưởng khi thực hiện truyền dữ liệu.

7.4. KÊNH VÀO/RA VÀ BỘ XỬ LÝ VÀO/RA

7.4.1. Quá trình phát triển của các chức năng I/O

Khi hệ thống máy tính đã trở nên phát triển, mô hình máy tính ngày càng phức tạp và tinh vi. Điều này thể hiện rõ nhất ở chức năng của các module vào/ra. Các bước phát triển của chức năng vào/ra có thể tóm tắt như sau:

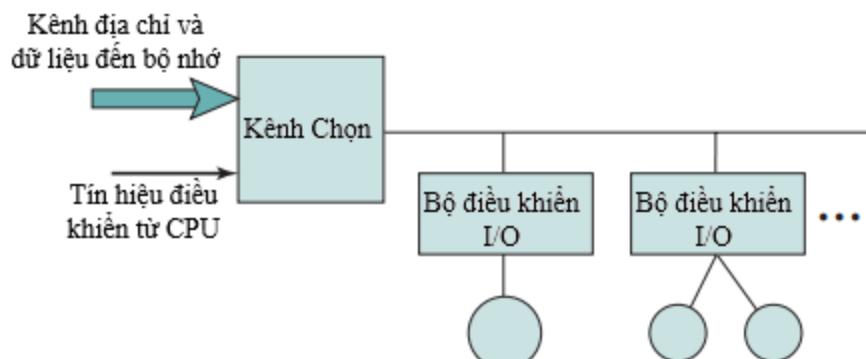
1. Ban đầu, CPU trực tiếp điều khiển một thiết bị ngoại vi. Điều này được thấy trong các vi điều khiển đơn giản.
2. Sử dụng các bộ điều khiển hoặc module I/O. CPU sử dụng cơ chế I/O chương trình. Từ thời điểm này, CPU bắt đầu tách ra khỏi việc điều khiển chi tiết thiết bị ngoài.
3. Cấu trúc tương tự như trong giai đoạn 2, tuy nhiên sử dụng cơ chế I/O điều khiển ngắn. CPU không phải tốn nhiều thời gian chờ đợi một hoạt động I/O được thực hiện, do đó tăng hiệu năng của hệ thống.
4. Module I/O được truy cập trực tiếp tới bộ nhớ thông qua cơ chế DMA. Nó có thể thực hiện việc trao đổi một khối dữ liệu đến hoặc đi từ bộ nhớ mà không cần có sự tham gia của CPU ngoại trừ thời điểm bắt đầu và kết thúc quá trình truyền.
5. Module I/O được cải tiến để trở thành một bộ xử lý I/O chuyên thực thi các hoạt động I/O. CPU khi này chỉ cần đưa ra chỉ thị cho bộ xử lý I/O thực hiện một chương trình I/O trong bộ nhớ. Bộ xử lý I/O sẽ nạp và thực hiện các chỉ thị này mà không cần sự can thiệp của CPU. Điều này cho phép CPU có thể chỉ thị một loạt các hoạt động vào/ra và chỉ bị ngắt khi nào toàn bộ công việc đã được thực hiện.
6. Module I/O có bộ nhớ cục bộ của riêng nó, thực chất nó có thể được coi là một máy tính với khả năng nhất định. Với kiến trúc này, một loạt các thiết bị ngoại vi có thể được điều khiển với sự tham gia tối thiểu của CPU. Bộ xử lý I/O sẽ thực hiện hầu hết các nhiệm vụ liên quan đến việc điều khiển các thiết bị đầu cuối.

Với quá trình phát triển này, điều dễ nhận thấy là sự tham gia của CPU vào các tác vụ vào/ra ngày càng ít, do đó giúp cải thiện hiệu suất CPU. Ở giai đoạn 5,6, một bước thay đổi lớn đối với khái niệm module I/O, module I/O từ đây có khả năng thực hiện các chương trình. Ở giai đoạn 5, module I/O sẽ được gọi kênh I/O. Còn với bước 6, người ta sử dụng thuật ngữ bộ xử lý I/O. Tuy nhiên, thông thường cả hai thuật ngữ này đều được sử dụng ngang nhau trong cả hai giai đoạn. Ở phần sau chúng tôi sẽ dùng chung thuật ngữ Kênh I/O.

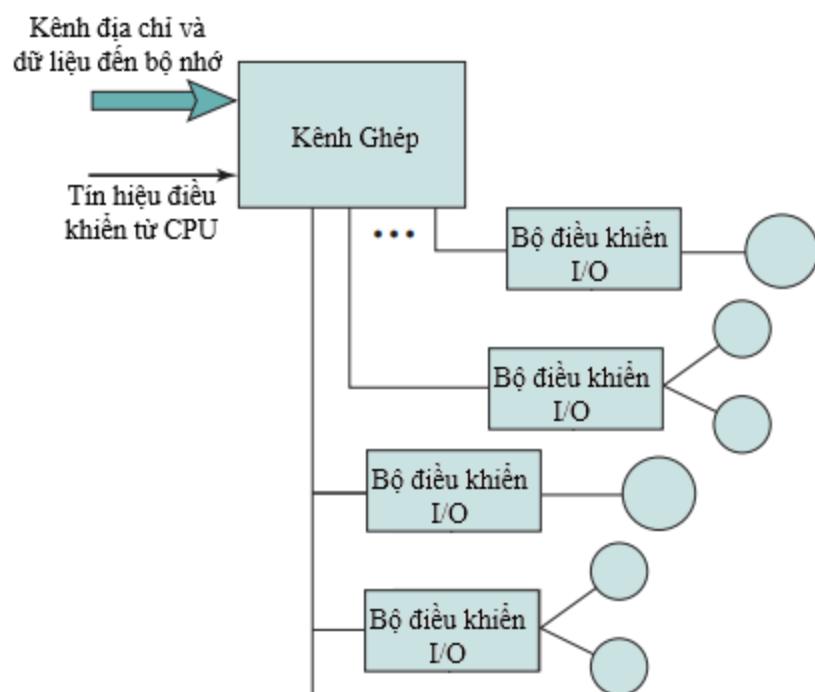
7.4.2. Đặc điểm của các Kênh I/O

Khái niệm Kênh I/O là mở rộng của khái niệm DMA. Kênh I/O có khả năng thực hiện các lệnh vào/ra, cho phép điều khiển hoàn toàn các hoạt động vào/ra. Trong một hệ thống máy tính với các thiết bị như vậy, CPU không thực hiện các lệnh vào/ra. Những chỉ thị này được lưu trữ trong bộ nhớ chính và được bộ xử lý của Kênh I/O thực thi. Do đó, khi có một

hoạt động vào/ra, CPU chỉ thị cho Kênh I/O để thực thi một chương trình trong bộ nhớ. Chương trình này sẽ định ra một hoặc nhiều thiết bị, một hoặc nhiều vùng bộ nhớ để lưu trữ, các mức ưu tiên và hoạt động sẽ được thực hiện trong trường hợp có các điều kiện lỗi nhất định. Kênh I/O sẽ theo các chỉ thị này và điều khiển việc truyền dữ liệu.



a. Kênh chọn



b. Kênh ghép

Hình 7.18 Kiến trúc Kênh I/O

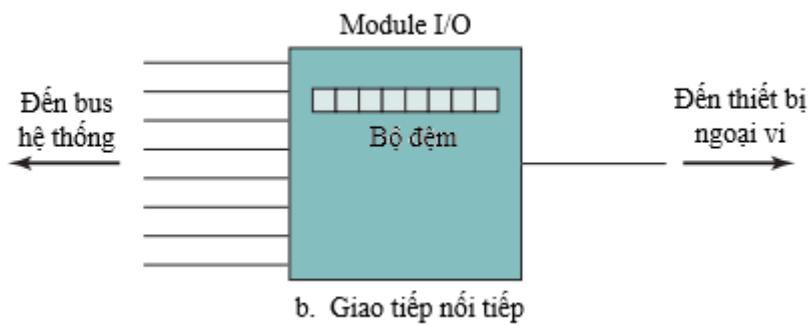
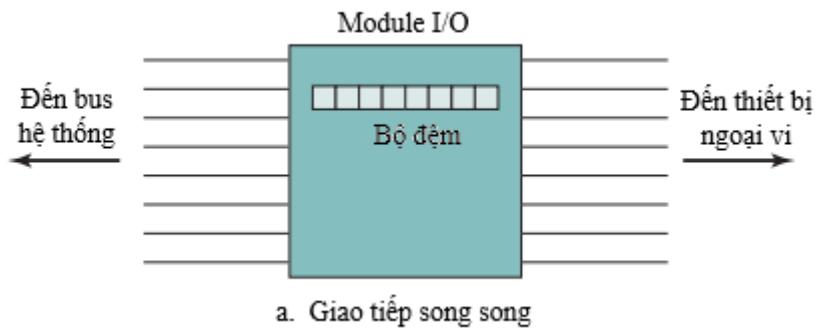
Có hai loại Kênh I/O phổ biến như trong Hình 7.18. **Kênh Chọn** điều khiển nhiều thiết bị tốc độ cao và được dành riêng cho việc truyền dữ liệu với một trong những thiết bị đó tại bất cứ thời điểm nào. Do đó, Kênh I/O lựa chọn một thiết bị và tác động đến việc truyền dữ liệu. Một thiết bị hoặc một nhóm thiết bị được điều khiển bởi một bộ điều khiển hoặc module I/O. Do đó, các Kênh I/O sẽ thay cho CPU trong việc kiểm soát, điều khiển các module I/O. **Kênh Ghép** có thể xử lý vào/ra nhiều thiết bị cùng một lúc. Với các

thiết bị tốc độ thấp, *Bộ ghép kênh* theo byte sẽ chấp nhận hoặc truyền các ký tự nhanh nhất có thể cho nhiều thiết bị. Ví dụ: ba luồng ký tự từ ba thiết bị có tốc độ khác nhau như sau: $A_1A_2A_3A_4 \dots$, $B_1B_2B_3B_4 \dots$ và $C_1C_2C_3C_4 \dots$ có thể được truyền như sau $A_1B_1C_1A_2C_2A_3B_2C_3A_4 \dots$. Với các thiết bị tốc độ cao, một khối dữ liệu của nó sẽ được ghép nối xen kẽ với một khối dữ liệu từ các thiết bị khác.

7.5. GIAO TIẾP NGOÀI

7.5.1. Các loại giao tiếp

Giao tiếp giữa một thiết bị ngoại vi và một module I/O phải được điều chỉnh phù hợp với tính chất và hoạt động của thiết bị ngoại vi. Đặc điểm chính của giao tiếp này là nó có thể là giao tiếp nối tiếp hoặc giao tiếp song song (Hình 7.19). Trong giao tiếp song song, có nhiều đường kết nối giữa module I/O và thiết bị ngoại vi, các bit được truyền đồng thời giống như việc truyền một từ trên bus dữ liệu. Trong một giao tiếp nối tiếp, chỉ có một đường truyền dữ liệu, mỗi bit được truyền một thời điểm. Giao tiếp song song thường được sử dụng cho các thiết bị ngoại vi tốc độ cao như băng từ, đĩa từ, còn giao tiếp nối tiếp thường được sử dụng cho máy in và các thiết bị đầu cuối khác. Ngày nay, với sự phát triển của các giao tiếp nối tiếp tốc độ cao, giao tiếp song song đang trở nên ít phổ biến hơn.



Hình 7.19 Giao tiếp song song và nối tiếp

Trong cả hai trường hợp, để thực hiện việc trao đổi dữ liệu, module I/O phải đối thoại với thiết bị ngoại vi. Ví dụ với trường hợp thực hiện hoạt động ghi như sau:

1. Module I/O gửi tín hiệu điều khiển yêu cầu cho phép gửi dữ liệu.
2. Thiết bị ngoại vi chấp nhận yêu cầu.

3. Module I/O truyền dữ liệu (một từ hoặc một khối dữ liệu tùy thuộc vào thiết bị ngoại vi).
4. Thiết bị ngoại vi nhận dữ liệu và gửi xác nhận việc đó.

Hoạt động đọc cũng được thực hiện với các thao tác tương tự.

Một thành phần quan trọng của module I/O là bộ đệm, nó có tác dụng lưu trữ dữ liệu tạm thời để cân bằng sự chênh lệch tốc độ giữa bus hệ thống và các đường kết nối ngoài.

7.5.2. Cấu hình Điểm – Điểm và Đa điểm

Kết nối giữa một module I/O và các thiết bị ngoại có thể là kết nối điểm-điểm hoặc kết nối đa điểm. Với giao tiếp điểm-điểm, module I/O được nối với mỗi thiết bị ngoại vi qua một đường riêng. Trên các hệ thống nhỏ (các máy tính cá nhân hoặc máy trạm), các kết nối điểm - điểm điển hình gồm có: kết nối với bàn phím, máy in và modem ngoài.

Giao tiếp đa điểm ngày càng phổ biến và trở nên quan trọng hơn. Giao tiếp này thường được sử dụng để hỗ trợ các thiết bị lưu trữ ngoài (như ổ đĩa và băng từ) và các thiết bị đa phương tiện (CD-ROM, video, audio). Các giao tiếp đa điểm này là các bus mở rộng và có cùng logic hoạt động giống bus.

7.6. CÂU HỎI

1. Liệt kê ba loại thiết bị ngoại vi hoặc thiết bị ngoại.
2. Bảng IRA là bảng gì?
3. Các chức năng chính của module Vào/ra là gì?
4. Liệt kê và nêu ngắn gọn ba kỹ thuật để thực hiện Vào/ra.
5. Sự khác biệt giữa I/O ánh xạ bộ nhớ và I/O riêng biệt là gì?
6. Khi xảy ra ngắt thiết bị, bộ xử lý xác định thiết bị đưa ra yêu cầu ngắt như thế nào?
7. Bộ xử lý sẽ làm gì trong khi một module DMA chiếm và duy trì quyền kiểm soát bus?

Chương 8. HỆ ĐỆM

8.1. HỆ THẬP PHÂN

Trong cuộc sống hàng ngày, chúng ta sử dụng một hệ thống dựa trên các chữ số thập phân ($0, 1, 2, 3, 4, 5, 6, 7, 8, 9$) để biểu diễn các số, và hệ thống như thế được gọi là hệ thập phân. Hãy xét ý nghĩa của số 83 . Nó có nghĩa là tám chục cộng với ba:

$$83 = (8 \times 10) + 3$$

Số 4728 có nghĩa là bốn nghìn, bảy trăm, hai chục, cộng với tám:

$$4728 = (4 \times 1000) + (7 \times 100) + (2 \times 10) + 8$$

Hệ thập phân có **cơ số** là 10 . Điều này có nghĩa là mỗi chữ số trong một số được nhân với luỹ thừa bậc i của 10 , với i là vị trí tương ứng của chữ số đó:

$$83 = (8 \times 10^1) + (3 \times 10^0)$$

$$4728 = (4 \times 10^3) + (7 \times 10^2) + (2 \times 10^1) + (8 \times 10^0)$$

Phần phân số thập phân cũng tuân theo nguyên tắc tương tự, nhưng số mũ i là âm. Như vậy, phân số thập phân 0.256 là 2 phần mươi cộng với 5 phần trăm cộng với 6 phần nghìn hay là:

$$0.256 = (2 \times 10^{-1}) + (5 \times 10^{-2}) + (6 \times 10^{-3})$$

Một số có cả phần nguyên và phần phân số thì các chữ số được nhân với 10 mũ i , với i có cả giá trị dương và âm:

$$442.256 = (4 \times 10^2) + (4 \times 10^1) + (2 \times 10^0) + (2 \times 10^{-1}) + (5 \times 10^{-2}) + (6 \times 10^{-3})$$

Trong một số bất kỳ, chữ số ngoài cùng bên trái được gọi là chữ số quan trọng nhất (chữ số có trọng số lớn nhất), bởi vì nó mang giá trị lớn nhất. Các chữ số ngoài cùng bên phải được gọi là chữ số ít quan trọng nhất (chữ số có trọng số nhỏ nhất). Trong số thập phân ở ví dụ trên, số 4 phía bên trái là chữ số có trọng số lớn nhất và số 6 bên phải là chữ số có trọng số nhỏ nhất.

Bảng 9.1 cho thấy mối quan hệ giữa vị trí của chữ số và giá trị được gán cho vị trí đó. Mỗi vị trí có trọng số lớn gấp 10 lần giá trị trọng số của vị trí bên phải nó và bằng một phần mươi giá trị trọng số của vị trí bên trái nó. Nếu chúng ta đánh thứ tự vị trí chữ số như ví dụ trong Bảng 9.1, thì vị trí i có trọng số là 10^i .

Bảng 8.1 Vị trí của một số thập phân

4	7	2	2	5	6
100	10	1i	1/10	1/100	1/1000
10^2	10^1	10^0	10^{-1}	10^{-2}	10^{-3}
Vị trí 2	Vị trí 1	Vị trí 0	Vị trí -1	Vị trí -2	Vị trí -3

Tổng quát, nếu số X có biểu diễn dạng thập phân là $X = \{...d_2d_1d_0.d_{-1}d_{-2}d_{-3}...\}$, giá trị của X là:

$$X = \sum_i (d_i \times 10^i)$$

Lưu ý, xét số 509 và hỏi có bao nhiêu giá trị hàng chục trong số đó. Bởi vì có một chữ số 0 ở vị trí hàng chục, có người sẽ nói rằng không có hàng chục. Nhưng trên thực tế ở đây có 50 chục. Chữ số 0 ở vị trí hàng chục có nghĩa là không có phần hàng chục lẻ ra mà không thể gộp vào hàng trăm, hoặc hàng ngàn, vv... Do đó, bởi vì mỗi vị trí chỉ chứa giá trị số lẻ ra mà không thể gộp vào các vị trí cao hơn, mỗi vị trí chữ số phải có giá trị không lớn hơn 9. Vậy 9 là giá trị lớn nhất mà một vị trí có thể chứa.

8.2. HỆ ĐÉM CÓ VỊ TRÍ

Trong hệ đếm có vị trí, mỗi số được biểu diễn bởi một dãy các chữ số trong đó mỗi vị trí chữ số i có trọng số tương ứng r^i , trong đó r là cơ số của hệ đếm đó. Dạng biểu diễn tổng quát của một số thuộc hệ đếm có cơ số r là

$$(\dots a_2a_1a_0.a_{-1}a_{-2}a_{-3}\dots)_r$$

trong đó giá trị của chữ số a_i bất kỳ là một số nguyên trong khoảng $0 \leq a_i < r$. Dấu chấm giữa a_0 và a_{-1} được gọi là dấu thập phân (dấu phân số)². Số này có giá trị là

$$\dots + a_2r^2 + a_1r^1 + a_0r^0 + a_{-1}r^{-1} + a_{-2}r^{-2} + a_{-3}r^{-3} + \dots = \sum_i (a_i \times r^i)$$

Hệ thập phân là một trường hợp đặc biệt của một hệ số đếm có vị trí với cơ số 10 và các chữ số trong khoảng từ 0 đến 9.

Ví dụ về một hệ đếm có vị trí khác, hãy xét hệ đếm với cơ số 7. Bảng 8.2 cho thấy giá trị trọng số của các vị trí từ -1 đến 4. Tại mỗi vị trí, chữ số có thể nhận giá trị trong khoảng 0 đến 6.

Bảng 8.2 Vị trí của một số trong hệ cơ số 7

Vị trí	4	3	2	2	0	-1
Trọng số	7^4	7^3	7^2	7^1	7^0	7^{-1}
Giá trị thập phân tương ứng	2401	343	49	7	1	$1/7$

8.3. HỆ NHỊ PHÂN

Trong hệ thập phân, 10 chữ số khác nhau được sử dụng để biểu diễn các số theo cơ số 10. Trong hệ nhị phân, ta chỉ dùng hai chữ số là 1 và 0. Do vậy, số trong hệ nhị phân được biểu diễn theo cơ số 2.

Để tránh nhầm lẫn, đôi khi chúng ta đặt một số nhỏ bên cạnh số cần biểu diễn để chỉ rõ cơ số của nó. Ví dụ: 83_{10} và 4728_{10} là các số được biểu diễn bằng ký hiệu thập phân

² Dấu thập phân trong tiếng Việt là dấu phẩy (,). Tuy nhiên, để dễ dàng tiếp thu và vận dụng kiến thức trong ngành công nghệ thông tin, chúng ta sử dụng dấu thập phân là dấu chấm (.) theo chuẩn chung quốc tế.

hoặc gọi ngắn gọn là số thập phân. Các chữ số 1 và 0 trong ký hiệu nhị phân có cùng ý nghĩa như trong ký hiệu thập phân:

$$0_2 = 0_{10}$$

$$1_2 = 1_{10}$$

Để biểu diễn các số lớn hơn, như với ký hiệu thập phân, mỗi chữ số trong một số nhị phân có một giá trị phụ thuộc vào vị trí của nó. Ví dụ:

$$10_2 = (1 \times 2^1) + (0 \times 2^0) = 2_{10}$$

$$11_2 = (1 \times 2^1) + (1 \times 2^0) = 3_{10}$$

$$100_2 = (1 \times 2^2) + (0 \times 2^1) + (0 \times 2^0) = 4_{10}$$

Ngoài ra, các giá trị phân số được biểu diễn với luỹ thừa bậc số mũ âm của cơ số:

$$1001.101 = 2^3 + 2^0 + 2^{-1} + 2^{-3} = 9.625_{10}$$

Tổng quát, nếu biểu diễn nhị phân của $Y = \{ \dots b_{m-1}b_{m-2}\dots b_1b_0.b_{-1}b_{-2}b_{-3}\dots \}$, giá trị của Y là

$$Y = \sum_i (b_i \times 2^i)$$

8.4. CHUYỂN ĐỔI GIỮA NHỊ PHÂN VÀ THẬP PHÂN

Việc chuyển đổi một số từ ký hiệu nhị phân sang ký hiệu thập phân được thực hiện rất dễ dàng. Thực tế, chúng ta đã gặp một số ví dụ trong mục trước. Để chuyển đổi một số từ nhị phân sang thập phân, ta chỉ cần nhân mỗi chữ số nhị phân với luỹ thừa của 2 và cộng vào kết quả.

Để chuyển từ thập phân sang nhị phân, ta cần thực hiện riêng việc chuyển đổi phần nguyên và phần phân số.

8.4.1. Phần nguyên

Đối với phần nguyên, nhớ lại rằng trong ký hiệu nhị phân, số nguyên biểu diễn bởi

$$b_{m-1}b_{m-2}\dots b_2b_1b_0 \quad b_i = 0 \text{ hoặc } 1$$

có giá trị là

$$(b_{m-1} \times 2^{m-1}) + (b_{m-2} \times 2^{m-2}) + \dots + (b_1 \times 2^1) + b_0$$

Giả sử ta cần chuyển đổi một số nguyên thập phân N thành dạng nhị phân. Nếu chia N cho 2, trong hệ thập phân, nhận được thương số N_1 và phần dư R_0 , ta có thể viết lại

$$N = 2 \times N_1 + R_0 \quad R_0 = 0 \text{ hoặc } 1$$

Tiếp theo, ta chia giá trị N_1 cho 2. Giả sử rằng thương số mới là N_2 và phần dư mới là R_1 . Khi đó,

$$N_1 = 2 \times N_2 + R_1 \quad R_1 = 0 \text{ hoặc } 1$$

vậy là

$$N = 2(2N_2 + R_1) + R_0 = (N_2 \times 2^2) + (R_1 \times 2^1) + R_0$$

Nếu tiếp tục

$$N_2 = 2 \times N_3 + R_2$$

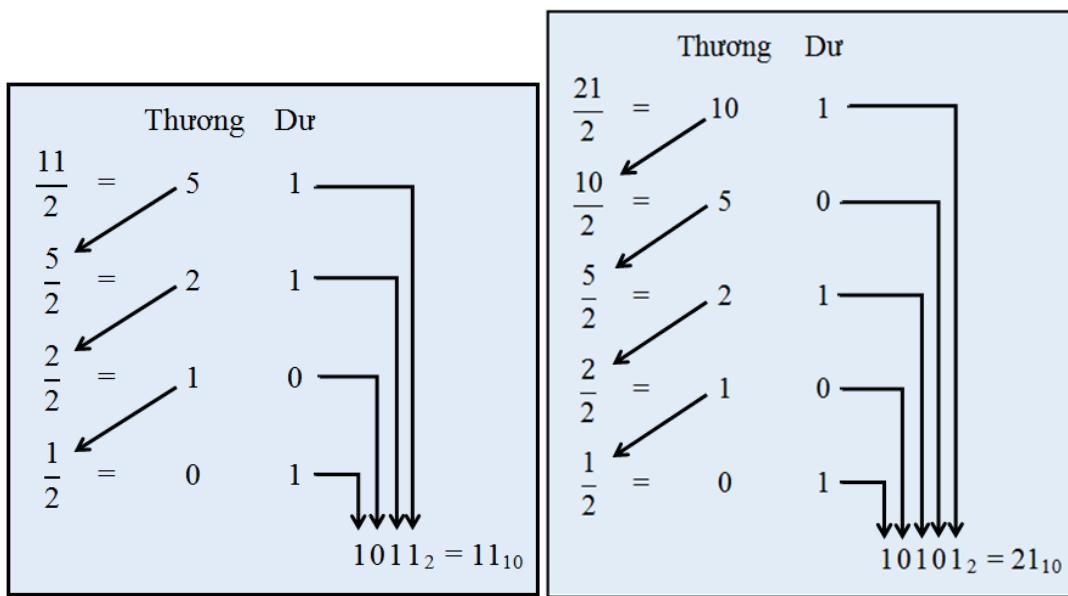
ta có

$$N = (N_3 \times 2^3) + (R_2 \times 2^2) + (R_1 \times 2^1) + R_0$$

Vì $N > N_1 > N_2 \dots$, tiếp tục thực hiện tương tự cuối cùng sẽ tạo ra một thương số $N_{m-1} = 1$ và phần dư R_{m-2} bằng 0 hoặc 1. Khi đó,

$$N = (1 \times 2^{m-1}) + (R_{m-2} \times 2^{m-2}) + \dots + (R_2 \times 2^2) + (R_1 \times 2^1) + R_0$$

là dạng nhị phân của N . Do đó, ta chuyển đổi từ cơ số 10 sang cơ số 2 bằng cách lặp lại việc chia một số cho 2. Thương số cuối cùng (bằng 1) và các số dư theo thứ tự ý nghĩa tăng dần chính là các chữ số nhị phân của N . Xét hai ví dụ trong Hình 8.1.

(a) 11_{10} (b) 21_{10}

Hình 8.1 Ví dụ chuyển đổi từ thập phân sang nhị phân đối với phần nguyên

8.4.2. Phần phân số

Đối với phần phân số, nhớ lại rằng trong ký hiệu nhị phân, một số có giá trị trong khoảng từ 0 đến 1 được biểu diễn bởi

$$0.b_{-1}b_{-2}b_{-3}\dots \quad b_i = 0 \text{ hoặc } 1$$

và có giá trị là

$$(b_{-1} \times 2^{-1}) + (b_{-2} \times 2^{-2}) + (b_{-3} \times 2^{-3})\dots$$

Có thể biến đổi thành

$$2^{-1} \times (b_{-1} + 2^{-1} \times (b_{-2} + 2^{-1} \times (b_{-3} + \dots)))$$

Cách biểu diễn này gợi ý cho ta về kỹ thuật chuyển đổi. Giả sử ta muốn chuyển đổi số F ($0 < F < 1$) từ ký hiệu thập phân sang nhị phân. Chúng ta biết rằng F có thể được biểu diễn dưới dạng

$$F = 2^{-1} \times (b_{-1} + 2^{-1} \times (b_{-2} + 2^{-1} \times (b_{-3} + \dots)))$$

Nếu ta nhân F với 2, ta thu được,

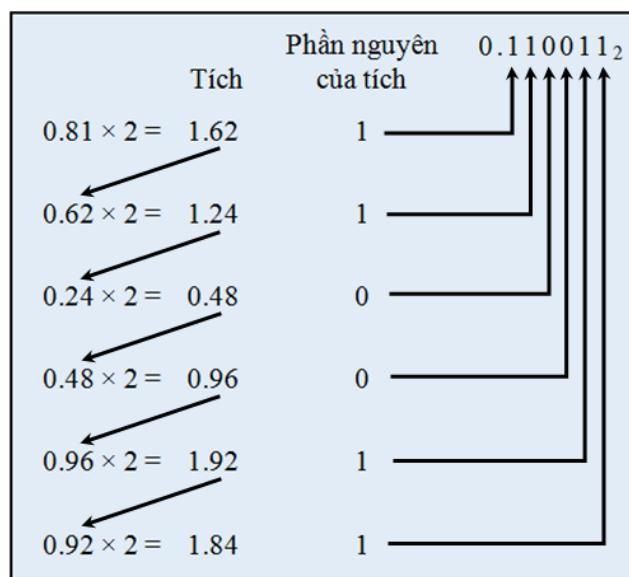
$$2 \times F = b_{-1} + 2^{-1} \times (b_{-2} + 2^{-1} \times (b_{-3} + \dots) \dots)$$

Từ phương trình này, chúng ta thấy rằng phần nguyên của $(2 \times F)$ phải là 0 hoặc 1 bởi vì $0 < F < 1$, và chính bằng b_{-1} . Vì vậy, chúng ta có thể viết $(2 \times F) = b_{-1} + F_1$, trong đó $0 < F_1 < 1$ và

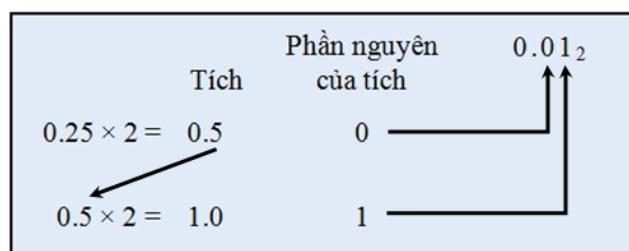
$$F_1 = 2^{-1} \times (b_{-2} + 2^{-1} \times (b_{-3} + 2^{-1} \times (b_{-4} + \dots) \dots))$$

Để tìm b_{-2} , chúng ta lặp lại quá trình trên. Do đó, thuật toán chuyển đổi phần phân số liên quan đến việc lặp lại phép nhân với 2. Ở mỗi bước, phần phân số của tích số trong phép nhân ở bước trước được nhân với 2. Chữ số bên trái dấu thập phân trong tích nhận được là 0 hoặc 1 và sẽ tham gia vào biểu diễn nhị phân của số cần chuyển đổi, bắt đầu với chữ số quan trọng nhất. Phần phân số của tích được sử dụng làm số bị nhân trong bước tiếp theo. Xét hai ví dụ trong Hình 8.2.

Kết quả chuyển đổi không nhất thiết phải chính xác; một phân số thập phân với giá trị hữu hạn có thể phải biểu diễn tương ứng bằng một phân số nhị phân vô hạn. Trong trường hợp như vậy, thuật toán chuyển đổi thường phải tạm dừng sau một số bước đã định, tùy thuộc vào độ chính xác mong muốn.



(a) $0.81_{10} = 0.110011_2$ (xấp xỉ)



(b) $0.25_{10} = 0.01_2$ (chính xác)

Hình 8.2 Ví dụ chuyển đổi từ thập phân sang nhị phân đối với phần phân số

8.5. KÝ HIỆU THẬP LỤC PHÂN

Do các thành phần máy vi tính kỹ thuật số đều thể hiện tính chất nhị phân, tất cả các dạng dữ liệu bên trong máy tính đều được biểu diễn bằng mã nhị phân. Tuy nhiên, cho dù hệ thống nhị phân của máy tính có thuận tiện như thế nào đối với máy tính, thì đối với con người, hệ thống số này lại quá cồng kềnh. Do đó, đa phần các chuyên gia máy tính, những người phải dành nhiều thời gian làm việc với các dữ liệu thô thực tế trong máy tính, đều muốn làm việc với dạng ký hiệu ngắn gọn hơn.

Vậy dạng ký hiệu nào được sử dụng? Ký hiệu thập phân là một khả năng. Nó chắc chắn là ngắn gọn hơn ký hiệu nhị phân, nhưng việc chuyển đổi giữa cơ số 2 và cơ số 10 lại khá nhầm chán.

Thay vào đó, ký hiệu thập lục phân đã được sử dụng. Các chữ số nhị phân được nhóm thành các nhóm bốn bit, được gọi là **nibble**. Mỗi tổ hợp có thể có của bốn chữ số nhị phân được gán cho một ký tự tương ứng như sau:

0000 = 0	0100 = 4	1000 = 8	1100 = C
0001 = 1	0101 = 5	1001 = 9	1101 = D
0010 = 2	0110 = 6	1010 = A	1110 = E
0011 = 3	0111 = 7	1011 = B	1111 = F

Bởi vì có 16 ký tự được sử dụng, nó được gọi là hệ thập lục phân, và 16 ký tự này chính là các chữ số thập lục phân.

Một dãy chữ số thập lục phân biểu diễn một số nguyên cơ số 16 (Bảng 8.3). Do vậy

$$\begin{aligned} 2 &= (2_{16} \times 16^1) + (C_{16} \times 16^0) \\ C_{16} &= (2_{10} \times 16^1) + (12_{10} \times 16^0) \\ &= 44 \end{aligned}$$

Do đó, một số thập lục phân là số trong hệ đếm có vị trí với cơ số 16 có thể biểu diễn tổng quát như sau

$$Z = \sum_i (h_i \times 16^i)$$

trong đó 16 là cơ số và mỗi chữ số thập lục phân h_i có giá trị trong khoảng thập phân $0 \leq h_i < 15$, tương đương với khoảng thập lục phân $0 \leq h_i < F$.

Bảng 8.3 Thập phân, nhị phân và thập lục phân

Thập phân (cơ số 10)	Nhi phân (cơ số 2)	Thập lục phân (cơ số 16)
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4

5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F
16	0001 0000	10
17	0001 0001	11
18	0001 0010	12
31	0001 1111	1F
100	0110 0100	64
255	1111 1111	FF
256	0001 0000 0000	100

Ký hiệu thập lục phân không chỉ được sử dụng để biểu diễn số nguyên mà còn được sử dụng để biểu diễn ngắn gọn một dãy số nhị phân bất kỳ. Lý do cho việc sử dụng ký hiệu thập lục phân là:

1. Nó ngắn gọn hơn ký hiệu nhị phân.
2. Trong hầu hết các máy tính, dữ liệu nhị phân thường ở dạng bội số của 4 bit, và do đó tương đương với một vài chữ số thập lục phân.
3. Việc chuyển đổi giữa ký hiệu nhị phân và thập lục phân rất dễ thực hiện.

Để làm sáng tỏ lý do cuối cùng, ta xét chuỗi nhị phân 110111100001. Chuỗi này tương đương với

$$1101 \quad 1110 \quad 0001 = DE1_{16}$$

D E 1

8.6. CÂU HỎI

1. Đếm từ 1 đến 20_{10} trong các hệ đếm có cơ số như sau:

- a. 8 b. 6 c. 5 d. 3

2. Sắp xếp các số sau theo thứ tự tăng dần: $(1.1)_2$, $(1.4)_{10}$, $(1.5)_{16}$

3. Thực hiện các chuyển đổi sau:

- a. 54_8 sang hệ cơ số 5
- b. 312_4 sang hệ cơ số 7
- c. 520_6 sang hệ cơ số 7

d. 12212_3 sang hệ cơ số 9

4. Đổi các số nhị phân sau ra số tương đương trong hệ thập phân:

- a. 001100 b. 000011 c. 011100 d. 111100

e. 101010

5. Đổi các số nhị phân sau ra số tương đương trong hệ thập phân:

- a. 11100.011 b. 110011.10011 c. 1010101010.1

6. Đổi các số thập phân sau ra số tương đương trong hệ nhị phân:

- a. 64 b. 100 c. 111 d. 145 e. 255

7. Đổi các số thập phân sau ra số tương đương trong hệ nhị phân:

- a. 34.75 b. 25.25 c. 27.1875

8. Biểu diễn các số bát phân (cơ số 8) sau ở dạng ký hiệu thập lục phân:

- a. 12 b. 5655 c. 2550276 d. 76545336

e. 3726755

9. Đổi các số thập lục phân sau ra số tương đương trong hệ thập phân:

- a. C b. 9F c. D52 d. 67E e.

ABCD

10. Đổi các số thập lục phân sau ra số tương đương trong hệ thập phân:

- a. F.4 b. D3.E c. 1111.1 d. 888.8

EBA.C

11. Đổi các số thập phân sau ra số tương đương trong hệ thập lục phân:

- a. 16 b. 80 c. 2560 d. 3000 e. 62500

12. Đổi các số thập phân sau ra số tương đương trong hệ thập lục phân:

- a. 204.125 b. 255.875 c. 631.25 d.

10000.00390625

13. Đổi các số thập lục phân sau ra số tương đương trong hệ nhị phân:

- a. E b. 1C c. A64 d. 1F.C e. 239.4

14. Đổi các số nhị phân sau ra số tương đương trong hệ thập lục phân:

- a. 1001.1111 b. 110101.011001 c.

10100111.111011

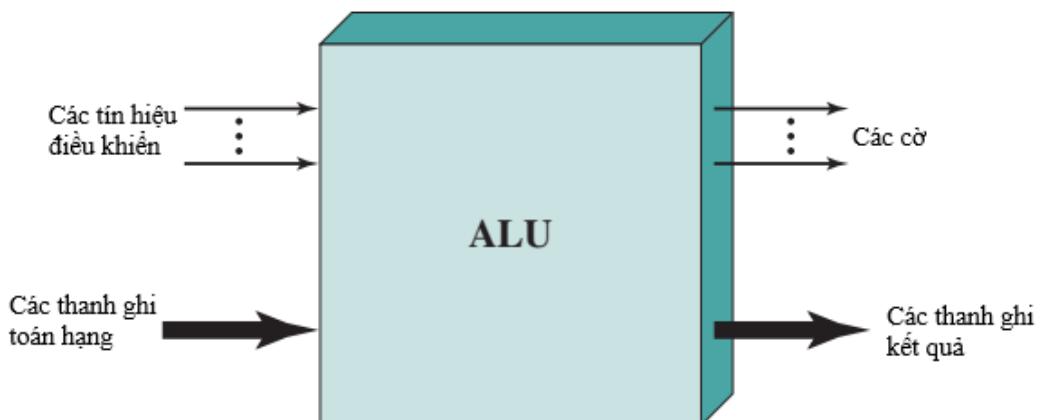
Chương 9. TÍNH TOÁN SỐ HỌC

9.1. KHỐI TÍNH TOÁN SỐ HỌC VÀ LOGIC

ALU là khối thực hiện các phép toán số học và logic dữ liệu trong hệ thống máy tính. Tất cả các thành phần khác của hệ thống máy tính như: khối CU, thanh ghi, bộ nhớ, I/O – có nhiệm vụ chủ yếu là đưa dữ liệu vào ALU để nó xử lý và sau đó đưa kết quả trở lại. Như vậy, với việc tìm hiểu ALU, ta có thể hiểu được cốt lõi của quá trình xử lý dữ liệu trong máy tính như thế nào.

Khối ALU cũng như các thành phần khác của máy tính được được xây dựng từ các phần tử logic số đơn giản, có thể lưu trữ các số nhị phân và thực hiện các phép toán logic Boolean đơn giản.

Hình 9.1 cho thấy cách ALU kết nối với phần còn lại của bộ xử lý. Các toán hạng của các phép toán số học và logic được lưu trong các thanh ghi, và kết quả tính toán của phép toán cũng được ALU đưa ra một thanh ghi nhất định. Các thanh ghi này lưu trữ dữ liệu tạm thời và có các đường kết nối đến ALU (ví dụ xem hình 2.3). ALU cũng có thể thiết lập các cờ trong thanh ghi cờ, ví dụ, cờ tràn được thiết lập thành 1 nếu kết quả tính toán vượt quá chiều dài của thanh ghi lưu trữ kết quả. Bộ xử lý cũng gửi các tín hiệu điều khiển hoạt động của ALU và di chuyển dữ liệu ra hoặc vào ALU.



Hình 9.1 Đầu vào và đầu ra khối ALU

9.2. BIỂU DIỄN SỐ NGUYÊN

Trong hệ thống số nhị phân, các số được biểu diễn dưới dạng các chữ số 0 hoặc 1 cùng với dấu âm (dành cho số âm) và dấu chấm (dành cho số thực có phần phân)

$$-1101.0101 = -13.3125_{10}$$

Với mục đích lưu trữ và xử lý trong máy tính, chúng ta không có các ký hiệu đặc biệt sử dụng để biểu diễn dấu âm và dấu chấm. Nếu chỉ giới hạn trong các số nguyên không âm thì việc biểu diễn hết sức đơn giản

Một từ 8-bit có thể biểu diễn các số từ 0 đến 255 như sau

$$00000000 = 0$$

$$00000001 = 1$$

$$\begin{aligned}00101001 &= 41 \\10000000 &= 128 \\11111111 &= 255\end{aligned}$$

Nghĩa là, nếu một số ta đổi chuỗi n-bit gồm các chữ số nhị phân $a_{n-1}a_{n-2} \dots a_1a_0$ sang số nguyên không dấu A ở hệ thập phân như sau

$$A = \sum_{i=0}^{n-1} 2^i a_i$$

9.2.1. Biểu diễn dấu – độ lớn

Có một số phương pháp biểu diễn số nguyên âm và dương trong máy tính, tất cả các phương pháp này đều có đặc điểm chung là sử dụng bit có trọng số lớn nhất (bit đầu tiên bên trái) làm **bit dấu**. Nếu bit dấu bằng 0, số đó là số dương, nếu bit dấu bằng 1, số đó là số âm

Dạng biểu diễn đơn giản nhất trong trường hợp này đó là biểu diễn dấu – độ lớn. Với một từ n -bit, $n - 1$ bit bên phải sẽ là giá trị độ lớn của số

$$\begin{aligned}+18 &= 00010010 \\-18 &= 10010010 \text{ (dấu – độ lớn)}\end{aligned}$$

Như vậy, ta có công thức quy đổi như sau:

$$\text{Đầu – độ lớn} \quad A = \begin{cases} \sum_{i=0}^{n-2} 2^i a_i & \text{nếu } a_{n-1} = 0 \\ -\sum_{i=0}^{n-2} 2^i a_i & \text{nếu } a_{n-1} = 1 \end{cases} \quad 9.1$$

Với phương pháp biểu diễn này, ta thấy có một số nhược điểm như sau. Thứ nhất, khi thực hiện các phép toán cộng hoặc trừ, ta cần phải xét dấu của hai số và sự tương quan về độ lớn để xác định phép toán cần thực hiện. Phần 9.3, chúng tôi sẽ trình bày rõ ràng hơn. Một nhược điểm nữa là phép biểu diễn này có hai dạng biểu diễn số 0:

$$\begin{aligned}+0_{10} &= 00000000 \\-0_{10} &= 10000000 \text{ (dấu – độ lớn)}\end{aligned}$$

Điều này cũng gây khó khăn cho ALU khi kiểm tra trường hợp 0 (phép toán logic này được thực hiện khá thường xuyên trong máy tính)

Vì những nhược điểm này, biểu diễn dấu – độ lớn hiếm khi được dùng để biểu diễn số nguyên trong ALU. Thay vào đó, dạng biểu diễn bù hai được sử dụng nhiều hơn

9.2.2. Biểu diễn bù hai

Giống như dấu – độ lớn, biểu diễn bù hai sử dụng bit quan trọng nhất làm bit dấu, nhờ đó, ta dễ dàng kiểm tra một số là số dương hay âm. Tuy nhiên, phần độ lớn sẽ được biểu diễn như sau:

- Với số nguyên dương, độ lớn của số được chuyển từ thập phân sang nhị phân theo cách thông thường, điều này giống với cách biểu diễn số dương của dạng dấu - độ lớn.
- Với số âm, ta lấy bù một của từng bit trong số nguyên dương tương ứng, sau đó cộng kết quả thu được với 1 như cộng giữa hai số nguyên không dấu (bù 2).

Ví dụ về biểu diễn số 18 và -18 dạng bù hai 8-bit như sau:

$$(18)_{10} = (00010010)_{bù 2}$$

Với -18, đầu tiên ta lấy bù 1 (bit 1 đổi thành bit 0 và ngược lại) của 00010010 ta được 11101101, sau đó cộng với 1, ta có

$$\begin{array}{r} + 11101101 \\ \hline 11101110 \end{array}$$

Như vậy

$$(-18)_{10} = (11101110)_{bù 2}$$

Bảng 9.1 liệt kê các đặc điểm chính của dạng biểu diễn bù hai. Chúng tôi sẽ giải thích kỹ hơn các đặc điểm này trong phần này và phần tiếp theo.

Bảng 9.1 Các đặc điểm chính của dạng biểu diễn bù hai

Miền giá trị	Từ -2^{n-1} đến $2^{n-1} - 1$
Biểu diễn số 0	Một cách
Số âm	Với số âm, ta lấy bù một của từng bit trong số nguyên dương tương ứng, sau đó cộng kết quả thu được với 1 như cộng giữa hai số nguyên không dấu (bù 2)
Mở rộng chiều dài bit	Các bit thêm được điền vào bên trái với giá trị là giá trị của bit dấu
Luật tràn	Khi cộng hai số có cùng dấu (cùng âm hoặc cùng dương), nếu kết quả có dấu ngược lại thì phép toán bị tràn
Luật trừ	Để trừ A cho B , ta lấy bù hai của B rồi cộng với A
Miền giá trị	

Một số nguyên $n - bit$ biểu diễn bù hai. Nếu A là số dương, bit dấu $a_{n-1} = 0$. Các bit còn lại biểu diễn độ lớn của số giống như dạng biểu diễn dấu - độ lớn. Như vậy,

$$A = \sum_{i=0}^{n-2} 2^i a_i \quad \text{với } A \geq 0$$

Số 0 được biểu diễn như số dương với bit dấu bằng 0 và tất cả các bit còn lại đều là 0. Từ công thức trên ta có thể thấy miền giá trị của số nguyên dương trong dạng biểu diễn

này là từ 0 (tất cả các bit phần độ lớn bằng 0) đến $2^{n-1} - 1$ (tất cả các bit phần độ lớn bằng 1). Với các số lớn hơn, ta cần phải biểu diễn bằng nhiều bit hơn.

