

TRƯỜNG ĐẠI HỌC AN GIANG
KHOA KỸ THUẬT – CÔNG NGHỆ – MÔI TRƯỜNG



Tài liệu giảng dạy
KIẾN TRÚC MÁY TÍNH

ThS. Nguyễn Văn Đông

An Giang, 02/2016

Tài liệu giảng dạy “Kiến trúc máy tính”, do tác giả Nguyễn Văn Đông, công tác tại Khoa Kỹ thuật – Công nghệ – Môi trường thực hiện. Tác giả đã báo cáo nội dung và được Hội đồng Khoa học và đào tạo Khoa thông qua ngày 26/2/2016, và được Hội đồng Khoa học và Đào tạo của trường Đại học An Giang thông qua ngày.

Tác giả biên soạn

Nguyễn Văn Đông

Trưởng đơn vị

Trưởng bộ môn

Hiệu trưởng

An Giang, 02-2016

LỜI MỞ ĐẦU

Trong hệ thống kiến thức trang bị cho sinh viên chuyên ngành công nghệ thông tin và kỹ thuật phần mềm, tài liệu Kiến trúc máy tính góp phần cung cấp những nội dung chung nhất về nguyên lý hoạt động cũng như tổ chức một hệ thống máy tính. Từ đó có thể tiếp cận các mô hình kiến trúc hiện đại đang được phát triển hiện nay.

Để phục vụ công tác giảng dạy và học tập, chúng tôi biên soạn tài liệu Kiến trúc máy tính nhằm cung cấp tới người học các kiến thức cơ bản nhất về lĩnh vực này.

Chúng tôi xin chân thành cảm ơn các thầy cô Bộ môn Công nghệ thông tin và Bộ môn Kỹ thuật phần mềm đã cho những ý kiến đóng góp quý báu để tài liệu giảng dạy được hoàn thiện hơn. Tài liệu giảng dạy này được biên soạn chia thành 5 chương. Tuy rằng chúng tôi đã có nhiều cố gắng trong biên soạn nhưng chắc chắn tài liệu vẫn còn nhiều thiếu sót, nên rất mong được bạn đọc cũng như các đồng nghiệp đóng góp ý kiến để tài liệu ngày càng hoàn thiện, nhằm mục đích phục vụ tốt hơn cho việc dạy và học tin học đang ngày càng phát triển ở nước ta.

Mọi sự góp ý hoặc thắc mắc xin gửi về địa chỉ email: nvdong@agu.edu.vn

Ngày 15 tháng 02 năm 2016

GV. biên soạn

Nguyễn Văn Đông

LỜI CAM KẾT

Tôi xin cam đoan đây là tài liệu giảng dạy của riêng tôi. Nội dung tài liệu giảng dạy có xuất xứ rõ ràng.

An Giang, ngày 15 tháng 02 năm 2016

Người biên soạn

Nguyễn Văn Đông

MỤC LỤC

CHƯƠNG 1: ĐẠI CƯƠNG	1
1.1 TỔNG QUAN	1
1.2 HIỆU SUẤT	2
1.2.1 Dẫn nhập	2
1.2.2 Đo đặc hiệu suất	3
1.3 BIỂU DIỄN HỆ SỐ	7
1.4 BIỂU DIỄN SỐ NGUYÊN	9
1.4.1 Biểu diễn số nguyên không dấu	9
1.4.2 Biểu diễn số nguyên có dấu	9
1.5 CÁC PHÉP TÍNH TRÊN SỐ NGUYÊN	10
1.6 BIỂU DIỄN SỐ VỚI DẤU CHẤM ĐỘNG	14
1.7 BIỂU DIỄN KÍ TỰ	15
CHƯƠNG 2: TẬP LỆNH VI XỬ LÝ	19
2.1 GIỚI THIỆU	19
2.2 TOÁN TỬ	19
2.3 TOÁN HẠNG	21
2.3.1 Toán hạng thanh ghi	21
2.3.2 Toán hạng bộ nhớ	21
2.3.3 Toán hạng trực tiếp	22
2.3.4 Định dạng lệnh	23
2.3.5 Lệnh luận lý	26
2.3.6 Các lệnh rẽ nhánh	27
2.4 GIẢI MÃ NGÔN NGỮ MÁY	28
CHƯƠNG 3: TỔ CHỨC BỘ XỬ LÝ	31
3.1 GIỚI THIỆU	31
3.2 ĐƯỜNG DẪN DỮ LIỆU	31
3.3 TỔ CHỨC BỘ TÍNH TOÁN VÀ LUẬN LÝ	33
3.4 BỘ ĐIỀU KHIỂN CHÍNH	34
3.5 KỸ THUẬT ỚNG DẪN	41
3.5.1 Tổng quan	41