Với một số âm A ($A < 0$), bit dấu $a_{n-1} = 1$. Trọng số của bit dấu này được quy ước là -2^{n-1} . Với quy ước này, ta hoàn toàn có thể sử dụng quy tắc chương 8 để tính giá trị của một số từ dạng biểu diễn bù 2: tổng của giá trị bit nhân với trọng số của mỗi bit. Vậy, giá trị của số A được tính như sau:

$$\text{Bù hai} \quad A = -2^{n-1}a_{n-1} + \sum_{i=0}^{n-2} 2^i a_i \quad 9.2$$

Phương trình (9.2) định nghĩa biểu diễn bù hai cho cả hai số âm và số dương. Với $a_{n-1} = 0$, $-2^{n-1}a_{n-1} = 0$, phương trình xác định giá trị số nguyên không âm. Với $a_{n-1} = 1$, giá trị -2^{n-1} bị trừ khỏi tổng, ta sẽ có số nguyên âm tương ứng.

Bảng 9.2 so sánh hai dạng biểu diễn dấu – độ lớn và bù hai với các số nguyên 4 – bit. Mặc dù, biểu diễn bù hai tạo cảm giác khó hiểu và khó sử dụng hơn nhưng lại rất phù hợp cho các phép tính số học quan trọng nhất là các phép cộng hoặc trừ. Chính vì vậy, dạng biểu diễn này được sử dụng rộng rãi để biểu diễn số nguyên trong máy tính.

Bảng 9.2 Các dạng biểu diễn khác nhau đối với số nguyên 4 – bit

Hệ thập phân	Biểu diễn dấu – độ lớn	Biểu diễn bù hai	Biểu diễn biased
+8	—	—	1111
+7	0111	0111	1110
+6	0110	0110	1101
+5	0101	0101	1100
+4	0100	0100	1011
+3	0011	0011	1010
+2	0010	0010	1001
+1	0001	0001	1000
+0	0000	0000	0111
-0	1000	—	—
-1	1001	1111	0110
-2	1010	1110	0101
-3	1011	1101	0100

-4	1100	1100	0011
-5	1101	1011	0010
-6	1110	1010	0001
-7	1111	1001	0000
-8	—	1000	—

Để xác định giá trị mà một số bù hai biểu diễn ta có thể sử dụng hộp giá trị, trong đó giá trị đầu tiên từ bên phải của hộp 1 (tương ứng với 2^0) và giá trị các vị trí kế tiếp sang phía trái lần lượt tăng gấp đôi giá trị, cho đến vị trí cuối cùng bên trái mang giá trị âm (như trong hình 9.2a). Số lượng cột trong hộp bằng chính số bit của biểu diễn bù hai, nếu có nhiều bit hơn thì ta thêm vào các cột có giá trị dương, cùng quy luật như vậy.

Hình 9.2b và c minh họa việc sử dụng hộp giá trị để chuyển đổi từ số dạng bù hai sang dạng thập phân và ngược lại.

-128 64 32 16 8 4 2 1

a. Hộp giá trị của biểu diễn bù hai 8-bit

-128	64	32	16	8	4	2	1
1	0	0	0	0	0	1	1

$$-128 + 2 + 1 = -125$$

b. Chuyển đổi từ 10000011 sang số thập phân

-128	64	32	16	8	4	2	1
1	0	0	0	1	0	0	0

$$-120 \equiv -128 \quad +8$$

c. Chuyển đổi từ số -120 hệ thập phân sang nhị phân

Hình 9.2 Sử dụng hộp giá trị để chuyển đổi thập phân – nhị phân

9.2.3. Mở rộng phạm vi

Đôi khi, trong một số trường hợp ta cần phải biểu diễn một số nguyên n-bit dưới dạng m-bit ($m > n$). Việc tăng chiều dài bit được gọi là mở rộng phạm vi, vì việc tăng số bit để biểu diễn số cho phép phạm vi biểu diễn của dãy bit tăng lên.

Với dạng biểu diễn dấu – độ lớn, để mở rộng phạm vi, ta chỉ cần đặt bit dấu thành bit đầu tiên bên trái và điền thêm các bit 0.

$$+18 = 00010010 \text{ (Dấu - độ lớn, 8 bit)}$$

$$+18 = 0000000000010010 \text{ (Dấu - độ lớn, 16 bit)}$$

$$-18 = 10010010 \text{ (Dấu - độ lớn, 8 bit)}$$

$$-18 = 1000000000010010 \text{ (Dấu - độ lớn, 16 bit)}$$

Với dạng bù hai, quy tắc trên không còn đúng, ta có ví dụ sau

$$+18 = 00010010 \text{ (Bù hai, 8 bit)}$$

$$+18 = 0000000000010010 \text{ (Bù hai, 16 bit)}$$

$$-18 = 11101110 \text{ (Bù hai, 8 bit)}$$

$$-32658 = 1000000011101110 \text{ (Bù hai, 16 bit)}$$

Vậy, với dạng bù hai, để mở rộng phạm vi, ta thêm vào bên trái của số đó các bit giống với bit dấu. Quy tắc này ta có thể gọi là quy tắc mở rộng dấu.

$$-18 = 11101110 \text{ (Bù hai, 8 bit)}$$

$$-18 = 111111111101110 \text{ (Bù hai, 16 bit)}$$

Để hiểu được quy tắc này, ta xét một chuỗi n-bit: $a_{n-1}a_{n-2} \dots a_1a_0$ dạng mã bù hai, vậy ta có giá trị A của số đó như sau:

$$A = -2^{n-1}a_{n-1} + \sum_{i=0}^{n-2} 2^i a_i$$

Khi mở rộng phạm vi chuỗi bit thành m-bit, ta có

$$A = -2^{m-1}a_{m-1} + \sum_{i=0}^{m-2} 2^i a_i$$

Nếu A là số dương, các bit thêm vào là các bit 0 nên không làm thay đổi gì tổng trên.
Nếu A là số âm, ta có hai tổng sau phải bằng nhau:

$$-2^{m-1} + \sum_{i=0}^{m-2} 2^i a_i = -2^{n-1} + \sum_{i=0}^{n-2} 2^i a_i$$

$$-2^{m-1} + \sum_{i=n-1}^{m-2} 2^i a_i = -2^{n-1}$$

$$2^{n-1} + \sum_{i=n-1}^{m-2} 2^i a_i = 2^{m-1}$$

$$1 + \sum_{i=0}^{n-2} 2^i + \sum_{i=n-1}^{m-2} 2^i a_i = 1 + \sum_{i=0}^{m-2} 2^i$$

$$\sum_{i=n-1}^{m-2} 2^i a_i = \sum_{i=n-1}^{m-2} 2^i$$

Như vậy, rõ ràng $a_{m-2} = \dots = a_{n-2} = a_{n-1} = 1$

9.3. CÁC PHÉP TOÁN SỐ HỌC VỚI SỐ NGUYÊN

Trong phần này, chúng tôi trình bày chủ yếu các phép toán số học với số nguyên biểu diễn dạng bù hai.

9.3.1. Phép phủ định

Trong biểu diễn dấu – độ lớn, quy tắc thực hiện phép phủ định khá đơn giản: đổi bit dấu. Quy tắc phép phủ định của số bù hai được thực hiện như sau:

1. Lấy bù của mỗi bit (bao gồm cả bit dấu) bằng cách đảo bit 0 thành bit 1 và ngược lại.
2. Cộng kết quả với 1 giống như việc cộng hai số nguyên thông thường.

Quá trình thực hiện hai bước trên còn được gọi là **phép toán bù hai** hoặc thực hiện bù hai của một số nguyên

$+18$	=	00010010 (bù hai)
Lấy bù	=	$\begin{array}{r} +11101101 \\ \hline 1 \end{array}$ $11101110 = -18$

Rõ ràng, phủ định của phủ định của một số là chính nó, ta có:

-18	=	11101110 (bù hai)
Lấy bù	=	$\begin{array}{r} +00010001 \\ \hline 1 \end{array}$ $00010010 = +18$

Chúng ta có thể chứng minh điều này bằng cách sử dụng Phương trình (9.2) với một số nguyên A biểu diễn dưới dạng mã bù hai n-bit như sau:

$$A = -2^{n-1}a_{n-1} + \sum_{i=0}^{n-2} 2^i a_i$$

Với quy tắc phủ định ở trên, ta được một dãy kết quả n-bit có giá trị B như sau

$$B = -2^{n-1}\bar{a}_{n-1} + \sum_{i=0}^{n-2} 2^i \bar{a}_i + 1$$

Điều ta cần chứng minh là $A = -B$, nghĩa là $A + B = 0$. Vậy

$$\begin{aligned}
 A + B &= -(a_{n-1} + \bar{a}_{n-1})2^{n-1} + 1 + \sum_{i=0}^{n-2} 2^i(a_i + \bar{a}_i) \\
 &= -2^{n-1} + 1 + (\sum_{i=0}^{n-2} 2^i) \\
 &= -2^{n-1} + 1 + (2^{n-1} - 1) = 0
 \end{aligned}$$

Tuy nhiên, phép phủ định đôi với các số biểu diễn bù hai có hai trường hợp đặc biệt cần xem xét. Trước tiên, với $A = 0$, biểu diễn 8-bit và thực hiện phép phủ định như sau:

0	=	00000000 (bù hai)
Lấy bù	=	$ \begin{array}{r} + 11111111 \\ \hline 1 \end{array} $
		100000000 = 0

Ta thấy bên trái dãy bit kết quả (gồm 9 bit) dư ra một bit 1, nếu bỏ qua bit này (chỉ lấy 8 bit sau) thì ta được kết quả phép phủ định của 0 là 0 là hợp lý.

Trường hợp đặc biệt thứ hai là nếu ta thực hiện phép phủ định của một số có bit đầu tiên là 1 còn các bit sau là 0 như dưới đây:

-128	=	10000000 (bù hai)
Lấy bù	=	$ \begin{array}{r} + 01111111 \\ \hline 1 \end{array} $
		10000000 = -128

Phủ định của -128 lại bằng chính nó, như vậy phép phủ định trong trường hợp này là sai. Nguyên nhân của điều này là do -128 được biểu diễn dưới dạng dãy 8-bit. Mà như phần trên đã trình bày, miền giá trị của biểu diễn bù 2 n-bit là từ -2^{n-1} đến $2^{n-1} - 1$. Nghĩa là với dãy 8-bit ta biểu diễn được các số từ $-2^{8-1} = -128$ đến $2^{8-1} - 1 = 127$. Trong khi đó, phủ định của -128 là 128 nằm ngoài phạm vi biểu diễn của mã bù hai 8-bit nên đã gây ra trường hợp sai sót trên. Giải pháp để giải quyết trường hợp này đó là mở rộng phạm vi biểu diễn bằng cách tăng kích thước dãy bit.

9.3.2. Phép toán cộng và phép toán trừ

Phép cộng hai số bù hai được minh họa như trong Hình 9.3. Việc thực hiện phép cộng bình thường giống như cộng hai số nguyên không dấu. Một ví dụ đầu tiên là các trường hợp cộng thành công, kết quả thu được vẫn là các số dạng bù hai. Chú ý rằng trong một số trường hợp, kết quả phép toán có một bit thừa ra (bit được bôi xanh), bit này được bỏ đi mà không làm ảnh hưởng đến kết quả.

Với phép toán cộng, kết quả có thể lớn hơn phạm vi biểu diễn của dãy bit, trường hợp này ta gọi là **Tràn**. Khi bị tràn, ALU phải báo tràn để bộ xử lý không sử dụng kết quả này. Vậy để phát hiện tràn, người ta sử dụng quy tắc sau:

$ \begin{array}{r} 1001 = -7 \\ +\underline{0101} = 5 \\ 1110 = -2 \\ \hline \end{array} $ <p>(a) $(-7) + (+5)$</p>	$ \begin{array}{r} 1100 = -4 \\ +\underline{0100} = 4 \\ \hline 10000 = 0 \end{array} $ <p>(b) $(-4) + (+4)$</p>
$ \begin{array}{r} 0011 = 3 \\ +\underline{0100} = 4 \\ 0111 = 7 \\ \hline \end{array} $ <p>(c) $(+3) + (+4)$</p>	$ \begin{array}{r} 1100 = -4 \\ +\underline{1111} = -1 \\ \hline 11011 = -5 \end{array} $ <p>(d) $(-4) + (-1)$</p>
$ \begin{array}{r} 0101 = 5 \\ +\underline{0100} = 4 \\ \hline 1001 = \text{Tràn} \end{array} $ <p>(e) $(+5) + (+4)$</p>	$ \begin{array}{r} 1001 = -7 \\ +\underline{1010} = -6 \\ \hline 10011 = \text{Tràn} \end{array} $ <p>(f) $(-7) + (-6)$</p>

Hình 9.3 Phép cộng hai số biểu diễn dạng Bù hai

QUY TẮC TRÀN: Nếu cộng hai số cùng dấu (cùng dương hoặc cùng âm), kết quả sẽ bị tràn nếu và chỉ nếu nó có dấu ngược lại.

Hình 9.3e và f là các trường hợp tràn, trong trường hợp ở hình f, bit tô màu là bit được bỏ đi, bit dấu là bit 0, ngược với bit dấu của hai toán hạng.

Phép trừ được thực hiện theo quy tắc sau:

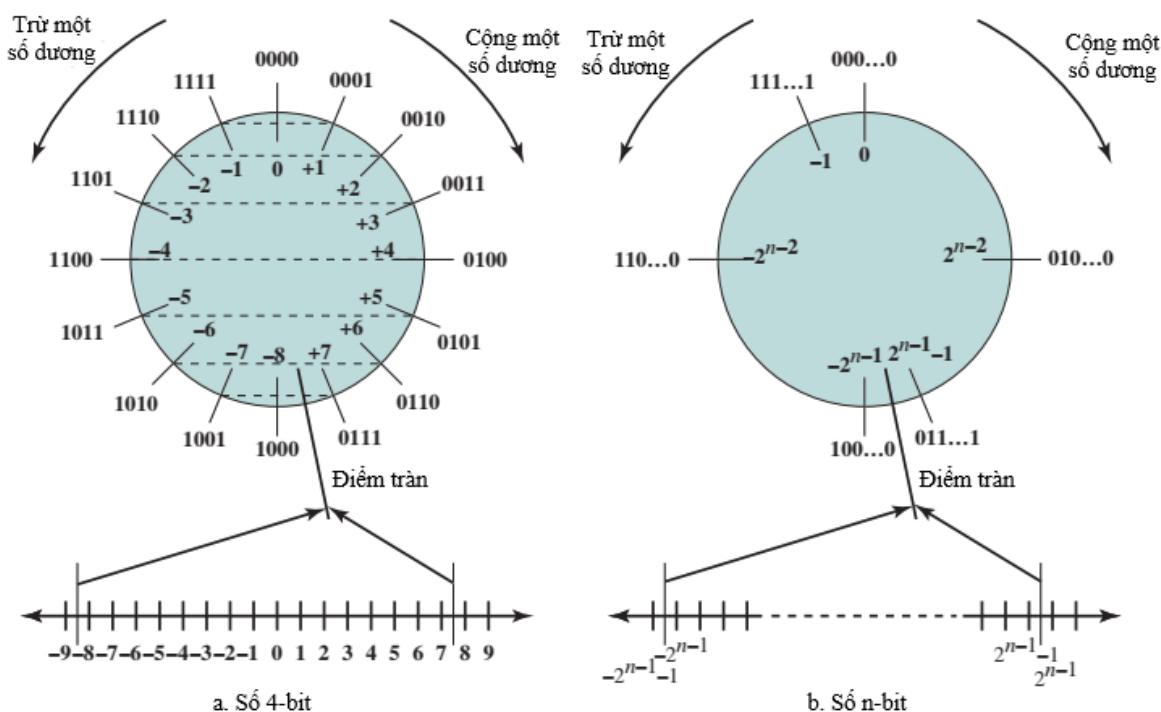
QUY TẮC TRỪ: Để thực hiện phép trừ giữa hai số, ta lấy bù hai (phủ định) của số trừ rồi cộng với số bị trừ.

Như vậy, phép trừ cũng được thực hiện thông qua phép cộng. Hình 9.4 gồm các ví dụ về phép trừ trong đó có hai trường hợp tràn

$ \begin{array}{r} 0010 = 2 \\ +\underline{1001} = -7 \\ \hline 1011 = -5 \end{array} $ <p>(a) $M = 2 = 0010$ $S = 7 = 0111$ $-S = 1001$</p>	$ \begin{array}{r} 0101 = 5 \\ +\underline{1110} = -2 \\ \hline 10011 = 3 \end{array} $ <p>(b) $M = 5 = 0101$ $S = 2 = 0010$ $-S = 1110$</p>
$ \begin{array}{r} 1011 = -5 \\ +\underline{1110} = -2 \\ \hline 11001 = -7 \end{array} $ <p>(c) $M = -5 = 1011$ $S = 2 = 0010$ $-S = 1110$</p>	$ \begin{array}{r} 0101 = 5 \\ +\underline{0010} = 2 \\ \hline 0111 = 7 \end{array} $ <p>(d) $M = 5 = 0101$ $S = -2 = 1110$ $-S = 0010$</p>
$ \begin{array}{r} 0111 = 7 \\ +\underline{0111} = 7 \\ \hline 1110 = \text{Tràn} \end{array} $ <p>(e) $M = 7 = 0111$ $S = -7 = 1001$ $-S = 0111$</p>	$ \begin{array}{r} 1010 = -6 \\ +\underline{1100} = -4 \\ \hline 10110 = \text{Tràn} \end{array} $ <p>(f) $M = -6 = 1010$ $S = 4 = 0100$ $-S = 1100$</p>

Hình 9.4 Phép trừ hai số biểu diễn Bù hai ($M - S$)

Hình 9.5 mô tả rõ hơn các phép cộng và trừ hai số bù hai qua đồ thị hình tròn. Vòng tròn ở trên được chia thành các đoạn tương ứng với số giá trị mà một mã có thể biểu diễn được. Ví dụ mã bù hai 4-bit biểu diễn được 16 giá trị, như vậy, vòng tròn được chia thành 16 đoạn và điền các giá trị tương ứng như trong hình 9.5a. Các số đối diện nhau theo chiều ngang (nối với nhau bằng đường nét đứt) sẽ tương ứng là các số bù hai của nhau. Để thực hiện phép cộng của một số với số k dương (hoặc trừ số k âm), ta bắt đầu từ vị trí số đó trên vòng tròn, di chuyển thêm k vị trí theo chiều kim đồng hồ. Ngược lại, để thực hiện phép trừ một số đi một giá trị k dương (hoặc cộng k âm), ta di chuyển từ số đó theo chiều ngược lại (ngược chiều kim đồng hồ) k vị trí. Nếu quá trình thực hiện phép toán đi qua điểm tràn thì phép toán này bị tràn. Tất cả các ví dụ trong Hình 9.3 và 9.4 đều có thể dễ dàng kiểm chứng thông qua Hình 9.5.



Hình 9.5 Biểu diễn hình học các số nguyên Bù hai

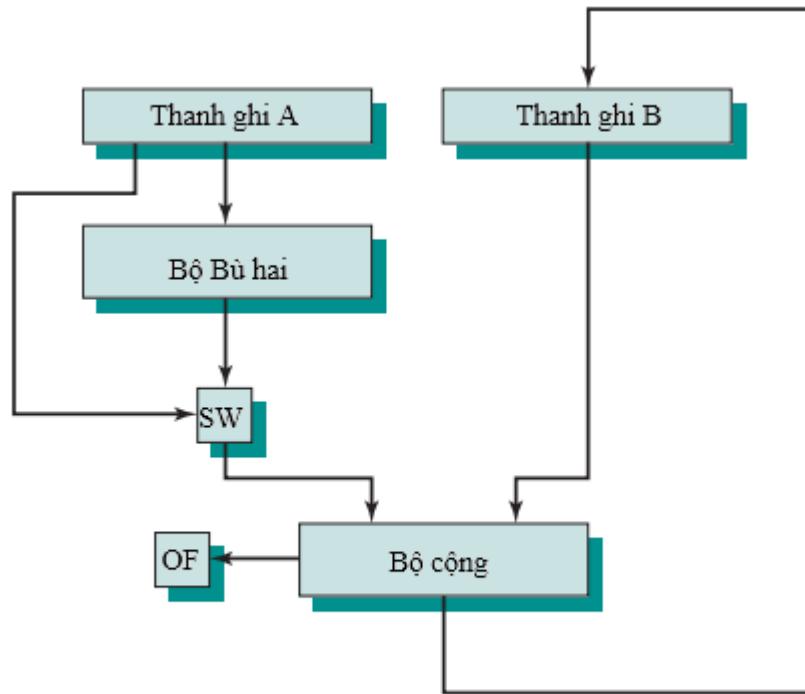
Hình 9.6 minh họa sơ đồ khói của phần cứng thực hiện phép cộng và phép trừ. Ta có một bộ cộng với hai đầu vào và hai đầu ra là tổng hai số đó và một cờ tràn. Bộ cộng hai số như hai số nguyên không dấu. Với phép cộng, các số hạng được lưu trữ ở thanh ghi A và thanh ghi B. Kết quả được ghi lại vào một trong hai thanh ghi này hoặc một thanh ghi thứ ba (trong trường hợp này, kết quả lưu vào thanh ghi A). Cờ tràn được thiết lập trên một bit, với 0 là không tràn và 1 là tràn. Với phép trừ, số bị trừ ghi vào thanh ghi A, số trừ ghi vào thanh ghi B. Số trừ được đưa đi qua bộ Bù hai sau đó được đưa vào bộ cộng. Ngoài ra, tín hiệu điều khiển sẽ được đưa qua bộ SW (bộ chọn phép cộng hay phép trừ) để cho phép tín hiệu từ thanh ghi B hoặc từ bộ Bù hai đi vào bộ cộng.

9.3.1. Phép nhân

So với phép cộng hoặc phép trừ, phép nhân tương đối phức tạp hơn cho dù thực hiện bằng phần cứng hay phần mềm. Có rất nhiều các thuật toán khác nhau được sử dụng trong

các hệ máy tính khác nhau. Ở phần này, chúng ta sẽ bắt đầu với việc tìm hiểu phép nhân giữa hai số nguyên không dấu và sau đó là phương pháp phổ biến nhất để thực hiện phép nhân giữa hai số biểu diễn dưới dạng mã bù hai.

SỐ NGUYÊN KHÔNG DẤU Hình 9.7 là các bước thực hiện phép nhân hai số không dấu mà ta hay làm bình thường trên giấy. Trong đó, có một số điểm ta có thể thấy được như sau:



SW: Bộ chuyển mạch (cho phép thực hiện phép cộng hoặc phép trừ)
OF: Cờ tràn

Hình 9.6 Sơ đồ khối phần cứng Bộ cộng và bộ trừ

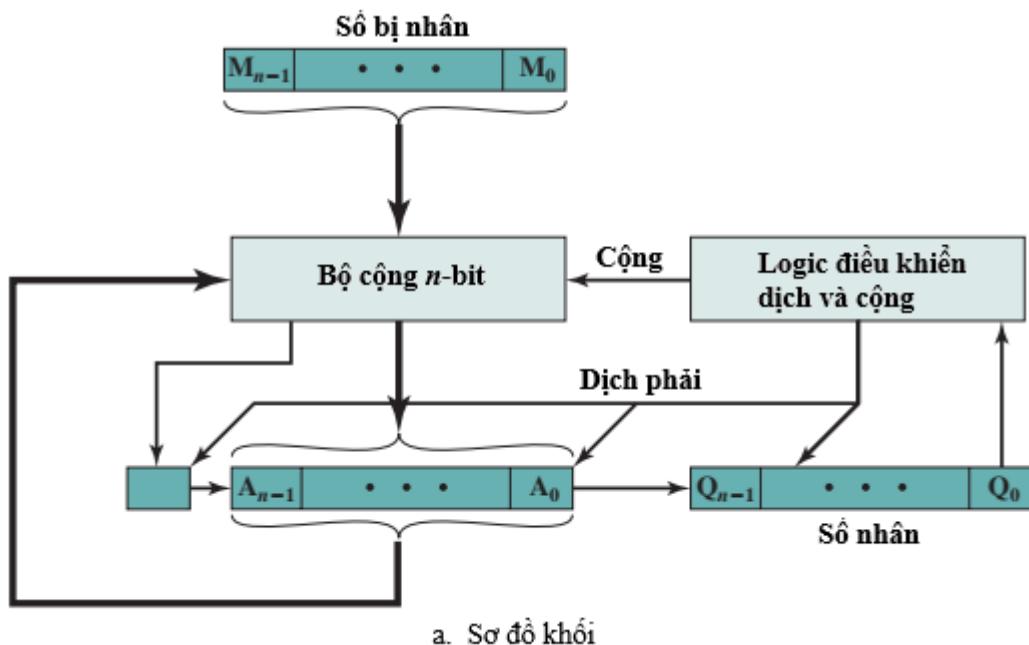
$$\begin{array}{r}
 1011 \\
 \times 1101 \\
 \hline
 1011 \\
 0000 \\
 1011 \\
 \hline
 1011 \\
 \hline
 10001111
 \end{array}
 \left.
 \begin{array}{l}
 \text{Số bị nhân (11)} \\
 \text{Số nhân (13)} \\
 \\ \\
 \text{Tích thành phần} \\
 \\ \\
 \text{Tích (143)}
 \end{array}
 \right\}$$

Hình 9.7 Nhân hai số nguyên nhị phân không dấu

1. Khi thực hiện phép nhân, ta cần xác định các tích thành phần của mỗi bit trong số nhân (số thứ hai trong phép nhân). Sau đó cộng các tích thành phần này ta được tích cần tìm.
 2. Việc xác định các tích thành phần rất dễ. Nếu bit của số nhân là 0, tích thành phần sẽ là 0. Nếu bit của số nhân là 1, tích thành phần sẽ là chính số bị nhân (số thứ nhất trong phép nhân).

3. Phép nhân giữa hai số nguyên nhị phân n -bit ta được kết quả là một số nguyên có kích thước $2n$ bit.

Như vậy, ta sẽ phải thực hiện một số điều để máy tính hóa phép nhân trên. Thứ nhất, ta có thể thực hiện việc cộng luôn các tích thành phần thay vì để đến lúc cuối. Việc này sẽ hạn chế việc phải sử dụng thêm các thanh ghi để lưu trữ từng tích thành phần. Thứ hai, chúng ta có thể tiết kiệm một chút thời gian khi xác định các tích thành phần. Với bit 1, ta cần thực hiện một hoạt động dịch và một hoạt động cộng, nhưng với bit 0, ta chỉ cần thực hiện một hoạt động dịch.



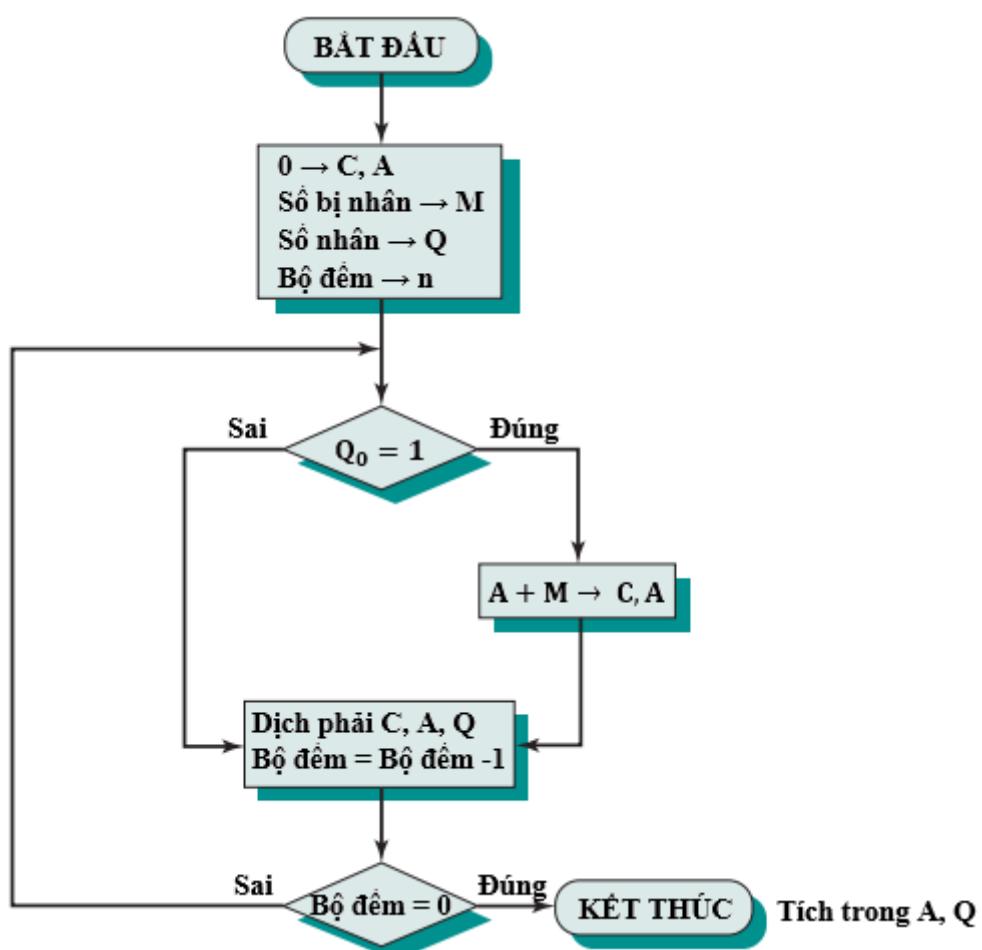
C	A	Q	M	Giá trị khởi tạo	
0	0000	1101	1011		
0	1011	1101	1011	Cộng	} Bước 1
	0101	1110	1011	Dịch	
0	0010	1111	1011	Dịch	} Bước 2
	1101	1111	1011	Cộng	
0	0110	1111	1011	Dịch	} Bước 3
	0001	1111	1011	Cộng	
1	1000	1111	1011	Dịch	} Bước 4
	0				

b. Ví dụ trong Hình 9.7 (Tích trong thanh ghi A và Q)

Hình 9.8 Phần cứng thực hiện phép nhân hai số nhị phân không dấu

Hình 9.8a là sơ đồ khói bộ nhân trong trường hợp trên. Số nhân và số bị nhân³ được lần lượt tải vào hai thanh ghi Q và M. Thanh ghi A được thiết lập bằng 0, thanh ghi 1 bit C để lưu trữ bit thừa ra do kết quả phép cộng vào thanh ghi A, ban đầu C cũng được thiết lập bằng 0.

Hoạt động của bộ nhân được thực hiện như sau: Mỗi một bước, logic điều khiển đọc giá trị bit Q_0 của số nhân. Nếu Q_0 bằng 1, số bị nhân được cộng vào thanh ghi A, bit C được sử dụng để lưu trữ giá trị tràn. Sau đó, dịch tất cả bit của thanh ghi C, A và Q sang bên phải một bit. Trong trường hợp $Q_0 = 0$, logic điều khiển sẽ không điều khiển việc thực hiện phép cộng và chỉ thực hiện hoạt động dịch sang phải một bit. Khi dịch phải, bit C sẽ thành bit A_{n-1} , A_0 thành Q_{n-1} còn Q_1 thành Q_0 , giá trị Q_0 cũ sẽ bỏ đi. Quá trình này sẽ lặp đi lặp lại với mỗi bit của số nhân ban đầu. Kết quả $2n$ -bit được lưu trữ trong thanh ghi A và Q. Hình 9.8b minh họa giá trị các thanh ghi C, A, Q, M trong mỗi bước. Hình 9.9 là sơ đồ thuật toán phép nhân vừa trình bày ở trên.



Hình 9.9 Sơ đồ thuật toán phép nhân hai số nhị phân không dấu

³ Với phép nhân, vai trò của hai toán hạng là ngang nhau, số nào cũng có thể là số nhân hoặc số bị nhân. Ở đây chúng tôi dùng thuật ngữ số nhân và số bị nhân để dễ dàng phân biệt chúng. Số bị nhân là số viết trên, số nhân là số viết dưới. Khi thực hiện phép nhân hai số, ta sẽ dùng từng bit của số nhân nhân với số bị nhân để xác định các tích thành phần

NHÂN HAI SỐ BÙ HAI Như ta đã biết, phép cộng và trừ hai số bù hai có thể được thực hiện giống như phép cộng và trừ hai số nguyên không dấu. Ví dụ

$$\begin{array}{r} 1001 \\ +0011 \\ \hline 1100 \end{array}$$

Nếu xem các số này là các số nguyên không dấu, ta có phép cộng $9(1001)$ và $3(0011)$ bằng $12(1100)$. Nếu xem đây là các số dạng bù hai, ta có $-7(1001) + 3(0011) = -4(1100)$

Tuy nhiên, điều này không đúng với phép nhân. Xét ví dụ trong Hình 9.7: nhân $11(1011)$ với $13(1101)$ ta được $143(10001111)$. Nếu xem các số đó là số biểu diễn dạng bù hai, ta có $-5(1011) \times (-3)(1101) = -113(10001111)$. Như vậy, thuật toán trên không thể sử dụng được với phép nhân hai số bù hai âm. Thực tế, phương pháp này không sử dụng được với một trong hai số là số bù hai âm.

1011 × 1101 ————— 00001011 $1011 \times 1 \times 2^0$ 00000000 $1011 \times 0 \times 2^1$ 00101100 $1011 \times 1 \times 2^2$ 01011000 $1011 \times 1 \times 2^3$ ————— 10001111
--

Hình 9.10 Nhân hai số nguyên không dấu 4-bit, ta được kết quả 8-bit

Để hiểu rõ hơn ta xét ví dụ trong Hình 9.11, ở trường hợp a là phép nhân giữa hai số nguyên không dấu, ta có thể thực hiện phép nhân này theo một cách khác như sau:

$$\begin{aligned} 1001 \times 0011 &= 1001 \times (2^3 \times 0 + 2^2 \times 0 + 2^1 \times 1 + 2^0 \times 1) \\ &= 1011 \times 2^1 + 1011 \times 2^0 \\ &= 10110 + 1011 \end{aligned}$$

Như vậy, các tích thành phần của phép nhân này là 10110 , 1011 . Nhưng do vấn đề biểu diễn số trong máy tính và thực hiện phép cộng, với các số ban đầu ta biểu diễn dạng 4-bit, để biểu diễn các tích thành phần này ra phải mở rộng phạm vi, tăng kích thước dãy bit thành 8 bit. Như vậy, các tích thành phần trường hợp a được biểu diễn dạng 8-bit bằng cách thêm các số 0 đằng trước. Tuy nhiên, khi thực hiện phép nhân giữa hai số bù hai có *số bị nhân âm* (trường hợp 9.11b), các tích thành phần biểu diễn dạng 8-bit phải là: 11110110 , 11111011 (thêm bit dấu vào đằng trước dãy bit), phép nhân mới cho ta kết quả chính xác.

$\begin{array}{r} 1001 \quad (9) \\ \times 0011 \quad (3) \\ \hline 00001001 \quad 1001 \times 2^0 \\ 00010010 \quad 1001 \times 2^1 \\ \hline 00011011 \quad (27) \end{array}$	$\begin{array}{r} 1001 \quad (-7) \\ \times 0011 \quad (3) \\ \hline 11111001 \quad (-7) \times 2^0 = (-7) \\ 11110010 \quad (-7) \times 2^1 = (-14) \\ \hline 11101011 \quad (-21) \end{array}$
---	--

a. Số nguyên không dấu

b. Số bù hai

Hình 9.11 So sánh phép nhân giữa các số nguyên không dấu và số bù hai

Trong trường hợp *số nhân là số âm*, xét ví dụ: 0101×1101 , nếu sử dụng phương pháp trên ta có:

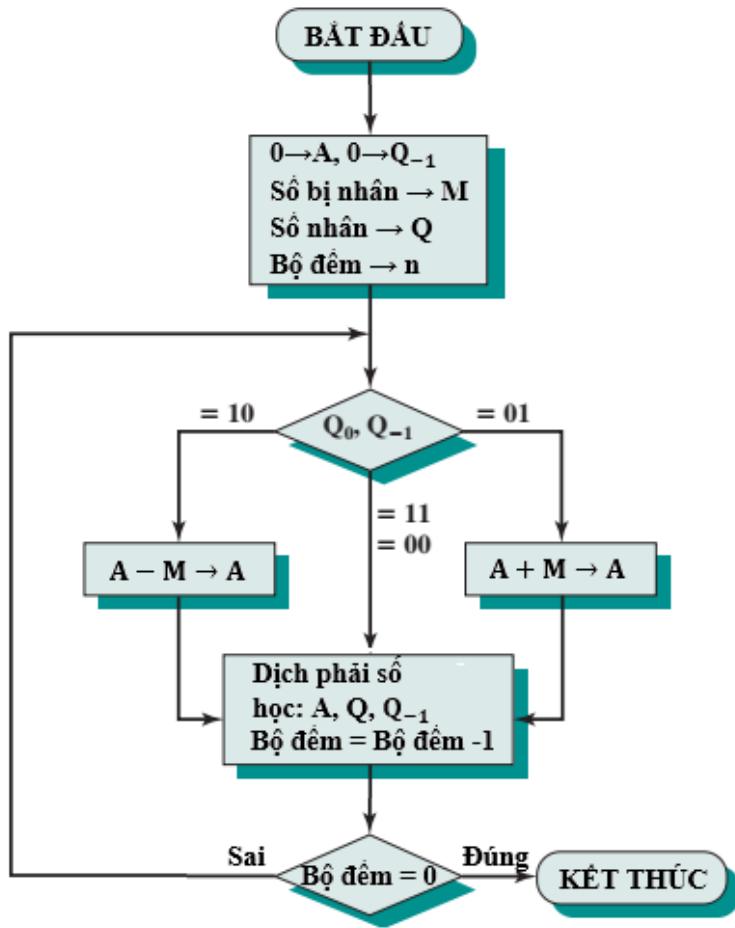
$$\begin{aligned}0101 \times 1101 &= 0101 \times (2^3 \times 1 + 2^2 \times 1 + 2^1 \times 0 + 2^0 \times 1) \\&= 0101 \times 2^3 + 0101 \times 2^2 + 0101 \times 2^0 \\&= 0101000 + 010100 + 0101\end{aligned}$$

Trong khi, thực tế với số bù hai phải như sau:

$$\begin{aligned}0101 \times 1101 &= 0101 \times (-2^3 \times 1 + 2^2 \times 1 + 2^1 \times 0 + 2^0 \times 1) \\&= -0101 \times 2^3 + 0101 \times 2^2 + 0101 \times 2^0 \\&= -0101000 + 010100 + 0101\end{aligned}$$

Như vậy, phương pháp trên không phù hợp với biểu diễn số bù hai.

Người ta đưa ra rất nhiều phương pháp để giải quyết vấn đề trên. Một cách đó là chuyển cả số nhân và số bị nhân sang thành số dương, thực hiện phép nhân sau đó thực hiện bù hai của kết quả trong trường hợp dấu của hai số trên khác nhau. Ngoài ra, có một số thuật toán không cần thực hiện việc chuyển đổi trên, thuật toán Booth's là một ví dụ. Thuật toán này còn có một ưu điểm là đẩy nhanh tốc độ thực hiện phép nhân. Chúng tôi xin trình bày thuật toán này như sau (Hình 9.12):



Hình 9.12 Thuật toán Booth's cho phép nhân hai số bù hai

Thuật toán Booth's Số nhân và số bị nhân lần lượt được đặt vào các thanh ghi Q và M . Thanh ghi 1-bit Q_{-1} đặt vào phía bên phải thanh ghi Q , cạnh bit Q_0 . Kết quả đặt tại thanh ghi A và Q . A và Q_{-1} được khởi tạo giá trị bằng 0. Ở thuật toán trên, logic điều khiển đọc từng bit của số nhân. Với thuật toán này, nó sẽ đọc đồng thời hai bit Q_0 và Q_{-1} . Nếu hai bit giống nhau (cùng 0 hoặc cùng 1), dịch tất cả các bit của ba thanh ghi A , Q và Q_{-1} sang bên phải 1 bit. Nếu hai bit là 0-1, số bị nhân sẽ được cộng vào thanh ghi A , ngược lại, nếu là 1-0, số bị nhân sẽ bị trừ khỏi A (nghĩa là $A - M$). Sau đó, cả ba thanh ghi A , Q , Q_{-1} dịch phải một bit. Khi thực hiện thao tác dịch, bit ngoài cùng bên trái của A (bit A_{n-1}) không chỉ được dịch vào A_{n-2} mà vẫn lưu lại ở A_{n-1} . Điều này để bảo lưu bit dấu của A (do là số bù hai) và được gọi là **dịch số học**.

A	Q	Q_{-1}	M	Khởi tạo giá trị	
0000	0011	0	0111		
1001	0011	0	0111	$A - M \rightarrow A$	Bước 1
1100	1001	1	0111	Dịch	
1110	0100	1	0111	Dịch	Bước 2
0101	0100	1	0111	$A + M \rightarrow A$	Bước 3
0010	1010	0	0111	Dịch	
0001	0101	0	0111	Dịch	Bước 4

Hình 9.13 Ví dụ Thuật toán Booth's (7×3)

Hình 9.13 là chuỗi các sự kiện khi dùng thuật toán Booth's thực hiện phép nhân 7 và 3. Các ví dụ khác với một trong hai số là số âm hoặc cả hai số là số âm cũng được minh họa trong Hình 9.14 và đều cho kết quả chính xác.

$ \begin{array}{r} 0111 \\ \times 0011 \\ \hline 11111001 \\ 0000000 \\ 000111 \\ \hline 00010101 \end{array} (0) $	$ \begin{array}{r} 0111 \\ \times 1101 \\ \hline 11111001 \\ 0000111 \\ 111001 \\ \hline 11101011 \end{array} (0) $
(a) $(7) \times (3) = (21)$	(b) $(7) \times (-3) = (-21)$
$ \begin{array}{r} 1001 \\ \times 0011 \\ \hline 00000111 \\ 0000000 \\ 111001 \\ \hline 11101011 \end{array} (0) $	$ \begin{array}{r} 1001 \\ \times 1101 \\ \hline 00000111 \\ 1111001 \\ 000111 \\ \hline 00010101 \end{array} (0) $
(c) $(-7) \times (3) = (-21)$	(d) $(-7) \times (-3) = (21)$

Hình 9.14 Ví dụ Thuật toán Booth's

Vậy, cơ sở lý thuyết của thuật toán Booth's như thế nào? Đầu tiên, xét trường hợp số nhân là số dương gồm một dãy các bit 1 liền nhau, còn lại và bit 0 (ví dụ: 00011110). Phép nhân một số M với số nhân trên có thể được viết như sau:

$$\begin{aligned}
 M \times (00011110) &= M \times (2^4 + 2^3 + 2^2 + 2^1) \\
 &= M \times (16 + 8 + 4 + 2) \\
 &= M \times 30
 \end{aligned}$$

Mặt khác, tổng một chuỗi lũy thừa hai có thể được viết như sau

$$2^n + 2^{n-1} + \dots + 2^{n-K} = 2^{n+1} - 2^{n-K}$$

.3

Nghĩa là,

$$\begin{aligned} M \times (00011110) &= M \times (2^5 - 2^1) \\ &= M \times (32 - 2) \\ &= M \times 30 \end{aligned}$$

Như vậy, $M \times 00011110 = M \times 2^5 - M \times 2^1$ nghĩa là với một dãy bit 1 liền nhau, thuật toán Booth's sẽ thực hiện một phép trừ ($M \times 2^1$) khi gặp bit 1 đầu tiên (1-0) và một phép cộng ($M \times 2^5$) khi kết thúc dãy bit (0-1).

Ta cũng có thể áp dụng điều này với một bit 1 đơn lẻ như trong ví dụ sau:

$$\begin{aligned} M \times (01111010) &= M \times (2^6 + 2^5 + 2^4 + 2^3 + 2^1) \\ &= M \times (2^7 - 2^3 + 2^2 - 2^1) \end{aligned}$$

Với trường hợp số nhân là số âm, giả sử được biểu diễn dạng bù hai như sau:

$$X = \{1x_{n-2}x_{n-3} \dots x_1x_0\}$$

Trong đó, x_0, x_1, \dots, x_{n-2} là các bit của X , bit dấu là 1. Như vậy giá trị của X là:

$$X = -2^{n-1} + (x_{n-2} \times 2^{n-2}) + (x_{n-3} \times 2^{n-3}) + \dots + (x_1 \times 2^1) + (x_0 \times 2^0)$$

.4

Do bit ngoài cùng bên trái của X là 1. Giả sử bit 0 đầu tiên tính từ bên trái của X ở vị trí thứ K. Nghĩa là

$$X = \{11 \dots 10x_{k-1}x_{k-2} \dots x_1x_0\}$$

.5

Như vậy, giá trị của X là

$$X = -2^{n-1} + 2^{n-2} + \dots + 2^{k+1} + (x_{k-1} \times 2^{k-1}) + \dots + (x_0 \times 2^0)$$

.6

Từ Phương trình (9.3), ta có

$$2^{n-2} + 2^{n-3} + \dots + 2^{k+1} = 2^{n-1} - 2^{k+1}$$

Như vậy

$$-2^{n-1} + 2^{n-2} + 2^{n-3} + \dots + 2^{k+1} = -2^{k+1}$$

.7

Thé vào Phương trình (9.6), ta có

$$X = -2^{k+1} + (x_{k-1} \times 2^{k-1}) + \cdots + (x_0 \times 2^0)$$

.8

Ta có phép nhân hai số

$$M \times X = M \times (-2^{k+1}) + M \times [(x_{k-1} \times 2^{k-1}) + \cdots + (x_0 \times 2^0)]$$

.9

Quay trở lại thuật toán của Booth. Với dạng của X trong Phương trình 9.5, tất cả các bit từ x_0 cho x_k đều được xử lý bình thường như phần trên. Đến bit x_{k+1} có giá trị bằng 1 và $x_k = 0$, ta được một cặp 1-0, do đó, thao tác cần thực hiện là trừ đi $M \times 2^{k-1}$ (Phương trình 9.9), phù hợp với thuật toán Booth's.

Xét ví dụ nhân một số M với (-6) biểu diễn bù hai dạng 8-bit: 11111010. Từ Phương trình 9.4, ta có

$$-6 = -2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^1$$

Vậy

$$M \times 11111010 = M \times (-2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^1)$$

Sử dụng phương trình 9.7 ta có

$$M \times 11111010 = M \times (-2^3 + 2^1)$$

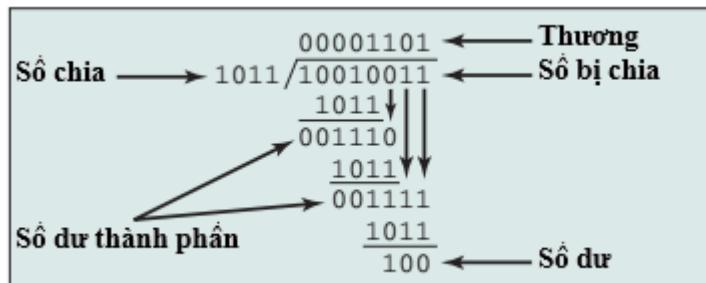
Cuối cùng, áp dụng lý thuyết như phần trên đã trình bày, ta có

$$M \times 11111010 = M \times (-2^3 + 2^2 - 2^1)$$

Như vậy, thuật toán Booth's hoàn toàn phù hợp với phép nhân hai số bù hai.Thêm vào đó, ta có thể thấy số lượng phép toán cộng và trừ của thuật toán Booth's ít hơn so với thuật toán trước đó

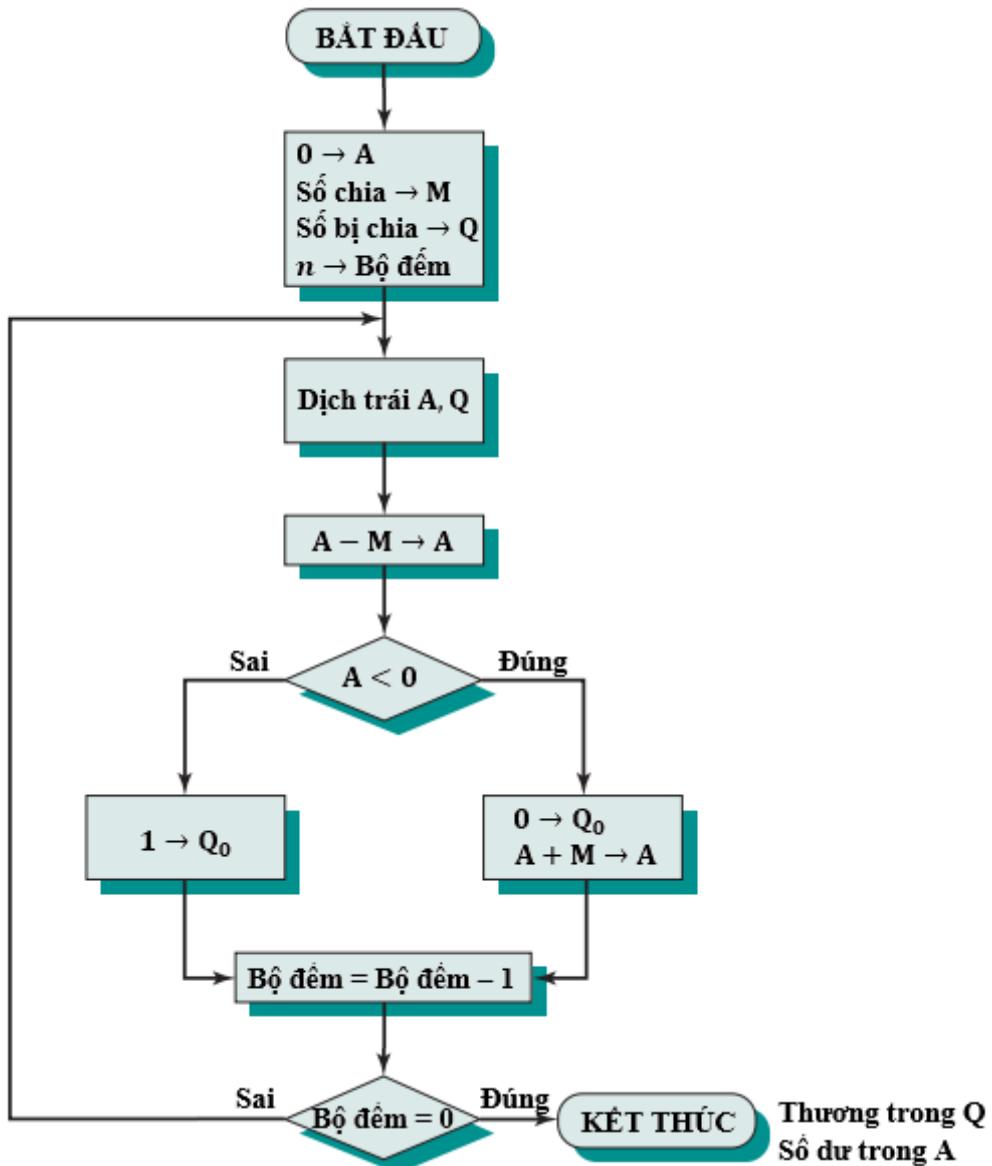
9.3.2. Phép chia

Phép chia phức tạp hơn một chút so với phép nhân tuy nhiên cũng có cùng một nguyên lý chung. Như trên, phép chia cũng dựa trên phương pháp tính toán trên giấy với các phép dịch, phép cộng và phép trừ.



Hình 9.15 Ví dụ phép chia hai số nguyên nhị phân không dấu

Hình 9.15 ví dụ một phép chia giữa hai số nguyên nhị phân không dấu. Quá trình thực hiện như sau: kiểm tra từng nhóm bit của số bị chia từ bên trái sang, nếu nhóm bit biểu diễn một số nhỏ hơn số chia thì ta điền 0 vào thương, sau đó thêm một bit tiếp theo vào nhóm bit trên. Nếu nhóm bit lớn hơn hoặc bằng số chia, ta viết 1 vào thương và lấy giá trị nhóm bit trừ đi số chia ta được *dư thành phần*, ghép thêm một bit tiếp theo của số bị chia vào dư thành phần và tiếp tục thực hiện các bước trên cho đến khi hết các bit của số bị chia.



Hình 9.16 Sơ đồ thuật toán phép chia số nhị phân không dấu

Thuật toán thực hiện phép chia trên máy được minh họa trong Hình 9.16 dựa trên nguyên tắc chia trên. Số chia được nạp vào thanh ghi M, số bị chia nạp vào thanh ghi Q. Tại mỗi bước, các thanh ghi A và Q cùng nhau dịch sang bên trái 1 bit. Lấy A trừ đi M để xác định xem liệu A có phải là dư thành phần. Nếu đúng, Q_0 được thiết lập là 1. Ngược lại, nếu không phải Q_0 được thiết lập là 0 và M được cộng lại vào A để khôi phục lại giá trị

trước đó. Bộ đếm giảm đi một đơn vị, quá trình này thực hiện qua n bước. Kết thúc, ta có thương trong thanh ghi Q và số dư trong thanh ghi A.

A	Q	
0000	0111	Khởi tạo giá trị
0000	1110	Dịch
<u>1101</u>		Trừ bằng cách cộng với bù hai của 0011
<u>1101</u>		Kết quả âm
0000	1110	Thiết lập $Q_0 = 0$, khôi phục lại A
0001	1100	Dịch
<u>1101</u>		Kết quả âm
<u>1110</u>		Thiết lập $Q_0 = 0$, khôi phục lại A
0001	1100	
0011	1000	Dịch
<u>1101</u>		Kết quả dương, thiết lập $Q_0 = 1$
<u>0000</u>	1001	
0001	0010	Dịch
<u>1101</u>		Kết quả âm
<u>1110</u>		Thiết lập $Q_0 = 0$, khôi phục lại A
0001	0010	

Hình 9.17 Ví dụ phép chia hai số bù hai (7/3)

Thuật toán này có thể cải tiến một chút cho trường hợp số âm. Chúng tôi xin trình bày một phương pháp cho số bù hai cùng với ví dụ trong Hình 9.17

Giả sử rằng số chia V và số bị chia D là hai số bù hai dương và $|V| < |D|$. Nếu $|V| = |D|$, thương $Q = 1$ và số dư $R = 0$. Nếu $|V| > |D|$, ta có $Q = 0$ và $R = D$. Thuật toán có thể tóm tắt như sau:

- Tải bù hai của số chia vào thanh ghi M, như vậy giá trị của M bằng trừ số chia. Tải số bị chia vào hai thanh ghi A, Q. Số chia được biểu diễn dưới dạng số nguyên 2-bit. Như vậy, với một số 4-bit 0111 sẽ trở thành 00000111.
- Dịch A, Q sang bên trái 1 bit
- Thực hiện $A + M \rightarrow A$: lấy A trừ M, kết quả ghi vào A.
- a. Nếu kết quả không âm (bit dấu của A = 0), thiết lập $Q_0 = 1$.
 - Nếu kết quả âm (bit dấu bằng 1), thiết lập $Q_0 = 0$ và khôi phục lại giá trị trước đó của A.
- Lặp lại từ bước 2 đến bước 4 với số lần bằng số bit của Q.
- Kết quả số dư nằm trong thanh ghi A, còn thương nằm trong thanh ghi Q.

Như vậy, thuật toán trên áp dụng với hai số bù hai dương. Trong phép chia $D:V = Q$ dư R , ta có

$$D = Q \times V + R$$

Dấu của Q và R được quyết định bởi dấu của D và V . Nghĩa là, D và R sẽ có cùng dấu. Dấu của Q là dương nếu D và V cùng dấu, âm nếu D và V trái dấu. Như vậy, một trong những cách giúp ta thực hiện phép chia số bù hai nói chung là ta chuyển các toán hạng

thành các số không dấu rồi thực hiện phép chia theo thuật toán ở trên, sau đó điều chỉnh dấu của kết quả.

9.4. BIỂU DIỄN DẤU CHẤM ĐỘNG⁴

9.4.1. Nguyên tắc

Với biểu diễn số dạng dấu chấm tĩnh (ví dụ: dạng bù hai) ta chỉ có thể biểu diễn một dải các số nguyên dương hoặc âm xung quanh 0. Ngoài ra, để biểu diễn số thực, ta có thể giả định một số bit nhất định cho phần phân số. Tuy nhiên, giải pháp này có một số hạn chế: không thể biểu diễn được các số quá lớn hoặc quá bé.

Với số thập phân, để biểu diễn các số lớn hoặc quá bé, thông thường chúng ta sử dụng cách như sau:

$$976,000,000,000,000 = 9.76 \times 10^{14}$$

Hay

$$0.0000000000976 = 9.76 \times 10^{-14}$$

Với cách này, ta có thể biểu diễn các số quá lớn hoặc quá bé bằng cách ngắn gọn hơn rất nhiều.

Tương tự ta có thể biểu diễn một số nhị phân như sau:

$$\pm S \times B^{\pm E}$$

Trong đó

- Dấu: cộng hoặc trừ
- Phần định trị S
- Phần số mũ E
- Cơ số B (với số nhị phân, cơ số bằng 2)



$$\begin{array}{rcl}
 1.1010001 \times 2^{10100} & = & 0\ 10010011\ 10100010000000000000000000000000 = 1.6328125 \times 2^{20} \\
 -1.1010001 \times 2^{10100} & = & 1\ 10010011\ 10100010000000000000000000000000 = -1.6328125 \times 2^{20} \\
 1.1010001 \times 2^{-10100} & = & 0\ 01101011\ 10100010000000000000000000000000 = 1.6328125 \times 2^{-20} \\
 -1.1010001 \times 2^{-10100} & = & 1\ 01101011\ 10100010000000000000000000000000 = -1.6328125 \times 2^{-20}
 \end{array}$$

b. Ví dụ

Hình 9.18 Định dạng biểu diễn dấu chấm động 32-bit

⁴ Với số thực, theo quy ước của Việt Nam sử dụng dấu phẩy để ngăn cách giữa phần nguyên và phần phân số. Tuy nhiên, với nhiều nước khác và biểu diễn trong máy tính, người ta sử dụng dấu chấm (point) cho số thực. Vì vậy, trong phần này, chúng tôi vẫn giữ nguyên là dấu chấm.

Như vậy, khi lưu trữ một số nhị phân, ta chỉ cần lưu trữ dấu, phần định trị S và số mũ E. Còn cơ số B mặc định bằng 2, ta không cần lưu trữ. Định dạng biểu diễn dấu chấm động 32-bit điển hình được minh họa trong Hình 9.18. Bit ngoài cùng bên trái là bit dấu, 8 bit tiếp theo được sử dụng để lưu trữ số mũ. Tuy nhiên, giá trị số mũ không được lưu trữ trực tiếp tại 8-bit này mà được điều chỉnh thành **số mũ lệch (biased)**. Số mũ thực tế sẽ bằng số mũ lệch trừ đi một giá trị cố định, gọi là **độ lệch**. Giá trị độ lệch bằng $(2^{k-1} - 1)$, với k là số bit biểu diễn phần mũ. Trong trường hợp này, có 8 bit phần mũ nên độ lệch là 127 ($2^7 - 1$). Như vậy, với 8-bit trường mũ lệch ta biểu diễn các số trong khoảng từ 0 đến 255, trừ đi độ lệch, giá trị số mũ thực tế sẽ từ -127 đến +128. 23 bit còn lại biểu diễn phần định trị.

Vậy, để biểu diễn một số thực nhị phân bất kỳ dưới định dạng 32-bit như trên, ta phải đưa số đó về dạng tiêu chuẩn như sau: số ngoài cùng bên trái phải bằng 1, sau đó là dấu chấm và phần phân số:

$$\pm 1. \text{bbb...b} \times 2^{\pm E}$$

Với b là các bit nhị phân (0 hoặc 1). Vì bit ngoài cùng bên trái luôn là 1 nên ta không cần thiết phải lưu trữ bit này. Như vậy, với trường 23-bit ta có thể lưu trữ được 24-bit phần định trị. Vậy nếu số chưa ở dạng tiêu chuẩn, ta phải dịch dấu chấm về vị trí sau bit 1 đầu tiên và điều chỉnh số mũ tương ứng.

Các số nhị phân sau đây có giá trị tương đương nhau:

$$0.11 \times 2^5$$

$$110 \times 2^2$$

$$0.0110 \times 2^6$$

$$1.1 \times 2^4 \text{ (Dạng tiêu chuẩn)}$$

Các ví dụ trong Hình 9.18b được lưu trữ theo định dạng trên. Trong đó, ở bên trái là số thực nhị phân, ở giữa là dạng biểu diễn số thực dấu chấm động 32-bit theo định dạng ở trên, bên phải là giá trị của số ở hệ thập phân. Chú ý một số điểm sau:

- Dấu được lưu trữ ở bit đầu tiên.
- Bit đầu tiên của phần định trị luôn là 1, do vậy không cần lưu trữ.
- Giá trị trong trường số mũ là giá trị số mũ thực tế cộng với 127.
- Cơ số là 2

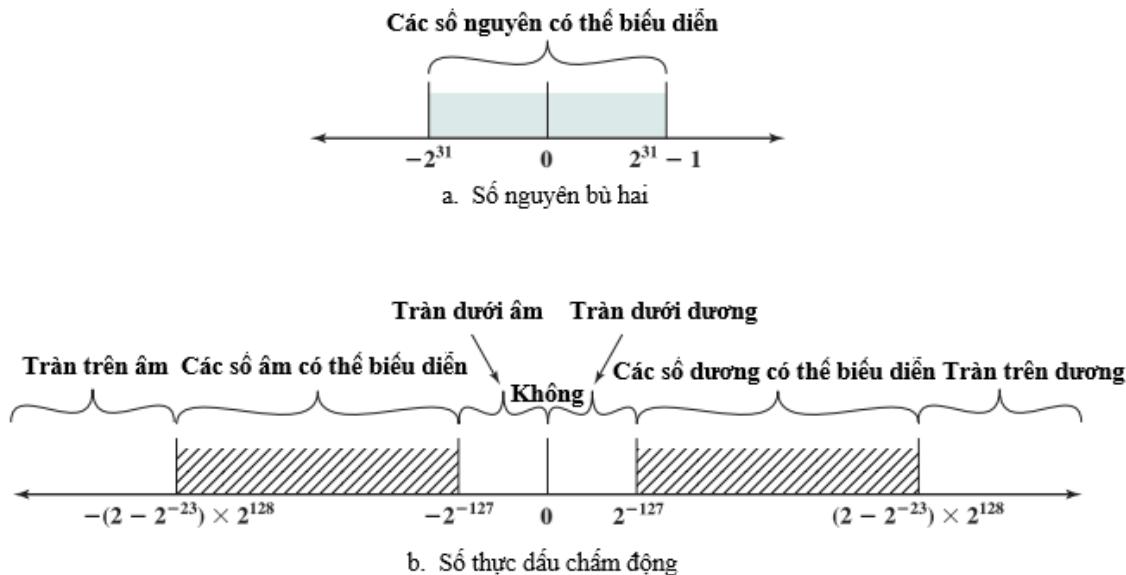
Trong Hình 9.19 là miền giá trị mà một từ 32-bit có thể biểu diễn được. Với dạng biểu diễn bù hai ta có thể biểu diễn được các giá trị từ -2^{-31} đến $2^{31} - 1$, như vậy có tất cả 2^{32} số khác nhau. Với định dạng dấu chấm động, miền giá trị như sau

- Các số âm từ $-(2 - 2^{-23}) \times 2^{128}$ đến -2^{-127}
- Các số dương từ 2^{-127} đến $(2 - 2^{-23}) \times 2^{128}$

Năm miền giá trị không thể biểu diễn:

- Các số âm nhỏ hơn $-(2 - 2^{-23}) \times 2^{128}$, gọi là **Tràn trên âm**
- Các số âm lớn hơn 2^{-127} , gọi là **Tràn dưới âm**

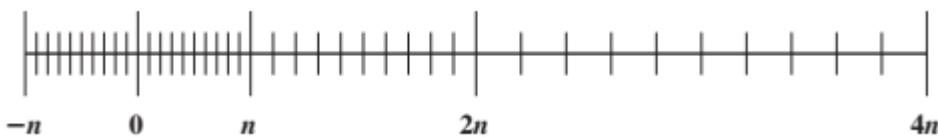
- Số không
- Các số dương nhỏ hơn 2^{-127} , gọi là **Tràn dưới dương**
- Các số dương lớn hơn $(2 - 2^{-23}) \times 2^{128}$, gọi là **Tràn trên dương**



Hình 9.19 Các số có thể biểu diễn với định dạng 32-bit

Như trên ta có thể thấy định dạng dấu chấm động không biểu diễn được số 0. Tuy nhiên, phần sau chúng tôi sẽ trình bày, ta sử dụng một biểu diễn quy ước cho số 0. Tuy nhiên, định dạng dấu chấm động không cho phép ta biểu diễn nhiều số hơn so với các dạng biểu diễn khác cùng số chiều dài bit. Số lượng tối đa các giá trị khác nhau một từ nhau 32 bit biểu diễn được vẫn là 2^{32} . Điều mà dạng biểu diễn này thực hiện được đó là trải các giá trị biểu diễn được ra thành miền dài, một miền âm và một miền dương. Trong thực tế, hầu như các số biểu diễn dưới dạng dấu chấm động chỉ là biểu diễn xấp xỉ.

Ngoài ra, vị trí các số được biểu diễn trên trực số không cách đều nhau (Hình 9.20). Các số ở gần gốc tọa độ có khoảng cách gần hơn và ngược lại. Đây là một trong những nhược điểm của các phép toán thực hiện trên biểu diễn dấu chấm động: Các phép toán có kết quả không bằng chính xác giá trị có thể biểu diễn được và phải được làm tròn về giá trị gần nhất có thể biểu diễn được.



Hình 9.20 Mật độ các số dấu chấm động

Với định dạng 32-bit ở trên, độ chính xác và miền giá trị có một mối quan hệ với nhau. Nếu chúng ta tăng số bit phân mũ, phạm vi biểu diễn của định dạng đó sẽ tăng lên. Tuy nhiên, với 32-bit ta chỉ biểu diễn được 2^{32} giá trị khác nhau nên khi phạm vi biểu diễn tăng thì độ chính xác sẽ giảm đi. Như vậy, với một định dạng nhất định, phân mũ càng lớn thì miền giá trị càng lớn tuy nhiên độ chính xác càng nhỏ. Cách duy nhất để tăng cả phạm vi

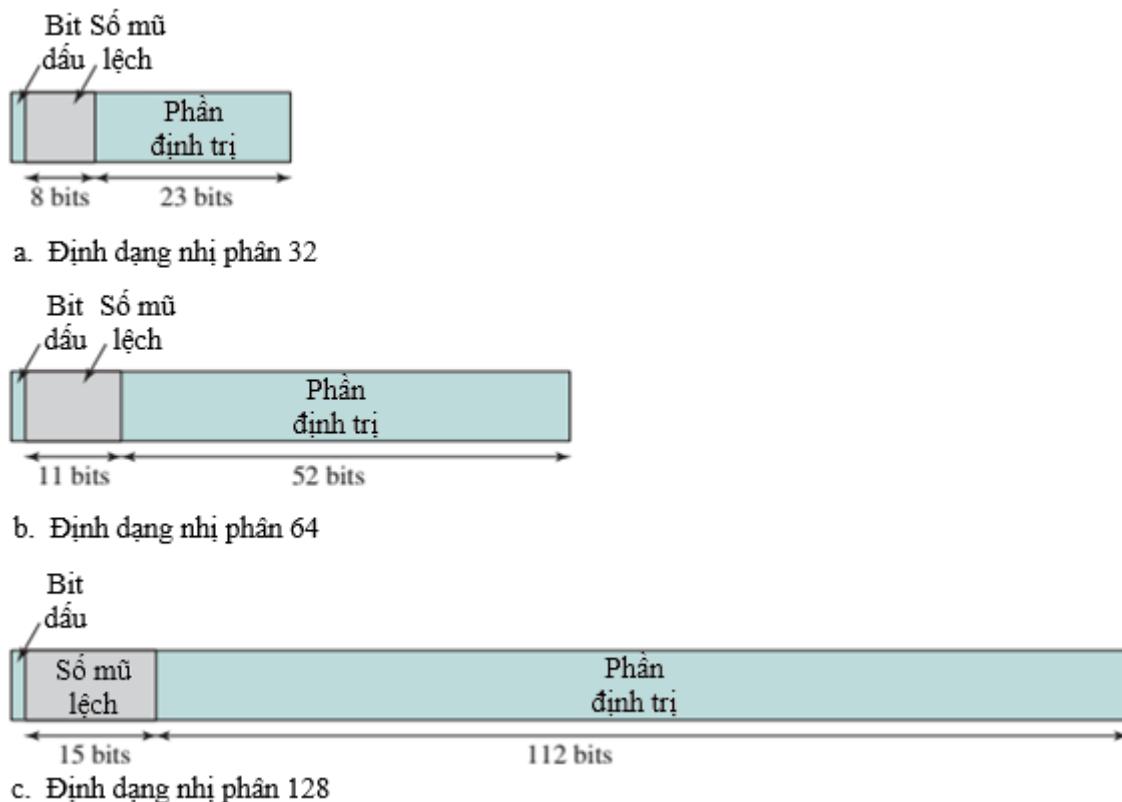
và độ chính xác là sử dụng nhiều bit hơn. Do đó, hầu hết các máy tính cung cấp các dạng biểu diễn có độ chính xác đơn và độ chính xác kép. Ví dụ, bộ xử lý có thể hỗ trợ định dạng chính xác đơn là 64 bit và một định dạng chính xác kép là 128 bit.

9.4.2. Chuẩn IEEE cho biểu diễn số nhị phân dấu chấm động

Chuẩn IEEE 754 được đưa ra năm 1985 và sửa đổi năm 2008 quy định về biểu diễn nhị phân dấu chấm động trong máy tính. Chuẩn này được phát triển với mục đích tạo thuận lợi cho các chương trình máy tính có thể tương thích với nhiều bộ xử lý khác nhau cũng như khuyến khích phát triển các chương trình định hướng số học tinh vi hơn Hiện nay, chuẩn IEEE 754 đã được áp dụng rộng rãi trên hầu hết các bộ xử lý và các bộ tính toán số học hiện tại. IEEE 754-2008 bao gồm các quy định về cả dạng biểu diễn nhị phân và thập phân dấu chấm động. Trong chương này, chúng ta chỉ quan tâm tới các dạng biểu diễn nhị phân.

IEEE 754-2008 định nghĩa các định dạng dấu chấm động khác nhau như sau:

- **Định dạng số học:** Các phép toán được quy định trong chuẩn cần được hỗ trợ các định dạng số học. Định dạng này được sử dụng để biểu diễn toán hạng hoặc kết quả của các phép toán dưới dạng dấu chấm động.
- **Định dạng cơ bản:** Định dạng này bao gồm năm dạng biểu diễn dấu chấm động: ba dạng nhị phân và hai dạng thập phân cùng với các chuẩn mã hóa của chúng và quy định định dạng nào được sử dụng trong số học.
- **Định dạng chuyển đổi:** Một định dạng mã hóa nhị phân có chiều dài cố định, cho phép chuyển đổi dữ liệu giữa các nền tảng khác nhau và có thể được sử dụng để lưu trữ.



Hình 9.21 Các định dạng IEEE 754

Ba **định dạng nhị phân** cơ bản lần lượt có kích thước bit là **32, 64 và 128 bit**, với số bit trường mũ là 8, 11 và 15 bit (hình 9.21). Bảng 9.3 tóm tắt các đặc tính của ba định dạng này. Hai **định dạng thập phân** cơ bản có độ dài bit là **64 và 128 bit**. Tất cả các định dạng cơ bản trên cũng đồng thời là các định dạng số học (có thể được sử dụng cho các phép toán số học) và định dạng chuyển đổi (độc lập với các nền tảng khác nhau).

Ngoài ra, chuẩn 754 cũng quy định một số định dạng khác như: **định dạng nhị phân 16**, đây là định dạng chuyển đổi và được dùng để lưu trữ các giá trị không cần độ chính xác cao. Định dạng chuyển đổi còn quy định hai **định dạng nhị phân {k}** và **thập phân {k}** có chiều dài k bit với kích thước phần định trị và phần mũ cụ thể. Số bit của các định dạng này phải là một bội số của 32 bit; do đó $k = 160, 192, \dots$. Các định dạng này cũng là định dạng số học.

Bảng 9.3 Quy định trong chuẩn IEEE 754

Thông số	Định dạng		
	Nhị phân 32	Nhị phân 64	Nhị phân 128
Kích thước (bit)	32	64	128
Số bit trường mũ	8	11	15
Độ lệch	127	1023	16383

Số mũ tối đa	127	1023	16383
Số mũ tối thiểu	-126	-1022	-16382
Miền giá trị xấp xỉ (hệ 10)	$10^{-38}, 10^{+38}$	$10^{-308}, 10^{+308}$	$10^{-4932}, 10^{+4932}$
Số bit trường định trị theo sau	23	52	112
Số lượng số mũ	254	2046	32766
Số lượng các số phân số	2^{23}	2^{52}	2^{112}
Số lượng các giá trị	1.98×3^{31}	1.99×2^{63}	1.99×2^{127}
Số dương nhỏ nhất	2^{-126}	2^{-1022}	2^{-16362}
Số dương lớn nhất	$2^{128} - 2^{104}$	$2^{1024} - 2^{971}$	$2^{16384} - 2^{16271}$

Một định dạng khác cũng được định nghĩa trong chuẩn là các **định dạng độ chính xác mở rộng** (gọi tắt là **định dạng mở rộng**) quy định việc mở rộng miền giá trị bằng cách thêm các bit phần mũ và tăng độ chính xác bằng cách tăng số bit phần định trị. Kích thước của phần định trị và phần mũ phụ thuộc vào việc thực hiện cụ thể, tuy nhiên chuẩn quy định một số ràng buộc về chúng. Định dạng mở rộng nằm trong các loại định dạng số học nên nó thường được sử dụng trong các phép toán trung gian. Do có độ chính xác cao hơn, định dạng mở rộng giảm sai số của phép toán do việc lòn tròn; và với phạm vi biểu diễn lớn hơn, định dạng này cũng giảm khả năng một hoạt động bị lỗi do tràn kết quả của phép toán trung gian. Ngoài ra, định dạng mở rộng vẫn mang các ưu điểm của định dạng cơ bản có kích thước lớn mà không bị ảnh hưởng về mặt thời gian xử lý do có độ chính xác cao hơn.

Cuối cùng, IEEE 754-2008 định nghĩa là **định dạng chính xác có thể mở rộng**: cho phép người dùng thiết lập độ chính xác và phạm vi biểu diễn. Các định dạng theo kiểu này có thể được sử dụng cho các phép tính trung gian nhưng chuẩn không đưa ra các ràng buộc về khuôn dạng hoặc kích thước của chúng.

Bảng 9.4 là quan hệ giữa các định dạng ở trên và các loại định dạng.

Bảng 9.4 Các định dạng IEEE

Định dạng	Loại định dạng		
	Định dạng số học	Định dạng cơ bản	Định dạng chuyên đổi
Nhi phân 16			X
Nhi phân 32	X	X	X
Nhi phân 64	X	X	X

Nhị phân 128	X	X	X
Nhị phân $\{k\}$ ($k = n \times 32$ với $n > 4$)	X		X
Thập phân 64	X	X	X
Thập phân 128	X	X	X
Thập phân $\{k\}$ ($k = n \times 32$ với $n > 4$)	X		X
Độ chính xác mở rộng	X		
Độ chính xác có thể mở rộng	X		

Không chỉ quy định về các định dạng, chuẩn IEEE còn quy định một số mẫu bit biểu diễn một số giá trị đặc biệt như trong Bảng 9.5. Cụ thể chuẩn IEEE quy định về giá trị biểu diễn của các định dạng như sau:

- Giá trị trường mũ của định dạng 32-bit là từ 1 đến 254, tương ứng với số mũ thực tế là từ -126 đến +127. Tương tự như vậy với định dạng 64-bit: từ 1 đến 2046 ứng với số mũ thực tế từ -1022 đến +1023 và 128-bit... Với các số mũ trong khoảng này, các định dạng trên sẽ biểu diễn các số theo dạng dấu chấm động tiêu chuẩn (phần trên đã đề cập đến) với bit 1 ở ngoài cùng bên trái trước dấu chấm. Như vậy, với số 1 là ngầm định, các định dạng trên cho phép ta biểu diễn được 24-bit, 53-bit, hoặc 113-bit phần định trị. Và do đó, người ta gọi trường này là **trường định trị theo sau**.
- Như phần trên đã đề cập, số nhị phân dấu chấm động không biểu diễn được giá trị 0, do đó, chuẩn này quy ước biểu diễn số không là số có phần mũ và phần phân số bằng 0. Tùy thuộc vào bit dấu, ta có số không âm hoặc dương.
- Với tất cả các bit phần mũ bằng 1 và phần phân số bằng 0 biểu diễn âm vô cùng hoặc dương vô cùng tùy thuộc vào bit dấu. Quy ước này cho phép người sử dụng quyết định khi nào xử lý tràn như một điều kiện lỗi hoặc gán giá trị vô cùng và tiếp tục thực thi chương trình.
- Trường mũ bằng không, trường phân số khác không biểu diễn số dạng *bán tiêu chuẩn*: bit trước dấu chấm bằng 0 (thay vì ngầm định là 1 trong phần trên) và số mũ thực tế là -126 (định dạng 32b) hoặc -1022 (định dạng 64b). Số được biểu diễn là số dương hoặc âm tùy thuộc vào bit dấu.
- Trường mũ gồm tất cả các bit 1, phần phân số là một số bất kỳ khác không được quy ước là giá trị NaN (Not a Number – Không phải là số) và được sử dụng để báo hiệu các điều kiện ngoại lệ khác.

Ý nghĩa của các số bán tiêu chuẩn và các số NaN được trình bày cụ thể trong Phần 9.5.

Bảng 9.5 Các quy ước biểu diễn số nhị phân dấu chấm động trong IEEE 754

a. Định dạng nhị phân 32 bit

	Dấu	Số mũ lệch	Phần phân số	Giá trị
Không dương	0	0	0	0

Không âm	1	0	0	-0
Dương vô cùng	0	Tất cả các bit là 1	0	$+\infty$
Âm vô cùng	1	Tất cả các bit là 1	0	$-\infty$
Quiet NaN	0 hoặc 1	Tất cả các bit là 1	$\neq 0$; bit đầu tiên = 1	qNaN
Signaling NaN	0 hoặc 1	Tất cả các bit là 1	$\neq 0$; bit đầu tiên = 0	sNaN
Số khác không dương	0	$0 < e < 255$	f	$1.f \times 2^{e-127}$
Số khác không âm	1	$0 < e < 255$	f	$-1.f \times 2^{e-127}$
Số bán tiêu chuẩn dương	0	0	$f \neq 0$	$0.f \times 2^{e-126}$
Số bán tiêu chuẩn âm	1	0	$f \neq 0$	$-0.f \times 2^{e-126}$

b. Định dạng nhị phân 64 bit

	Dấu	Số mũ lệch	Phần phân số	Giá trị
Không dương	0	0	0	0
Không âm	1	0	0	-0
Dương vô cùng	0	Tất cả các bit là 1	0	$+\infty$
Âm vô cùng	1	Tất cả các bit là 1	0	$-\infty$
Quiet NaN	0 hoặc 1	Tất cả các bit là 1	$\neq 0$; bit đầu tiên = 1	qNaN
Signaling NaN	0 hoặc 1	Tất cả các bit là 1	$\neq 0$; bit đầu tiên = 0	sNaN
Số khác không dương	0	$0 < e < 2047$	f	$1.f \times 2^{e-1023}$
Số khác không âm	1	$0 < e < 2047$	f	$-1.f \times 2^{e-1023}$
Số bán tiêu chuẩn dương	0	0	$f \neq 0$	$0.f \times 2^{e-1022}$
Số bán tiêu chuẩn âm	1	0	$f \neq 0$	$-0.f \times 2^{e-1022}$

c. Định dạng nhị phân 128 bit

	Dấu	Số mũ lệch	Phần phân số	Giá trị
Không dương	0	0	0	0
Không âm	1	0	0	-0
Dương vô cùng	0	Tất cả các bit là 1	0	$+\infty$
Âm vô cùng	1	Tất cả các bit là 1	0	$-\infty$
Quiet NaN	0 hoặc 1	Tất cả các bit là 1	$\neq 0$; bit đầu tiên = 1	qNaN
Signaling NaN	0 hoặc 1	Tất cả các bit là 1	$\neq 0$; bit đầu tiên = 0	sNaN

Số khác không dương	0	Tất cả các bit là 1	f	$1.f \times 2^{e-16383}$
Số khác không âm	1	Tất cả các bit là 1	f	$-1.f \times 2^{e-16383}$
Số bán tiêu chuẩn dương	0	0	$f \neq 0$	$0.f \times 2^{e-16383}$
Số bán tiêu chuẩn âm	1	0	$f \neq 0$	$-0.f \times 2^{e-16383}$

9.5. CÁC PHÉP TOÁN VỚI DẤU CHẤM ĐỘNG

Các phép toán cơ bản với số nhị phân dấu chấm động được tóm tắt trong Bảng 9.6. Để thực hiện các phép cộng hoặc phép trừ, hai toán hạng phải có cùng giá trị phần mũ. Như vậy, nếu số mũ ban đầu khác nhau, ta cần dịch dấu chấm của một trong hai toán hạng. Phép nhân và phép chia có thể đơn giản hơn.

Bảng 9.6 Thực hiện các phép toán dấu chấm động

Số dấu chấm động	Các phép toán
$X = X_s \times B^{X_E}$ $Y = Y_s \times B^{Y_E}$	$X + Y = (X_s \times B^{X_E - Y_E} + Y_s) \times B^{Y_E}$ $X - Y = (X_s \times B^{X_E - Y_E} - Y_s) \times B^{Y_E}$ $X \times Y = (X_s \times Y_s) \times B^{X_E + Y_E}$ $\frac{X}{Y} = \left(\frac{X_s}{Y_s} \right) \times B^{X_E - Y_E}$

Ví dụ

$$X = 0.3 \times 10^2 = 30$$

$$Y = 0.2 \times 10^3 = 200$$

$$X + Y = (0.3 \times 10^{2-3} + 0.2) \times 10^3 = 0.23 \times 10^3 = 230$$

$$X - Y = (0.3 \times 10^{2-3} - 0.2) \times 10^3 = (-0.17) \times 10^3 = -170$$

$$X \times Y = (0.3 \times 0.2) \times 10^{2+3} = 0.06 \times 10^5 = 6000$$

$$X \div Y = (0.3 \div 0.2) \times 10^{2-3} = 1.5 \times 10^{-1} = 0.15$$

Phép toán dấu chấm động có thể tạo ra một số trường hợp lỗi như sau:

- **Tràn trên phần mũ:** Số mũ dương vượt quá giá trị số mũ tối đa trường mũ có thể lưu trữ. Trong một số hệ thống, số mũ lúc này được thiết lập giá trị $+\infty$ hoặc $-\infty$.
- **Tràn dưới phần mũ:** Số mũ âm nhỏ hơn giá trị số mũ tối thiểu trường mũ có thể biểu diễn (ví dụ: nhỏ hơn -127 với định dạng nhị phân 32-bit). Như vậy, số đó quá nhỏ và có thể quy về 0.
- **Tràn dưới phần định trị:** Khi hiệu chỉnh phần định trị, các số có thể dịch ra ngoài dãy bit. Khi đó, giá trị định trị có thể cần được làm tròn.
- **Tràn trên phần định trị:** Khi thực hiện phép cộng hai phần định trị cùng dấu, kết quả có thể vượt qua bit có giá trị lớn nhất. Để giải quyết vấn đề này ta cần dịch chuyển dấu chấm và hiệu chỉnh lại phần mũ.

9.5.1. Phép cộng và phép trừ

Với dấu chấm động, phép toán cộng và trừ phức tạp hơn so với phép nhân và chia. Phép cộng và trừ được thực hiện qua bốn giai đoạn như sau:

1. Kiểm tra 0
2. Hiệu chỉnh phần định trị
3. Cộng hoặc trừ phần định trị
4. Chuẩn hóa kết quả

Hình 9.22 là sơ đồ thuật toán phép cộng và trừ theo từng bước. Chúng ta giả thiết rằng các toán hạng có định dạng tương tự như định dạng trong hình 9.21. Với phép toán cộng hoặc trừ, hai toán hạng phải được chuyển tới các thanh ghi được ALU sử dụng. Do định dạng dấu chấm động chứa một bit định trị ngầm định, bit đó cần được thiết lập rõ trước khi thực hiện phép toán.

Bước 1: Kiểm tra 0. Về bản chất phép cộng và phép trừ giống nhau chỉ khác là với phép trừ, trước khi thực hiện các bước, ta cần đổi dấu của số trừ sau đó thực hiện như phép cộng. Vì vậy, bước đầu tiên là bước kiểm tra xem một trong hai toán hạng có toán hạng nào bằng 0 không? Nếu có, kết quả chính là toán hạng còn lại.

Bước 2: Hiệu chỉnh phần định trị sao cho số mũ của hai toán hạng bằng nhau.

Để hiểu tại sao cần hiệu chỉnh phần định trị và phần mũ, ta theo dõi ví dụ về phép cộng giữa hai số hệ thập phân như sau:

$$(123 \times 10^0) + (456 \times 10^{-2})$$

Rõ ràng, ta không thể cộng ngay lập tức phần định trị của hai số trên. Ta có thể hiệu chỉnh phần định trị, phần mũ của hai toán hạng trên và thực hiện phép cộng như sau:

$$(123 * 10^0) + (456 * 10^{-2}) = (123 * 10^0) + (4.56 * 10^0) = 127.56 * 10^0$$

Việc hiệu chỉnh có thể thực hiện bằng cách dịch phần định trị của số nhỏ hơn sang bên phải (tăng số mũ) hoặc dịch phần định trị của số lớn hơn sang bên trái (giảm số mũ). Vì phép dịch có thể làm mất một số chữ số nên ta sẽ chọn dịch phần định trị của số nhỏ hơn vì nếu bị mất, các chữ số mất sẽ có giá trị nhỏ hơn. Bước hiệu chỉnh sẽ được thực hiện bằng cách dịch từng bit phần định trị sang bên phải đồng thời tăng số mũ cho đến khi số mũ của hai toán hạng bằng nhau. Nếu bước dịch này làm cho phần định trị bị dịch bằng 0 thì số còn lại sẽ là kết quả của phép toán. Do đó, nếu hai số có số mũ chênh lệch khá lớn thì số nhỏ hơn có thể sẽ bị mất.

Bước 3: Cộng. Tiếp theo, hai phần định trị được cộng vào với nhau, xét cả phần dấu. Vì dấu của hai toán hạng có thể khác nhau nên kết quả có thể bằng 0. Ngoài ra, kết quả của phép cộng có thể tràn 1 chữ số, cần được dịch phải và tăng số mũ lên một đơn vị. Khi tăng số mũ, nếu tràn trên phần mũ thì cần báo tràn và kết thúc phép toán.

Bước 4: Chuẩn hóa. Bước này thực hiện việc chuẩn hóa kết quả, bao gồm: dịch phần định trị của kết quả sao cho chữ số quan trọng nhất (bit ngoài cùng bên trái) khác không. Mỗi bước dịch ta cần đi kèm với việc giảm số mũ đi một đơn vị nên có thể gây ra trường

hợp tràn dưới số mũ. Cuối cùng là làm tròn và trả về kết quả. Cụ thể việc làm tròn kết quả chúng tôi sẽ trình bày thêm ở phần thực hiện phép nhân và phép chia.

9.5.2. Phép nhân và phép chia

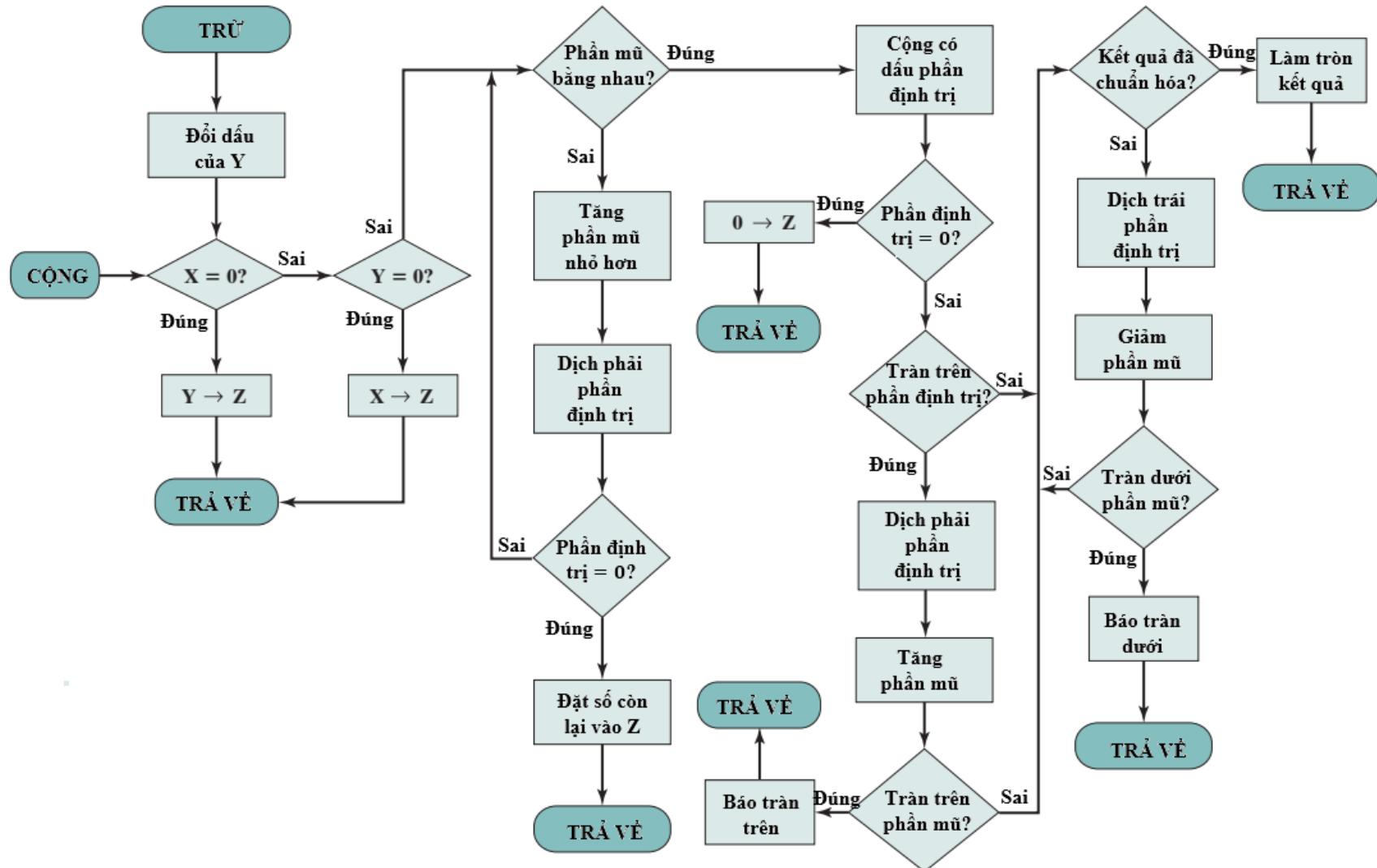
Phép nhân và phép chia dấu chấm động đơn giản hơn nhiều so với phép cộng và phép trừ như sẽ chỉ ra dưới đây.

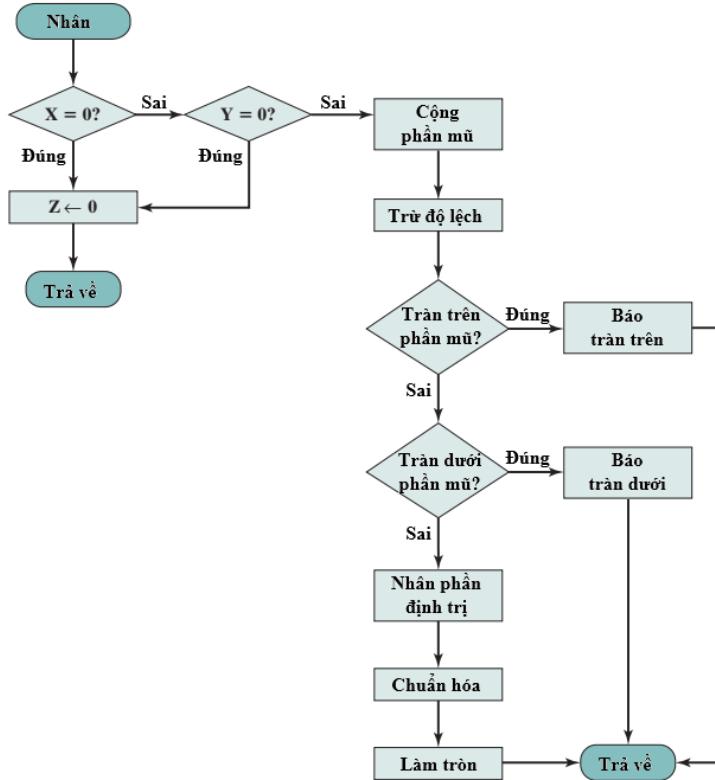
Đầu tiên, ta xét phép nhân (Hình 9.23) gồm các bước sau:

- Kiểm tra nếu một trong hai toán hạng là 0 thì kết quả bằng 0.
- Tiếp theo là cộng hai số mũ. Nếu số mũ được lưu trữ dưới dạng số mũ lệch thì tổng của chúng sẽ tăng gấp đôi giá trị độ lệch. Do đó, phải trừ bớt đi một giá trị độ lệnh này. Kết quả của bước này có thể tràn trên (do phép cộng) hoặc tràn dưới (do phép trừ), cần được báo tràn và kết thúc thuật toán.
- Nếu số mũ của bước trên không bị tràn, ta thực hiện việc nhân có dấu phần định trị giống như nhân hai số nguyên. Hai toán hạng của phép nhân này được biểu diễn dưới dạng dấu và độ lớn nhưng các bước chi tiết lại giống như biểu diễn bù hai. Kết quả có chiều dài gấp đôi chiều dài hai toán hạng và cần được làm tròn.
- Kết quả được chuẩn hóa và làm tròn giống như với phép cộng và phép trừ. Lưu ý rằng việc chuẩn hóa có thể gây ra tràn dưới phần mũ.