3.5.2 Đường đi dữ liệu của kỹ thuật ống dẫn	43
3.5.3 Điều khiển trong kỹ thuật ống dẫn	54
CHƯƠNG 4: BỘ NHỚ	64
4.1 GIỚI THIỆU	64
4.2 BỘ NHỚ CACHE.....	65
4.2.1 Tổng quan.....	65
4.2.2 Truy cập bộ nhớ Cache.....	67
4.2.3 Xử lý thất bại cache	68
4.2.4 Xử lý ghi.....	69
4.2.5 Thiết kế bộ nhớ hỗ trợ cache	69
4.3 ĐO LƯỜNG VÀ CẢI TIẾN HIỆU SUẤT CACHE	71
4.3.1 Thay thế khối.....	73
4.3.2 Xác định khối trong cache.....	76
4.3.3 Thay thế khối.....	77
4.3.4 Cache nhiều mức	78
CHƯƠNG 5: HỆ THỐNG LƯU TRỮ VÀ NHẬP - XUẤT	81
5.1 GIỚI THIỆU	81
5.2 Đĩa từ.....	81
5.3 BỘ NHỚ FLASH.....	82
5.4 KẾT NỐI GIỮA BỘ XỬ LÝ, BỘ NHỚ VÀ THIẾT BỊ NHẬP/XUẤT.....	83
5.5 GIAO TIẾP THIẾT BỊ NHẬP/XUẤT VỚI BỘ XỬ LÝ, BỘ NHỚ VÀ HỆ ĐIỀU HÀNH	84
5.5.1 Ra lệnh cho thiết bị nhập xuất	85
5.5.2 Giao tiếp với bộ xử lý.....	85
5.5.3 Độ ưu tiên ngắt	86
5.5.4 Truyền dữ liệu giữa thiết bị và bộ nhớ	87
5.5.5 Truy cập bộ nhớ trực tiếp DMA và hệ thống bộ nhớ	88

DANH SÁCH HÌNH

Hình 1.1: Giải thuật Booth.....	12
Hình 1.2: Giải thuật thực hiện phép chia	13
Hình 2.1: Địa chỉ ô nhớ và dữ liệu tương ứng	21
Hình 3.1: Sơ đồ tổ chức tổng quát của MIPS.....	31
Hình 3.2: Đường dẫn dữ liệu của ba loại lệnh cơ bản.....	33
Hình 3.3: Ba định dạng lệnh R-type, load và store, branch	35
Hình 3.4: Các tín hiệu điều khiển trong hệ thống	36
Hình 3.5: Các tín hiệu điều khiển của các dạng lệnh	37
Hình 3.6: Đường đi dữ liệu của dạng lệnh R-format	38
Hình 3.7: Đường đi dữ liệu của lệnh load.....	39
Hình 3.8: Đường đi dữ liệu của lệnh beq	40
Hình 3.9: đường đi dữ liệu của lệnh jump	41
Hình 3.10: Thực hiện tuần tự so với kỹ thuật ống dẫn.....	42
Hình 3.11: Các giai đoạn thực thi một lệnh	43
Hình 3.12: Ba lệnh được thực thi trong kỹ thuật ống dẫn.....	44
Hình 3.13: Các thanh ghi ống dẫn.....	45
Hình 3.14: Giai đoạn duyệt lệnh của lệnh lw	46
Hình 3.15: Giai đoạn giải mã lệnh của lệnh lw	47
Hình 3.16: Giai đoạn thực thi của lệnh lw	48
Hình 3.17: Giai đoạn truy cập bộ nhớ của lệnh lw	49
Hình 3.18: Giai đoạn ghi dữ liệu vào thanh ghi	50
Hình 3.19: Kết hợp các giai đoạn của lệnh lw	51
Hình 3.20: Giản đồ ống dẫn biểu diễn bằng đa chu kỳ	52
Hình 3.21: Giản đồ ống dẫn đơn chu kỳ tương ứng với chu kỳ 5 của hình 3.20	53
Hình 3.22: Các tín hiệu điều khiển trong kỹ thuật ống dẫn	54
Hình 3.23: Các tín hiệu điều khiển của ba giai đoạn ống dẫn sau cùng.....	57
Hình 3.24: Các tín hiệu điều khiển được nối vào các phần tương ứng.....	57
Hình 4.1: Cấu trúc của cấp bậc bộ nhớ	64
Hình 4.2: Bộ nhớ Cache trước và sau khi CPU tham khảo từ X_n	65
Hình 4.3: Ánh xạ trực tiếp từ bộ nhớ sang cache có 8 khối	66
Hình 4.4: Bộ nhớ cache với lần lượt 9 địa chỉ được truy cập	68
Hình 4.5: Ba cách tổ chức bộ nhớ khác nhau.....	70
Hình 4.6: Khối 12 được đặt vào cache tương ứng với ba mô hình	73
Hình 4.7: Cache 8 khối được chia thành tập hợp có 1,2,4,8 phần tử	74
Hình 4.8: Mô hình kết hợp theo tập hợp với tập hợp có bốn phần tử.....	77
Hình 5.1: Hệ thống nhập/xuất điển hình	81
Hình 5.2: Thanh ghi trạng thái và thanh ghi nguyên nhân.....	87