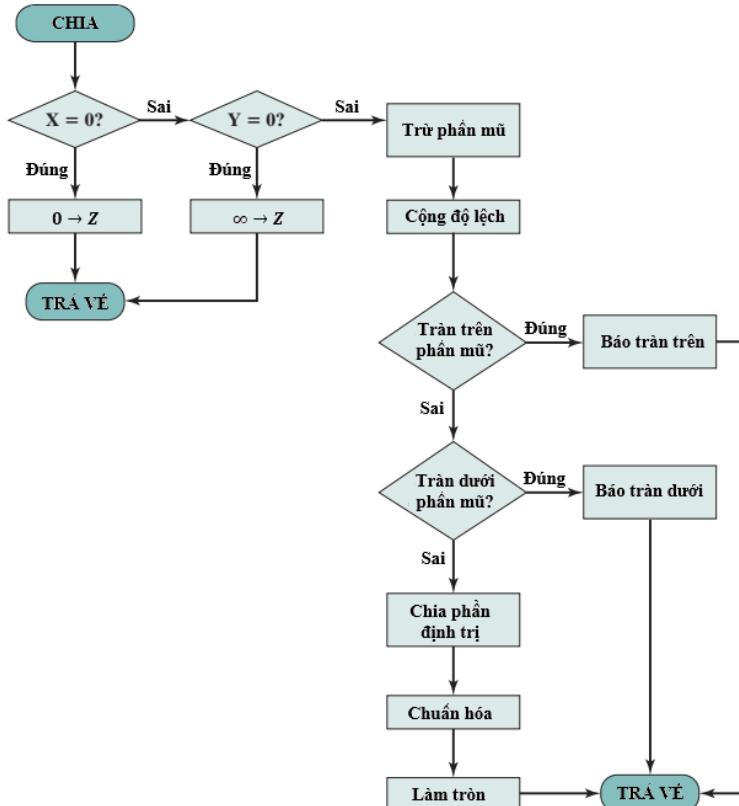
Phép chia được mô tả trong Hình 9.24 bao gồm:

- Kiểm tra 0: nếu số chia bằng 0 thì phát ra một tín hiệu báo lỗi hoặc thiết lập kết quả bằng vô cùng. Nếu số bị chia bằng 0 thì kết quả thiết lập bằng 0.
- Nếu không gặp hai trường hợp trên thì tiếp theo lấy số mũ của số bị chia trừ đi số mũ của số chia. Khi trừ hai số mũ lệch, độ lệch sẽ bị trừ đi khỏi kết quả nên ta cần cộng bù giá trị này vào. Cũng như phép nhân, việc cộng và trừ sẽ có thể gây ra tràn trên và tràn dưới. Ta cần kiểm tra tràn và báo lỗi nếu có.
- Bước cuối cùng là chia hai phần định trị, sau đó là chuẩn hóa và làm tròn kết quả.

Hình 9.22 Sơ đồ thuật toán phép cộng và phép trừ ($X \pm Y \rightarrow Z$)



Hình 9.23 Phép nhân dấu chấm động ($X * Y \rightarrow Z$)



Hình 9.24 Phép chia dấu chấm động ($X/Y \rightarrow Z$)

9.5.3. Độ chính xác của phép toán

GUARD BIT Như đã đề cập, trước khi thực hiện các phép toán dấu chấm động, số mũ và phần định trị của mỗi toán hạng được nạp vào các thanh ghi của ALU. Chiều dài của thanh ghi chứa phần định trị thường lớn hơn chiều dài của phần định trị cộng với một bit ngầm định. Các thanh ghi này chứa một số bit thêm được gọi là guard bit (hay còn gọi là bit bảo vệ) ở phía bên phải của phần định trị với các giá trị bằng 0.

Lý do sử dụng các guard bit được minh họa trong Hình 9.25. Xét phép toán với các toán hạng định dạng theo chuẩn IEEE gồm 24-bit định trị (đã bao gồm 1 bit ngầm định trước dấu chấm) như sau $x = 1.00 \dots 00 * 2^1$ và $y = 1.11 \dots 11 * 2^0$. Nếu số lớn trừ đi số bé thì ta cần hiệu chỉnh phần định trị để số mũ bằng nhau (Hình 10.25a). Do việc hiệu chỉnh này, y mất đi 1 bit 1 phần định trị. Kết quả của phép toán là 2^{-22} . Trong 9.25b, phép toán được thực hiện lại với 4 guard bit ở bên phải. Như vậy, khi hiệu chỉnh phần định trị, bit cuối cùng được dịch sang phần guard bit, ta được kết quả phép toán là 2^{-23} , chính xác hơn so với kết quả phần a.

$$\begin{array}{r} x = 1.000\dots00 \times 2^1 \\ -y = \underline{0.111\dots11} \times 2^1 \\ z = 0.000\dots01 \times 2^1 \\ = 1.000\dots00 \times 2^{-22} \end{array}$$

a. Ví dụ phép từ không sử dụng Guard bit

$$\begin{array}{r} x = 1.000\dots00\ 0000 \times 2^1 \\ -y = \underline{0.111\dots11\ 1000} \times 2^1 \\ z = 0.000\dots00\ 1000 \times 2^1 \\ = 1.000\dots00\ 0000 \times 2^{-23} \end{array}$$

b. Ví dụ phép từ có sử dụng Guard bit

Hình 9.25 Guard bit

LÀM TRÒN Một yếu tố khác ảnh hưởng đến độ chính xác của phép toán là việc làm tròn. Kết quả của phép toán thường được lưu trữ trong một thanh ghi dài hơn. Khi chuẩn hóa trở lại định dạng dấu chấm động, các bit thừa ra cần được bỏ bớt, kết quả được làm tròn sao cho sai số nhỏ nhất có thể. Một số kỹ thuật làm tròn được IEEE quy định như sau:

- **Làm tròn đến giá trị gần nhất:** Kết quả được làm tròn đến số biểu diễn gần nhất.
- **Làm tròn về phía $+\infty$:** Kết quả được làm tròn lên.
- **Làm tròn về phía $-\infty$:** Kết quả được làm tròn xuống.
- **Làm tròn về phía 0:** Kết quả được làm tròn về phía 0.

Làm tròn đến giá trị gần nhất là kỹ thuật làm tròn mặc định được chuẩn IEEE quy định và được định nghĩa như sau: Giá trị biểu diễn số đó là giá trị gần với kết quả nhất.

Một trường hợp đặc biệt xảy ra là khi các guard bit có dạng 10000. Giá trị này có thể làm tròn lên hoặc làm tròn xuống đều được. IEEE quy định như sau: Nếu kết quả tính toán nằm ở chính giữa giá trị hai số có thể biểu diễn, nó sẽ được làm tròn lên nếu bit cuối cùng trước phần guard bit hiện tại là 1 và không làm tròn nếu bit đó là 0.

9.5.4. Chuẩn IEEE với các phép toán số học nhị phân dấu chấm động

IEEE 754 không chỉ đưa ra các định nghĩa đơn giản về các định dạng mà còn đặt ra các quy trình và quy ước cụ thể để các phép toán dấu chấm động cho kết quả có thể dự đoán và thống nhất độc lập với nền tảng phần cứng. Vấn đề làm tròn được thảo luận ở trên là một trong những quy ước này. Phần này sẽ đề cập đến một số quy ước khác: vô cùng, các NaN và các số bán tiêu chuẩn.

VÔ CÙNG Vô cùng là khái niệm về giá trị giới hạn của số thực, như sau:

$$-\infty < (\text{các số hữu hạn bất kỳ}) < +\infty$$

Ngoại trừ một số trường hợp đặc biệt được đề cập ở phần sau, phép toán số học bất kỳ với số vô cùng sẽ có kết quả định sẵn như ví dụ sau:

Ví dụ:

$5 + (+\infty) = +\infty$	$5 \div (+\infty) = +0$
$5 - (+\infty) = -\infty$	$(+\infty) + (+\infty) = +\infty$
$5 + (-\infty) = -\infty$	$(-\infty) + (-\infty) = -\infty$
$5 - (-\infty) = +\infty$	$(-\infty) - (+\infty) = -\infty$
$5 \times (+\infty) = +\infty$	$(+\infty) - (-\infty) = +\infty$

QUIET NAN VÀ SIGNALING NAN NaN là một ký hiệu được mã hóa theo định dạng dấu chấm động, trong đó có hai loại: **signaling NaN** và **quiet NaN**. Signaling NaN báo hiệu một phép toán không hợp lệ do một toán hạng nào đó. Signaling NaN được gán cho các biến không được khởi tạo và các phép tăng giống như số học nhưng không được quy định trong chuẩn. Quiet NaN sinh ra và đi qua hầu hết các phép toán số học mà không đưa ra báo hiệu. Bảng 9.7 liệt kê các phép toán sẽ tạo ra quiet NaN.

Bảng 9.7 Các phép toán sinh ra Quiet NaN

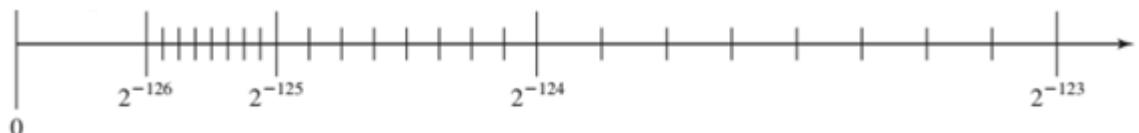
Phép toán	Quiet NaN được sinh ra bởi
Bất kỳ	Bất kỳ phép toán nào với Signaling NaN
Cộng hoặc trừ	Phép trừ độ lớn của vô cùng $(+\infty) + (-\infty)$ $(-\infty) + (+\infty)$ $(+\infty) - (+\infty)$ $(-\infty) - (-\infty)$
Nhân	$0 \times \infty$
Chia	$\frac{0}{0}$ hoặc $\frac{\infty}{\infty}$

Phép chia lấy dư	x chia 0 hoặc ∞ chia y (chia lấy dư)
Lấy căn	\sqrt{x} với $x < 0$

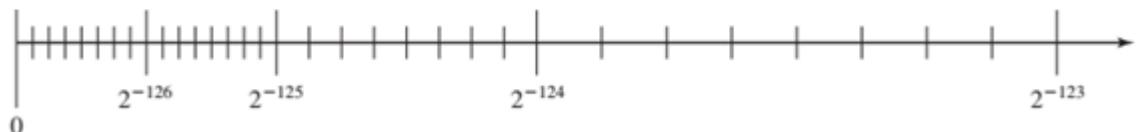
Lưu ý rằng cả hai loại NaN có cùng một định dạng chung (Bảng 10.4): tất cả các bit phần mũ bằng 1 và phần phân số khác không. Giá trị phần phân số được sử dụng để phân biệt signaling NaN và quiet NaN và chỉ ra các trường hợp báo hiệu cụ thể.

CÁC SỐ BÁN TIÊU CHUẨN Các số bán tiêu chuẩn trong IEEE 754 để xử lý các trường hợp tràn dưới phần mũ. Khi phần mũ của kết quả quá nhỏ (số mũ âm với giá trị quá lớn), kết quả sẽ được quy về dạng bán tiêu chuẩn bằng cách dịch phải phần phân số và tăng số mũ cho đến khi số mũ nằm trong phạm vi có thể biểu diễn được.

Hình 9.26 minh họa phân bố các số của định dạng 32-bit gồm cả các số bán tiêu chuẩn. Các số có thể biểu diễn được nhóm lại thành các khoảng $[2^n, 2^{n+1}]$. Trong mỗi khoảng đó, phần số mũ của mỗi số không thay đổi, phần phân số khác nhau, tạo ra khoảng bằng nhau giữa các số có thể biểu diễn được trong khoảng đó. Theo chiều tiến dần đến không, khoảng sau sẽ có chiều rộng bằng một nửa chiều rộng của khoảng trước đó nhưng số các số có thể biểu diễn được thì vẫn bằng nhau. Do đó mật độ số có thể biểu diễn tăng dần lên khi tiến đến không. Tuy nhiên, nếu chỉ mã hóa các số theo chuẩn thì sẽ có một khoảng trống giữa số nhỏ nhất và 0 (Hình 9.26a) không có giá trị nào có thể biểu diễn được. Với định dạng 32-bit IEEE 754, mỗi khoảng biểu diễn được 2^{23} số và số dương nhỏ nhất là 2^{-126} . Với việc quy ước bổ sung các số bán tiêu chuẩn, ta biểu diễn thêm được $2^{23} - 1$ số trong khoảng từ 0 đến 2^{-126} .



a. Định dạng 32-bit không có số bán tiêu chuẩn



b. Định dạng 32-bit có số bán tiêu chuẩn

Hình 9.26 Phân bố số của định dạng nhị phân 32-bit theo chuẩn IEEE 754

9.6. CÂU HỎI

- Giải thích tóm tắt các phương pháp biểu diễn sau: dấu-độ lớn, bù hai, lệch.
- Giải thích cách làm thế nào để xác định xem một số có phải là số âm không trong biểu diễn dấu-độ lớn, bù hai, lệch.
- Nguyên tắc mở rộng phạm vi của số biểu diễn dạng bù hai là gì?

4. Phép phủ định một số bù hai được thực hiện như thế nào?
5. Trường hợp nào phép bù hai đối với một số n-bit cho ta chính số đó?
6. Sự khác biệt giữa biểu diễn bù hai một số và lấy bù hai của một số là gì?
7. Khi thực hiện phép cộng hai số bù hai, ta có thể cộng hai số đó giống như cộng hai số nguyên không dấu, kết quả ta được một số dạng bù hai chính xác. Tuy nhiên điều này không đúng với phép nhân. Tại sao?
8. Bốn thành phần quan trọng của một số biểu diễn dạng dấu chấm động là gì?
9. Lợi ích của việc biểu diễn số mũ lệch là gì?
10. Phân biệt tràn dương, tràn trên phần mũ và tràn phần định trị.
11. Các yếu tố cơ bản của phép cộng và trừ dấu chấm động là gì?
12. Lý do sử dụng các guard bit.
13. Liệt kê các phương pháp làm tròn kết quả của phép toán dấu chấm động.

Chương 10. TẬP LỆNH: ĐẶC ĐIỂM VÀ CHỨC NĂNG

Phần lớn nội dung được thảo luận trong tài liệu này không trực quan, **rõ ràng** đối với lập trình viên hoặc người sử dụng máy tính. Nếu lập trình viên sử dụng một ngôn ngữ bậc cao, như Pascal hay Ada, họ chỉ có thể nhìn thấy được rất ít kiến trúc bên dưới của máy tính.

Tập lệnh máy chính là điểm chung mà người thiết kế máy tính và người lập trình máy tính có thể cùng nhìn thấy khi xem xét một máy tính. Từ quan điểm của người thiết kế, tập lệnh máy cung cấp các yêu cầu chức năng cho bộ vi xử lý: việc thực hiện hoạt động của bộ xử lý gắn liền với việc thực hiện tập lệnh máy. Người sử dụng máy tính muốn lập trình bằng ngôn ngữ máy (hợp ngữ assembly) cần quan tâm đến cấu trúc thanh ghi và bộ nhớ, các loại dữ liệu được máy tính hỗ trợ trực tiếp và chức năng của ALU.

Mô tả về tập lệnh máy sẽ giúp giải thích rõ hơn về bộ xử lý của máy tính. Do đó, chương này và chương kế tiếp sẽ tập trung thảo luận về lệnh máy.

10.1. ĐẶC ĐIỂM CỦA LỆNH MÁY

Hoạt động của bộ xử lý được quyết định bởi các lệnh mà nó thực thi, được gọi là *lệnh máy*. Tập hợp nhiều lệnh khác nhau mà bộ xử lý có thể thực thi được gọi là *tập lệnh* của bộ xử lý.

10.1.1. Các thành phần của lệnh máy

Mỗi lệnh phải chứa những thông tin cần thiết để bộ xử lý có thể thực thi. Hình 10.1, cho thấy các bước trong quá trình thực thi lệnh và đồng thời cũng thể hiện các thành phần của một lệnh máy. Những thành phần đó là:

- **Mã opcode:** Chỉ ra hành động cần thực hiện (ví dụ: ADD, I/O). Hành động cần được thực hiện được xác định bởi một mã nhị phân.
- **Địa chỉ tham chiếu toán hạng nguồn:** Một hành động có thể liên quan đến một hoặc nhiều toán hạng nguồn. Toán hạng nguồn là đầu vào của hành động.
- **Địa chỉ tham chiếu toán hạng đích:** Một hành động có thể tạo ra một kết quả.
- **Địa chỉ tham chiếu lệnh tiếp theo:** Chỉ cho bộ xử lý biết vị trí lệnh tiếp theo cần truy xuất sau khi lệnh này đã hoàn thành.

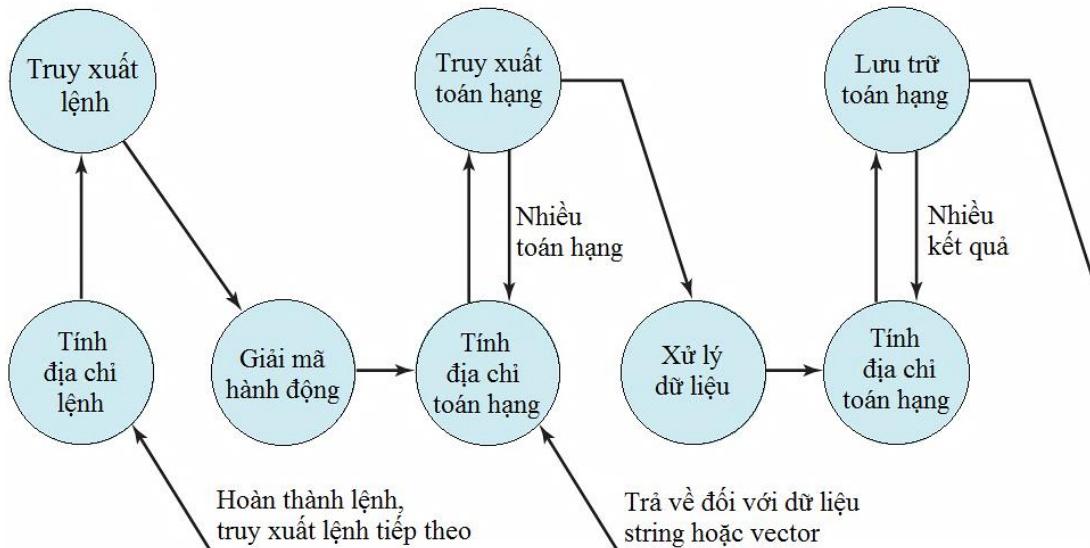
Địa chỉ của lệnh tiếp theo cần truy xuất có thể là địa chỉ thực hoặc địa chỉ ảo, tùy thuộc vào kiến trúc đang sử dụng. Nói chung, sự phân biệt là minh bạch đối với kiến trúc tập lệnh. Trong hầu hết các trường hợp, lệnh tiếp theo sẽ được truy xuất ngay sau lệnh hiện tại. Trong trường hợp này, không có sự tham chiếu rõ ràng tới lệnh tiếp theo. Nếu cần có sự tham chiếu rõ ràng, khi đó địa chỉ bộ nhớ chính hoặc địa chỉ bộ nhớ ảo phải được cung cấp. Hình thức cung cấp địa chỉ đó sẽ được thảo luận trong Chương 11.

Các toán hạng nguồn và đích có thể nằm trong các vùng sau:

- **Bộ nhớ chính/bộ nhớ ảo:** Phải cung cấp địa chỉ bộ nhớ chính/ảo để phục vụ cho các tham chiếu lệnh tiếp theo.
- **Thanh ghi trong bộ xử lý:** Một bộ xử lý chứa một hoặc nhiều thanh ghi, các thanh ghi này có thể được các lệnh tham chiếu đến. Nếu chỉ có duy nhất một thanh ghi, tham chiếu đến thanh ghi đó có thể là ngầm định. Nếu có nhiều thanh ghi, mỗi

thanh ghi phải được gán một tên gọi hoặc chỉ số riêng. Khi đó, lệnh tham chiếu đến thanh ghi nào phải chứa chỉ số của thanh ghi đó.

- **Tức thời:** Giá trị của toán hạng đặt trong một trường nhất định trong lệnh đang thực hiện.
- **Thiết bị vào/ra:** Lệnh phải chỉ ra thiết bị và mô-đun vào/ra liên quan đến hành động. Nếu sử dụng I/O ánh xạ bộ nhớ thì địa chỉ cần xác định chính là một địa chỉ bộ nhớ chính/ảo khác.

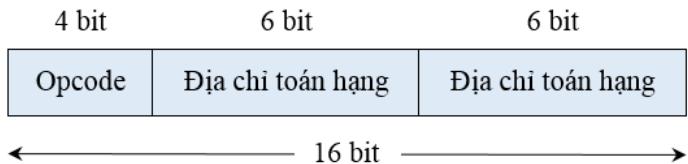


Hình 10.1 Sơ đồ trạng thái chu kỳ lệnh

10.1.2. Biểu diễn lệnh

Trong máy tính, mỗi lệnh được biểu diễn bằng một chuỗi các bit. Lệnh được chia ra thành các trường tương ứng với các thành phần cấu thành nên lệnh. Chúng ta đã xét một ví dụ về định dạng lệnh ở máy IAS trong hình 2.2. Hình 10.2 cho ta một ví dụ khác về định dạng lệnh. Đa số các tập lệnh đều sử dụng nhiều định dạng lệnh. Trong quá trình thực thi lệnh, một lệnh sẽ được đọc vào thanh ghi lệnh (IR) trong bộ xử lý. Bộ xử lý phải có khả năng trích xuất dữ liệu từ các trường lệnh khác nhau để thực hiện thao tác yêu cầu. Sẽ rất khó khăn cho lập trình viên và người học nếu phải làm việc với các lệnh máy biểu diễn ở dạng nhị phân. Do đó, việc biểu diễn lệnh máy ở dạng ký tự gợi nhớ đã được sử dụng và phổ biến trong thực tế. Ví dụ như *biểu diễn ký tự gợi nhớ* được sử dụng cho tập lệnh IAS, trong Bảng 2.1. Opcode được biểu diễn bằng các từ viết tắt, gợi nhớ về hành động tương ứng. Ví dụ,

ADD	Cộng
SUB	Trừ
MUL	Nhân
DIV	Chia
LOAD	Tải dữ liệu từ bộ nhớ
STOR	Lưu dữ liệu vào bộ nhớ



Hình 10.2 Một định dạng lệnh đơn giản

Các toán tử cũng được biểu diễn bằng ký tự. Ví dụ, lệnh

ADD R, Y

có nghĩa là cộng giá trị trong vị trí dữ liệu Y với giá trị nội dung thanh ghi R. Trong ví dụ này, Y là địa chỉ của một vị trí trong bộ nhớ, và R là một thanh ghi cụ thể. Lưu ý rằng hành động này được thực hiện trên nội dung của một vị trí chứ không phải trên địa chỉ của nó.

Do đó, một chương trình ngôn ngữ máy có thể được tạo ra gồm các câu lệnh dạng ký tự. Mỗi opcode ký tự gắn với một biểu diễn nhị phân cố định, và lập trình viên sẽ xác định vị trí của mỗi toán hạng ký tự. Ví dụ, lập trình viên có thể bắt đầu với một loạt các khai báo:

$X = 513$

$Y = 514$

Một chương trình đơn giản sẽ nhận đầu vào dạng ký tự này, chuyển đổi các opcode và địa chỉ tham chiếu toán hạng sang dạng nhị phân và tạo ra các lệnh máy nhị phân.

Rất hiếm, thậm chí không có, lập trình viên lập trình bằng ngôn ngữ máy. Hầu hết các chương trình hiện nay đều được viết bằng ngôn ngữ bậc cao hoặc chí ít là hợp ngữ assembly. Tuy nhiên, ngôn ngữ máy dạng ký tự vẫn là một công cụ hữu ích để mô tả lệnh máy.

10.1.3. Phân loại lệnh

Giả sử một lệnh ngôn ngữ bậc cao có thể được diễn đạt bằng một ngôn ngữ như BASIC hay FORTRAN. Ví dụ,

$X = X + Y$

Câu lệnh này chỉ thị cho máy tính cộng giá trị lưu trữ trong Y với giá trị được lưu trữ trong X và đặt kết quả vào X. Hành động này có thể được thực hiện với lệnh máy như thế nào? Giả sử rằng các biến số X và Y tương ứng với các vị trí 513 và 514. Giả sử sử dụng một tập lệnh máy đơn giản, hành động này có thể được thực hiện bằng ba câu lệnh:

1. Tải nội dung của vị trí bộ nhớ 513 vào một thanh ghi.
2. Cộng nội dung của vị trí bộ nhớ 514 vào thanh ghi đó.
3. Lưu trữ nội dung của thanh ghi vào vị trí bộ nhớ 513.

Có thể thấy, một lệnh đơn BASIC lại đòi hỏi đến ba câu lệnh máy. Đây là đặc điểm điển hình của mối quan hệ giữa ngôn ngữ bậc cao và ngôn ngữ máy. Ngôn ngữ bậc cao diễn đạt hành động ở dạng đại số ngắn gọn, sử dụng các biến số. Ngôn ngữ máy diễn đạt hành động ở dạng cơ bản gắn với sự di chuyển của dữ liệu đến hoặc từ thanh ghi.

Chúng ta hãy xem xét các loại lệnh có trong một máy tính thực tế. Máy tính cần có một tập hợp các lệnh cho phép người sử dụng tạo ra mọi thao tác xử lý dữ liệu cần thiết. Hoặc cách khác, ta xem xét khả năng của ngôn ngữ lập trình bậc cao. Bất kỳ chương trình viết bằng ngôn ngữ bậc cao nào cũng phải được dịch sang ngôn ngữ máy để được thực hiện. Do đó, tập lệnh máy phải đủ lớn để diễn đạt được mọi lệnh trong ngôn ngữ bậc cao. Chúng ta có thể phân loại các loại lệnh như sau:

- **Xử lý dữ liệu:** Lệnh số học và lệnh logic
- **Lưu trữ dữ liệu:** Chuyển dữ liệu vào hoặc ra khỏi thanh ghi và/hoặc vị trí bộ nhớ
- **Di chuyển dữ liệu:** Lệnh vào/ra
- **Điều khiển:** Lệnh kiểm tra và lệnh rẽ nhánh

Lệnh số học cung cấp khả năng tính toán để xử lý dữ liệu số. *Lệnh logic* (Boolean) thực hiện trên từng bit của một word chứ không thực hiện trên từng số, do đó chúng cung cấp khả năng xử lý bất kỳ loại dữ liệu nào mà người dùng muốn. Các hành động này được thực hiện chủ yếu trên dữ liệu trong thanh ghi bộ xử lý. Vì vậy, phải có các lệnh bộ nhớ để di chuyển dữ liệu giữa bộ nhớ và thanh ghi. *Lệnh vào/ra* được sử dụng để truyền chương trình và dữ liệu vào bộ nhớ, và truyền kết quả tính toán lại cho người dùng. *Lệnh kiểm tra* được sử dụng để kiểm tra giá trị của một word dữ liệu hoặc trạng thái của một phép toán. *Lệnh rẽ nhánh* được sử dụng để rẽ nhánh tới một nhóm câu lệnh khác dựa vào quyết định được đưa ra.

Chúng ta sẽ xem xét các loại lệnh cụ thể hơn trong phần sau của chương này.

10.1.4. Số lượng địa chỉ trong lệnh

Mô tả số lượng địa chỉ trong mỗi lệnh là một trong những cách truyền thống để mô tả kiến trúc bộ xử lý. Khi thiết kế bộ xử lý ngày càng phức tạp, vai trò của yếu tố này đã giảm dần. Tuy nhiên, việc phân tích số lượng địa chỉ trong lệnh lại rất hữu ích cho người học tại thời điểm này.

Vậy số lượng địa chỉ tối đa cần có trong một lệnh là bao nhiêu? Rõ ràng, lệnh số học và lệnh logic sẽ yêu cầu nhiều toán hạng nhất. Hầu như tất cả các phép tính số học và logic đều có một hoặc hai toán hạng nguồn. Do đó, chúng ta cần tối đa hai địa chỉ để tham chiếu toán hạng nguồn. Kết quả của một hành động phải được lưu trữ, vì vậy cần có một địa chỉ thứ ba, tham chiếu toán hạng đích. Cuối cùng, sau khi hoàn thành một lệnh, lệnh tiếp theo phải được truy xuất, và cần phải có địa chỉ của nó.

Từ đó, ta thấy rằng một lệnh cần phải có tối đa bốn tham chiếu địa chỉ: hai toán hạng nguồn, một toán hạng đích, và địa chỉ của lệnh tiếp theo. Trong hầu hết các kiến trúc, các lệnh thường có một, hai hoặc ba địa chỉ toán hạng, còn địa chỉ của lệnh tiếp theo được ngầm định (lấy từ bộ đếm chương trình). Hầu hết các kiến trúc cũng có một vài lệnh chuyên dụng với nhiều toán hạng hơn. Ví dụ, việc tải và lưu trữ nhiều lệnh của kiến trúc ARM, **được mô tả trong Chương 11**, có 17 toán hạng thanh ghi trong một lệnh duy nhất.

Hình 10.3 lần lượt so sánh các lệnh có một, hai và ba địa chỉ được dùng để tính biểu thức $Y = (A - B) > [C + (D \times E)]$. Với ba địa chỉ, mỗi lệnh xác định hai vị trí toán hạng nguồn và một vị trí toán hạng đích. Bởi vì chúng ta giả thiết không thay đổi giá trị của bất

kỳ vị trí toán hạng nào, một vị trí tạm thời T được sử dụng để lưu trữ một số kết quả trung gian. Lưu ý rằng có tất cả bốn lệnh và biểu thức ban đầu có năm toán hạng.

Lệnh	Giải thích
SUB Y, A, B	$Y \leftarrow A - B$
MPY T, D, E	$T \leftarrow D \times E$
ADD T, T, C	$T \leftarrow T + C$
DIV Y, Y, T	$Y \leftarrow Y \div T$

(a) Lệnh có ba địa chỉ

Lệnh	Giải thích
MOVE Y, A	$Y \leftarrow A$
SUB Y, B	$Y \leftarrow Y - B$
MOVE T, D	$T \leftarrow D$
MPY T, E	$T \leftarrow T \times E$
ADD T, C	$T \leftarrow T + C$
DIV Y, T	$Y \leftarrow Y \div T$

(b) Lệnh có hai địa chỉ

Lệnh	Giải thích
LOAD D	$AC \leftarrow D$
MPY E	$AC \leftarrow AC \times E$
ADD C	$AC \leftarrow AC + C$
STOR Y	$Y \leftarrow AC$
LOAD A	$AC \leftarrow A$
SUB B	$AC \leftarrow AC - B$
DIV Y	$AC \leftarrow AC \div Y$
STOR Y	$Y \leftarrow AC$

(c) Lệnh có một địa chỉ

$$\text{Hình 10.3} \quad \text{Chương trình tính biểu thức } Y = \frac{A - B}{C + (D \times E)}$$

Định dạng lệnh ba địa chỉ không phổ biến vì để chứa được ba tham chiếu địa chỉ chúng đòi hỏi định dạng lệnh tương đối dài. Với lệnh hai địa chỉ, và đối với các xử lý nhị phân, một địa chỉ sẽ phải đảm nhiệm gấp đôi nhiệm vụ, vừa cho toán hạng nguồn vừa cho kết quả đích. Do đó, lệnh SUB Y, B thực hiện phép tính Y - B và lưu trữ kết quả vào Y. Định dạng hai địa chỉ làm giảm độ dài lệnh nhưng cũng gây ra một số phiền toái. Để tránh thay đổi giá trị của toán hạng, lệnh MOVE được sử dụng để di chuyển một trong các giá trị đó đến vị trí kết quả hoặc một vị trí tạm thời trước khi thực hiện xử lý. Chương trình ví dụ lúc này mở rộng lên tới sáu lệnh.

Đơn giản hơn nữa là lệnh một địa chỉ. Với định dạng này, địa chỉ thứ hai phải được ngầm định. Điều này rất phổ biến trong các máy trước đây, với địa chỉ ngầm định là một thanh ghi bộ xử lý được gọi là AC (accumulator). Thanh ghi AC chứa một trong hai toán hạng và được sử dụng để lưu trữ kết quả. Trong ví dụ của chúng ta, cần tổng cộng tám lệnh để hoàn thành tính toán Y.

Trên thực tế, một số lệnh có thể thực hiện mà không cần địa chỉ. Lệnh không địa chỉ được áp dụng cho một tổ chức bộ nhớ đặc biệt được gọi là ngăn xếp. Ngăn xếp là một tập

hợp các vị trí LIFO (vào sau ra trước). Ngăn xếp ở một vị trí đã biết, và ít nhất hai phần tử trên cùng của ngăn xếp thường nằm trong thanh ghi bộ xử lý. Do đó, lệnh không địa chỉ sẽ tham chiếu đến hai phần tử ở đỉnh ngăn xếp. Việc sử dụng ngăn xếp sẽ được trình bày chi tiết hơn trong chương này và chương 11.

Bảng 10.1 Cách sử dụng địa chỉ lệnh (Lệnh không rẽ nhánh)

Số lượng địa chỉ	Biểu diễn ký tự	Giải thích
3	OP A, B, C	A ← B OP C
2	OP A, B	A ← A OP B
1	OP A	AC ← AC OP A
0	OP	T ← (T - 1) OP T

AC = Thanh ghi tích lũy

T = đỉnh ngăn xếp

(T - 1) = phần tử thứ hai trong ngăn xếp

A, B, C = các vị trí bộ nhớ hoặc thanh ghi

Bảng 10.1 tóm tắt cách giải thích các lệnh có không, một, hai hoặc ba địa chỉ. Trong các trường hợp trong bảng, giả sử rằng địa chỉ của lệnh kế tiếp là ngầm định, và một phép toán với hai toán hạng nguồn và một toán hạng kết quả sẽ được thực hiện.

Số lượng địa chỉ trong câu lệnh là một yếu tố thiết kế cơ bản. Càng ít địa chỉ trong lệnh dẫn tới các lệnh đơn giản hơn, đòi hỏi bộ xử lý ít phức tạp hơn. Nó cũng làm cho lệnh ngắn gọn hơn. Mặt khác, chương trình lại có tổng cộng nhiều lệnh hơn, khiến cho thời gian thực hiện lâu hơn, chương trình phức tạp và dài hơn. Ngoài ra, còn có sự khác biệt quan trọng giữa lệnh một địa chỉ và nhiều địa chỉ. Với lệnh một địa chỉ, lập trình viên thường chỉ có sẵn một thanh ghi đa năng là AC. Với lệnh nhiều địa chỉ, thường có nhiều thanh ghi đa năng hơn. Điều này cho phép một số hành động chỉ cần thực hiện trong thanh ghi. Bởi vì tham chiếu thanh ghi nhanh hơn nhiều so với tham chiếu bộ nhớ, tốc độ thực hiện lệnh sẽ tăng lên. Nhằm có được tính linh hoạt và khả năng sử dụng nhiều thanh ghi, hầu hết các máy tính hiện đại sử dụng đồng thời các lệnh hai và ba địa chỉ.

Việc lựa chọn số lượng địa chỉ cho mỗi lệnh trong quá trình thiết kế còn phức tạp bởi các yếu tố khác. Có một câu hỏi đặt ra là khi nào thì tham chiếu đến địa chỉ một vị trí bộ nhớ và khi nào thì tham chiếu địa chỉ một thanh ghi. Bởi số lượng thanh ghi là rất ít (so với số lượng vị trí bộ nhớ), nên địa chỉ tham chiếu thanh ghi cần ít bit biểu diễn hơn. Ngoài ra, như chúng ta sẽ thấy trong Chương 11, một máy có thể có nhiều chế độ địa chỉ khác nhau, và việc chỉ định chế độ cụ thể sẽ cần dùng một hoặc nhiều bit. Kết quả là hầu hết các thiết kế bộ xử lý đều có nhiều định dạng lệnh khác nhau.

10.1.5. Các vấn đề thiết kế tập lệnh

Một trong những khía cạnh thú vị nhất và thường được phân tích nhất trong thiết kế máy tính là thiết kế tập lệnh. Việc thiết kế tập lệnh là rất phức tạp vì nó có ảnh hưởng đến rất nhiều khía cạnh khác của hệ thống máy tính. Tập lệnh định nghĩa nhiều chức năng mà bộ xử lý thực hiện và do đó có ảnh hưởng đáng kể đến việc thực hiện của bộ xử lý. Tập lệnh là phương tiện để lập trình viên có thể điều khiển bộ xử lý. Do đó, các yêu cầu của lập trình viên phải được xem xét khi thiết kế tập lệnh.

Có một thực tế là một số vấn đề cơ bản nhất trong thiết kế tập lệnh vẫn chưa được thống nhất. Trong những năm gần đây, mức độ bất đồng liên quan đến các vấn đề cơ bản này dần tăng lên. Các vấn đề thiết kế cơ bản quan trọng nhất bao gồm:

- **Danh sách các hành động:** Bao nhiêu hành động? Hành động nào? Độ phức tạp của hành động?
- **Kiểu dữ liệu:** Hành động được thực hiện trên kiểu dữ liệu nào
- **Định dạng lệnh:** chiều dài lệnh (theo bit), số lượng địa chỉ, kích thước các trường khác nhau, v.v ...
- **Thanh ghi:** Số lượng thanh ghi của bộ xử lý mà lệnh có thể tham chiếu đến? Cách sử dụng thanh ghi?
- **Địa chỉ:** Một hoặc nhiều chế độ địa chỉ được dùng để xác định địa chỉ của toán hạng

Những vấn đề này có mối liên hệ rất chặt chẽ và phải được xem xét cùng nhau khi thiết kế tập lệnh.

Có thể nói, các vấn đề thiết kế tập lệnh là rất quan trọng, do đó chúng sẽ được thảo luận chi tiết hơn trong các nội dung tiếp theo. Sau mục tổng quan này, chương này sẽ khảo sát các kiểu dữ liệu và danh sách hoạt động. Chương 11 trình bày về các chế độ địa chỉ (đồng thời đề cập cả về thanh ghi) và các định dạng lệnh. Chương 13 giới thiệu về máy tính tập lệnh rút gọn (RISC).

10.2. CÁC KIỂU TOÁN HẠNG

Lệnh máy thao tác trên dữ liệu. Các kiểu dữ liệu quan trọng nhất là

- Địa chỉ
- Số
- Ký tự
- Dữ liệu logic

Chúng ta sẽ thấy, trong các thảo luận về chế độ địa chỉ ở chương 11, rằng địa chỉ thực ra cũng là một dạng dữ liệu. Trong nhiều trường hợp, tham chiếu toán hạng trong một lệnh phải được tính toán (bằng một số phép tính) để xác định địa chỉ bộ nhớ chính hoặc ảo. Trong ngữ cảnh này, địa chỉ có thể được coi là số nguyên không dấu.

Các kiểu dữ liệu phổ biến khác là số, ký tự, và dữ liệu lôgic, và mục này sẽ trình bày ngắn gọn về từng kiểu.

10.2.1. Số

Tất cả các ngôn ngữ máy đều có kiểu dữ liệu dạng số. Ngay cả trong xử lý dữ liệu không phải dạng số, vẫn cần phải có số để thực hiện bộ đếm, độ rộng trường, vv. Một điểm khác biệt quan trọng giữa các con số được sử dụng trong toán học thông thường và số được lưu trữ trong máy tính là số trong máy tính bị giới hạn. Điều này có thể hiểu theo hai nghĩa. Thứ nhất, độ lớn của số biểu diễn được trên máy bị giới hạn và thứ hai, trong trường hợp số dấu chấm động, độ chính xác của số cũng bị hạn chế. Vì vậy, lập trình viên phải hiểu biết rõ về những ảnh hưởng của việc làm tròn, tràn trên (overflow), và tràn dưới (underflow).

Ba kiểu dữ liệu số thông thường trong máy tính là:

- Số nguyên nhị phân hoặc số thực dấu chấm tĩnh nhị phân
- Số thực dấu chấm động nhị phân
- Số thập phân

Hai kiểu dữ liệu số đầu tiên đã được trình bày trong chương 9. Ở đây, ta thảo luận thêm một chút về số thập phân.

Mặc dù bản chất tất cả các hoạt động bên trong máy tính là nhị phân, con người sử dụng hệ thống vẫn cần làm việc với số thập phân. Vì vậy, việc chuyển đổi đầu vào từ thập phân sang nhị phân và đầu ra từ nhị phân sang thập phân là rất cần thiết. Đối với các ứng dụng có rất nhiều hoạt động vào/ra và tương đối ít tính toán hay tính toán tương đối đơn giản, việc lưu trữ và xử lý trên các con số sẽ hiệu quả hơn nếu số ở dạng thập phân. Dạng biểu diễn phổ biến nhất cho mục đích này là **packed decimal** (thập phân đóng gói).

Trong packed decimal, mỗi *chữ số thập phân* được biểu diễn bằng một mã 4 bit, hai chữ số được lưu trữ thành một byte. Theo đó, 0 = 0000, 0001,..., 8 = 1000 và 9 = 1001. Lưu ý rằng đây là kiểu mã không hiệu quả vì chỉ sử dụng 10 trong 16 tổ hợp 4 bit có thể có. Để tạo thành số, các mã 4 bit được ghép với nhau, số bit trong một số thường là bội số của 8 bit. Ví dụ, số 246 được biểu diễn là 0000 0010 0100 0110. Mã này rõ ràng là cồng kềnh hơn so với biểu diễn nhị phân bình thường, nhưng nó giảm bớt được độ phức tạp trong chuyển đổi. Các số âm có thể được biểu diễn bằng cách thêm 4 bit dấu vào ngoài cùng bên trái hoặc bên phải của chuỗi các chữ số packed decimal. Giá trị dấu chuẩn là 1100 đối với dấu dương (+) và 1101 đối với dấu âm (-).

Nhiều máy tính cung cấp các lệnh số học để thực hiện xử lý trực tiếp trên số packed decimal. Thuật toán sẽ tương tự như các thuật toán **đã mô tả trong mục 9.3**, tuy nhiên phải có thêm phép nhớ thập phân.

10.2.2. Ký tự

Dữ liệu cũng thường ở dạng văn bản hoặc chuỗi ký tự. Mặc dù dữ liệu dạng văn bản là tiện lợi nhất cho con người, việc lưu trữ hoặc truyền dữ liệu này qua hệ thống xử lý dữ liệu và truyền thông không thể thực hiện được một cách dễ dàng. Các hệ thống đó chỉ được thiết kế cho dữ liệu nhị phân. Do đó, một số loại mã đã được tạo ra sao cho mỗi ký tự được biểu diễn bởi một chuỗi bit. Mã Morse có lẽ là xuất hiện sớm nhất và là ví dụ rất quen thuộc cho điều này. Ngày nay, mã ký tự IRA (International Reference Alphabet) được sử dụng phổ biến nhất là mã ASCII (American Standard Code for Information Interchange).

Mỗi ký tự trong mã này được biểu diễn bởi một mẫu 7 bit duy nhất; do đó, có thể biểu diễn được tổng cộng 128 ký tự khác nhau. Con số này lớn hơn số lượng cần thiết để biểu diễn các ký tự in được, bởi một số mẫu 7 bit được dùng để biểu diễn ký tự điều khiển. Một số ký tự điều khiển liên quan đến điều khiển việc in các ký tự trên một trang; một số khác liên quan đến các thủ tục truyền thông. Các ký tự mã hoá IRA hầu như luôn được lưu trữ và truyền đi theo từng nhóm 8 bit cho mỗi ký tự. Bit thứ tám có thể được đặt thành 0 hoặc được sử dụng như là bit chẵn lẻ (parity) để phát hiện lỗi. Trong trường hợp thứ hai, bit được đặt sao cho tổng số các số 1 nhị phân trong mỗi cụm 8 bit luôn lẻ (parity lẻ) hoặc luôn chẵn (parity chẵn).

Lưu ý, trong bảng mã ASCII, các chữ số thập phân từ 0 đến 9 được biểu diễn bằng mẫu bit 011XXXX với 4 bit ngoài cùng bên phải là các số nhị phân tương ứng từ 0000 đến 1001. Tức là, thập phân 0 tương ứng 0110000, 1 tương ứng 0110001, ..., 9 tương ứng 0111001. Điều này rất giống với mã packed decimal. Chính vì vậy, việc chuyển đổi giữa biểu diễn ASCII 7 bit và biểu diễn thập phân 4 bit rất dễ dàng.

Một mã khác được sử dụng để mã hoá ký tự là mã EBCDIC (Extended Binary Coded Decimal Interchange Code). EBCDIC được sử dụng trên máy mainframe của IBM. Nó là một mã 8 bit. Giống như IRA, EBCDIC tương thích với số packed decimal. Trong trường hợp EBCDIC, các mã từ 11110000 đến 11111001 biểu diễn các chữ số từ 0 đến 9.

10.2.3. Dữ liệu logic

Thông thường, mỗi từ hoặc mỗi đơn vị địa chỉ khác (byte, halfword, v.v...) được coi như một đơn vị dữ liệu. Tuy nhiên, đôi khi ta cần xem xét một đơn vị dữ liệu n bit như là n phần dữ liệu 1 bit riêng, mỗi phần có giá trị 0 hoặc 1. Khi dữ liệu được xem xét theo cách này, chúng được gọi là dữ liệu logic.

Việc xét dữ liệu theo hướng bit có hai ưu điểm. Thứ nhất, đôi khi chúng ta mong muốn lưu trữ một mảng các phần tử dữ liệu nhị phân hoặc Boolean, trong đó mỗi phần tử chỉ có thể nhận giá trị 1 (true) và 0 (false). Với dữ liệu logic, bộ nhớ có thể được sử dụng hiệu quả nhất cho kiểu lưu trữ này. Thứ hai, đôi khi chúng ta cũng mong muốn thao tác trên các bit của một dữ liệu. Ví dụ, nếu các phép tính dấu chấm động được thực hiện trong phần mềm, ta cần phải có khả năng dịch bit trong một số thao tác. Một ví dụ khác: Để chuyển đổi từ IRA sang dạng packed decimal, chúng ta cần trích xuất 4 bit ngoài cùng bên phải của mỗi byte.

Lưu ý rằng, trong các ví dụ trước, cùng một dữ liệu đôi khi được xử lý như là logic và đôi khi lại như là số hoặc văn bản. "Kiểu" của một đơn vị dữ liệu được xác định bởi thao tác đang được thực hiện trên nó. Mặc dù điều này hầu như luôn gặp phải trong ngôn ngữ máy, nó không phải là trường hợp thường gặp trong các ngôn ngữ bậc cao.

10.3. KIỂU DỮ LIỆU TRONG INTEL X86 VÀ ARM

10.3.1. Các kiểu dữ liệu trong Intel x86

x86 có thể xử lý các kiểu dữ liệu với độ dài bit là 8 (byte), 16 (word), 32 (doubleword), 64 (quadword) và 128 (đôi quadword). Để có cấu trúc dữ liệu linh hoạt tối đa và việc sử dụng bộ nhớ hiệu quả, các word không cần phải bắt đầu ở các địa chỉ số chẵn; doubleword không cần phải bắt đầu ở các địa chỉ chia hết cho 4; và quadword không

cần phải bắt đầu ở các địa chỉ chia hết cho 8; v.v... Cũng giống như tất cả các máy tính Intel 80x86, x86 sử dụng kiểu little-endian; nghĩa là, byte ít quan trọng nhất được lưu giữ ở địa chỉ nhỏ nhất.

Byte, word, doubleword, quadword, và double quadword được gọi là các kiểu dữ liệu tổng quát. Ngoài ra, x86 còn hỗ trợ nhiều kiểu dữ liệu cụ thể được nhận biết và hoạt động theo các lệnh cụ thể. Bảng 10.2 tóm tắt các kiểu dữ liệu này.

Bảng 10.2 Các kiểu dữ liệu trong Intel x86

Kiểu dữ liệu	Mô tả
Tổng quát	Byte, word (16 bit), doubleword (32 bit), quadword (64 bit), và double quadword (128 bit) với các nội dung nhị phân tùy ý
Số nguyên	Một giá trị nhị phân có dấu có độ dài byte, word, hoặc doubleword, sử dụng biểu diễn bù hai.
Ordinal	Một số nguyên không dấu có độ dài byte, word, hoặc doubleword.
BCD	Biểu diễn chữ số BCD từ 0 đến 9, mỗi chữ số trong một byte.
Packed BCD	Biểu diễn đóng gói hai chữ số BCD (mỗi chữ số 4 bit, gộp hai chữ số trong một byte); giá trị trong khoảng từ 0 đến 99.
Near pointer (Con trỏ gần)	Địa chỉ hiệu dụng 16 bit, 32 bit hoặc 64 bit biểu diễn giá trị offset trong một segment. Được sử dụng cho tất cả các con trỏ trong bộ nhớ không phân đoạn và dùng trong tham chiếu một segment trong bộ nhớ phân đoạn.
Far pointer (Con trỏ xa)	Địa chỉ logic bao gồm segment selector 16 bit và offset 16, 32 hoặc 64 bit. Con trỏ xa được sử dụng cho tham chiếu bộ nhớ trong mô hình bộ nhớ phân đoạn, do trong mô hình này, nhận dạng của segment cần truy cập phải được xác định rõ ràng.
Bit field	Một dãy bit liên tục, trong đó vị trí của mỗi bit được xem như là một đơn vị độc lập. Một dãy bit có thể bắt đầu tại bất kỳ vị trí bit nào trong một byte bất kỳ và có thể chứa đến 32 bit.
Bit string	Một dãy bit liên tục chứa từ 0 đến $2^{32} - 1$ bit.
Byte string	Một dãy byte, word, hoặc doubleword liên tục, chứa từ 0 đến $2^{32} - 1$ byte.
Dấu chấm động	Xem hình 10.4
Packed SIMD	SIMD (single instruction, multiple data) là một lệnh, nhiều dữ liệu. Các kiểu dữ liệu 64 bit và 128 bit đóng gói.

Hình 10.4 minh họa các kiểu dữ liệu trong x86. Các số nguyên có dấu được biểu diễn ở dạng bù hai và có thể dài 16, 32 hoặc 64 bit. Kiểu dấu chấm động thực ra là đê cập đến

một tập hợp các kiểu dữ liệu được sử dụng bởi khói xử lý dấu chấm động và để thực hiện trên các lệnh dấu chấm động. Ba dạng biểu diễn dấu chấm động này tuân theo tiêu chuẩn IEEE 754.

10.3.1. Các kiểu dữ liệu trong ARM

Bộ xử lý ARM hỗ trợ cho các kiểu dữ liệu có độ dài bit là 8 (byte), 16 (halfword) và 32 (word). Cả ba kiểu dữ liệu byte, halfword và word đều hỗ trợ việc biểu diễn không dấu để biểu diễn một số nguyên không âm, không dấu. Cả ba kiểu dữ liệu này cũng có thể được sử dụng để biểu diễn số nguyên có dấu, bù hai.

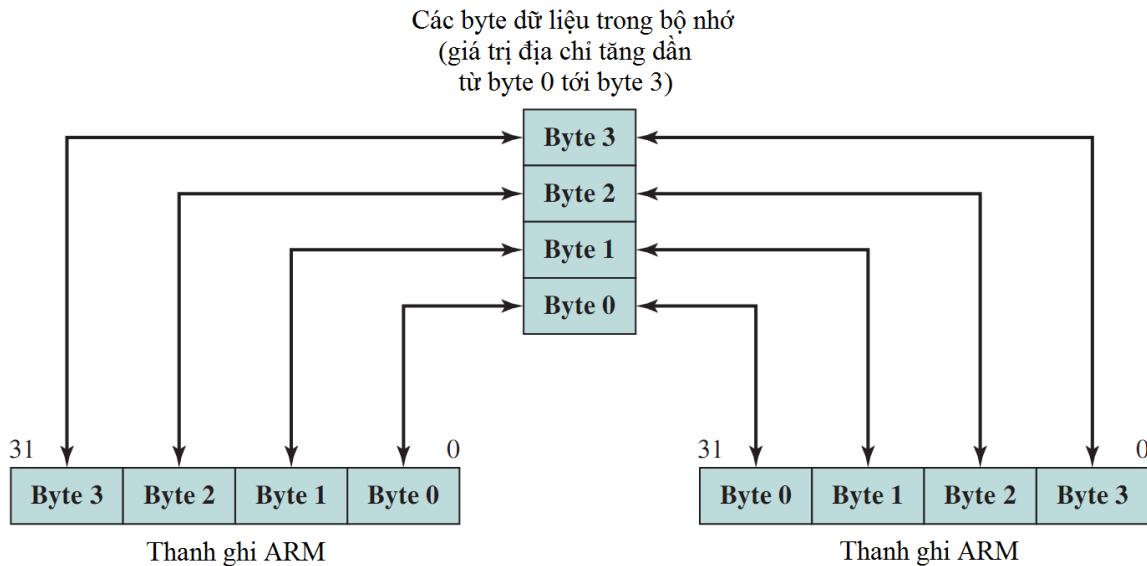
Phần lớn bộ xử lý ARM không cung cấp phần cứng xử lý dấu chấm động. Nếu bộ xử lý ARM được yêu cầu xử lý số học dấu chấm động, việc xử lý này phải được thực hiện trong phần mềm (mặc dù nếu xử lý bằng phần cứng thì tiết kiệm năng lượng và diện tích hơn). ARM hỗ trợ một bộ đồng xử lý dấu chấm động tùy chọn để hỗ trợ các kiểu dữ liệu dấu chấm động độ chính xác đơn và gấp đôi được định nghĩa trong IEEE 754.



Hình 10.4 Các kiểu dữ liệu trong x86

10.3.2. Hỗ trợ Endian

Một bit trạng thái (bit E) trong thanh ghi điều khiển hệ thống được thiết lập và xóa bằng sự điều khiển của chương trình sử dụng lệnh SETEND. Bit E xác định sẽ tải và lưu trữ dữ liệu vào endian nào. Hình 10.5 mô tả các chức năng liên quan đến bit E đối với một hành động tải hoặc lưu trữ một word. Cơ chế này đem lại khả năng tải/lưu trữ dữ liệu hiệu quả cho các nhà thiết kế hệ thống, những người biết rằng họ cần phải truy cập vào các cấu trúc dữ liệu theo chiều endian ngược lại với môi trường/hệ điều hành của chúng. Lưu ý rằng địa chỉ của mỗi byte dữ liệu được cố định trong bộ nhớ. Tuy nhiên, **byte lane** trong thanh ghi lại khác nhau.



Hình 10.5 Hỗ trợ Endian trong ARM – Tải/lưu trữ word với bit E

10.4. CÁC LOẠI HÀNH ĐỘNG

Tổng số lượng opcode trên mỗi máy là rất khác nhau. Tuy nhiên, các loại hành động tổng quát lại giống nhau và có thể tìm thấy trên tất cả các máy. Chúng được phân loại thành các loại điển hình như sau:

- Truyền dữ liệu
- Xử lý số học
- Xử lý logic
- Điều khiển vào/ra
- Chuyển điều khiển (rẽ nhánh)
- Điều khiển hệ thống

Bảng 10.3 liệt kê các câu lệnh phổ biến của mỗi loại hành động. Mục này sẽ trình bày ngắn gọn về các loại hành động này, đồng thời thảo luận ngắn gọn về hoạt động của bộ xử lý để thực hiện từng loại hành động cụ thể (tóm tắt trong Bảng 10.4).

Bảng 10.3 Các hành động của tập lệnh

Loại	Lệnh	Mô tả
Truyền dữ liệu	MOVE	Copy dữ liệu từ nguồn đến đích
	LOAD	Nạp dữ liệu từ bộ nhớ đến bộ xử lý
	STORE	Cắt dữ liệu từ bộ xử lý đến bộ nhớ
	EXCHANGE	Trao đổi nội dung của nguồn và đích
	CLEAR	Chuyển các bit 0 vào toán hạng đích
	SET	Chuyển các bit 1 vào toán hạng đích
	PUSH	Cắt nội dung toán hạng nguồn vào ngăn xếp
	POP	Lấy nội dung đỉnh ngăn xếp đưa đến toán hạng đích
Xử lý số học	ADD	Cộng hai toán hạng
	SUBTRACT	Trừ hai toán hạng
	MULTIPLY	Nhân hai toán hạng
	DIVIDE	Chia hai toán hạng
	ABSOLUTE	Lấy trị tuyệt đối toán hạng
	NEGATE	Đổi dấu toán hạng (lấy bù 2)
	INCREMENT	Tăng toán hạng thêm 1
	DECREMENT	Giảm toán hạng đi 1
Xử lý logic	AND	Thực hiện phép AND hai toán hạng
	OR	Thực hiện phép OR hai toán hạng
	NOT	Đảo bit của toán hạng (lấy bù 1)
	XOR	Thực hiện phép XOR hai toán hạng
	TEST	Kiểm tra điều kiện cụ thể; thiết lập cờ dựa trên kết quả
	COMPARE	So sánh logic hoặc số học của hai hoặc nhiều toán hạng; thiết lập cờ dựa trên kết quả
	SHIFT	Dịch trái (phải) toán hạng
	ROTATE	Quay vòng trái (phải) toán hạng

Điều khiển vào/ra	INPUT	Truyền dữ liệu từ một cổng hoặc thiết bị I/O xác định đến đích (VD: Bộ nhớ chính hoặc thanh ghi bộ xử lý)
	OUTPUT	Truyền dữ liệu từ nguồn xác định đến cổng hoặc thiết bị I/O
	START I/O	Truyền lệnh đến bộ xử lý I/O để bắt đầu hoạt động I/O
	TEST I/O	Truyền thông tin trạng thái từ hệ thống I/O đến đích xác định
Truyền điều khiển	JUMP (BRANCH)	Truyền không điều kiện; tải địa chỉ xác định vào PC
	JUMP CONDITIONAL	Kiểm tra điều kiện cụ thể; tải địa chỉ xác định vào PC hoặc không làm gì tùy thuộc vào điều kiện
	CALL	Cắt nội dung của PC (địa chỉ trả về) vào một vị trí xác định; Nạp địa chỉ lệnh đầu tiên của chương trình con vào PC
	RETURN	Đặt địa chỉ trả về trả lại cho PC để trả về chương trình chính
	SKIP	Tăng PC để bỏ qua lệnh tiếp theo
	SKIP CONDITIONAL	Kiểm tra điều kiện cụ thể; bỏ qua lệnh hoặc không làm gì tùy thuộc vào điều kiện
	HALT	Dừng thực thi chương trình
	WAIT (HOLD)	Dừng thực thi chương trình; liên tục kiểm tra điều kiện cụ thể; quay lại thực thi tiếp khi điều kiện được thỏa mãn
	NO OPERATION	Không có hành động nào được thực hiện, nhưng việc thực thi chương trình vẫn được tiếp tục

Bảng 10.4 Hoạt động của bộ xử lý tương ứng với từng loại hành động

Truyền dữ liệu	Truyền dữ liệu từ vị trí này sang vị trí khác
	Nếu có liên quan đến bộ nhớ:
	Xác định địa chỉ bộ nhớ
	Thực hiện chuyển đổi địa chỉ bộ nhớ từ địa chỉ bộ nhớ ảo sang thực
	Kiểm tra bộ nhớ cache
Số học	Khởi tạo lệnh đọc/ghi bộ nhớ
	Có thể liên quan đến việc truyền dữ liệu (trước và/hoặc sau truyền dữ liệu)
	Thực hiện hàm trong ALU
	Thiết lập mã điều kiện và cờ

Logic	Tương tự như số học
Truyền điều khiển	Cập nhật PC. Đổi với lệnh gọi/trả về chương trình con (Call/Return), quản lý việc liên kết và truyền tham số
Điều khiển vào/ra	Ra mệnh lệnh cho mô-đun I/O Nếu sử dụng I/O ánh xạ bộ nhớ, xác định địa chỉ ánh xạ bộ nhớ

10.4.1. Truyền dữ liệu

Loại lệnh máy cơ bản nhất là lệnh truyền dữ liệu. Lệnh chuyển dữ liệu phải xác định một vài thông tin. Thứ nhất, nó phải xác định vị trí của các toán hạng nguồn và toán hạng đích. Mỗi vị trí này có thể là vị trí bộ nhớ, thanh ghi, hoặc định ngăn xếp. Thứ hai, nó cũng phải chỉ ra độ dài dữ liệu cần truyền. Thứ ba, giống như với tất cả các lệnh có toán hạng, chế độ định địa chỉ cho mỗi toán hạng phải được xác định. Điểm thứ ba này được trình bày chi tiết hơn trong Chương 11.

Trong các loại hành động xử lý, hành động truyền dữ liệu có lẽ là đơn giản nhất. Nếu cả toán hạng nguồn và toán hạng đích đều ở thanh ghi, thì bộ xử lý chỉ đơn giản là chuyển dữ liệu từ thanh ghi này sang thanh ghi khác; đây là hành động diễn ra bên trong bộ xử lý. Nếu một hoặc cả hai loại toán hạng này nằm trong bộ nhớ, khi đó bộ xử lý phải thực hiện một số hoặc tất cả các hành động sau đây:

1. Tính toán địa chỉ bộ nhớ, dựa trên chế độ địa chỉ (thảo luận trong Chương 11).
2. Nếu địa chỉ liên quan đến bộ nhớ ảo, dịch địa chỉ từ địa chỉ bộ nhớ ảo sang thực.
3. Xác định xem toán hạng mong muốn có trong bộ nhớ cache hay không.
4. Nếu không, ra một mệnh lệnh cho mô-đun bộ nhớ.

10.4.2. Xử lý số học

Hầu hết máy tính cung cấp các phép tính số học cơ bản như cộng, trừ, nhân và chia. Các phép tính này được cung cấp cho tính toán liên quan đến số nguyên (dấu chấm tĩnh) có dấu, số thực dấu chấm động và số packed decimal.

Bên cạnh đó, còn có các xử lý khác, bao gồm một loạt các lệnh chỉ có một toán hạng; ví dụ,

- Tuyệt đối (Absolute): Lấy giá trị tuyệt đối của toán hạng.
- Đảo (Negate): Đổi dấu toán hạng.
- Tăng (Increment): Tăng toán hạng thêm 1.
- Giảm (Decrement): Giảm toán hạng đi 1.

Việc thực thi một lệnh số học có thể bao gồm vài thao tác truyền dữ liệu để đặt các toán hạng tới đầu vào khói ALU, và để vận chuyển kết quả ở đầu ra ALU. Hình 3.5 minh họa cho sự dịch chuyển liên quan đến việc truyền dữ liệu và xử lý số học. Ngoài ra, khói ALU của bộ xử lý tất nhiên sẽ thực hiện các xử lý mong muốn.

10.4.3. Xử lý logic

Hầu hết máy tính cung cấp một loạt các hành động để xử lý các bit riêng lẻ trong một word hoặc một đơn vị địa chỉ khác. Chúng dựa trên các phép toán đại số logic Boolean.

Bảng 10.5 Các phép toán logic cơ bản

P	Q	NOT P	P AND Q	P OR Q	P XOR Q	P = Q
0	0	1	0	0	0	1
0	1	1	0	1	1	0
1	0	0	0	1	1	0
1	1	0	1	1	0	1

Một số phép toán logic cơ bản có thể được thực hiện trên dữ liệu nhị phân hoặc Boolean được trình bày trong Bảng 10.5. Phép toán NOT đảo ngược giá trị một bit. AND, OR, và XOR là các hàm logic có hai toán hạng phổ biến nhất. EQUAL là một phép thử nhị phân hữu ích.

Những phép toán logic này có thể được áp dụng (trên từng bit) cho các đơn vị dữ liệu logic n bit. Vì vậy, nếu hai thanh ghi chứa dữ liệu như sau:

$$(R1) = 10100101$$

$$(R2) = 00001111$$

thì

$$(R1) \text{ AND } (R2) = 00000101$$

trong đó ký hiệu (X) có nghĩa là nội dung của vị trí X. Do đó, phép toán AND có thể được sử dụng làm mặt nạ để chọn một số bit xác định trong một word và lọc đi các bit còn lại. Xét ví dụ khác, nếu hai thanh ghi là

$$(R1) = 10100101$$

$$(R2) = 11111111$$

thì

$$(R1) \text{ XOR } (R2) = 01011010$$

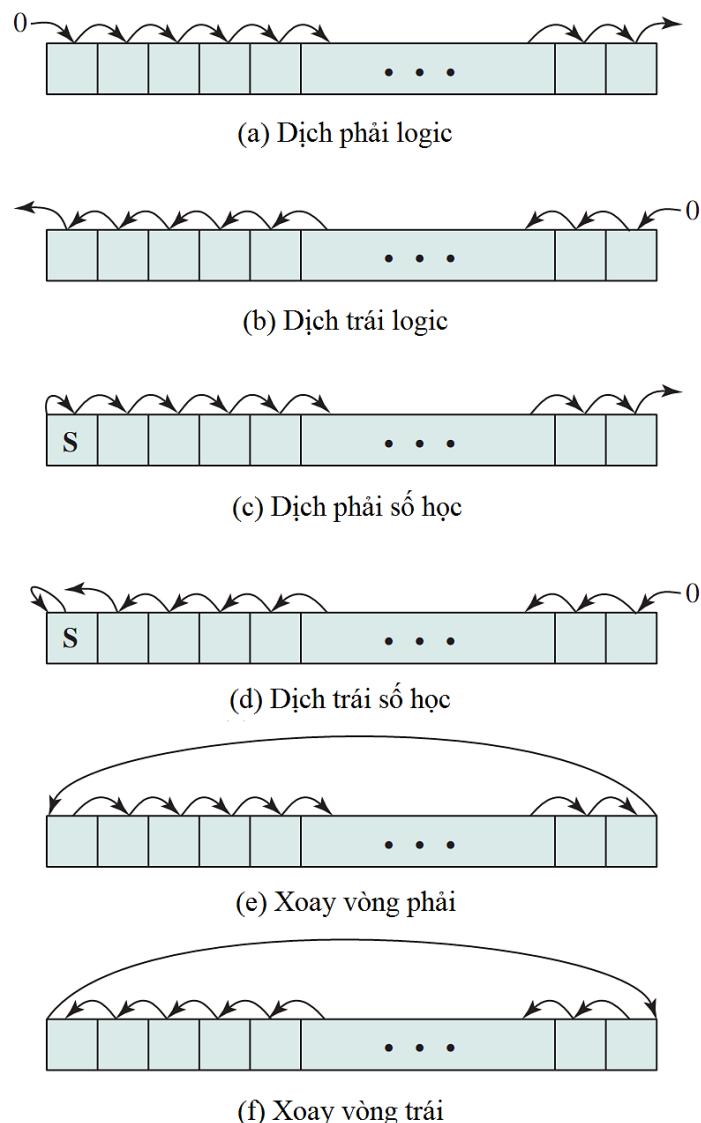
Với một word được thiết lập để tất cả các bit là 1, phép toán XOR đảo ngược giá trị tất cả các bit trong word còn lại (bù một).

Ngoài phép toán logic trên từng bit, hầu hết máy tính đều cung cấp một loạt các chức năng dịch và xoay vòng. Các phép dịch và xoay vòng cơ bản nhất được minh họa trong Hình 10.6. Với **phép dịch logic**, các bit của một word được dịch chuyển sang trái hoặc phải. Trên một đầu, bit dịch ra ngoài bị mất đi. Ở đầu kia, một bit 0 được dịch vào. Phép dịch logic chủ yếu hữu ích trong việc tách các trường trong một word. Các bit 0 được dịch

vào một word thay thế cho những thông tin không mong muốn được dịch ra ngoài ở đầu bên kia.

Ví dụ, giả sử chúng ta muốn truyền các ký tự của dữ liệu tới một thiết bị I/O, mỗi lần truyền một ký tự. Nếu mỗi word của bộ nhớ dài 16 bit và chứa hai ký tự, chúng ta phải tách rời từng ký tự trước khi gửi đi. Để gửi đi hai ký tự trong một word,

1. Tải word đó vào thanh ghi.
2. Dịch sang phải tám lần. Việc này sẽ làm dịch ký tự còn lại sang nửa bên phải của thanh ghi.
3. Thực hiện hành động I/O. Mô-đun I/O đọc 8 bit trọng số thấp hơn từ bus dữ liệu.



Hình 10.6 Phép dịch và xoay vòng

Các bước vừa nêu tương ứng với việc gửi đi ký tự bên trái. Để gửi ký tự bên phải,

1. Tải lại word đó vào thanh ghi.
2. AND với 00000001111111. Việc này sẽ lọc bỏ ký tự bên trái.

3. Thực hiện hành động I/O.

Các **phép dịch số học** xử lý dữ liệu như là một số nguyên có dấu và không dịch chuyển bit dấu. Với phép dịch phải số học, bit dấu được sao chép vào vị trí bit ở bên phải nó. Phép dịch trái số học giống như phép dịch trái logic được thực hiện trên tất cả các bit, ngoại trừ bit dấu, tức là bit dấu giữ nguyên. Các phép dịch này có thể làm tăng tốc độ xử lý của các phép toán số học nhất định. Với các số biểu diễn bù hai, phép dịch phải số học tương ứng với một phép chia cho 2 (làm tròn với số lẻ). Cả dịch trái số học và dịch trái logic tương ứng với một phép nhân với 2 khi không có tràn trên (overflow). Nếu xảy ra tràn, phép dịch trái số học và dịch trái logic sẽ tạo ra các kết quả khác nhau, nhưng dịch trái số học vẫn giữ lại dấu của số.

Phép xoay vòng, hoặc dịch vòng, bảo toàn tất cả các bit mà nó xử lý. Phép xoay vòng được sử dụng để lần lượt đưa từng bit về vị trí bit ngoài cùng bên trái, tại đó, nó có thể được nhận diện giá trị bằng cách kiểm tra dấu của dữ liệu (coi như một số có dấu).

Giống như các phép tính số học, phép tính logic cũng liên quan đến hoạt động của khối ALU và có thể bao gồm các thao tác truyền dữ liệu. Bảng 10.6 đưa ra ví dụ minh họa cho các phép dịch và xoay vòng này.

Bảng 10.6 Ví dụ cho phép dịch và xoay vòng

Đầu vào	Hành động	Kết quả
10100110	Dịch phải logic (3 bit)	00010100
10100110	Dịch trái logic (3 bit)	00110000
10100110	Dịch phải số học (3 bit)	11110100
10100110	Dịch trái số học (3 bit)	10110000
10100110	Xoay vòng phải (3 bit)	11010100
10100110	Xoay vòng trái (3 bit)	00110101

10.4.4. Điều khiển vào/ra

Các lệnh đầu vào/đầu ra đã được thảo luận chi tiết trong Chương 7. Như chúng ta đã thấy, có nhiều phương pháp được thực hiện, bao gồm I/O bằng lập trình, tách biệt bộ nhớ, I/O bằng lập trình, ánh xạ bộ nhớ, DMA, và việc sử dụng bộ xử lý I/O. Nhiều cách thực hiện chỉ cung cấp một vài lệnh I/O, với các hành động cụ thể được xác định bởi tham số, mã hoặc các word lệnh.

10.4.5. Điều khiển hệ thống

Lệnh điều khiển hệ thống là lệnh có thể được thực thi chỉ khi bộ xử lý đang ở trong trạng thái đặc quyền (privileged) nhất định hoặc đang thi hành một chương trình trong vùng đặc quyền (privileged) đặc biệt của bộ nhớ. Thông thường, các lệnh này được dành riêng cho việc sử dụng hệ điều hành.

Ví dụ về hành động điều khiển hệ thống, một lệnh điều khiển hệ thống có thể đọc hoặc thay đổi thanh ghi điều khiển. Một ví dụ khác là lệnh để đọc hoặc thay đổi khóa bảo vệ bộ nhớ.

10.4.6. Chuyển điều khiển (rẽ nhánh)

Đối với tất cả các loại hành động đã thảo luận trên đây, lệnh tiếp theo được thực hiện là lệnh liền ngay phía sau lệnh hiện tại trong bộ nhớ. Tuy nhiên, một số lệnh trong chương trình có chức năng thay đổi trình tự thi hành lệnh. Đối với các lệnh này, bộ xử lý sẽ cập nhật thanh ghi PC bằng địa chỉ của một số lệnh trong bộ nhớ.

Tại sao cần phải có các hành động chuyển điều khiển? Trong số nhiều lý do, một số lý do quan trọng nhất là:

1. Trong thực tế sử dụng máy tính, việc thực hiện một lệnh nhiều hơn một lần và thậm chí hàng nghìn lần là rất cần thiết. Một ứng dụng có thể đòi hỏi thực hiện hàng nghìn đến hàng triệu câu lệnh. Vì vậy, nếu phải viết riêng từng lệnh kể cả lệnh được lặp lại thì sẽ rất khó khăn. Nếu cần xử lý một bảng hoặc danh sách các mục, ta cần một vòng lặp chương trình. Một chuỗi lệnh sẽ được thi hành lặp đi lặp lại để xử lý tất cả dữ liệu.
2. Hầu như mọi chương trình đều liên quan đến việc đưa ra quyết định. Chúng ta muốn máy tính thực hiện một hành động nếu điều kiện này được thỏa mãn, và hành động khác nếu điều kiện khác được thỏa mãn. Ví dụ, một chuỗi lệnh tính toán căn bậc hai của một số. Ở đầu chuỗi, dấu của số đó được kiểm tra. Nếu là số âm, phép tính không được thực hiện, và sẽ báo lỗi.
3. Tạo ra một chương trình máy tính lớn là một nhiệm vụ cực kỳ khó khăn. Sẽ rất hiệu quả nếu có cơ chế để phân chia công việc thành các phần nhỏ hơn và thực hiện lần lượt từng việc.

Tiếp theo, chúng ta sẽ thảo luận về các lệnh chuyển điều khiển thông dụng nhất trong các tập lệnh: lệnh rẽ nhánh, skip, và gọi thủ tục.

10.4.6.1. Lệnh rẽ nhánh (Branch)

Lệnh rẽ nhánh, còn được gọi là lệnh nhảy (jump), có một trong các toán hạng của nó là địa chỉ của lệnh tiếp theo được thực thi. Thông thường, lệnh này là lệnh **rẽ nhánh có điều kiện**. Tức là, chỉ rẽ nhánh (cập nhật địa chỉ đã chỉ định trong toán hạng vào thanh ghi PC) khi một điều kiện nhất định được thỏa mãn. Nếu không, lệnh tiếp theo trong chuỗi lệnh được thực thi (tăng giá trị PC như bình thường). Một lệnh rẽ nhánh mà việc rẽ nhánh luôn được thực hiện được gọi là **rẽ nhánh không điều kiện**.

Có hai cách phổ biến để đặt ra điều kiện trong lệnh rẽ nhánh có điều kiện. Thứ nhất, hầu hết các máy đều có mã điều kiện 1 bit hoặc nhiều bit được thiết lập như là kết quả của một số phép tính. Mã này có thể được coi là một thanh ghi ngắn, hiển thị với người dùng. Ví dụ, một phép tính số học (ADD, SUBTRACT, v.v.) có thể thiết lập mã điều kiện 2 bit nhận một trong bốn giá trị sau: 0, dương, âm, tràn (overflow). Trên máy này, có thể có bốn lệnh rẽ nhánh có điều kiện khác nhau như sau:

BRP X Rẽ nhánh đến vị trí X nếu kết quả dương.

BRN X Rẽ nhánh đến vị trí X nếu kết quả âm.

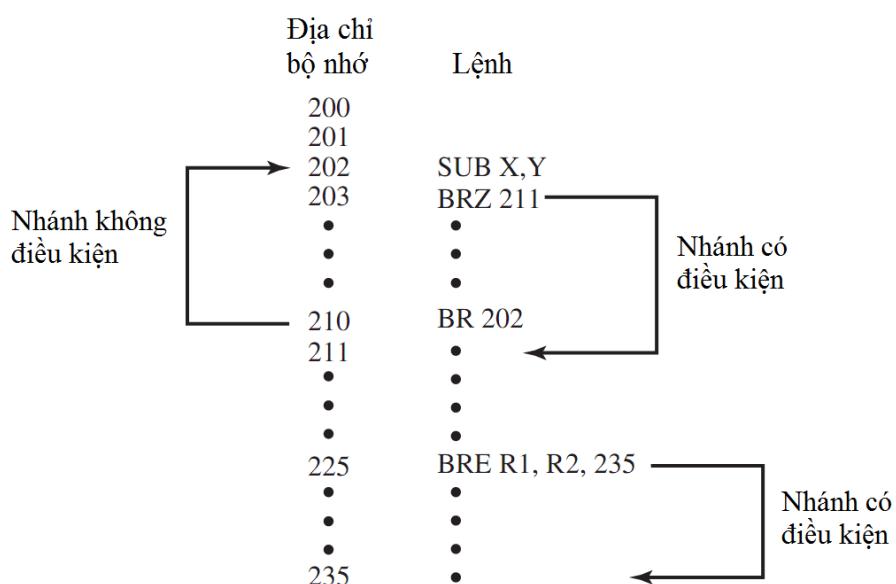
BRZ X Rẽ nhánh đến vị trí X nếu kết quả là 0.

BRO X Rẽ nhánh đến vị trí X nếu xảy ra tràn.

Trong tất cả các trường hợp này, kết quả được đề cập đến là kết quả của hành động thiết lập mã điều kiện gần đây nhất. Với định dạng lệnh ba địa chỉ, một phương pháp khác có thể được sử dụng là thực hiện so sánh và chỉ định rẽ nhánh trong cùng một lệnh. Ví dụ,

BRE R1, R2, X Rẽ nhánh đến X nếu nội dung của R1 = nội dung của R2.

Hình 10.7 cho thấy ví dụ về các hành động này. Lưu ý rằng nhánh có thể tiến (một lệnh với địa chỉ cao hơn) hoặc lùi (địa chỉ thấp hơn). Ví dụ này cho ta thấy nhánh vô điều kiện và nhánh có điều kiện có thể được sử dụng để tạo ra một vòng lặp các lệnh như thế nào. Các lệnh nằm tại các vị trí từ 202 đến 210 sẽ được thi hành lặp đi lặp lại cho đến khi kết quả của phép tính X trừ Y bằng 0.



Hình 10.7 Lệnh rẽ nhánh

10.4.6.2. Lệnh skip

Một dạng lệnh chuyển điều khiển khác là lệnh skip. Lệnh skip gồm có một địa chỉ ngầm định. Thông thường, skip được ngầm hiểu rằng một lệnh được bỏ qua; do đó, địa chỉ ngầm định bằng với địa chỉ của lệnh tiếp theo cộng với chiều dài một lệnh.

Bởi vì lệnh skip không đòi hỏi trường địa chỉ đích, nên trường này có thể được dành cho những thứ khác. Một ví dụ điển hình là lệnh ISZ (increment-and-skip-if-zero hay là tăng lên và bỏ qua nếu bằng không). Xét đoạn chương trình sau:

```

301
:
:
309 ISZ R1
310 BR 301

```

Trong đoạn chương trình này, hai lệnh chuyển điều khiển được sử dụng để thực hiện một vòng lặp. R1 được thiết lập bằng một số âm, bằng số lần lặp sẽ được thực hiện. Ở cuối vòng lặp, R1 được tăng lên. Nếu nó khác 0, chương trình trở lại phần đầu của vòng lặp. Nếu nó bằng 0, việc rẽ nhánh bị bỏ qua, vòng lặp kết thúc và chương trình tiếp tục thực thi lệnh tiếp theo.

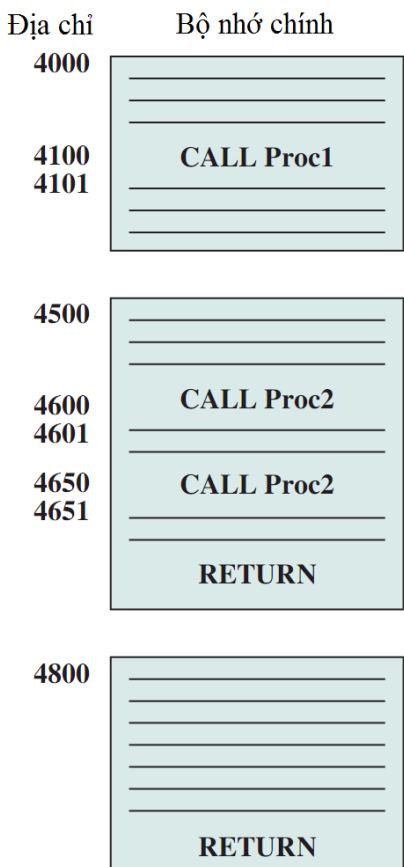
10.4.6.3. Lệnh gọi thủ tục

Có lẽ, thủ tục là điểm đổi mới quan trọng nhất trong sự phát triển của ngôn ngữ lập trình. Thủ tục là một chương trình máy tính được kết hợp vào một chương trình lớn hơn. Thủ tục có thể được gọi tại bất kỳ vị trí nào trong chương trình. Bộ xử lý được chỉ dẫn để đi tới và thực hiện toàn bộ thủ tục rồi sau đó quay trở về điểm phát sinh lệnh gọi thủ tục.

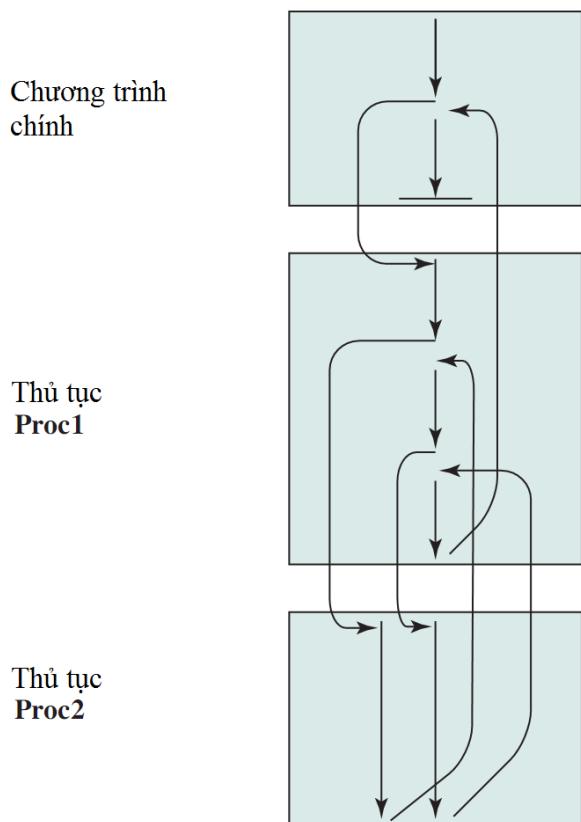
Sử dụng thủ tục có hai ưu điểm là tính kinh tế và tính mô đun. Một thủ tục cho phép sử dụng nhiều lần cùng một đoạn mã. Điều này giúp sử dụng hiệu quả nhất không gian lưu trữ trong hệ thống (chương trình phải được lưu trữ). Thủ tục cũng cho phép công việc lập trình quy mô lớn được chia nhỏ thành các khối công việc nhỏ hơn. Việc này làm giảm đáng kể khối lượng công việc lập trình.

Cơ chế của thủ tục liên quan đến hai lệnh cơ bản: lệnh gọi thủ tục (call) rẽ nhánh từ vị trí hiện tại đến thủ tục, và lệnh trả về (return) để từ thủ tục trở về nơi mà nó được gọi. Cả hai lệnh này đều là các hình thức của lệnh rẽ nhánh.

Hình 10.8a minh họa cho việc sử dụng thủ tục để xây dựng một chương trình. Trong ví dụ này, chương trình chính bắt đầu từ vị trí 4000. Chương trình này gồm có một lệnh gọi thủ tục PROC1, với PROC1 bắt đầu tại vị trí 4500. Khi gặp phải lệnh gọi này, bộ xử lý ngừng thực hiện chương trình chính và bắt đầu thực hiện PROC1 bằng cách truy xuất lệnh tiếp theo tại vị trí 4500. Trong PROC1, có hai lệnh gọi PROC2 biết rằng PROC2 ở vị trí 4800. Trong mỗi trường hợp này, việc thực thi PROC1 bị tạm ngưng và PROC2 được thực thi. Câu lệnh RETURN khiến cho bộ xử lý quay trở lại chương trình gọi và tiếp tục thi hành câu lệnh sau lệnh CALL tương ứng. Hành vi này được minh họa trong hình 10.8b.



(a) Lệnh Call và Return



(b) Chuỗi thực thi

Hình 10.8 Thủ tục lồng nhau

Cần lưu ý ba điểm sau:

1. Một thủ tục có thể được gọi từ nhiều vị trí.
 2. Một thủ tục có thể gọi một thủ tục khác. Điều này cho phép các thủ tục lồng nhau.
 3. Mỗi lệnh gọi thủ tục phải gắn với một lệnh trả về (return) trong chương trình bị gọi.

Do chúng ta muốn gọi một thủ tục từ nhiều điểm trong chương trình, bộ xử lý cần phải lưu lại địa chỉ trả về để có thể trở lại vị trí thích hợp. Có ba nơi để lưu trữ địa chỉ trả về là:

- **Thanh ghi**
 - **Điểm bắt đầu thủ tục được gọi**
 - **Định ngăn xếp**

Xét một lệnh ngôn ngữ máy CALL X, là lệnh gọi thủ tục tại vị trí X. Nếu phương án thanh ghi được sử dụng, CALL X tạo ra các hành động sau:

$$\begin{array}{lcl} \text{RN} & \leftarrow & \text{PC} + \Delta \\ \text{PC} & \leftarrow & \text{X} \end{array}$$

trong đó RN là thanh ghi luôn được sử dụng cho mục đích này, PC là bộ đếm chương trình, và Δ là chiều dài lệnh. Thủ tục được gọi lúc này có thể lưu lại nội dung thanh ghi RN để sử dụng khi trở lại sau này.

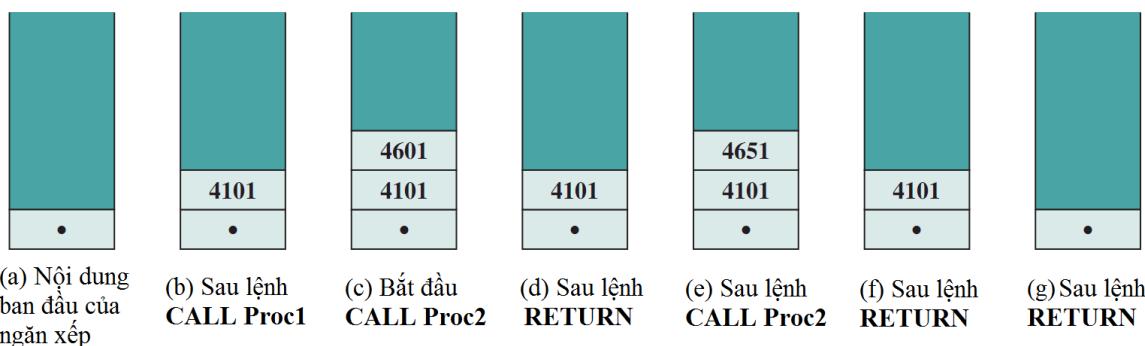
Khả năng thứ hai là lưu trữ địa chỉ trả về tại vị trí bắt đầu thủ tục. Trong trường hợp này, CALL X tương ứng với:

$$\begin{aligned} X &\leftarrow PC + \Delta \\ PC &\leftarrow X + 1 \end{aligned}$$

Cách làm này khá tiện lợi. Địa chỉ trả về đã được cất giữ an toàn.

Cả hai phương án này đều hoạt động và đã được sử dụng. Hạn chế duy nhất của các phương án này là chúng làm cho việc sử dụng các thủ tục dùng chung (reentrant) phức tạp hơn. Thủ tục dùng chung là thủ tục mà có thể được mở ra cùng một lúc bởi nhiều lệnh gọi. Thủ tục đệ quy (thủ tục tự gọi chính nó) là một ví dụ về việc sử dụng tính năng này.

Phương pháp tổng quát và mạnh hơn là sử dụng ngăn xếp. Khi bộ xử lý thi hành một lệnh gọi, nó đặt địa chỉ trả về vào ngăn xếp. Khi nó thi hành một lệnh trả về (return), nó sử dụng địa chỉ trên ngăn xếp. Hình 10.9 minh họa việc sử dụng ngăn xếp.



Hình 10.9 Sử dụng ngăn xếp để thực thi các chương trình con lồng nhau ở Hình 10.8

10.5. CÂU HỎI

1. Các thành phần điển hình của một lệnh máy?
2. Toán hạng nguồn và đích có thể được đặt ở đâu?
3. Nếu một lệnh có 4 địa chỉ, mục đích của từng địa chỉ là gì?
4. Trình bày ngắn gọn 5 vấn đề thiết kế tập lệnh quan trọng.
5. Các loại toán hạng điển hình trong một tập lệnh máy?
6. Mối quan hệ giữa mã ký tự IRA và biểu diễn packed decimal?
7. Phân biệt phép dịch số học và dịch logic?
8. Tại sao cần phải có lệnh chuyển điều khiển?
9. Trình bày hai cách thường dùng để đặt điều kiện để kiểm tra trong một lệnh nhánh có điều kiện?
10. Ý nghĩa của thuật ngữ *thủ tục lồng nhau* là gì?

244 Chương 10. Tập lệnh: Đặc điểm và chức năng

11. Liệt kê ba vị trí có thẻ lưu trữ địa chỉ trả về của thủ tục có trả về.
12. Thủ tục dùng chung (reentrant) là gì?
13. Biểu diễn thập lục phân của:
 - a. Định dạng packed decimal của 23
 - b. Kí tự ASCII 23
14. Xác định giá trị thập phân tương ứng của các số packed decimal sau:
 - a. 0111 0011 0000 1001
 - b. 0101 1000 0010
 - c. 0100 1010 0110
15. Một vi xử lý có độ dài word là 1 byte. Xác định số nguyên nhỏ nhất và lớn nhất có thể biểu diễn được trong các dạng biểu diễn sau:
 - a. Không dấu
 - b. Dấu – độ lớn
 - c. Bù một
 - d. Bù hai
 - e. Packed decimal không dấu
 - f. Packed decimal có dấu

Chương 11. TẬP LỆNH: CHÉ ĐỘ ĐỊA CHỈ VÀ ĐỊNH DẠNG

11.1. CHÉ ĐỘ ĐỊA CHỈ

Một hay nhiều trường địa chỉ trong định dạng lệnh thông thường có kích thước tương đối nhỏ. Tuy nhiên, chúng ta luôn muốn tham chiếu đến một dải địa chỉ lớn của các vị trí trong bộ nhớ chính hoặc bộ nhớ ảo. Để đạt được mục tiêu này, người ta đã sử dụng một loạt các kỹ thuật địa chỉ. Tuy nhiên, việc sử dụng các kỹ thuật này sẽ dẫn đến một sự đánh đổi giữa: (1) dải địa chỉ và/hoặc tính linh hoạt, (2) số lượng các tham chiếu bộ nhớ trong một lệnh và/hoặc sự phức tạp trong việc tính toán địa chỉ. Trong phần này, chúng ta sẽ tìm hiểu các kỹ thuật hay chế độ địa chỉ phổ biến nhất:

- Chế độ tức thì
- Chế độ trực tiếp
- Chế độ gián tiếp
- Chế độ thanh ghi
- Chế độ gián tiếp thanh ghi
- Chế độ dịch chuyển
- Chế độ ngăn xếp

Các chế độ này được minh họa trong Hình 11.1. Trong phần này, chúng ta sẽ sử dụng một số ký hiệu sau:

A = nội dung trường địa chỉ trong lệnh

R = nội dung trường địa chỉ trong lệnh tham chiếu đến một thanh ghi

EA = địa chỉ hiệu dụng của một vị trí chứa toán hạng được tham chiếu

(X) = nội dung của vị trí bộ nhớ X hoặc thanh ghi X

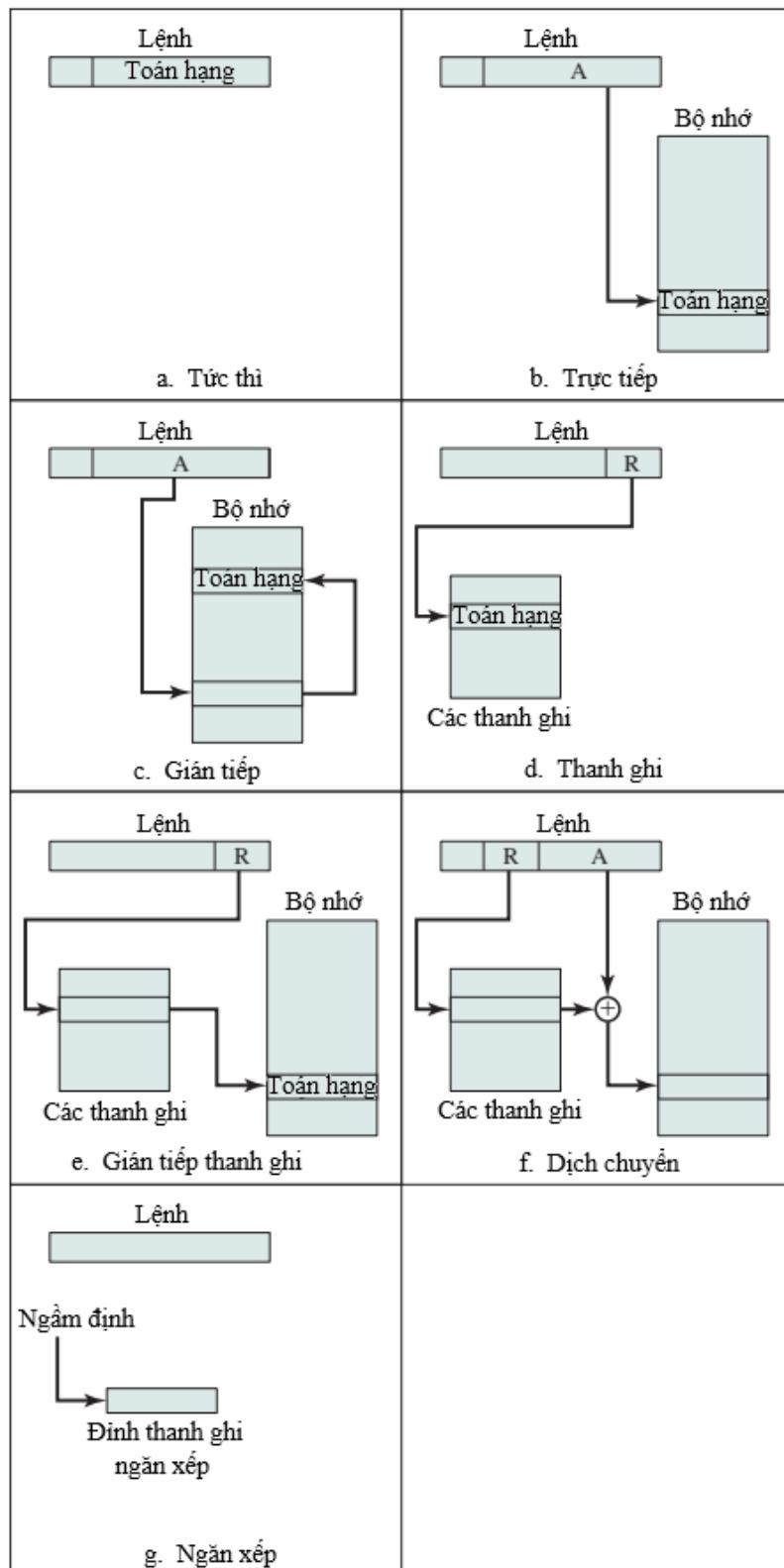
Việc tính toán địa chỉ của mỗi chế độ được chỉ ra trong Bảng 11.1.

Bảng 11.1 Các chế độ địa chỉ cơ bản

Chế độ	Thuật toán	Ưu điểm	Nhược điểm
Tức thì	Toán hạng = A	Không cần tham chiếu bộ nhớ	Hạn chế về giá trị của toán hạng
Trực tiếp	EA = A	Đơn giản	Không gian địa chỉ hạn chế
Gián tiếp	EA = (A)	Không gian địa chỉ lớn	Tham chiếu bộ nhớ nhiều lần
Thanh ghi	EA = R	Không cần tham chiếu bộ nhớ	Không gian địa chỉ hạn chế
Gián tiếp thanh ghi	EA = A + (R)	Linh hoạt	Phức tạp
Ngăn xếp	EA = đỉnh ngăn xếp	Không cần tham chiếu bộ nhớ	Khả năng ứng dụng ít

Trước khi tìm hiểu về các chế độ địa chỉ, có hai vấn đề cần được giải thích. Thứ nhất, hầu như tất cả các kiến trúc máy tính đều hỗ trợ nhiều chế độ địa chỉ. Câu hỏi đặt ra là làm sao để bộ xử lý quyết định chế độ địa chỉ nào đang được sử dụng với một lệnh cụ thể. Có một số kỹ thuật để giải quyết vấn đề này. Thông thường các opcode (mã lệnh) khác nhau

sẽ sử dụng các chế độ địa chỉ khác nhau. Ngoài ra, trong định dạng lệnh có thể có một trường *mode* bao gồm một hoặc nhiều bit. Giá trị của trường này xác định chế độ địa chỉ nào được sử dụng.



Hình 11.1 Các chế độ địa chỉ

Vấn đề thứ hai liên quan đến việc phân giải địa chỉ hiệu dụng (EA). Với một hệ thống không có bộ nhớ ảo, **địa chỉ hiệu dụng** sẽ là địa chỉ bộ nhớ chính hoặc địa chỉ thanh ghi. Trong hệ thống có sử dụng bộ nhớ ảo, **địa chỉ hiệu dụng** là địa chỉ ảo hoặc địa chỉ thanh ghi. Việc ánh xạ từ địa chỉ ảo đến địa chỉ vật lý là chức năng của bộ quản lý bộ nhớ (MMU - Memory Management Unit), lập trình viên không thể thấy được.

11.1.1. Địa chỉ tức thì

Dạng đơn giản nhất của chế độ địa chỉ là **địa chỉ tức thì**, trong đó giá trị toán hạng được đặt luôn trong lệnh



Hình 11.2 Địa chỉ tức thì

$$\text{Toán hạng} = A$$

Chế độ này có thể được sử dụng để định nghĩa và sử dụng các hằng số hoặc thiết lập các giá trị ban đầu cho biến. Thông thường, số sẽ được lưu trữ dưới dạng bù hai với một bit dấu. Khi toán hạng được nạp vào thanh ghi, bit dấu được mở rộng sang bên trái để điền đủ vào kích thước thanh ghi. Trong một số trường hợp, giá trị nhị phân tức thì được hiểu là một số nguyên dương không dấu.

Ưu điểm của chế độ này là: trong một chu kỳ lệnh, chỉ cần tham chiếu bộ nhớ để lấy lệnh mà không cần tham chiếu bộ nhớ để lấy toán hạng, do đó tiết kiệm một chu kỳ bộ nhớ hoặc chu kỳ cache. Nhược điểm của chế độ này là kích thước của số bị hạn chế vì chỉ bằng kích thước của trường địa chỉ, mà đối với hầu hết các tập lệnh, kích thước này nhỏ hơn so với độ dài một word (từ nhớ).

Ví dụ:

ADD R1, 5 ; R1←R1+5

Trong đó, 5 là một toán hạng nguồn được tham chiếu trực tiếp trong câu lệnh: địa chỉ tức thì

11.1.2. Địa chỉ trực tiếp

Một dạng khác cũng khá đơn giản là chế độ **địa chỉ trực tiếp**, trong đó trường địa chỉ chứa **địa chỉ hiệu dụng** của toán hạng:

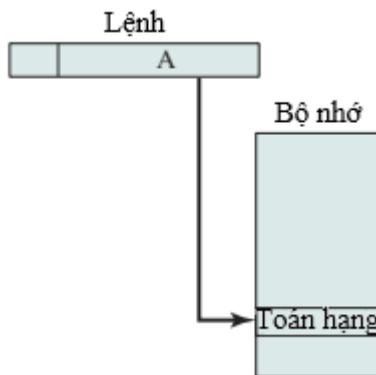
$$EA = A$$

Kỹ thuật này khá phổ biến trong các máy tính thế hệ trước đây nhưng ít được sử dụng trong các kiến trúc hiện đại. Chế độ này cần có một tham chiếu bộ nhớ và không có gì đặc biệt. Nhược điểm ở đây là chế độ này có không gian địa chỉ hạn chế.

Ví dụ:

ADD R1, A ; R1←R1+(A)

Trong đó, A là địa chỉ một vị trí bộ nhớ, (A) là nội dung của vị trí đó. CPU tham chiếu đến toán hạng có địa chỉ A trong bộ nhớ.



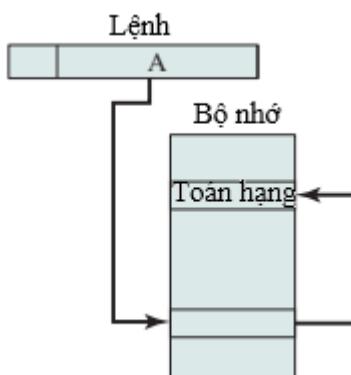
Hình 11.3 Địa chỉ trực tiếp

11.1.3. Địa chỉ gián tiếp

Với chế độ địa chỉ trực tiếp, kích thước trường địa chỉ thường ngắn hơn chiều dài từ, do đó hạn chế dài địa chỉ có thể tham chiếu tới. Một giải pháp cho vấn đề này là trường địa chỉ sẽ tham chiếu đến một word trong bộ nhớ mà nội dung của word này chứa địa chỉ toán hạng cần tham chiếu tới. Chế độ này được gọi là **chế độ địa chỉ gián tiếp**:

$$EA = (A)$$

Như trên đã định nghĩa, dấu ngoặc đơn được hiểu là *nội dung của*. Ưu điểm của kỹ thuật này là với một từ có độ dài N-bit, không gian địa chỉ là 2^N . Nhược điểm của nó là quá trình thực hiện lệnh đòi hỏi phải tham chiếu bộ nhớ hai lần mới lấy được toán hạng: một lần để lấy địa chỉ toán hạng và một lần lấy giá trị toán hạng.



Hình 11.4 Địa chỉ gián tiếp

Một cải tiến khác của kỹ thuật này là địa chỉ gián tiếp nhiều cấp hoặc địa chỉ gián tiếp nhiều tầng:

$$EA = (\dots (A) \dots)$$

Trong trường hợp này, một bit trong từ nhớ địa chỉ là một cờ gián tiếp (I). Nếu cờ I là 0, thì địa chỉ trong từ đó là địa chỉ hiệu dụng EA. Nếu bit cờ I bằng 1 thì đây vẫn là một địa chỉ gián tiếp. Rõ ràng, việc sử dụng chế độ địa chỉ nhiều tầng này không có ưu điểm nào

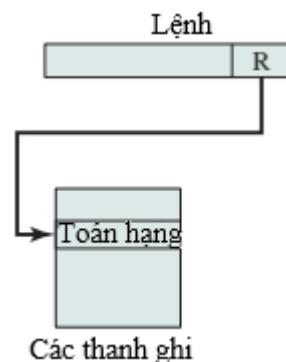
mà lại có nhược điểm rất lớn đó là số lần truy cập bộ nhớ để lấy ra toán hạng nhiều do đó cài tiến này không được sử dụng nhiều.

11.1.4. Địa chỉ thanh ghi

Chế độ địa chỉ thanh ghi cũng tương tự như chế độ địa chỉ trực tiếp. Điểm khác nhau duy nhất là trường địa chỉ dùng để tham chiếu đến thanh ghi chứ không phải tham chiếu đến bộ nhớ chính:

$$EA = R$$

Ví dụ, trường địa chỉ của một lệnh là 5 và lệnh này sử dụng chế độ địa chỉ thanh ghi thì giá trị toán hạng của lệnh nằm trong thanh ghi 5 (R5). Thông thường, một trường địa chỉ tham chiếu đến thanh ghi sẽ có từ 3 đến 5 bit, như vậy có thể tham chiếu đến từ 8 đến 32 thanh ghi đa nhiệm.



Hình 11.5 Địa chỉ thanh ghi

Các ưu điểm của chế độ địa chỉ thanh ghi là (1) trường địa chỉ của lệnh chỉ cần kích thước nhỏ, và (2) không mất một khoảng thời gian tham chiếu bộ nhớ. Như đã đề cập trong Chương 4, thời gian truy cập vào thanh ghi trong bộ xử lý ít hơn nhiều so với thời gian tham chiếu đến một địa chỉ bộ nhớ chính. Nhược điểm của địa chỉ thanh ghi là không gian địa chỉ là rất hạn chế (ít thanh ghi).

Do số lượng thanh ghi hạn chế (ít hơn rất nhiều so với số lượng vị trí bộ nhớ chính) nên việc sử dụng chúng theo cách này chỉ có ý nghĩa nếu chúng được sử dụng hiệu quả. Nếu một toán hạng được nạp vào thanh ghi từ bộ nhớ, sau đó nó chỉ được gọi một lần trong hoạt động và ghi lại vào bộ nhớ thì rõ ràng kỹ thuật này không hiệu quả và còn mất thời gian hơn so với kỹ thuật địa chỉ trực tiếp. Do đó, chế độ này thường được sử dụng khi một toán hạng được gọi lại trong nhiều hoạt động, ví dụ một kết quả trung gian trong một phép tính. Cụ thể với phép nhân hai số bù hai thực hiện bằng phần mềm. Địa chỉ bộ nhớ có nhãn A trong Hình 9.12 được tham chiếu nhiều lần trong thuật toán, nên nếu ta đưa toán hạng này vào thanh ghi thì thuật toán được thực hiện nhanh hơn rất nhiều so với việc sử dụng vị trí bộ nhớ chính.

Việc quyết định giá trị nào được giữ lại trong thanh ghi và giá trị nào cần được lưu trữ trong bộ nhớ chính sẽ do trình lập trình hoặc trình biên dịch đưa ra. Hầu hết các bộ xử lý hiện đại đều sử dụng nhiều thanh ghi đa năng nên việc sử dụng chúng sao cho hiệu quả là một vấn đề mà các trình hợp ngữ phải đối mặt.

Ví dụ:

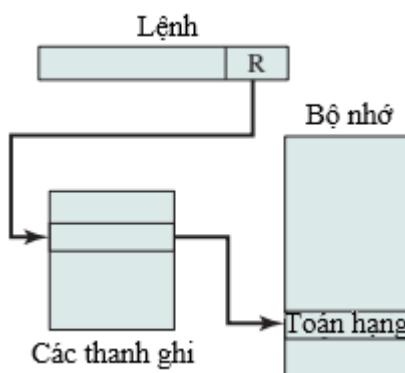
ADD R1, A ; $R1 \leftarrow R1 + (A)$

Trong đó, R1 là một thanh ghi trong bộ xử lý, câu lệnh trên tham chiếu giá trị (A) trong bộ nhớ cộng với giá trị lưu trữ trong thanh ghi R1, kết quả được ghi vào R1.

11.1.5. Địa chỉ gián tiếp thanh ghi

Chế độ này cũng tương tự như chế độ địa chỉ gián tiếp, chỉ có một điểm khác biệt là trường địa chỉ của lệnh tham chiếu đến một thanh ghi, thanh ghi này chứa địa chỉ hiệu dụng của toán hạng trong bộ nhớ (khác với chế độ địa chỉ gián tiếp sử dụng một vị trí bộ nhớ để chứa địa chỉ hiệu dụng). Thanh ghi này được gọi là con trỏ. Ta có

$$EA = (R)$$



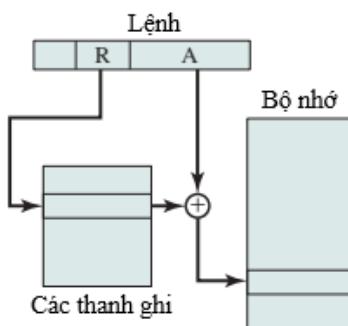
Hình 11.6 Địa chỉ gián tiếp thanh ghi

Giống như chế độ địa chỉ gián tiếp, việc sử dụng thanh ghi lưu trữ địa chỉ hiệu dụng tăng không gian địa chỉ của lệnh.Thêm vào đó, kỹ thuật này có tốc độ nhanh hơn do chỉ cần một lần truy xuất bộ nhớ (so với hai lần truy xuất bộ nhớ của chế độ gián tiếp).

11.1.6. Địa chỉ dịch chuyển

Đây là một chế độ địa chỉ rất mạnh mẽ kết hợp giữa địa chỉ trực tiếp và địa chỉ gián tiếp thanh ghi. Nó được biết với nhiều tên khác nhau tùy thuộc vào bối cảnh sử dụng nhưng về cơ chế là như nhau. Chúng tôi tạm gọi chế độ này là chế độ **địa chỉ dịch chuyển**:

$$EA = A + (R)$$



Hình 11.7 Địa chỉ dịch chuyển

Kỹ thuật này đòi hỏi định dạng lệnh phải có hai trường địa chỉ hoặc phải có một địa chỉ ngầm định. Giá trị của một trường địa chỉ là A, được sử dụng trực tiếp. Trường địa chỉ còn lại hoặc địa chỉ ngầm định (thường quy định theo opcode) tham chiếu đến một thanh ghi. Giá trị lưu trữ trong thanh ghi cộng với A cho ta địa chỉ hiệu dụng của toán hạng.

Có ba cách sử dụng phổ biến nhất của chế độ địa chỉ dịch chuyển:

- Địa chỉ tương đối
- Địa chỉ thanh ghi cơ sở
- Địa chỉ indexing

ĐỊA CHỈ TƯƠNG ĐỐI hay còn được gọi là địa chỉ PC-tương đối, thanh ghi tham chiếu được ngầm định là thanh ghi PC. Nghĩa là, địa chỉ lệnh tiếp theo (nội dung PC) được cộng với trường địa chỉ của lệnh để tạo ra EA. Thông thường, trường địa chỉ được coi là một số bù hai trong hoạt động này. Như vậy, địa chỉ hiệu dụng của toán hạng dịch chuyển một đoạn so với địa chỉ của lệnh.

Địa chỉ tương đối khai thác tính địa phương của các tham chiếu bộ nhớ như đã đề cập trong Chương 4. Nếu hầu hết các toán hạng được tham chiếu gần với lệnh dạng thực hiện, thì việc sử dụng địa chỉ tương đối sẽ giúp giảm bớt số lượng bit địa chỉ cần thiết trong lệnh.

ĐỊA CHỈ THANH GHI CƠ SỞ chế độ này được giải thích là như sau: nội dung của thanh ghi tham chiếu là một địa chỉ bộ nhớ, trường địa chỉ của lệnh chứa một giá trị dịch chuyển (thường là số nguyên không dấu) so với địa chỉ đó. Thanh ghi được tham chiếu có thể được chỉ rõ trong lệnh hoặc ngầm định.

Địa chỉ thanh ghi cơ sở cũng khai thác tính địa phương của các tham chiếu bộ nhớ. Chế độ này cũng giúp thực hiện một kỹ thuật phân đoạn bộ nhớ chính. Ở một số hệ thống, một thanh ghi đoạn ngầm định sẽ chứa địa chỉ đoạn. Một số hệ thống khác, người lập trình có thể lựa chọn một thanh ghi chứa địa chỉ cơ sở đoạn và lệnh phải tham chiếu đến thanh ghi này. Trong trường hợp thứ hai, nếu chiều dài của trường địa chỉ là K và số lượng thanh ghi là N, một lệnh có thể tham chiếu đến một vùng bất kỳ trong N vùng, mỗi vùng có 2^K từ.

ĐỊA CHỈ INDEXING Trường địa chỉ tham chiếu đến một địa chỉ bộ nhớ chính và đến một thanh ghi chứa một giá trị dịch chuyển dương từ địa chỉ đó. Ta có thể thấy kỹ thuật này ngược lại so với kỹ thuật địa chỉ thanh ghi cơ sở. Do trường địa chỉ của lệnh tham chiếu đến một địa chỉ bộ nhớ nên cần nhiều bit hơn trường địa chỉ tham chiếu đến thanh ghi cơ sở. Tuy nhiên, cả hai kỹ thuật đều có cách tính EA giống nhau và thanh ghi tham chiếu có thể là một thanh ghi ngầm định hoặc cần phải chỉ rõ trong lệnh (tùy thuộc vào từng bộ xử lý khác nhau).

Một ứng dụng quan trọng của chế độ indexing là nó cung cấp một cơ chế hiệu quả để thực hiện các hoạt động lặp đi lặp lại. Ví dụ, một mảng các số được lưu trữ bắt đầu từ vị trí A. Giả sử ta muốn cộng thêm 1 vào mỗi phần tử trong danh sách, ta phải lấy từng giá trị, thực hiện phép cộng với 1 sau đó lưu giá trị lại vào vị trí đó. Như vậy, ta cần thực hiện hoạt động với từng phần tử có địa chỉ hiệu dụng lần lượt là A, A + 1, A + 2, ..., đến vị trí cuối cùng trong mảng. Với chế độ indexing, các hoạt động này có thể được thực hiện dễ dàng.

Giá trị A sẽ được lưu trữ trong trường địa chỉ của lệnh, một thanh ghi được chọn là thanh ghi index và được khởi tạo giá trị bằng 0. Sau mỗi hoạt động, thanh ghi index tăng lên 1 đơn vị.

Vì các thanh ghi index thường được sử dụng cho các tác vụ lặp nên nó thường phải được tăng hoặc giảm giá trị sau mỗi lần được tham chiếu. Vì hoạt động này khá phổ biến nên một số hệ thống sẽ tự động làm điều này như một phần của chu trình lệnh, được gọi là autoindexing. Nghĩa là nếu thanh ghi nào đó được dành riêng cho index, hoạt động autoindexing được thực hiện ngầm định và tự động. Nếu sử dụng thanh ghi đa dụng làm thanh ghi index thì hoạt động autoindexing được thiết lập bằng một bit báo hiệu trong lệnh. Như vậy, chế độ indexing như sau:

$$EA = A + (R)$$

$$(R) + 1 \rightarrow (R)$$

Một số hệ thống máy tính cho phép sử dụng cả chế độ địa chỉ gián tiếp và địa chỉ indexing và có thể sử dụng cả hai chế độ này trong cùng một lệnh. Có hai khả năng: việc định địa chỉ theo chế độ indexing có thể thực hiện trước hoặc sau định địa chỉ gián tiếp.

Nếu được thực hiện sau chế độ địa chỉ gián tiếp, ta gọi là **postindexing**:

$$EA = (A) + (R)$$

Trường địa chỉ của lệnh sẽ trả đến một word trong bộ nhớ, nội dung của word này là địa chỉ một từ khác trong bộ nhớ. Địa chỉ này cộng với giá trị thanh ghi index sẽ là địa chỉ toán hạng cần truy xuất. Kỹ thuật này rất hữu ích để truy cập các block dữ liệu có định dạng cố định. Ví dụ, hệ điều hành cần thực hiện các hoạt động giống nhau với mỗi block dữ liệu, các địa chỉ trong các lệnh tham chiếu đến block cần trả đến một vị trí đầu tiên của block (địa chỉ = A). Thanh ghi index chứa giá trị dịch chuyển đến các vị trí bộ nhớ trong block đó.

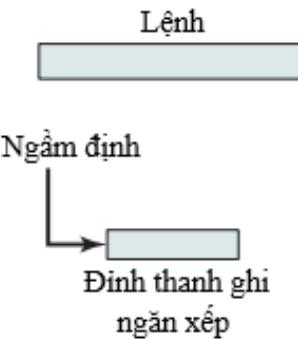
Nếu chế độ indexing được thực hiện trước chế độ gián tiếp, ta gọi là **preindexing**:

$$EA = (A + (R))$$

Kỹ thuật này thực hiện việc indexing đến một địa chỉ giống như chế độ indexing thông thường. Tuy nhiên, giá trị được tham chiếu đến chưa phải là toán hạng, mà là địa chỉ của toán hạng. Ví dụ về việc sử dụng kỹ thuật này là để xây dựng một bảng rẽ nhánh đa đường. Tại một điểm cụ thể trong một chương trình, ta cần phải rẽ nhánh đến một trong các vị trí nhất định tùy thuộc vào điều kiện. Một bảng các địa chỉ được thiết lập bắt đầu từ vị trí A. Bằng cách indexing vào bảng này, ta có thể tìm thấy vị trí cần thiết.

Thông thường, một tập lệnh sẽ không bao gồm cả preindexing và postindexing.

11.1.7. Địa chỉ ngăn xếp



Hình 11.8 Địa chỉ ngăn xếp

Chế độ địa chỉ cuối cùng mà chúng tôi đề cập tới là địa chỉ ngăn xếp. Ngăn xếp là một mảng các vị trí trong bộ nhớ, còn được gọi là *hàng đợi vào sau ra trước* – *LIFO*. Cơ chế làm việc của ngăn xếp như sau: các phần tử sẽ được đưa vào vùng ngăn xếp với cơ chế vào sau ra trước, một con trỏ luôn tham chiếu đến vị trí đỉnh của ngăn xếp (con trỏ này là giá trị địa chỉ được lưu trữ trong một thanh ghi – thanh ghi SP – Stack Pointer). Ngoài ra, trong một số trường hợp, hai phần tử trên cùng của ngăn xếp có thể được lưu trữ trên hai thanh ghi của bộ xử lý, như vậy, phần tử ở đỉnh ngăn xếp là phần tử thứ ba. Chế độ địa chỉ tham chiếu đến các vị trí trong bộ nhớ ngăn xếp về bản chất là chế độ địa chỉ gián tiếp thanh ghi.

Chế độ địa chỉ ngăn xếp là một dạng địa chỉ ngầm định. Các lệnh của máy không cần phải tham chiếu đến địa chỉ bộ nhớ ngăn xếp mà nó sẽ mặc định thao tác với đỉnh ngăn xếp

11.2. ĐỊNH DẠNG LỆNH

Một định dạng lệnh định nghĩa bối cảnh của các bit của lệnh, được chia dưới dạng các trường. Định dạng lệnh bao gồm một opcode (mã lệnh) có thể ngầm định hoặc định rõ, không có hoặc gồm nhiều toán hạng. Mỗi toán hạng được chỉ ra trong lệnh sẽ được tham chiếu bằng một trong các chế độ địa chỉ trong Phần 13.1. Chế độ địa chỉ được sử dụng phải được chỉ ra trong lệnh hoặc phải được quy ước theo một cách nào đó. Hầu hết các tập lệnh đều có nhiều định dạng lệnh.

Việc thiết kế định dạng lệnh là một việc hết sức phức tạp, tuy nhiên có rất nhiều tập lệnh được thiết kế tuyệt vời đã được các nhà phát triển đưa ra. Trong phần này chúng tôi sẽ trình bày một số vấn đề chính trong việc thiết kế các tập lệnh và đưa ra một số ví dụ về điều này.

11.2.1. Kích thước lệnh

Vấn đề thiết kế cơ bản đầu tiên người thiết kế phải đối mặt là chiều dài định dạng lệnh. Kích thước lệnh ảnh hưởng đến và bị ảnh hưởng bởi kích thước bộ nhớ, tổ chức bộ nhớ, cấu trúc bus, độ phức tạp của bộ xử lý và tốc độ xử lý. Điều này cũng quyết định sự đa dạng và linh hoạt của hệ thống máy tính mà người lập trình hợp ngữ sẽ gặp phải.

Vấn đề quan trọng thứ hai là phải cân bằng giữa mong muốn có một danh sách cách lệnh mạnh mẽ và yêu cầu tiết kiệm không gian bộ nhớ. Các lập trình viên luôn muốn có thêm nhiều opcode, nhiều toán hạng hơn, sử dụng các chế độ địa chỉ khác nhau và dài địa

chỉ lớn hơn. Việc có thêm nhiều opcode và toán hạng làm cho công việc của các lập trình viên dễ dàng hơn vì họ có thể viết các chương trình ngắn gọn hơn đối với cùng một công việc. Tương tự, việc sử dụng các chế độ địa chỉ khác nhau cho phép lập trình các chức năng khác nhau linh hoạt hơn, ví dụ như các thao tác với bảng hay các câu lệnh rẽ nhánh. Và, rõ ràng, với sự tăng kích thước bộ nhớ chính cùng với việc sử dụng bộ nhớ ảo, các tập lệnh phải đảm bảo việc tham chiếu đến nhiều địa chỉ bộ nhớ hơn. Để giải quyết tất cả các yêu cầu trên (opcodes, toán hạng, chế độ địa chỉ, phạm vi địa chỉ) đều kéo theo yêu cầu nhiều bit hơn và làm chiều dài lệnh dài hơn. Nhưng chiều dài lệnh dài hơn lại có thể lãng phí trong một số trường hợp. Một lệnh 64-bit chiếm không gian gấp đôi lệnh 32-bit nhưng hiệu quả chưa chắc đã gấp đôi.

Ngoài những vấn đề trên còn có một số điểm khác cũng cần được cân nhắc. Chiều dài lệnh phải bằng chiều dài một đơn vị truyền của bộ nhớ (trong hệ thống bus là kích thước bus dữ liệu) hoặc phải là bội số của nhau. Nếu không, ta sẽ không nhận được một số nguyên các lệnh trong một chu kỳ truy xuất. Một vấn đề khác đó là tốc độ truyền của bộ nhớ. Tốc độ này không tăng theo tốc độ vi xử lý. Do đó, bộ nhớ có thể trở thành nút cản chia nếu bộ xử lý thực hiện lệnh nhanh hơn việc truy xuất chúng. Một giải pháp đó là sử dụng bộ nhớ cache (đã trình bày trong chương 4) hoặc sử dụng các lệnh ngắn hơn. Một lệnh 16-bit có thể được truy xuất với tốc độ nhanh gấp đôi so với lệnh 32-bit nhưng lại được thực hiện chậm hơn hai lần.

Một điểm quan trọng khác cần cân nhắc đó là chiều dài lệnh. Chiều dài lệnh thường là bội số của chiều dài ký tự (thường là 8 bit) và chiều dài của dạng biểu diễn dấu chấm tĩnh. Như ta đã biết, word trong bộ nhớ có thể được hiểu là đơn vị "tự nhiên" của tổ chức máy tính. Kích thước của word quyết định kích thước dạng biểu diễn số dấu chấm tĩnh (thường là kích thước của chúng bằng nhau). Kích thước word cũng thường bằng kích thước một đơn vị truyền tải đến hoặc đi từ bộ nhớ.Thêm vào đó, dữ liệu ký tự là dạng phổ biến trong máy tính nên để lưu trữ hiệu quả ký tự trong máy tính thì kích thước từ thường là bội số của dữ liệu ký tự (bội số của 8 bit). Ví dụ như IBM giới thiệu kiến trúc System/360, sau đó để sử dụng dạng ký tự 8-bit nên đã quyết định chuyển từ kiến trúc 36-bit sang kiến trúc 32-bit.

11.2.2. Phân bổ các bit

Vấn đề thứ hai quan trọng không kém chiều dài lệnh là việc bố trí và phân bổ các bit trong định dạng lệnh. Điều này phụ thuộc vào rất nhiều yếu tố như sau:

Với một lệnh có chiều dài nhất định, trường opcode và các trường địa chỉ tham chiếu đến các toán hạng sẽ có ràng buộc với nhau. Nếu một tập lệnh có nhiều lệnh, số lượng bit trường opcode sẽ phải nhiều hơn, do đó làm giảm số lượng bit của các trường địa chỉ. Một giải pháp cho điều này là người ta thiết kế các tập lệnh có kích thước trường opcode thay đổi. Với phương pháp này, có độ dài opcode là nhỏ nhất có thể nhưng với một số opcode nhất định được bổ sung thêm một số bit. Do đó, việc sử dụng thêm các bit bổ sung thường được sử dụng cho các lệnh ít toán hạng và/hoặc sử dụng chế độ địa chỉ đơn giản, ít bit.

Một số yếu tố sau đây ảnh hưởng đến việc xác định số bit địa chỉ:

- Số lượng chế độ địa chỉ: Một số lệnh có thể ngầm định chế độ địa chỉ. Ví dụ: một opcode nhất định sử dụng chế độ index. Các trường hợp khác, cần phải chỉ rõ chế độ địa chỉ được sử dụng thông qua một hoặc nhiều bit mode.
- Số lượng toán hạng: trước đây, chúng tôi đã trình bày: lệnh càng ít toán hạng, chương trình dài hơn, phức tạp hơn. Các định dạng lệnh ngày nay thường bao gồm hai toán hạng. Mỗi địa chỉ toán hạng có thể cần một số bit mode của riêng nó hoặc trường mode sẽ chỉ định chế độ địa chỉ của một trường bất kỳ.
- Số lượng thanh ghi: Các thanh ghi có nhiệm vụ lưu trữ dữ liệu để xử lý trong bộ xử lý. Với chế độ một thanh ghi (thường là thanh ghi AC), địa chỉ toán hạng ứng với thanh ghi này sẽ ngầm định nên không tồn bit nào trong lệnh. Tuy nhiên, lập trình chế độ này rất rắc rối và đòi hỏi nhiều lệnh hơn. Ngay cả khi sử dụng nhiều thanh ghi ta cũng chỉ cần ít bit để tham chiếu đến các thanh ghi. Càng sử dụng nhiều thanh ghi để lưu trữ toán hạng thì cần càng ít bit địa chỉ. Một số nghiên cứu chỉ ra rằng số lượng thanh ghi cần thiết thường là từ 8 đến 32 thanh ghi. Các kiến trúc ngày nay có ít nhất 32 thanh ghi.
- Số lượng các tập thanh ghi: Hầu hết các máy tính hiện đại có một nhóm gồm ít nhất 32 thanh ghi đa năng. Các thanh ghi này có thể được sử dụng để lưu trữ dữ liệu hoặc lưu trữ địa chỉ trong chế độ địa chỉ dịch chuyển. Một số kiến trúc như x86 có bộ gồm hai hoặc nhiều nhóm riêng biệt (cho dữ liệu riêng và cho chế độ dịch chuyển riêng). Ưu điểm của việc sử dụng nhiều nhóm đó là: với số lượng thanh ghi nhất định, việc chia chức năng đòi hỏi ít bit hơn để sử dụng trong lệnh. Ví dụ, với hai nhóm tám thanh ghi chỉ cần 3 bit để xác định một thanh ghi; opcode hoặc thanh ghi mode sẽ xác định nhóm thanh ghi nào đang được tham chiếu đến.
- Dải địa chỉ: với các địa chỉ tham chiếu đến bộ nhớ, dải địa chỉ có thể tham chiếu được phụ thuộc vào số lượng bit địa chỉ. Vì hạn chế này nên địa chỉ trực tiếp ít khi được sử dụng. Với chế độ địa chỉ dịch chuyển, dải địa chỉ sẽ phụ thuộc vào kích thước thanh ghi địa chỉ. Mặc dù vậy, nếu giá trị dịch chuyển từ địa chỉ trong thanh ghi lớn thì lệnh vẫn cần số bit địa chỉ tương đối lớn.
- Độ chi tiết của địa chỉ: Với các chế độ địa chỉ tham chiếu đến bộ nhớ, một yếu tố quan trọng khác đó là độ chi tiết của việc định địa chỉ. Trong một hệ thống 16 hoặc 32-bit, một giá trị địa chỉ có thể tham chiếu tới một word hoặc một byte trong bộ nhớ, tùy thuộc vào việc thiết kế. Địa chỉ byte phù hợp với các hoạt động liên quan đến ký tự nhưng lại cần nhiều bit địa chỉ hơn.

Các nhà thiết kế phải cân nhắc cân bằng các yếu tố nêu trên để đạt được hiệu suất cao nhất. Phần tiếp theo chúng tôi sẽ ví dụ một vài tập lệnh được thiết kế dựa trên các yếu tố trên.

PDP-8 Đây là một tập lệnh đơn giản nhất, được thiết kế cho máy tính đa nhiệm như PDP-8. PDP-8 sử dụng các lệnh có 12-bit và dữ liệu là các word 12-bit. Hệ thống này chỉ có một thanh ghi đa năng là thanh ghi AC.

Mặc dù có một số hạn chế về mặt thiết kế, việc định địa chỉ lại khá linh hoạt. Mỗi địa chỉ tham chiếu bộ nhớ gồm 7 bit và 2 bit căn chỉnh. Bộ nhớ được chia thành các page có độ dài cố định là $2^7 = 128$ word. Định dạng lệnh PDP-8 hỗ trợ các chế độ địa chỉ gián tiếp, dịch chuyển và indexing. Hình 11. minh họa các định dạng lệnh PDP-8, trong đó: 3-

bit phần opcode và 3 loại lệnh. Với các opcode từ 0 đến 5 là các lệnh chỉ tham chiếu đến một địa chỉ bộ nhớ thông qua một bit page và một bit gián tiếp. Như vậy, chỉ có 6 hoạt động cơ bản. Để thêm vào các hoạt động khác, opcode 7 định nghĩa việc tham chiếu thanh ghi hoặc vi lệnh. Trong định dạng này, các bit còn lại được sử dụng để mã hóa các hoạt động mở rộng khác. Mỗi bit định nghĩa một hoạt động đặc biệt – vi lệnh (ví dụ: xóa thanh ghi AC) và các bit này có thể được kết hợp trong một lệnh đơn. Opcode 6 là hoạt động vào/ra: 6 bit được dùng để chọn một trong 64 thiết bị, 3 bit chỉ ra lệnh vào/ra cụ thể.

Định dạng lệnh của PDP-8 khá hiệu quả. Với phần opcode mở rộng, nó hỗ trợ tổng cộng khoảng 35 lệnh. Với ràng buộc về chiều dài lệnh chỉ 12 bit, các nhà thiết kế khó có thể đưa ra được định dạng lệnh nào tốt hơn.

PDP-10 PDP-10 được thiết kế là hệ thống chia sẻ cỡ lớn, nhấn mạnh vào việc tạo ra một hệ thống dễ dàng lập trình, ngay cả nếu thêm vào các phần cứng.

Tập lệnh được thiết kế với một số thông số như sau:

- Tính trực giao: Đây là một nguyên tắc mà nếu ta có hai biến, hai biến này được gọi là trực giao nếu chúng độc lập với nhau. Trong trường hợp tập lệnh, thuật ngữ này có nghĩa là các phần tử của lệnh độc lập với opcode (nghĩa là opcode không có quyết định gì với các trường còn lại trong lệnh). Các nhà thiết kế PDP-10 sử dụng thuật ngữ này để mô tả rằng chế độ địa chỉ được sử dụng sẽ độc lập với opcode. Điều này khác với nhiều hệ thống máy tính khác mà chế độ địa chỉ đôi khi phụ thuộc hoàn toàn vào hoạt động được thực hiện.

Các lệnh tham chiếu bộ nhớ

Opcode	D/I	Z/C	Dịch chuyển					
0	2	3	4	5				11

Các lệnh vào/ra

1	1	0	Thiết bị				Opcode		
0	1	2	3				8	9	11

Các vi lệnh nhóm 1

1	1	1	0	CLA	CLL	CMA	CML	RAR	RAL	BSW	IAC
0	1	2	3	4	5	6	7	8	9	10	11

Các vi lệnh nhóm 2

1	1	1	0	CLA	SMA	SZA	SNL	RSS	OSR	HLT	0
0	1	2	3	4	5	6	7	8	9	10	11

Các vi lệnh nhóm 3

1	1	1	0	CLA	MQA	0	MQL	0	0	0	1
0	1	2	3	4	5	6	7	8	9	10	11

D/I = Địa chỉ trực tiếp/gián tiếp
 Z/C = Page 0 hoặc page hiện tại
 CLA = Xóa thanh ghi AC
 CLL = Xóa liên kết
 CMA = Thực thi AC
 CML = Thực thi liên kết
 RAR = Dịch vòng AC sang phải
 RAL = Dịch vòng AC sang trái
 BSW = Đảo byte

IAC = Tăng AC
 SMA = Bỏ qua nếu AC âm
 SZA = Bỏ qua nếu AC bằng không
 SNL = Bỏ qua nếu liên kết khác không
 RSS = Reverse Skip Sense
 OSR = Phép Hoặc với thanh ghi Switch
 HLT = Dừng
 MQA = Nạp số nhân, thương vào AC
 MQL = Tải số nhân, thương

Hình 11.9 Định dạng lệnh PDP-8

- Tính hoàn chỉnh: Mỗi loại dữ liệu số học (số nguyên, dấu chấm tĩnh, dấu chấm động) phải có một tập hợp các hoạt động hoàn chỉnh và giống hệt nhau.
- Chế độ địa chỉ trực tiếp: PDP-10 sử dụng chế độ địa chỉ trực tiếp, nhờ vậy việc tổ chức bộ nhớ đơn giản hơn khi lập trình.

Mã lệnh	Thanh ghi	I	Thanh ghi Index	Địa chỉ bộ nhớ				
0	8 9	12 13 14	17 18					35

I = bit gián tiếp

Hình 11.10 Định dạng lệnh PDP-10

Độ dài word và lệnh trong PDP-10 đều là 36 bit (Hình 11.10). Trường opcode chiếm 9 bit, như vậy cho phép số lượng hoạt động là 512. Trong thực tế, PDP-10 có tổng cộng 365 lệnh khác nhau được định nghĩa. Hầu hết các lệnh đều có hai trường địa chỉ, một trường địa chỉ tham chiếu đến một trong 16 thanh ghi đa năng. Do đó, trường địa chỉ này chỉ cần kích thước 4 bit. Toán hạng thứ hai được tham chiếu bằng trường địa chỉ 18-bit, trong đó có thể là toán hạng tức thì hoặc một địa chỉ bộ nhớ. Tập lệnh này cũng cho phép sử dụng

chế độ địa chỉ indexing hoặc gián tiếp. Khi đó, thanh ghi đa năng được sử dụng làm thanh ghi index.

Chiều dài lệnh 36-bit khá là xa xỉ. Trường opcode 9-bit là quá đủ cho một tập lệnh nên ta không cần phải áp dụng bất cứ kỹ thuật nào để tăng thêm các opcode. Vấn đề định địa chỉ cũng khá đơn giản. Trường địa chỉ 18 bit đủ để dùng cho chế độ địa chỉ trực tiếp. Nếu dung lượng bộ nhớ lớn hơn 2^{18} , ta sẽ có thể thiết lập chế độ địa chỉ gián tiếp hoặc indexing. Thêm vào đó, với 18 bit cũng phù hợp với toán hạng sử dụng chế độ tíc thỉ.

Các đặc điểm trên của tập lệnh PDP-10 đều hướng đến mục đích làm cho công việc lập trình trở nên dễ dàng hơn. Ngược lại, việc sử dụng không gian địa chỉ không hiệu quả là nhược điểm của tập lệnh này. Tuy nhiên, đây là thiết kế có mục đích của các nhà hệ thống nên không phải là một thiết kế kém.

11.2.3. Các lệnh có độ dài thay đổi

Các tập lệnh trên đều có một đặc điểm là kích thước lệnh cố định. Tuy nhiên, các nhà thiết kế có thể thực hiện các định dạng lệnh có kích thước khác nhau, ta gọi chúng là tập lệnh có độ dài thay đổi. Với cách này, các nhà thiết kế có thể có nhiều opcode hơn và độ dài opcode khác nhau. Các chế độ địa chỉ sẽ linh hoạt hơn và có thể kết hợp giữa việc tham chiếu thanh ghi và bộ nhớ cùng với các chế độ địa chỉ. Các lệnh có độ dài thay đổi có nhiều biến thể hiệu quả và gọn gàng. Tuy nhiên, nhược điểm của dạng này là nó làm tăng độ phức tạp của bộ xử lý.

Việc sử dụng các lệnh có độ dài thay đổi vẫn phải hướng đến việc độ dài lệnh bằng bội số của kích thước word vì bộ xử lý sẽ không biết độ dài của lệnh tiếp theo được truy xuất. Một giải pháp là kích thước lệnh dài nhất sẽ bằng bội số của từ hoặc byte. Như vậy, trong một số trường hợp nhiều lệnh có kích thước ngắn được truy xuất cùng một lúc.

PDP-11 PDP-11 được thiết kế để mang lại một tập lệnh mạnh mẽ và linh hoạt cho dòng máy tính cỡ nhỏ 16-bit. PDP-11 sử dụng một bộ tám thanh ghi đa năng 16-bit. Hai trong số các thanh ghi này có thêm ý nghĩa khác: một thanh ghi được sử dụng như một thanh ghi SP (con trỏ ngắn xép) cho các hoạt động với ngắn xép cho mục đích đặc biệt và một thanh ghi khác là thanh ghi PC chứa địa chỉ của lệnh tiếp theo.

Hình 11.11 là các định dạng lệnh PDP-11. Có 13 định dạng khác nhau được sử dụng gồm các loại lệnh một, hai địa chỉ hoặc không có địa chỉ. Opcode có thể có độ dài từ 4 đến 16 bit. Tham chiếu thanh ghi qua một trường 6-bit. Ba bit để xác định một thanh ghi, 3 bit còn lại để xác định chế độ địa chỉ được sử dụng. PDP-11 sử dụng đa dạng các chế độ địa chỉ và liên kết các chế độ địa chỉ với toán hạng. Ưu điểm của việc này so với liên kết với opcode đó là bất cứ chế độ địa chỉ nào cũng có thể được sử dụng với bất cứ opcode nào. Như đã nhắc tới ở trên, tính độc lập này chính là tính trực giao.

Các lệnh PDP-11 thường có kích thước bằng một word (16 bit). Một vài lệnh khác có độ dài 32 bit hoặc 48 bit, việc truy xuất lệnh sẽ phải kèm thêm một hoặc hai địa chỉ.

Chương 11. Tập lệnh: chế độ địa chỉ

và định dạng 259

1	Opcode 4	Nguồn 6	Đích 6
---	-------------	------------	-----------

2	Opcode 7	R 3	Nguồn 6
---	-------------	--------	------------

4	Opcode 8	FP 2	Đích 6
---	-------------	---------	-----------

5	Opcode 10		Đích 6
---	--------------	--	-----------

7	Opcode 13	R 3	
---	--------------	--------	--

8	Opcode 16		
---	--------------	--	--

9	Opcode 4	Nguồn 6	Đích 6	Địa chỉ bộ nhớ 16
---	-------------	------------	-----------	----------------------

10	Opcode 7	R 3	Nguồn 6	Địa chỉ bộ nhớ 16
----	-------------	--------	------------	----------------------

11	Opcode 8	FP 2	Nguồn 6	Địa chỉ bộ nhớ 16
----	-------------	---------	------------	----------------------

12	Opcode 10		Đích 6	Địa chỉ bộ nhớ 16
----	--------------	--	-----------	----------------------

13	Opcode 4	Nguồn 6	Đích 6	Địa chỉ bộ nhớ 1 16	Địa chỉ bộ nhớ 2 16
----	-------------	------------	-----------	------------------------	------------------------

Các số ở dưới các trường biểu thị số bit của trường tương ứng

Nguồn, Đích: toán hạng nguồn và toán hạng đích. Các trường này gồm 3-bit xác định chế độ địa chỉ và 3-bit xác định một thanh ghi

FP chỉ ra một trong bốn thanh ghi dấu chấm động

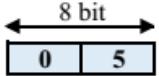
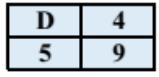
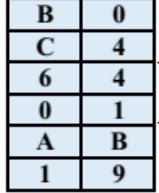
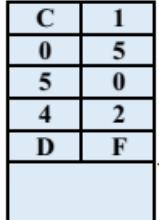
R chỉ ra một trong các thanh ghi đa năng

CC là trường mã điều kiện

Hình 11.11 Tập lệnh của PDP-11

Tập lệnh và các chế độ địa chỉ của PDP-11 khá là phức tạp. Nó làm tăng cả giá thành phần cứng lẫn độ phức tạp trong lập trình. Ưu điểm là các chương trình được viết ra sẽ hiệu quả và gọn gàng hơn.

VAX Hầu hết các kiến trúc có số lượng các định dạng lệnh cố định tương đối ít. Điều này có thể gây ra hai vấn đề cho lập trình viên. Thứ nhất, chế độ địa chỉ và opcode không trực giao. Nghĩa là, với một hoạt động cụ thể, chế độ địa chỉ cho mỗi toán hạng trong lệnh được định sẵn. Ví dụ: lệnh ADD quy định hai toán hạng được truy xuất trực tiếp từ bộ nhớ, lệnh READ có toán hạng nguồn là từ bộ nhớ, toán hạng đích là một thanh ghi đa năng. Như vậy, ta muốn sử dụng chế độ địa chỉ khác cho các toán hạng là không thể. Thứ hai, chỉ có một số nhất định các toán hạng trong định dạng lệnh: thường là hai hoặc ba. Vì một số hoạt động cần nhiều toán hạng hơn nên người lập trình phải sử dụng hai hoặc nhiều lệnh để đạt được kết quả mong muốn.

Biểu diễn hệ 16	Giải thích	Giải thích và mô tả câu lệnh
	Opcode của lệnh RSB	RSB Trở về từ chương trình con
	Opcode của lệnh CLRL Tham chiếu thanh ghi R9	CLRL R9 Xóa nội dung thanh ghi R9
	Opcode của lệnh MOVW Chế độ dịch chuyển word với thanh ghi R4 Biểu diễn hệ 16 của 356 Chế độ dịch chuyển byte với thanh ghi R11 Biểu diễn hệ 16 của 25	MOVW 356(R4), 25(R11) Chuyển một word từ địa chỉ: "356 cộng với nội dung của R4" đến địa chỉ: "25 cộng nội dung của R11"
	Opcode của lệnh ADDL3 Số 5 Chế độ thanh ghi R0 Chế độ preindex R2 Word gián tiếp tương đối (dịch chuyển từ PC) Số lần dịch chuyển từ PC tương ứng từ vị trí A	ADDL3 #5, R0, @A[R2] Cộng 5 với một số nguyên 32-bit trong R0 và lưu trữ kết quả vào vị trí có địa chỉ: "tổng của A và 4 lần nội dung của R2"

Hình 11.12 Ví dụ một số lệnh VAX

Để tránh các vấn đề trên, hai tiêu chí được sử dụng để thiết kế định dạng lệnh VAX:

1. Tất cả các lệnh phải có đủ số lượng toán hạng cần thiết.
2. Tất cả các toán hạng phải có cùng tính tổng quát trong đặc tả.

Với các tiêu chí trên, kết quả ta có một định dạng lệnh rất linh hoạt. Một lệnh gồm một trường opcode độ dài 1 hoặc 2 byte và từ 0 đến 6 toán hạng tùy thuộc vào opcode. Chiều dài lệnh tối thiểu là 1 byte và tối đa là 37 byte. Hình 11.12 là một vài ví dụ về định dạng này.

Với các lệnh VAX bắt đầu bằng opcode 1 byte. Với kích thước có thể mã hóa hầu hết các lệnh VAX. Tuy nhiên, vì có hơn 300 lệnh khác nhau, 8 bit không đủ nên người ta sử dụng thêm 1 byte thứ hai và quy định như sau: nếu byte đầu tiên là FD và FF (biểu diễn dưới dạng thập lục phân) thì đây là opcode mở rộng, và lúc này opcode thực tế được xác định trong byte thứ hai.

Phần còn lại của lệnh gồm tối đa sáu toán hạng. Một toán hạng tối thiểu là 1 byte, trong đó 4 bit ngoài cùng bên trái là chế độ địa chỉ. Ngoại lệ duy nhất của quy định này là chế độ literal, được nhận biết bằng 2 bit ngoài cùng bên trái có giá trị 00 và để lại phần trống cho literal 6-bit. Vì điều này, định dạng lệnh có thể thiết lập một trong 12 chế độ địa chỉ khác nhau cho toán hạng.

Lệnh VAX có thể có đến sáu toán hạng, ví dụ:

ADDP6 OP1, OP2, OP3, OP4, OP5, OP6

Lệnh này thực hiện việc cộng hai số thập phân đóng. OP1 và OP2 chỉ ra chiều dài và địa chỉ bắt đầu của một chuỗi thập phân; OP3 and OP4 chỉ ra chuỗi thứ hai. Hai chuỗi này được cộng và kết quả được lưu vào chuỗi thập phân có chiều dài và địa chỉ bắt đầu được chỉ ra trong OP5 và OP6.

Tập lệnh VAX cung cấp nhiều các hoạt động và chế độ địa chỉ khác nhau. Điều này mang lại cho người lập trình, như là người viết trình biên dịch, một công cụ mạnh mẽ và linh hoạt trong quá trình phát triển chương trình. Về lý thuyết, điều này sẽ mang lại một trình biên dịch ngôn ngữ máy hiệu quả cho các ngôn ngữ lập trình bậc cao và thông thường sẽ giúp sử dụng hiệu quả các tài nguyên của bộ xử lý. Tuy nhiên, nhược điểm của nó là làm tăng độ phức tạp của bộ xử lý so với các tập lệnh khác.

11.3. CÂU HỎI

1. Định nghĩa ngắn gọn chế độ địa chỉ tức thì.
2. Định nghĩa ngắn gọn chế độ địa chỉ trực tiếp.
3. Định nghĩa ngắn gọn chế độ địa chỉ gián tiếp.
4. Định nghĩa ngắn gọn chế độ địa chỉ thanh ghi.
5. Định nghĩa ngắn gọn chế độ địa chỉ gián tiếp thanh ghi.
6. Định nghĩa ngắn gọn chế độ địa chỉ dịch chuyển.
7. Định nghĩa ngắn gọn chế độ địa chỉ tương đối.
8. Ưu điểm của chế độ autoindexing là gì?
9. Sự khác nhau giữa chế độ postindexing và preindexing là gì?
10. Sự kiện nào để xác định việc sử dụng các bit địa chỉ của một lệnh?
11. Những lợi thế và bất lợi của việc sử dụng một định dạng hướng dẫn thay đổi chiều dài là gì?

Chương 12. CẤU TRÚC VÀ CHỨC NĂNG CỦA BỘ XỬ LÝ

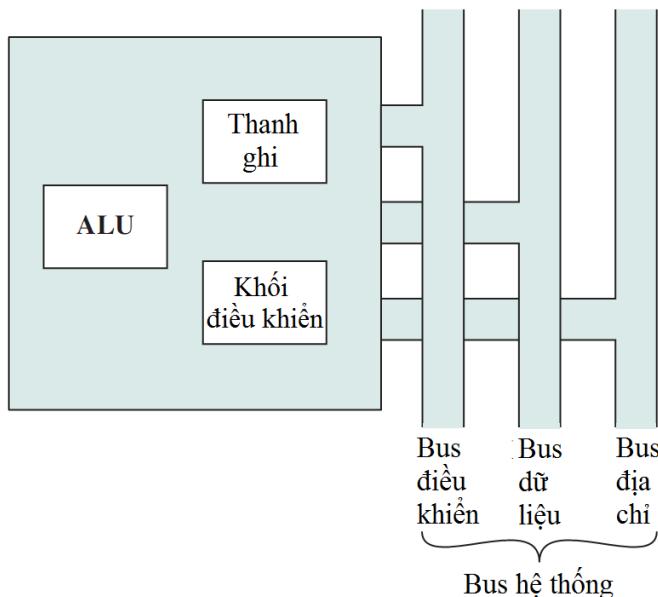
Chúng ta bắt đầu bằng việc tổng kết về tổ chức bộ xử lý. Sau đó, ta phân tích về các thành phần tạo thành bộ nhớ nội bộ bên trong bộ xử lý, mô tả về chu kỳ lệnh và một kỹ thuật phổ biến được gọi là pipelining lệnh.

12.1. TỔ CHỨC CỦA BỘ XỬ LÝ

Để hiểu hơn về tổ chức bộ xử lý, ta xét các nhiệm vụ mà nó phải thực hiện:

- **Truy xuất lệnh:** Bộ xử lý đọc lệnh từ bộ nhớ (thanh ghi, bộ nhớ cache, bộ nhớ chính).
- **Giải mã lệnh:** Lệnh được giải mã để xác định hành động nào được yêu cầu.
- Truy xuất dữ liệu: Việc thực thi một lệnh có thể yêu cầu đọc dữ liệu từ bộ nhớ hoặc một mô-đun I/O.
- **Xử lý dữ liệu:** Việc thực thi một lệnh có thể yêu cầu thực hiện một số phép tính số học hoặc logic trên dữ liệu.
- **Ghi dữ liệu:** Kết thúc việc thực hiện có thể yêu cầu ghi dữ liệu vào bộ nhớ hoặc một mô-đun I / O.

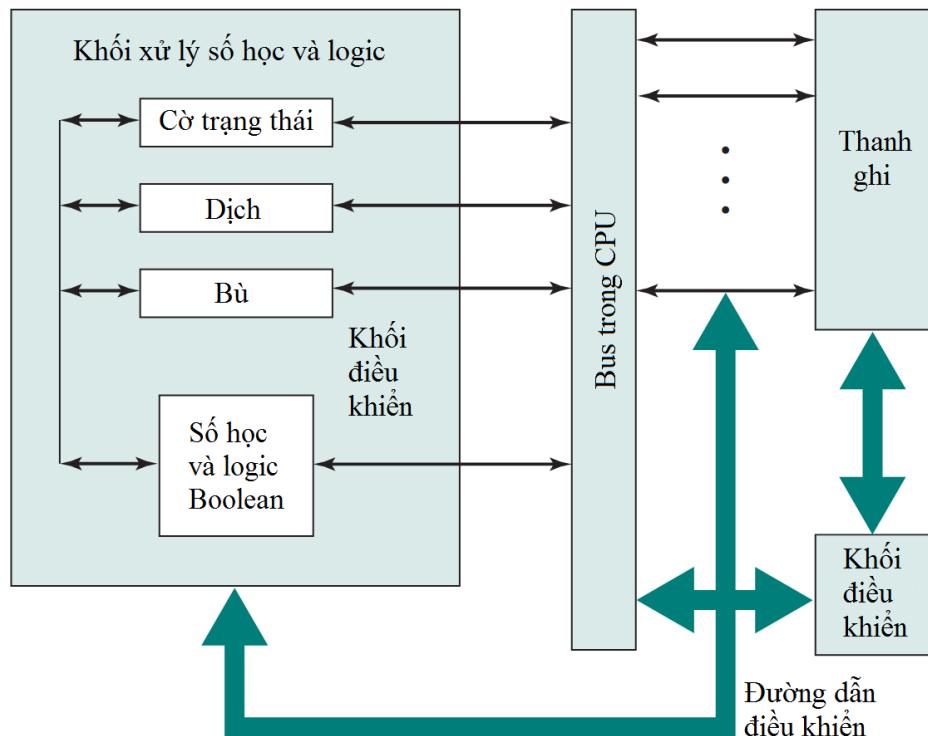
Để thực hiện những việc này, rõ ràng là bộ xử lý cần phải lưu tạm thời một số dữ liệu. Nó phải nhớ vị trí của lệnh gần nhất vừa thực thi để có thể biết được nơi truy xuất lệnh tiếp theo. Nó cần lưu trữ tạm thời các lệnh và dữ liệu trong khi một lệnh đang được thực thi. Nói cách khác, bộ xử lý cần một bộ nhớ nhỏ bên trong nó.



Hình 12.1 CPU và bus hệ thống

Hình 12.1 là sơ đồ khái niệm đơn giản của một bộ xử lý, cho thấy kết nối của nó với phần còn lại của hệ thống thông qua bus hệ thống. Như đã đề cập ở chương 3, các thành phần chính của bộ xử lý là *khối số học và logic* (ALU) và *khối điều khiển* (CU). ALU thực hiện tính toán hoặc xử lý dữ liệu. Khối điều khiển kiểm soát việc di chuyển dữ liệu và lệnh vào và ra khỏi bộ xử lý và điều khiển hoạt động của ALU. Ngoài ra, hình vẽ này cũng thể hiện bộ nhớ bên trong CPU, bao gồm một tập hợp các vị trí lưu trữ, được gọi là *thanh ghi*.

Hình 12.2 là sơ đồ khái chi tiết hơn của bộ xử lý. Các đường truyền dữ liệu và các đường điều khiển logic được thể hiện, bao gồm một thành phần được gọi là bus trong bộ xử lý. Thành phần này là cần thiết để truyền dữ liệu giữa các thanh ghi khác nhau với ALU vì trên thực tế, ALU chỉ hoạt động trên dữ liệu nằm trong bộ nhớ nội bộ của bộ xử lý. Hình vẽ này cũng cho thấy các thành phần cơ bản điển hình của ALU. Lưu ý sự tương đồng giữa cấu trúc bên trong của máy tính và cấu trúc bên trong của bộ xử lý. Cả hai trường hợp đều có một tập hợp nhỏ vài thành phần chính (máy tính: bộ xử lý, I/O, bộ nhớ; bộ xử lý: khối điều khiển, ALU, thanh ghi) được kết nối với nhau bằng đường dẫn dữ liệu.



Hình 12.2 Cấu trúc bên trong CPU

12.2. TỔ CHỨC THANH GHI

Như chúng ta đã thảo luận trong Chương 4, hệ thống máy tính sử dụng một hệ thống phân cấp bộ nhớ. Bộ nhớ ở cấp độ cao hơn trong hệ phân cấp thì nhanh hơn, nhỏ hơn và đắt hơn (trên mỗi bit). Tập hợp các thanh ghi trong bộ xử lý có chức năng như một cấp độ bộ nhớ phía trên bộ nhớ chính và cache trong hệ thống phân cấp. Các thanh ghi trong bộ xử lý thực hiện hai vai trò:

- **Thanh ghi hiển thị với người dùng:** Cho phép lập trình viên hợp ngữ hoặc ngôn ngữ máy giảm thiểu các tham chiếu bộ nhớ chính bằng cách tối ưu hóa việc sử dụng thanh ghi.
- **Thanh ghi điều khiển và trạng thái:** Được khối điều khiển sử dụng để điều khiển hoạt động của bộ xử lý và các chương trình hệ điều hành sử dụng để kiểm soát việc thực thi chương trình.

Tuy nhiên không thể phân loại rõ ràng các thanh ghi vào hai loại này. Ví dụ, trên một số máy tính, bộ đếm chương trình là hiển thị với người dùng (ví dụ, x86), nhưng trong

nhiều máy khác thì không hiển thị. Tuy nhiên, ta sẽ sử dụng các hai loại này để tiện cho việc trình bày nội dung tiếp theo.

12.2.1. Thanh ghi hiển thị với người dùng

Thanh ghi hiển thị với người dùng là thanh ghi được tham chiếu bằng ngôn ngữ máy mà bộ xử lý thực hiện. Ta có thể phân loại các thanh ghi này thành các loại sau:

- Đa năng
- Dữ liệu
- Địa chỉ
- Mã điều kiện

Thanh ghi đa năng có thể được lập trình viên gán cho nhiều chức năng khác nhau. Đôi khi việc sử dụng chúng trong tập lệnh là trực giao với hoạt động. Tức là, thanh ghi đa năng bất kỳ có thể chứa toán hạng cho bất kỳ opcode nào. Tuy nhiên, việc sử dụng thanh ghi đa năng thường có những hạn chế. Ví dụ, có thể có thanh ghi chuyên dụng cho các phép toán dấu chấm động và ngắn xép.

Trong một số trường hợp, thanh ghi đa năng có thể được sử dụng cho chức năng định địa chỉ (ví dụ: thanh ghi gián tiếp, thanh ghi dịch). Trong trường hợp khác, thanh ghi dữ liệu và thanh ghi địa chỉ được phân biệt một phần hoặc phân biệt hoàn toàn. **Thanh ghi dữ liệu** có thể được sử dụng chỉ để chứa dữ liệu và không được sử dụng trong tính toán địa chỉ toán hạng. **Thanh ghi địa chỉ** có thể là đa năng, hoặc là chuyên dụng cho một chế độ địa chỉ cụ thể. Ví dụ, một vài thanh ghi địa chỉ gồm:

- **Con trỏ segment:** Trong máy sử dụng địa chỉ phân đoạn, thanh ghi segment chứa địa chỉ của base của segment. Có thể có nhiều thanh ghi: chẳng hạn, một cho hệ điều hành và một cho tiến trình đang chạy
- **Thanh ghi index:** được sử dụng trong địa chỉ index và autoindex
- **Con trỏ ngắn xép:** Nếu sử dụng chế độ địa chỉ ngắn xép thì thường có một thanh ghi chuyên dụng trỏ tới đầu ngắn xép. Điều này cho phép sử dụng địa chỉ ngầm định; tức là, lệnh push, pop và các lệnh ngắn xép khác không cần phải có toán hạng ngắn xép rõ ràng.

Loại thanh ghi cuối cùng, hiển thị ít nhất một phần với người dùng, chứa **mã điều kiện** (còn được gọi là cờ). Mã điều kiện là các bit mà phần cứng của bộ xử lý thiết lập theo kết quả của hành động. Ví dụ, một phép tính số học có thể tạo ra kết quả dương, âm, bằng không, hoặc tràn. Ngoài việc kết quả được lưu trữ trong thanh ghi hoặc bộ nhớ, một mã điều kiện cũng được thiết lập tương ứng. Sau đó, mã này có thể được sử dụng trong lệnh rẽ nhánh có điều kiện.

Các bit mã điều kiện được tập hợp lại trong một hoặc nhiều thanh ghi. Thông thường, chúng tạo thành một phần của thanh ghi điều khiển. Nói chung, các lệnh máy cho phép đọc các bit này bằng tham chiếu ngầm định, nhưng lập trình viên không thể thay đổi chúng.

Trong một số máy, một lệnh gọi chương trình con sẽ dẫn đến việc tắt cả các thanh ghi hiển thị với người dùng được tự động lưu lại, để có thể khôi phục khi trở về. Bộ xử lý thực hiện việc lưu và khôi phục thanh ghi này như một phần công việc trong quá trình thực thi

các lệnh gọi và trả về. Điều này cho phép các chương trình con sử dụng thanh ghi hiển thị với người dùng một cách độc lập. Trên các máy khác, người lập trình có trách nhiệm lưu lại nội dung của các thanh ghi hiển thị với người dùng trước một lệnh gọi chương trình con, bằng cách thêm vào chương trình các lệnh phù hợp.

12.2.2. Thanh ghi điều khiển và trạng thái

Nhiều thanh ghi bộ xử lý được sử dụng để điều khiển hoạt động của bộ xử lý. Phần lớn thanh ghi không hiển thị với người dùng. Một số khác có thể hiển thị với các lệnh máy được thực hiện trong chế độ điều khiển hoặc chế độ hệ điều hành.

Tất nhiên, các máy khác nhau sẽ có tổ chức thanh ghi khác nhau và sử dụng các thuật ngữ khác nhau. Bốn thanh ghi sau đây là cần thiết để thực thi lệnh:

- **Bộ đếm chương trình (PC):** Chứa địa chỉ của lệnh sắp được truy xuất
- **Thanh ghi lệnh (IR):** Chứa lệnh vừa được truy xuất gần đây nhất
- **Thanh ghi địa chỉ bộ nhớ (MAR):** Chứa địa chỉ của một vị trí trong bộ nhớ
- **Thanh ghi đệm bộ nhớ (MBR):** Chứa một từ dữ liệu sắp được ghi vào bộ nhớ hoặc từ vừa được đọc ra.

Không phải mọi bộ xử lý đều có các thanh ghi kiểu MAR và MBR. Tuy nhiên, mọi CPU cần phải có một cơ chế đệm tương tự như MAR và MBR sao cho các bit truyền lên bus hệ thống được phân tầng và các bit đọc từ bus dữ liệu được lưu tạm thời.

Thông thường, bộ xử lý cập nhật PC sau mỗi lần truy xuất lệnh để PC luôn trỏ đến lệnh tiếp theo sẽ được thực thi. Lệnh rẽ nhánh hoặc lệnh skip cũng sẽ thay đổi nội dung của PC. Lệnh đã truy xuất được tải vào thanh ghi IR, nơi phân tích các opcode và nhận diện toán hạng. Dữ liệu được trao đổi với bộ nhớ nhờ thanh ghi MAR và MBR. Trong một hệ thống sử dụng bus, MAR kết nối trực tiếp tới bus địa chỉ, và MBR kết nối trực tiếp với bus dữ liệu. Các thanh ghi hiển thị với người dùng trao đổi dữ liệu với MBR.

Bốn thanh ghi này được sử dụng trong việc di chuyển dữ liệu giữa bộ xử lý và bộ nhớ. Trong bộ xử lý, dữ liệu phải được đưa đến ALU để xử lý. ALU có thể có quyền truy cập trực tiếp vào MBR và các thanh ghi hiển thị với người dùng. Ngoài ra, có thể có thêm các thanh ghi đệm đặt gần ALU; các thanh ghi này phục vụ như thanh ghi đầu vào và đầu ra cho ALU và trao đổi dữ liệu với MBR và các thanh ghi hiển thị với người dùng.

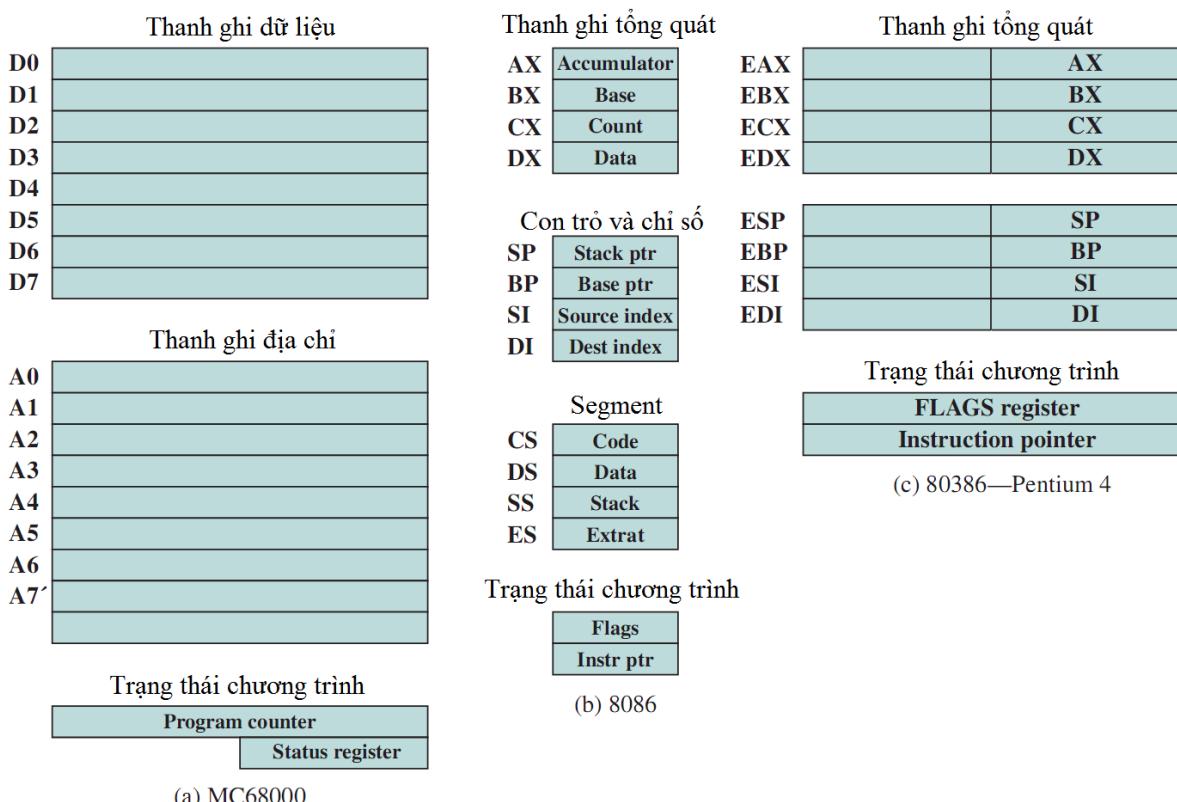
Nhiều bộ xử lý được thiết kế bao gồm một thanh ghi hoặc một tập hợp thanh ghi gọi là *từ trạng thái chương trình* (program status word - PSW), có chứa thông tin trạng thái. PSW thường chứa mã điều kiện cùng với các thông tin trạng thái khác. Các trường hoặc cờ phổ biến bao gồm:

- **Sign:** Chứa bit dấu của kết quả của phép tính số học cuối cùng.
- **Zero:** Thiết lập khi kết quả bằng 0.
- **Carry:** Thiết lập nếu một phép tính có nhớ (phép cộng) hoặc vay (phép trừ) vào bit có bậc lớn hơn. Được sử dụng cho các phép tính số học nhiều từ.
- **Equal:** Thiết lập nếu kết quả so sánh logic là bằng nhau.
- **Overflow:** Được sử dụng để chỉ định sự tràn số học.
- **Interrupt Enable/Disable:** Được sử dụng để cho phép hoặc vô hiệu hóa gián đoạn.

- Supervisor:** Cho biết bộ xử lý đang thực hiện trong chế độ giám sát hay chế độ người dùng. Một số lệnh privileged chỉ có thể được thực hiện trong chế độ giám sát, và một số vùng bộ nhớ chỉ có thể được truy cập trong chế độ giám sát.

12.2.3. Ví dụ tổ chức thanh ghi vi xử lý

Ta sẽ dễ dàng hình dung hơn nếu xem xét và so sánh tổ chức thanh ghi của một số hệ thống. Trong phần này, ta xem xét hai vi xử lý 16 bit được thiết kế trong cùng khoảng thời gian: Motorola MC68000 và Intel 8086. Hình 12.3a và b mô tả tổ chức thanh ghi của từng vi xử lý; các thanh ghi nội bộ thuần túy, chẳng hạn như MAR, không được thể hiện.



Hình 12.3 Ví dụ tổ chức thanh ghi vi xử lý

MC68000 phân chia các thanh ghi 32 bit của nó thành 8 thanh ghi dữ liệu và 9 thanh ghi địa chỉ. 8 thanh ghi dữ liệu được sử dụng chủ yếu cho thao tác dữ liệu và cũng được sử dụng để định địa chỉ (như thanh ghi index). Độ rộng của thanh ghi cho phép các hoạt động dữ liệu 8, 16 và 32 bit, được xác định bởi opcode. Các thanh ghi địa chỉ chứa địa chỉ 32 bit (không phân đoạn); hai trong số các thanh ghi này cũng được sử dụng làm con trỏ ngăn xếp, một cho người dùng và một cho hệ điều hành, tùy thuộc vào chế độ thực thi hiện tại. Cả hai thanh ghi này được đánh số 7, bởi vì tại một thời điểm chỉ có thể sử dụng một trong hai. MC68000 cũng có thanh ghi PC 32 bit và thanh ghi trạng thái 16 bit.

Intel 8086 có cách tổ chức thanh ghi khác. Mỗi thanh ghi có mục đích riêng, mặc dù một số thanh ghi cũng có thể được sử dụng như là đa năng. 8086 chia bốn thanh ghi dữ liệu 16 bit, có thể đánh địa chỉ cho từng byte hoặc từng 16 bit, và bốn thanh ghi index và con trỏ 16 bit. Trong một số lệnh, các thanh ghi dữ liệu có thể là đa năng. Trong các lệnh khác, thanh ghi được sử dụng ngầm định. Ví dụ, lệnh nhân luôn sử dụng thanh ghi tích luỹ

(accumulator). Bốn thanh ghi con trả cũng được sử dụng ngầm trong một số hành động, mỗi thanh ghi chứa một offset phân đoạn. Cũng có bốn thanh ghi segment 16 bit. Ba trong bốn thanh ghi segment được sử dụng một cách riêng biệt, ngầm định, lần lượt trả đến segment của lệnh hiện tại (hữu ích cho lệnh rẽ nhánh), một segment chứa dữ liệu, và một segment chứa ngắn xếp. Việc sử dụng ngầm định và dành riêng này cho phép mã hoá ngắn gọn nhưng tính linh hoạt giảm. 8086 cũng bao gồm một con trả lệnh và một tập hợp các cờ điều khiển và cờ trạng thái 1 bit.

Cần nhấn mạnh quan điểm của sự so sánh này. Không có cách tổ chức thanh ghi bộ xử lý nào được cho là tốt nhất. Cũng như việc thiết kế tập lệnh và rất nhiều vấn đề thiết kế bộ xử lý khác, lựa chọn thiết kế nào vẫn là vấn đề thuộc về quan điểm và thị hiếu.

Một điểm tham khảo thứ hai liên quan đến thiết kế tổ chức thanh ghi được minh họa trong hình 12.3c. Hình này cho thấy tổ chức thanh ghi hiển thị với người dùng của Intel 80386, bộ xử lý 32 bit được phát triển mở rộng từ 8086. 80386 sử dụng thanh ghi 32 bit. Tuy nhiên, để cung cấp khả năng tương thích cho các chương trình được viết trên máy tính cũ, 80386 vẫn giữ lại tổ chức thanh ghi cũ và nhúng vào tổ chức mới. Do điểm hạn chế trong thiết kế này, độ linh hoạt trong thiết kế tổ chức thanh ghi trong các kiến trúc bộ xử lý 32 bit giảm đi.

12.3. CHU KÌ LỆNH

Trong mục 3.2, ta đã trình bày về chu kì lệnh của bộ xử lý (Hình 3.9). Nhắc lại, một chu kì lệnh gồm các giai đoạn sau:

- **Truy xuất:** Đọc lệnh tiếp theo từ bộ nhớ vào bộ xử lý
- **Thi hành:** Giải nghĩa opcode và thực hiện hành động được chỉ định
- **Gián đoạn:** Nếu gián đoạn được kích hoạt và đã xảy ra một gián đoạn, lưu trạng thái xử lý hiện tại và phục vụ gián đoạn đó.

Bây giờ ta sẽ tìm hiểu kỹ hơn về chu kì lệnh. Trước tiên, cần phải giới thiệu một giai đoạn bổ sung, được gọi là chu kỳ gián tiếp.

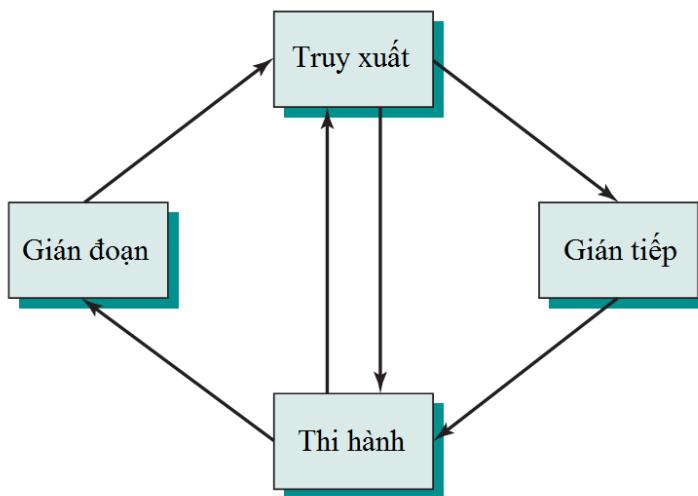
12.3.1. Chu kì gián tiếp

Ta đã thấy trong chương 13 rằng việc thực thi một lệnh có thể liên quan đến một hoặc nhiều toán hạng trong bộ nhớ, mỗi toán hạng yêu cầu một lần truy cập bộ nhớ. Hơn nữa, nếu sử dụng địa chỉ gián tiếp thì cần có thêm các truy cập bộ nhớ bổ sung.

Có thể coi việc truy xuất địa chỉ gián tiếp như một giai đoạn lệnh khác như thể hiện trong hình 12.4. Luồng hoạt động chính bao gồm truy xuất lệnh và thực thi lệnh diễn ra luân phiên. Sau khi một lệnh được truy xuất, nó được kiểm tra để xác định xem có liên quan đến địa chỉ gián tiếp hay không. Nếu có, các toán hạng yêu cầu được truy xuất bằng cách sử dụng địa chỉ gián tiếp. Sau khi thực thi, có thể xử lý một gián đoạn trước khi truy xuất lệnh tiếp theo.

Quá trình này cũng có thể được mô tả như hình 12.5. Hình vẽ này mô tả chính xác hơn bản chất của chu kỳ lệnh. Khi một lệnh được truy xuất, các nhận diện toán hạng của lệnh đó phải được xác định. Sau đó, từng toán hạng đầu vào trong bộ nhớ sẽ được truy xuất, và quá trình này có thể đòi hỏi địa chỉ gián tiếp. Các toán hạng lưu ở thanh ghi thì không cần

phải truy xuất. Còn khi opcode được thực hiện, có thể cần một quá trình tương tự để lưu kết quả trong bộ nhớ chính.



Hình 12.4 Chu kỳ gián tiếp

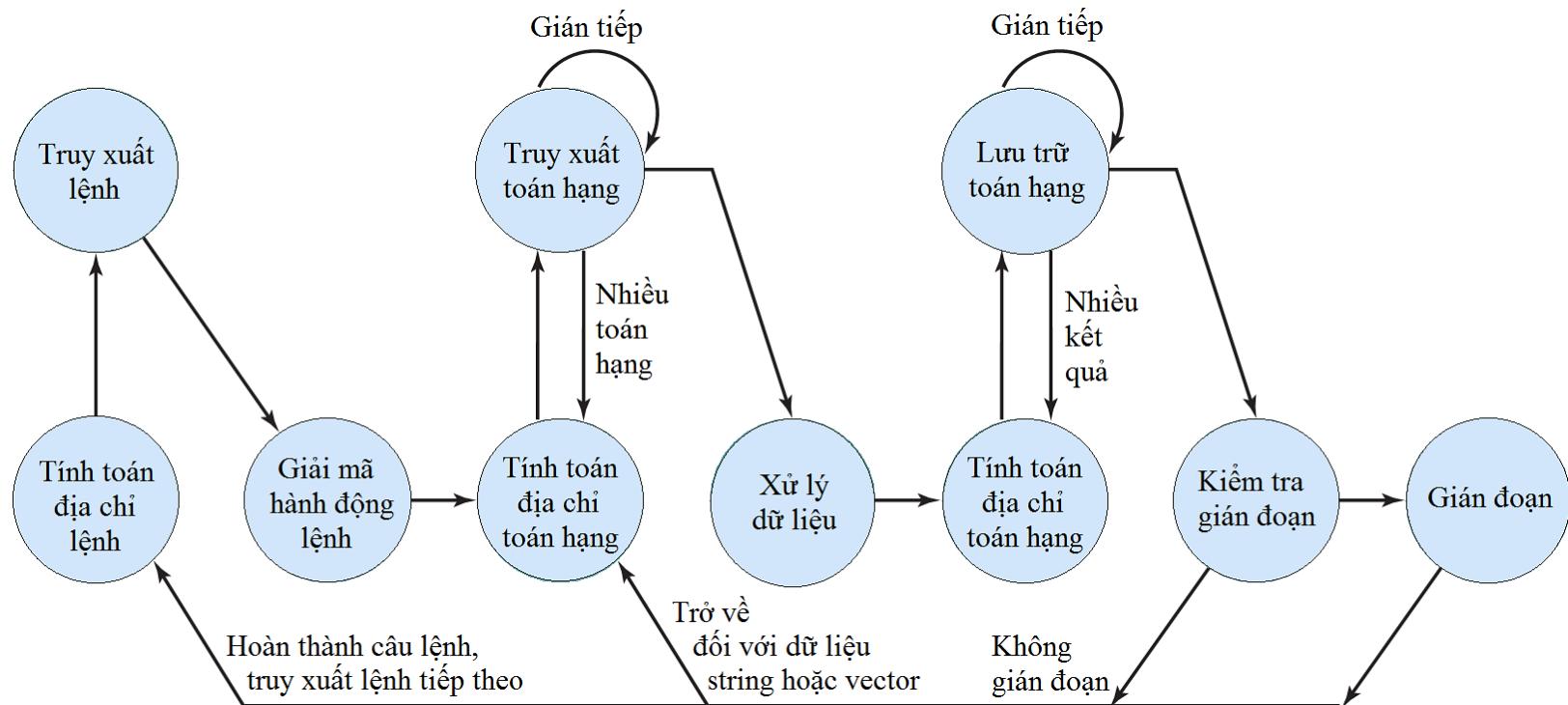
12.3.2. Luồng dữ liệu

Chuỗi sự kiện chi tiết diễn ra trong chu kỳ lệnh phụ thuộc vào thiết kế của bộ xử lý. Tuy nhiên, ta có thể chỉ ra những sự kiện chính phải diễn ra. Giả sử một bộ xử lý sử dụng thanh ghi địa chỉ bộ nhớ (MAR), thanh ghi đệm bộ nhớ (MBR), bộ đếm chương trình (PC) và thanh ghi lệnh (IR).

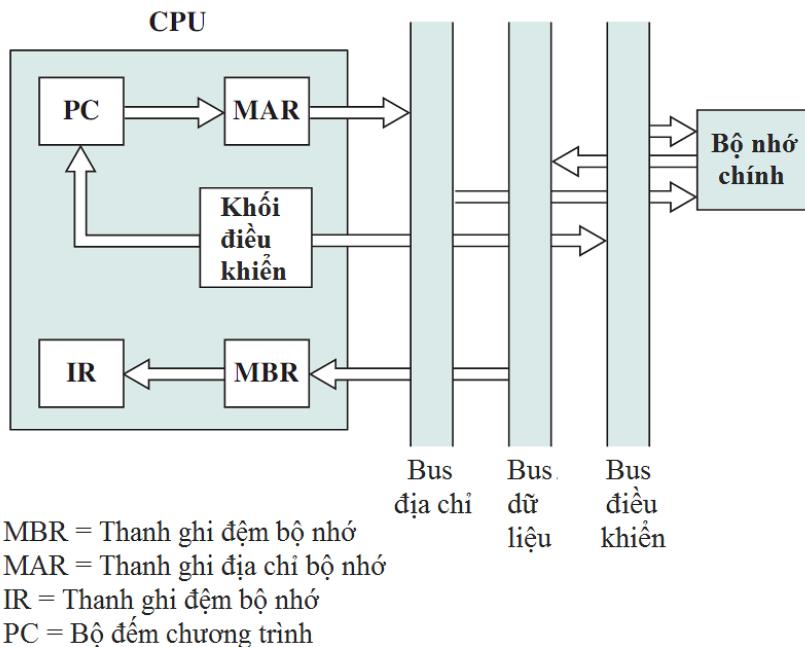
Trong *chu kỳ truy xuất*, một lệnh được đọc từ bộ nhớ. Hình 12.6 thể hiện luồng dữ liệu trong chu kỳ này. PC chứa địa chỉ của lệnh tiếp theo được truy xuất. Địa chỉ này được chuyển đến MAR và được đặt lên bus địa chỉ. Khối điều khiển yêu cầu một lần đọc bộ nhớ; kết quả được đặt trên bus dữ liệu và sao chép vào MBR rồi sau đó chuyển tới IR. Trong khi đó, PC được tăng lên 1, chuẩn bị cho lần truy xuất tiếp theo.

Khi chu kỳ truy xuất kết thúc, khối điều khiển kiểm tra nội dung của IR để xác định xem nó có chứa một nhận diện toán hạng sử dụng địa chỉ gián tiếp hay không. Nếu có, một *chu kỳ gián tiếp* sẽ được thực hiện. Như thể hiện trong hình 12.7, đây là một chu kỳ đơn giản. N bit ngoài cùng bên phải của MBR, chứa tham chiếu địa chỉ, được chuyển tới MAR. Sau đó khối điều khiển yêu cầu một lần đọc bộ nhớ, để lấy được địa chỉ của toán hạng mong muốn vào MBR.

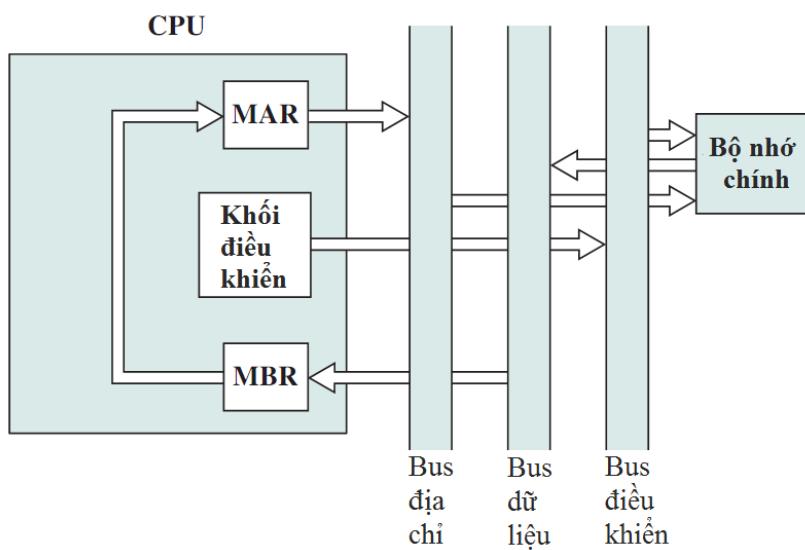
Chu kỳ truy xuất và gián tiếp rất đơn giản và có thể dự đoán được. *Chu kỳ thực thi* có nhiều hình thức diễn ra; hình thức đó phụ thuộc vào lệnh máy nào đang nằm trong IR. Chu kỳ này có thể liên quan đến việc truyền dữ liệu giữa các thanh ghi, đọc hoặc ghi vào bộ nhớ hoặc I/O, và/hoặc sự tham gia của ALU.



Hình 12.5 Sơ đồ trạng thái chu kỳ lệnh

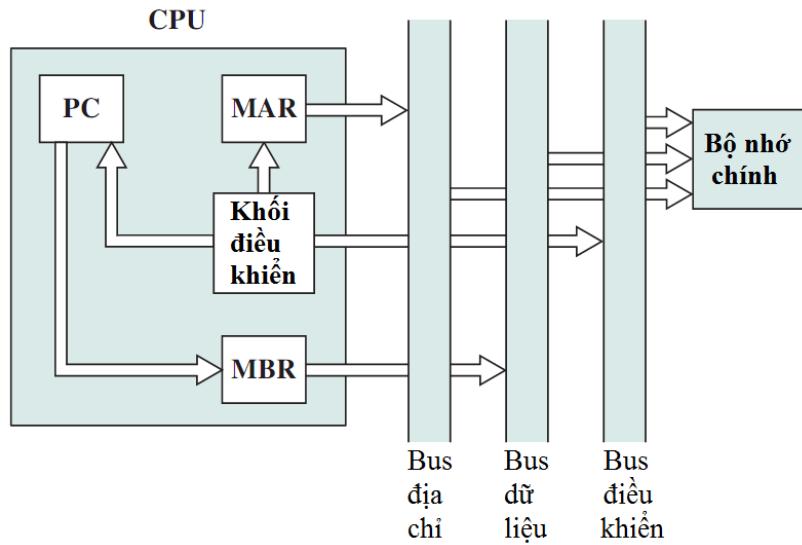


Hình 12.6 Luồng dữ liệu, chu kỳ truy xuất



Hình 12.7 Luồng dữ liệu, chu kỳ gián tiếp

Giống với chu kỳ truy xuất và gián tiếp, chu kỳ gián đoạn đơn giản và có thể dự đoán được (Hình 12.8). Nội dung hiện tại của PC phải được lưu lại để bộ xử lý có thể tiếp tục hoạt động bình thường sau gián đoạn. Do đó, nội dung của PC được chuyển sang MBR để ghi vào bộ nhớ. Vị trí bộ nhớ đặc biệt dành riêng cho mục đích này (chẳng hạn, con trỏ ngăn xếp) được nạp vào MAR từ bộ điều khiển. Địa chỉ của trình xử lý gián đoạn được nạp vào PC. Kết quả là chu kỳ lệnh tiếp theo sẽ bắt đầu bằng việc truy xuất lệnh thích hợp (lệnh đầu tiên trong xử lý gián đoạn).



Hình 12.8 Luồng dữ liệu, chu kỳ gián đoạn

12.4. PIPELINE LỆNH

Khi hệ thống máy tính phát triển, ta có thể đạt được hiệu suất cao hơn bằng cách tận dụng các cải tiến trong công nghệ, ví dụ như sử dụng mạch điện nhanh hơn. Bên cạnh đó, cải tiến tổ chức bộ xử lý cũng giúp cải thiện hiệu suất. Ta đã thấy một số ví dụ về điều này, chẳng hạn như việc sử dụng nhiều thanh ghi thay vì một thanh ghi tích lũy ACC đơn lẻ, và việc sử dụng bộ nhớ cache. Một phương pháp tổ chức khác, khá phổ biến, là pipeline lệnh.

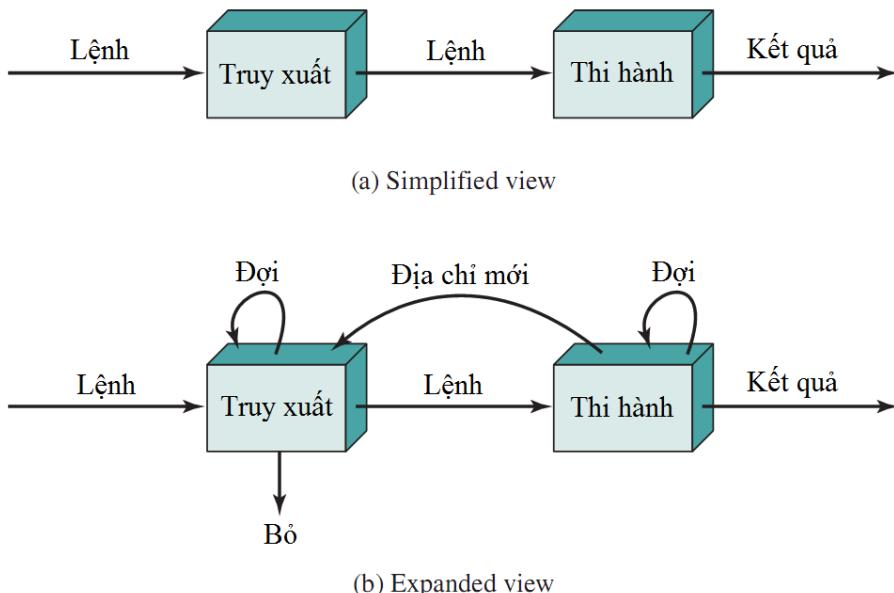
12.4.1. Chiến lược pipelining

Pipelining lệnh tương tự như việc sử dụng dây chuyền lắp ráp trong nhà máy sản xuất. Dây chuyền lắp ráp dựa trên việc một sản phẩm đi qua nhiều khâu sản xuất khác nhau. Bằng cách đưa quá trình sản xuất vào dây chuyền lắp ráp, các sản phẩm ở các khâu khác nhau có thể được thực hiện đồng thời. Quá trình này được gọi là pipelining (đường ống), bởi vì, giống như trong một đường ống, các đầu vào mới được tiếp nhận ở một phía trước khi các đầu vào trước đã trở thành đầu ra ở phía còn lại.

Để áp dụng khái niệm này trong thực thi lệnh, ta phải ghi nhớ rằng một lệnh gồm có nhiều giai đoạn. Ví dụ, hình 12.5 chia chu kỳ lệnh thành 10 hành động theo trình tự. Rõ ràng, việc áp dụng kỹ thuật pipelining là hoàn toàn khả thi.

Chẳng hạn, xét một phương án đơn giản, chia quá trình xử lý lệnh thành hai giai đoạn: truy xuất lệnh và thi hành lệnh. Trong quá trình thi hành một lệnh, có nhiều khoảng thời gian trong đó hành động được thực hiện không đòi hỏi việc truy cập vào bộ nhớ chính. Trong thời gian này, ta có thể truy xuất lệnh tiếp theo song song với việc thi hành lệnh hiện tại. Hình 12.9a mô tả phương pháp này. Pipeline có hai giai đoạn độc lập. Giai đoạn thứ nhất truy xuất một lệnh và lưu đệm nó lại. Khi giai đoạn thứ hai đang rảnh, giai đoạn thứ nhất chuyển lệnh vừa đệm cho nó. Trong khi giai đoạn thứ hai đang thi hành lệnh, giai đoạn thứ nhất sẽ tận dụng mọi chu kỳ bộ nhớ không được sử dụng để truy xuất và đệm lệnh tiếp theo. Điều này được gọi là truy xuất lệnh trước (prefetch) hoặc *truy xuất nối chòng* (*fetch overlap*). Lưu ý rằng phương pháp này, do liên quan đến việc đệm lệnh, đòi

hỏi nhiều thanh ghi hơn. Nói chung, pipelining đòi hỏi nhiều thanh ghi để lưu trữ dữ liệu giữa các giai đoạn.



Hình 12.9 Pipeline lệnh hai giai đoạn

Rõ ràng là quá trình này sẽ tăng tốc việc thực thi lệnh. Nếu các giai đoạn truy xuất và thi hành diễn ra trong khoảng thời gian bằng nhau, thời gian chu kỳ lệnh sẽ giảm một nửa. Tuy nhiên, nếu ta xét kỹ hơn về pipeline (Hình 12.9b), ta sẽ thấy rằng việc tăng gấp đôi tốc độ thực thi là không thực tế vì hai lý do:

1. Thời gian thi hành thường dài hơn thời gian truy xuất. Việc thi hành liên quan đến việc đọc, lưu trữ toán hạng và thực hiện một số phép toán. Do đó, giai đoạn truy xuất có thể phải đợi một lúc trước khi có thể giải phóng bộ đệm.
2. Trong một lệnh rẽ nhánh có điều kiện, địa chỉ của lệnh tiếp theo cần truy xuất là không biết trước. Do đó, giai đoạn truy xuất phải đợi cho đến khi nó nhận được địa chỉ lệnh tiếp theo từ giai đoạn thi hành. Giai đoạn thi hành sau đó có thể phải đợi để lệnh tiếp theo được truy xuất.

Dự đoán có thể làm giảm thời gian lãng phí ở điều thứ hai nêu trên. Một quy tắc đơn giản như sau: Khi một lệnh rẽ nhánh có điều kiện chuyển từ giai đoạn truy xuất sang thi hành, giai đoạn truy xuất lệnh tiếp theo ngay sau lệnh rẽ nhánh trong bộ nhớ. Sau đó, nếu việc rẽ nhánh không được thực hiện thì cũng không mất thời gian. Nếu việc rẽ nhánh được thực hiện, lệnh đã truy xuất phải được loại bỏ và một lệnh khác được truy xuất.

Mặc dù các yếu tố này có làm giảm hiệu quả tiềm năng của pipeline hai giai đoạn, tốc độ thực thi lệnh vẫn tăng lên đáng kể. Để tăng tốc nhanh hơn nữa, pipeline phải có nhiều giai đoạn hơn. Giả sử việc xử lý lệnh được phân chia thành 6 giai đoạn như sau:

- **Truy xuất lệnh (FI – Fetch instruction):** Đọc lệnh mong đợi tiếp theo vào bộ đệm.
- **Giải mã lệnh (DI – Decode instruction):** Xác định opcode và nhận diện toán hạng.

- **Tính toán các toán hạng (CO – Calculate operands):** Tính toán địa chỉ hiệu dụng của từng toán hạng nguồn. Điều này có thể bao gồm tính toán địa chỉ dịch, gián tiếp thanh ghi, gián tiếp hoặc các hình thức địa chỉ khác.
- **Truy xuất toán hạng (FO – Fetch operands):** Truy xuất từng toán hạng từ bộ nhớ. Không cần truy xuất toán hạng từ thanh ghi
- **Thực thi lệnh (EI – Execute instruction):** Thực hiện hành động được chỉ định và lưu trữ kết quả (nếu có) trong vị trí toán hạng đích đã định.
- **Ghi toán hạng (WO – Write operand):** Lưu kết quả vào bộ nhớ.

Với sự phân chia này, các giai đoạn khác nhau diễn ra trong khoảng thời gian gần bằng nhau. Để tiện cho việc minh họa, ta giả sử các khoảng thời gian là bằng nhau. Hình 12.10 cho thấy rằng một pipeline 6 giai đoạn có thể làm giảm thời gian thực thi của 9 câu lệnh từ 54 đơn vị thời gian xuống còn 14 đơn vị thời gian.

Thời gian →

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Lệnh 1	FI	DI	CO	FO	EI	WO								
Lệnh 2		FI	DI	CO	FO	EI	WO							
Lệnh 3			FI	DI	CO	FO	EI	WO						
Lệnh 4				FI	DI	CO	FO	EI	WO					
Lệnh 5					FI	DI	CO	FO	EI	WO				
Lệnh 6						FI	DI	CO	FO	EI	WO			
Lệnh 7							FI	DI	CO	FO	EI	WO		
Lệnh 8								FI	DI	CO	FO	EI	WO	
Lệnh 9									FI	DI	CO	FO	EI	WO

Hình 12.10 Biểu đồ thời gian của hành động pipeline lệnh

Sơ đồ này giả thiết rằng mỗi lệnh đi qua tất cả sáu giai đoạn của pipeline. Điều này không phải lúc nào cũng đúng. Ví dụ, một lệnh LOAD không cần giai đoạn WO. Tuy nhiên, để đơn giản hóa phần cứng của pipeline, ta giả sử rằng mỗi lệnh cần có cả sáu giai đoạn này. Ngoài ra, giả thiết rằng tất cả các giai đoạn có thể được thực hiện song song, và không có xung đột bộ nhớ. Giả thiết, các giai đoạn FI, FO, và WO có liên quan đến việc truy cập bộ nhớ. Trong sơ đồ trên, tất cả các truy cập có thể xảy ra đồng thời. Hầu hết các hệ thống bộ nhớ sẽ không cho phép điều này. Tuy nhiên, giá trị mong muốn có thể nằm

trong cache, hoặc giai đoạn FO hoặc WO có thể rỗng. Do đó, phần lớn thời gian, xung đột bộ nhớ sẽ không làm chậm pipeline.

Còn một số yếu tố khác gây hạn chế việc nâng cao hiệu suất. Nếu sáu giai đoạn không diễn ra trong khoảng thời gian bằng nhau, sẽ có khoảng thời gian chờ đợi giữa các giai đoạn pipeline, như đã thảo luận ở phần pipeline hai giai đoạn. Lệnh rẽ nhánh có điều kiện cũng là một khó khăn khác, vì nó có thể loại bỏ một vài lệnh đã truy xuất. Tương tự, gián đoạn cũng là một sự kiện không đoán trước được. Hình 12.11 cho thấy ảnh hưởng của nhánh có điều kiện, sử dụng chương trình giống như Hình 12.10. Giả sử lệnh 3 là nhánh có điều kiện rẽ tới lệnh 15. Nếu lệnh này chưa được thi hành, sẽ không có cách nào biết được lệnh nào tiếp theo cần được thực thi. Trong ví dụ này, pipeline chỉ cần tải lệnh tiếp theo theo thứ tự (lệnh 4) và xử lý. Trong hình 12.10, nhánh không được thực hiện, khi đó ta nhận được lợi ích hiệu suất toàn phần. Trong hình 12.11, nhánh được chọn. Điều này chỉ được xác định khi kết thúc đơn vị thời gian 7. Tại thời điểm này, pipeline phải xóa sạch các lệnh không hữu ích. Trong đơn vị thời gian 8, lệnh 15 đi vào pipeline. Không có lệnh nào được hoàn thành trong các đơn vị thời gian từ 9 đến 12; đây là sự trả giá về hiệu quả hoạt động do ta không thể đoán trước được nhánh. Hình 12.12 cho thấy logic cần thiết để pipelining xử lý nhánh và gián đoạn.

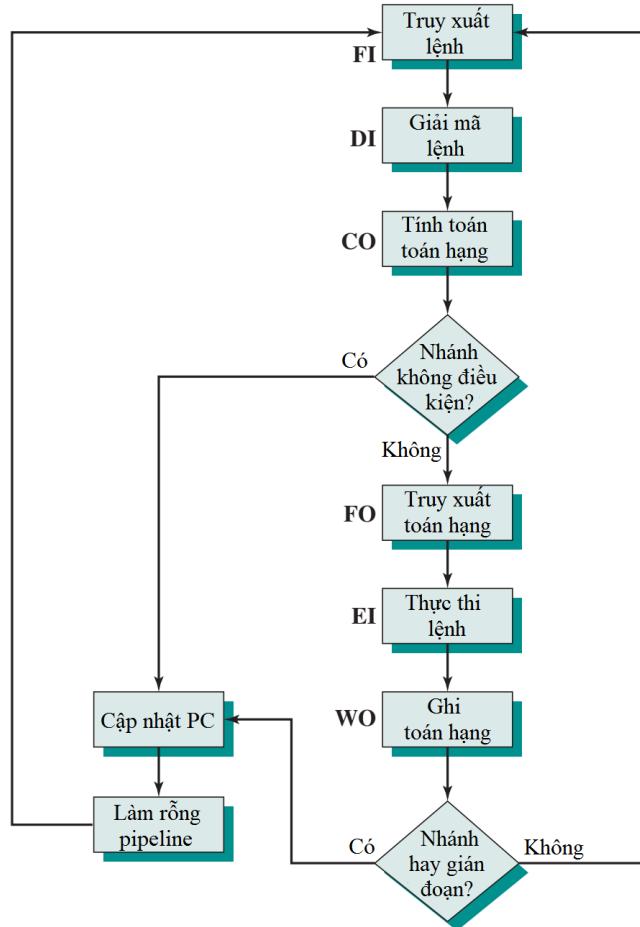
Cũng có một số vấn đề khác phát sinh trong pipeline sáu giai đoạn. Giai đoạn CO có thể phụ thuộc vào nội dung của một thanh ghi mà nội dung này có thể bị thay đổi bởi một lệnh phía trước vẫn còn nằm trong pipeline. Các xung đột thanh ghi và xung đột bộ nhớ khác có thể xảy ra. Hệ thống phải có logic để xử lý đối với kiểu xung đột này.

	Thời gian							Trả giá cho rẽ nhánh						
	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Lệnh 1	FI	DI	CO	FO	EI	WO								
Lệnh 2		FI	DI	CO	FO	EI	WO							
Lệnh 3			FI	DI	CO	FO	EI	WO						
Lệnh 4				FI	DI	CO	FO							
Lệnh 5					FI	DI	CO							
Lệnh 6						FI	DI							
Lệnh 7							FI							
Lệnh 15								FI	DI	CO	FO	EI	WO	
Lệnh 16								FI	DI	CO	FO	EI	WO	

Hình 12.11 Ảnh hưởng của nhánh có điều kiện tới hoạt động của pipeline lệnh

Để làm rõ hoạt động của pipeline, ta có thể xem xét nó bằng một cách mô tả khác. Hình 12.10 và 12.11 cho thấy diễn biến theo thời gian trên trực ngang, với mỗi dòng là diễn biến của một lệnh. Hình 12.13 thể hiện cùng một chuỗi sự kiện, nhưng diễn biến thời gian theo chiều dọc từ trên xuống, mỗi dòng thể hiện trạng thái của pipeline ở một thời điểm nhất định. Trong hình 12.13a (tương ứng với Hình 12.10), ở thời điểm 6, pipeline lấp

đầy bởi 6 lệnh khác nhau trong các giai đoạn thi hành khác nhau; và pipeline vẫn đầy cho tới thời điểm 9; ta giả sử rằng lệnh I9 là lệnh cuối cùng được thực thi. Trong hình 12.13b, (tương ứng với hình 12.11), pipeline đầy ở thời điểm 6 và 7. Tại thời điểm 7, lệnh 3 đang trong giai đoạn thực thi và thực hiện rẽ nhánh đến lệnh 15. Tại đây, lệnh I4 đến I7 được giải phóng khỏi pipeline, do đó tại thời điểm 8, chỉ có hai lệnh trong pipeline, I3 và I15.



Hình 12.12 Pipeline lệnh CPU sáu giai đoạn

	FI	DI	CO	FO	EI	WO
1	I1					
2	I2	I1				
3	I3	I2	I1			
4	I4	I3	I2	I1		
5	I5	I4	I3	I2	I1	
6	I6	I5	I4	I3	I2	I1
7	I7	I6	I5	I4	I3	I2
8	I8	I7	I6	I5	I4	I3
9	I9	I8	I7	I6	I5	I4
10		I9	I8	I7	I6	I5
11			I9	I8	I7	I6
12				I9	I8	I7
13					I9	I8
14						I9

(a) Không rẽ nhánh

	FI	DI	CO	FO	EI	WO
1	I1					
2	I2	I1				
3	I3	I2	I1			
4	I4	I3	I2	I1		
5	I5	I4	I3	I2	I1	
6	I6	I5	I4	I3	I2	I1
7	I7	I6	I5	I4	I3	I2
8	I15					I3
9	I16	I15				
10		I16	I15			
11			I16	I15		
12				I16	I15	
13					I16	I15
14						I16

(b) Rẽ nhánh có điều kiện

Hình 12.13 Một cách mô tả khác về pipeline

12.4.2. Hazard trong pipeline

Trong tiêu mục trước, chúng ta đã đề cập đến một số tình huống có thể làm giảm hiệu suất pipeline. Trong tiêu mục này, ta xem xét vấn đề này một cách có hệ thống hơn.

Hazard trong pipeline, hay là xung đột trong pipeline, xảy ra khi một phần hoặc toàn bộ pipeline phải ngừng lại vì điều kiện không cho phép tiếp tục thực hiện. Sự ngừng pipeline như vậy còn được gọi là *pipeline bubble*. Có ba loại hazard trong pipeline: tài nguyên, dữ liệu và điều khiển.

12.4.2.1. Hazard tài nguyên

Hazard tài nguyên (xung đột tài nguyên) xảy ra khi hai lệnh (hoặc nhiều hơn) đang nằm trong pipeline cần sử dụng nguồn tài nguyên giống nhau. Kết quả là các lệnh này không thể thực hiện song song mà phải được thực hiện lần lượt trong một phần của pipeline. Hazard tài nguyên đôi khi được gọi là *hazard cấu trúc*.

Xét một ví dụ đơn giản về hazard tài nguyên. Giả sử một pipeline đơn giản gồm năm giai đoạn, trong đó mỗi giai đoạn chiếm một chu kỳ đồng hồ. Hình 12.14a mô tả trường hợp lý tưởng, trong đó cứ mỗi chu kỳ đồng hồ lại có một lệnh mới đi vào pipeline. Bây giờ giả sử rằng bộ nhớ chính có một cổng duy nhất và tất cả các hành động truy xuất lệnh, đọc và ghi dữ liệu phải được thực hiện lần lượt từng hành động một. Đồng thời, bỏ qua bộ nhớ cache. Trong trường hợp này, hành động đọc toán hạng hoặc ghi toán hạng từ bộ nhớ không thể được thực hiện song song với hành động truy xuất lệnh. Điều này được mô tả ở hình 12.14b, trong đó giả sử rằng toán hạng nguồn của lệnh I1 nằm trong bộ nhớ chung không nằm ở thanh ghi. Do đó, giai đoạn truy xuất lệnh của pipeline phải nhàn rỗi (idle) trong một chu kỳ trước khi bắt đầu truy xuất lệnh đối với lệnh I3. Hình vẽ này giả thiết rằng tất cả các toán hạng khác đều nằm trong thanh ghi.

		Chu kỳ đồng hồ								
		1	2	3	4	5	6	7	8	9
Lệnh	I1	FI	DI	FO	EI	WO				
	I2		FI	DI	FO	EI	WO			
	I3			FI	DI	FO	EI	WO		
	I4				FI	DI	FO	EI	WO	

(a) Pipeline năm giai đoạn, trường hợp lý tưởng

		Chu kỳ đồng hồ								
		1	2	3	4	5	6	7	8	9
Lệnh	I1	FI	DI	FO	EI	WO				
	I2		FI	DI	FO	EI	WO			
	I3			Idle	FI	DI	FO	EI	WO	
	I4				FI	DI	FO	EI	WO	

(b) Toán hạng nguồn của lệnh I1 trong bộ nhớ

Hình 12.14 Ví dụ về hazard tài nguyên

Một ví dụ khác về xung đột tài nguyên là tình huống trong đó nhiều lệnh đã sẵn sàng vào giai đoạn thi hành lệnh EI và chỉ có một ALU duy nhất. Giải pháp cho hazard tài nguyên này là tăng cường các tài nguyên sẵn có, chẳng hạn như sử dụng nhiều cổng vào bộ nhớ chính và nhiều khối ALU.

12.4.2.2. Hazard dữ liệu

Hazard dữ liệu xảy ra khi có xung đột trong việc truy cập vị trí toán hạng. Nói chung, hazard có thể ở dạng này: Hai lệnh trong một chương trình sẽ được thi hành theo trình tự và cả hai đều truy cập vào một toán hạng bộ nhớ hoặc toán hạng thanh ghi cụ thể. Nếu hai lệnh này được thi hành theo trình tự nghiêm ngặt thì không có vấn đề xảy ra. Tuy nhiên, nếu chúng được thực hiện trong pipeline, giá trị toán hạng có thể được cập nhật và sẽ tạo ra kết quả khác với kết quả của việc thi hành tuân tự nghiêm ngặt. Nói cách khác, chương trình tạo ra kết quả không chính xác do sử dụng pipeline.

Ví dụ, xét chuỗi lệnh máy x86 sau:

ADD EAX,	EBX /* EAX = EAX + EBX
SUB ECX,	EAX /* ECX = ECX - EAX

Lệnh đầu tiên cộng nội dung của thanh ghi 32 bit EAX với EBX và lưu trữ kết quả trong EAX. Lệnh thứ hai lấy nội dung của ECX trừ EAX từ và lưu trữ kết quả trong ECX. Hình 12.16 mô tả diễn biến trong pipeline.

Lệnh ADD chỉ cập nhật thanh ghi EAX khi kết thúc giai đoạn 5, xảy ra ở chu kỳ đồng hồ 5. Tuy nhiên, lệnh SUB cần giá trị đó vào đầu giai đoạn 2, xảy ra ở chu kỳ đồng hồ 4. Để duy trì hoạt động chính xác, pipeline phải ngừng lại trong hai chu kỳ đồng hồ. Do đó, khi không có các phần cứng đặc biệt và các thuật toán phòng tránh cụ thể, hazard dữ liệu kiểu này dẫn đến việc sử dụng pipeline không hiệu quả.

Có ba loại hazard dữ liệu;

- **Đọc sau ghi (RAW – Read after write):** Một lệnh thay đổi nội dung của một vị trí bộ nhớ hoặc thanh ghi và lệnh tiếp theo đọc dữ liệu trong vị trí bộ nhớ hoặc thanh ghi đó. Hazard xảy ra nếu việc đọc diễn ra trước khi hành động ghi hoàn tất.
- **Ghi sau đọc (WAR – Write after read):** Một lệnh đọc nội dung của một vị trí bộ nhớ hoặc thanh ghi và lệnh tiếp theo ghi vào vị trí đó. Hazard xảy ra nếu hành động ghi hoàn thành trước khi diễn ra hành động đọc.
- **Ghi sau ghi (WAW – Write after write):** Hai lệnh cùng ghi vào một vị trí. Hazard xảy ra nếu các hành động ghi diễn ra theo trình tự ngược lại với trình tự đã định.

Hình 12.15 đưa ra ví dụ về hazard RAW.

12.4.2.1. Hazard điều khiển

Hazard điều khiển, còn được gọi là hazard rẽ nhánh, xảy ra khi pipeline đưa ra quyết định sai về dự báo nhánh và do đó đưa sai các lệnh vào pipeline rồi sau này phải bỏ chúng đi. Chúng ta sẽ thảo luận các giải pháp để đối phó với hazard điều khiển trong phần tiếp theo.

Chu kỳ đồng hồ										
	1	2	3	4	5	6	7	8	9	10
ADD EAX, EBX	FI	DI	FO	EI	WO					
SUB ECX, EAX		FI	DI	Idle		FO	EI	WO		
I3			FI			DI	FO	EI	WO	
I4						FI	DI	FO	EI	WO

Hình 12.15 Ví dụ về hazard dữ liệu

12.4.3. Đối phó với hazard rẽ nhánh

Đảm bảo một luồng lệnh ổn định đưa đến các giai đoạn đầu tiên của pipeline là một trong những vấn đề chính trong thiết kế pipeline lệnh. Như chúng ta đã thấy, trở ngại chính đối với vấn đề này là lệnh rẽ nhánh có điều kiện. Chừng nào mà lệnh này còn chưa được thi hành, sẽ không thể xác định nhánh đó có được thực hiện hay không.

Có nhiều phương pháp để đối phó với nhánh có điều kiện như sau:

- Sử dụng nhiều luồng
- Truy xuất trước **mục tiêu** rẽ nhánh
- Bộ đệm vòng lặp
- Dự báo rẽ nhánh
- **Rẽ nhánh chậm**

12.4.3.1. Sử dụng nhiều luồng

Lệnh rẽ nhánh có thể gây ảnh hưởng cho một pipeline đơn giản vì nó phải chọn một trong hai lệnh cho lần truy xuất tiếp theo và có thể chọn sai. Một phương pháp được đưa ra

là sao chép lại phần mở đầu của pipeline và cho phép pipeline truy xuất cả hai lệnh, sử dụng hai luồng khác nhau. Phương pháp này có hai vấn đề:

- Nhiều pipeline kéo theo việc có thể tranh chấp để truy cập vào thanh ghi và bộ nhớ.
- Có thể có thêm các lệnh rẽ nhánh đi vào pipeline (thêm luồng khác) trước khi quyết định rẽ nhánh ban đầu được giải quyết. Mỗi lệnh như vậy lại cần thêm một luồng bổ sung nữa.

Mặc dù có những hạn chế như vậy, phương pháp này thực sự có thể cải thiện hiệu suất. Ví dụ về máy có nhiều luồng pipeline là IBM 370/168 và IBM 3033.

12.4.3.2. Truy xuất trước mục tiêu rẽ nhánh

Khi nhận được một lệnh rẽ nhánh có điều kiện, mục tiêu của lệnh rẽ nhánh đó được truy xuất trước, thêm vào lệnh phía sau lệnh rẽ nhánh. Mục tiêu này sau đó được lưu lại cho đến khi lệnh nhánh được thi hành. Nếu rẽ nhánh được thực hiện thì mục tiêu của nó đã được truy xuất trước.

IBM 360/91 sử dụng phương pháp này.

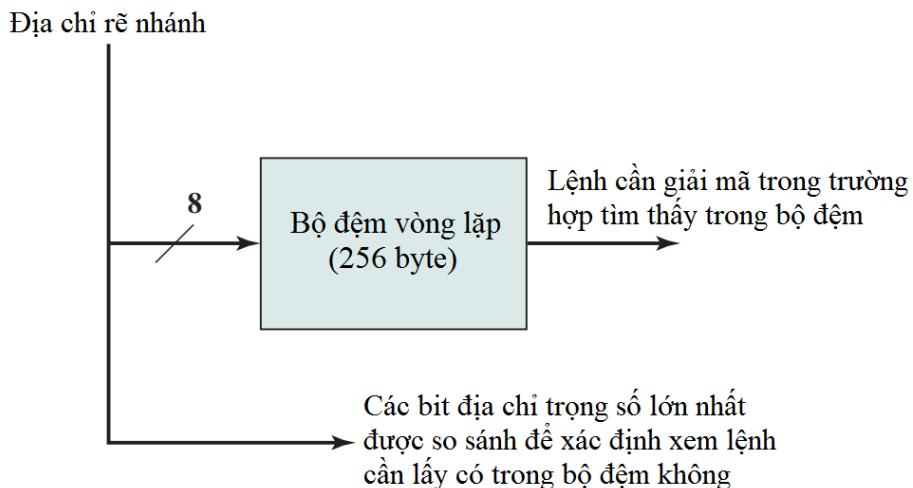
12.4.3.3. Bộ đệm vòng lặp

Bộ đệm vòng lặp là một bộ nhớ nhỏ, tốc độ rất cao và chứa n lệnh được truy xuất gần đây nhất, theo thứ tự. Nó được duy trì bởi giai đoạn truy xuất lệnh của pipeline. Nếu một rẽ nhánh được thực hiện, trước tiên, phần cứng phải kiểm tra xem mục tiêu rẽ nhánh có nằm trong bộ đệm hay không. Nếu có, lệnh tiếp theo sẽ được truy xuất ra từ bộ đệm. Bộ đệm vòng lặp có ba lợi ích:

1. Do có sử dụng truy xuất trước, bộ đệm vòng lặp sẽ chứa một vài lệnh tuân tự phía trước địa chỉ truy xuất lệnh hiện tại. Do đó, sẽ có sẵn một số lệnh được truy xuất theo trình tự mà không mất thời gian truy cập bộ nhớ như thông thường.
2. Nếu một rẽ nhánh xảy ra mà mục tiêu của nó chỉ đứng trước địa chỉ của lệnh rẽ nhánh một vài vị trí, mục tiêu sẽ ở sẵn trong bộ đệm. Điều này rất hữu ích cho chuỗi khá thường gặp là IF-THEN và IF-THEN-ELSE.
3. Phương pháp này đặc biệt phù hợp để xử lý các vòng lặp; do đó được gọi tên là bộ đệm vòng lặp. Nếu bộ đệm vòng lặp đủ lớn để chứa tất cả các lệnh trong một vòng lặp, thì những lệnh đó chỉ cần được truy xuất từ bộ nhớ một lần, cho lần lặp đầu tiên. Đối với những lần lặp tiếp theo, tất cả các lệnh cần thiết đã nằm trong bộ đệm.

Bộ đệm vòng lặp hoạt động theo nguyên lý tương tự với bộ nhớ cache dành riêng cho lệnh. Điểm khác biệt là bộ đệm vòng lặp chỉ giữ lại các lệnh theo trình tự và kích thước nhỏ hơn nhiều và do đó giá rẻ hơn.

Hình 12.16 đưa ra ví dụ về bộ đệm vòng lặp. Nếu bộ đệm chứa 256 byte, và địa chỉ được đánh theo byte, khi đó 8 bit địa chỉ có trọng số nhỏ nhất được sử dụng để đánh số bộ đệm. Các bit địa chỉ có trọng số lớn nhất còn lại được kiểm tra để xác định xem mục tiêu rẽ nhánh có nằm trong vùng bao trùm bởi bộ đệm hay không.



Hình 12.16 Bộ đếm vòng lặp

Bộ đếm vòng lặp được sử dụng trong các máy CDC (Star-100, 6600, 7600), CRAY-1 và Motorola 68010.

12.4.3.4. Dự đoán nhánh

Nhiều kỹ thuật khác nhau có thể được sử dụng để dự đoán liệu một rẽ nhánh có được thực hiện hay không. Một số kỹ thuật phổ biến hơn cả là:

- Dự đoán không bao giờ được thực hiện
- Dự đoán luôn luôn được thực hiện
- Dự đoán theo opcode
- Công tắc được thực hiện/không được thực hiện
- Bảng lịch sử rẽ nhánh

Ba kỹ thuật đầu tiên là phương pháp tĩnh: chúng không phụ thuộc vào lịch sử thi hành tính đến trước thời điểm lệnh rẽ nhánh có điều kiện. Hai kỹ thuật sau cùng là phương pháp động: chúng phụ thuộc vào lịch sử thi hành. Hai phương pháp đầu tiên là đơn giản nhất. Chúng hoặc luôn luôn giả định rằng rẽ nhánh sẽ không được thực hiện và tiếp tục truy xuất các lệnh theo trình tự hoặc luôn luôn giả định rằng rẽ nhánh sẽ được thực hiện và luôn truy xuất từ mục tiêu rẽ nhánh. Phương pháp dự đoán không bao giờ được thực hiện là phương pháp phổ biến nhất trong tất cả các phương pháp dự báo nhánh.

Phương pháp tĩnh cuối cùng đưa ra quyết định dựa trên opcode của lệnh rẽ nhánh. Bộ xử lý giả định rằng đối với một số loại opcode rẽ nhánh nhất định thì nhánh sẽ luôn được thực hiện, và đối với một số loại khác thì nhánh sẽ không được thực hiện. Chiến lược này đạt được tỷ lệ truy xuất trước thành công lớn hơn 75%.

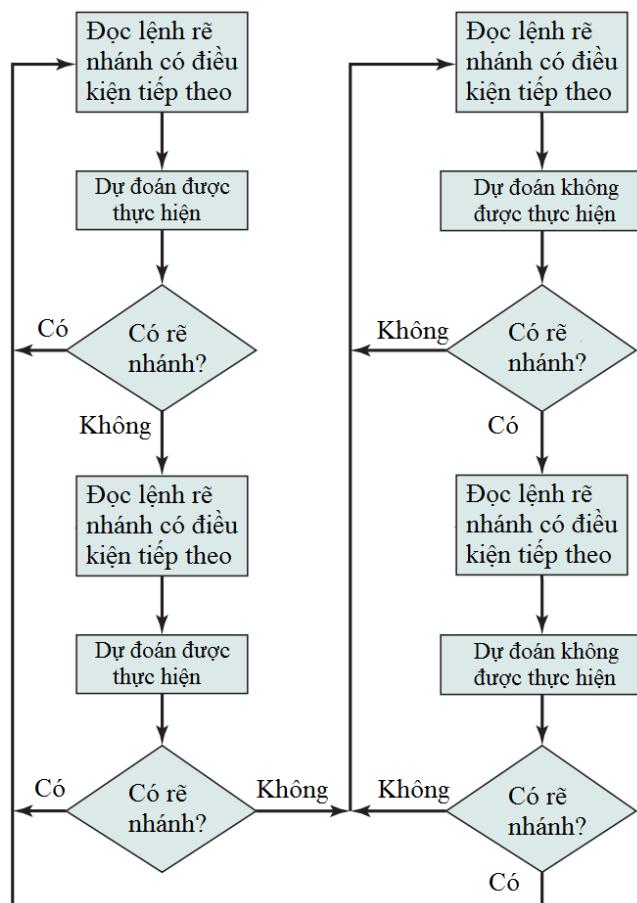
Các phương pháp rẽ nhánh động cố gắng nâng cao độ chính xác của dự đoán bằng cách ghi lại lịch sử các lệnh rẽ nhánh có điều kiện trong một chương trình. Chẳng hạn, một hoặc nhiều bit được gắn với một lệnh rẽ nhánh có điều kiện và có thể phản ánh lịch sử gần đây của lệnh đó. Các bit này được gọi là công tắc được thực hiện/không được thực hiện, chúng định hướng cho bộ xử lý đưa ra một quyết định cụ thể vào lần tiếp theo gấp lại lệnh. Thông thường, các bit lịch sử này không được gắn với lệnh trong bộ nhớ chính. Thay vào đó, chúng được giữ trong bộ nhớ tạm thời. Một khả năng có thể là gắn các bit này với mọi

lệnh rẽ nhánh có điều kiện nằm trong cache. Khi một lệnh này được thay thế khỏi cache, lịch sử của nó mất đi. Một khả năng khác là duy trì một bảng nhỏ chứa các lệnh rẽ nhánh được thi hành gần đây với mỗi lệnh nhập vào có một hoặc nhiều bit lịch sử. Bộ xử lý có thể truy cập vào bảng một cách kết hợp, giống như bộ nhớ cache, hoặc bằng cách sử dụng các bit trọng số nhỏ trong địa chỉ lệnh của rẽ nhánh.

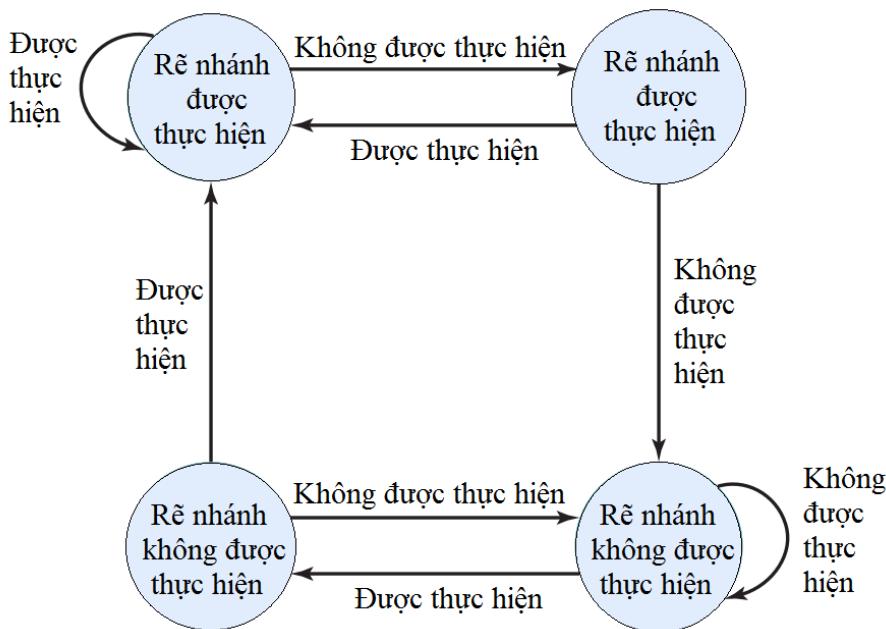
Nếu sử dụng một bit đơn, thông tin duy nhất có thể ghi lại được là liệu lần thi hành gần nhất của lệnh này có rẽ nhánh hay không. Việc sử dụng một bit đơn xuất hiện như một điểm trong trường hợp một lệnh rẽ nhánh có điều kiện gần như luôn luôn được thực hiện, như một lệnh lặp chẳng hạn. Nếu chỉ dùng một bit lịch sử, mỗi lần sử dụng vòng lặp sẽ xảy ra hai lần lỗi dự đoán: một lần khi bắt đầu vào vòng lặp, và một khi ra khỏi vòng lặp.

Nếu sử dụng hai bit, hai bit này có thể được dùng để ghi lại kết quả của hai lần thực thi lệnh rẽ nhánh gần nhất, hoặc để ghi lại một trạng thái nhất định. Hình 12.17 minh họa cho một phương pháp điển hình. Giả sử rằng thuật toán bắt đầu ở góc phía trên bên trái của lưu đồ. Chứng nào mà một lệnh rẽ nhánh có điều kiện tiếp theo gấp phải có thực hiện rẽ nhánh, quá trình ra quyết định sẽ dự đoán rằng lệnh tiếp theo sẽ lại thực hiện rẽ nhánh.

Nếu một lần dự đoán dự đoán sai, thuật toán tiếp tục dự đoán rằng rẽ nhánh tiếp theo được thực hiện. Chỉ khi hai rẽ nhánh liên tiếp không được thực hiện thì thuật toán sẽ chuyển sang phía tay phải của lưu đồ. Sau đó, thuật toán sẽ dự đoán rằng các nhánh không được thực hiện cho đến khi hai nhánh liên tiếp được thực hiện. Do đó, thuật toán này yêu cầu hai lần dự đoán sai liên tiếp để thay đổi quyết định dự đoán. Quá trình ra quyết định có thể được mô tả ngắn gọn hơn như trong hình 12.18.



Hình 12.17 Lưu đồ dự đoán nhánh



Hình 12.18 Lưu đồ trạng thái dự đoán nhánh

Việc sử dụng các bit lịch sử, như vừa mô tả, có một nhược điểm: Nếu quyết định được đưa ra là sẽ thực hiện rẽ nhánh thì sẽ không thể truy xuất được lệnh mục tiêu chừng nào còn chưa giải mã được địa chỉ của lệnh mục tiêu (chính là một toán hạng trong lệnh rẽ nhánh có điều kiện). Rõ ràng là, hiệu quả sẽ cao hơn nếu việc truy xuất lệnh có thể được bắt đầu ngay sau khi có quyết định rẽ nhánh. Để làm được như vậy, cần phải lưu lại nhiều thông tin hơn trong một bộ đệm mục tiêu rẽ nhánh hoặc bảng lịch sử rẽ nhánh.

Bảng lịch sử rẽ nhánh là một bộ nhớ cache nhỏ gắn với giai đoạn truy xuất lệnh trong pipeline. Mỗi mục trong bảng gồm có ba phần: địa chỉ của một lệnh rẽ nhánh, một vài bit lịch sử ghi lại trạng thái sử dụng lệnh đó, và thông tin về lệnh mục tiêu. Trường thứ ba thường chứa địa chỉ của lệnh mục tiêu. Trong một số đề xuất, trường thứ ba này có thể chứa luôn lệnh mục tiêu. Quan hệ ràng buộc trong vấn đề này rất rõ ràng: Lưu trữ địa chỉ mục tiêu sẽ làm cho kích thước bảng nhỏ hơn nhưng thời gian truy xuất lệnh lâu hơn so với lưu trữ lệnh mục tiêu.

12.4.3.5. Nhánh trẽ

Có thể cải thiện hiệu suất pipeline bằng cách tự động sắp xếp lại các lệnh trong một chương trình, sao cho lệnh rẽ nhánh xuất hiện muộn hơn.

12.4.4. Pipeline trong Intel 80486

Xét một ví dụ minh họa cho pipeline lệnh của Intel 80486. Bộ xử lý 80486 thực hiện một pipeline 5 giai đoạn:

- Truy xuất:** Các lệnh được truy xuất từ bộ nhớ cache hoặc từ bộ nhớ ngoài và được đặt vào một trong hai bộ đệm truy xuất trước 16 byte. Mục tiêu của giai đoạn truy xuất là điền dữ liệu mới vào các bộ đệm truy xuất trước ngay khi dữ liệu cũ được bộ giải mã lệnh sử dụng. Bởi vì các lệnh có chiều dài thay đổi (từ 1 đến 11 byte)

không tính tiền tố), trạng thái của bộ truy xuất trước tương ứng với các giai đoạn pipeline khác cũng thay đổi theo từng lệnh. Giai đoạn truy xuất hoạt động độc lập với các giai đoạn khác để giữ cho bộ đệm truy xuất luôn đầy.

- **Giai đoạn giải mã 1 (D1):** Mọi thông tin về opcode và chế độ địa chỉ được giải mã trong giai đoạn D1. Các thông tin cần thiết cùng với thông tin về độ dài của lệnh được đặt trong nhiều nhất là 3 byte đầu tiên của lệnh. Do đó, 3 byte này được chuyển tới giai đoạn D1 từ bộ đệm truy xuất trước. Giai đoạn D1 có thể nối thẳng tới giai đoạn D2 để lấy nốt phần còn lại của lệnh (dữ liệu đích và dữ liệu tức thời) mà không liên quan đến giải mã D1.
- **Giai đoạn giải mã 2 (D2):** Giai đoạn D2 diễn giải mỗi mã opcode thành các tín hiệu điều khiển cụ thể cho ALU. Nó cũng điều khiển việc tính toán các chế độ địa chỉ phức tạp hơn.
- **Thi hành:** Giai đoạn này bao gồm các thao tác ALU, truy cập bộ nhớ cache và cập nhật thanh ghi.
- **Write back:** Nếu cần thiết, giai đoạn này cập nhật các thanh ghi và cờ trạng thái đã bị thay đổi trong giai đoạn thi hành phía trước. Nếu lệnh hiện tại cập nhật bộ nhớ, giá trị tính toán được sẽ cùng một lúc được gửi đến cache và các bộ đệm ghi giao diện bus.

Nhờ việc sử dụng hai giai đoạn giải mã, pipeline có thể duy trì thông lượng ở mức gần một lệnh trong một chu kỳ đồng hồ. Lệnh phức tạp và lệnh rẽ nhánh có điều kiện có thể làm chậm tốc độ này.

Hình 12.19 đưa ra các ví dụ về hoạt động của pipeline. Hình 12.19a cho thấy rằng việc tải vào pipeline sẽ không có độ trễ khi một truy cập bộ nhớ được yêu cầu. Tuy nhiên, như trong hình 12.19b, có thể có độ trễ đối với các giá trị được sử dụng để tính địa chỉ bộ nhớ. Nghĩa là, nếu một giá trị được nạp từ bộ nhớ vào thanh ghi và thanh ghi đó sau đó được sử dụng như một thanh ghi cơ sở trong lệnh tiếp theo, bộ xử lý sẽ ngừng lại trong một chu kỳ. Trong ví dụ này, bộ xử lý truy cập bộ nhớ cache trong giai đoạn EX của lệnh đầu tiên và lưu giá trị đã lấy được vào thanh ghi trong giai đoạn WB. Tuy nhiên, lệnh tiếp theo cần sử dụng thanh ghi này trong giai đoạn D2 của nó. Khi giai đoạn D2 giống thẳng với giai đoạn WB của lệnh trước, các đường dẫn vòng cho phép giai đoạn D2 truy cập vào cùng một dữ liệu mà giai đoạn WB đang sử dụng để ghi, lưu một giai đoạn trong pipeline.

Hình 12.19c mô tả thời gian của lệnh rẽ nhánh, giả sử rằng nhánh được thực hiện. Lệnh so sánh cập nhật mã điều kiện trong giai đoạn WB, và cùng thời điểm ấy, giai đoạn EX của lệnh nhảy Jcc cần dùng mã điều kiện này. Đường dẫn vòng giúp cho việc hai việc này có thể diễn ra cùng lúc. Đồng thời, bộ xử lý chạy một chu kỳ truy xuất trước tới mục tiêu của lệnh nhảy trong giai đoạn EX của lệnh nhảy. Nếu bộ xử lý xác định rằng điều kiện rẽ nhánh là sai, nó sẽ loại bỏ phần truy xuất trước này và tiếp tục thi hành lệnh tuần tự tiếp theo (đã được truy xuất và giải mã).

Truy xuất	D1	D2	EX	WB		MOV Reg1, Mem1
Truy xuất	D1	D2	EX	WB		MOV Reg1, Reg2
Truy xuất	D1	D2	EX	WB		MOV Mem2, Reg1

(a) Không có trễ tải dữ liệu trong pipeline

Truy xuất	D1	D2	EX	WB		MOV Reg1, Mem1
Truy xuất	D1		D2	EX		MOV Reg2, (Reg1)

(b) Có trễ tải con trỏ

Truy xuất	D1	D2	EX	WB		CMP Reg1, Imm
Truy xuất	D1	D2	EX			Jcc Target
			Truy xuất	D1	D2	EX

(c) Lệnh rẽ nhánh

Hình 12.19 Ví dụ pipeline lệnh 80486

12.5. CÂU HỎI

- Chức năng của thanh ghi trong bộ xử lý là gì?
- Thanh ghi hiển thị với người dùng thường hỗ trợ những loại dữ liệu nào?
- Mã điều kiện là gì?
- Từ trạng thái chương trình PSW là gì?
- Tại sao một pipeline lệnh hai giai đoạn không rút ngắn một nửa thời gian chu kỳ lệnh so với khi không sử dụng pipeline?
- Liệt kê và giải thích ngắn gọn các giải pháp để pipeline lệnh giải quyết các vấn đề xảy ra với lệnh rẽ nhánh có điều kiện?
 - Carry
 - Zero
 - Overflow
 - Sign
- Dự báo nhánh sử dụng các bit lịch sử như thế nào?
 - Giả sử một máy tính với độ dài word là 8 bit. Nếu phép tính cuối cùng được thực hiện trên máy này một phép cộng, trong đó hai toán hạng là 00000010 và 00000011. Hãy xác định giá trị của các cờ sau?
 - Carry
 - Zero
 - Overflow
 - Sign
 - Lặp lại câu hỏi trên cho phép cộng -1 (bù hai) và +1?

9. Lắp lại ván đè ở Câu 8 đối với phép tính A – B, trong đó A là 11110000, B là 0010100.