DANH SÁCH BẢNG

Bảng 1.1: Số nhị phân, thập phân và thập lục phân	8
Bảng 1.2: Phép cộng 2 số được biểu diễn bằng phương pháp bù 2	11
Bảng 1.3: Phép trừ 2 số (M-S) được biểu diễn bằng bù 2.....	11
Bảng 2.1: Các thanh ghi và địa chỉ bộ nhớ trong MIPS.....	20
Bảng 2.2: Tập lệnh vi xử lý MIPS	20
Bảng 2.3: Định dạng lệnh MIPS	26
Bảng 2.4: mã lệnh <i>op</i> của các lệnh MIPS	28
Bảng 2.5: các giá trị của trường funct với R-format	29
Bảng 3.1: Thiết lập các bit điều khiển ALU dựa vào ALUOp và trường funct.....	34
Bảng 3.2: Tổng thời gian thực hiện của mỗi lệnh.....	42
Bảng 3.3: Thiết lập các bit ALU control.....	55
Bảng 3.4: Các tín hiệu điều khiển tương ứng theo 3 giai đoạn sau cùng.....	56
Bảng 4.1: So sánh giữa ba kỹ thuật trong cấp bậc bộ nhớ	65
Bảng 4.2: CPU lần lượt truy cập 9 địa chỉ bộ nhớ	67

DANH MỤC TỪ VIẾT TẮT

Từ viết tắt	Tiếng Anh	Tiếng Việt
ALU	Algorithm logic unit	Đơn vị luận lý và số học
AMAT	Average memory access time	Thời gian truy cập bộ nhớ trung bình
CISC	Complex Instruction	Tập lệnh máy tính phức tạp
CPI	Clock cycle per instruction	Số chu kỳ xung nhịp thực thi lệnh
CPU	Central processing unit	Bộ xử lý
CU	Control unit	Đơn vị điều khiển
DM	Data memory	Vùng nhớ dữ liệu
DMA	Direct memory access	Truy cập bộ nhớ trực tiếp
DRAM	Dynamic random access memory	Bộ nhớ truy cập ngẫu nhiên động
EX	Execution	Thực thi
ID	Instruction decode	Giải mã lệnh
IM	Instruction memory	Vùng nhớ lệnh
IF	Instruction fetch	Duyệt lệnh
I/O	Input/Output	Thiết bị nhập/xuất
LRU	Least recently used	Ít được sử dụng gần đây nhất
MEM	Memory	Truy cập bộ nhớ
MIPS	Million instructions per second	Triệu lệnh mỗi giây
MIPS	Microprocessor without Interlocked Pipeline Stages	Bộ xử lý không đồng bộ kỹ thuật ống dẫn
Mux	Multiplexor	Bộ điều hợp
PC	Personal computer	Máy tính cá nhân
Reg	Register	Thanh ghi
RISC	Reduced Instruction Set Computer	Tập lệnh máy tính rút gọn
SRAM	Static random access memory	Bộ nhớ truy cập ngẫu nhiên tĩnh
V	Valid bit	Bit hợp lý
WB	Write-back	Ghi kết quả trở lại

NỘI DUNG

➤ **CHƯƠNG 1: ĐẠI CƯƠNG**

Giới thiệu về lịch sử máy tính, cách phân loại máy tính, hiệu suất máy tính. Biểu diễn số nguyên và số với dấu chấm động.

➤ **CHƯƠNG 2: TẬP LỆNH VI XỬ LÝ**

Giới thiệu về tập lệnh vi xử lý MIPS: các loại toán hạng trong lệnh, định dạng lệnh, các dạng lệnh khác nhau và cách giải mã ngôn ngữ máy.

➤ **CHƯƠNG 3: TỔ CHỨC BỘ XỬ LÝ**

Giới thiệu khái niệm đường dẫn dữ liệu, cách tổ chức bộ tính toán và luận lý, hoạt động của bộ điều khiển chính. Nguyên lý hoạt động của kỹ thuật ống dẫn.

➤ **CHƯƠNG 4: BỘ NHỚ**

Trình bày tổng quan về nguyên lý hoạt động của bộ nhớ cache: truy cập bộ nhớ, xử lý thất bại, xử lý ghi. Cách thức đo lường và phương pháp cải tiến hiệu suất bộ nhớ cache.

➤ **CHƯƠNG 5: HỆ THỐNG LƯU TRỮ VÀ NHẬP – XUẤT**

Giới thiệu về hệ thống lưu trữ và nhập xuất. Cấu tạo và nguyên lý hoạt động của đĩa từ, bộ nhớ flash. Kết nối giữa bộ xử lý, bộ nhớ và thiết bị nhập – xuất. Giao tiếp thiết bị nhập – xuất với bộ xử lý, bộ nhớ và hệ điều hành.

KẾ HOẠCH GIẢNG DẠY

(lý thuyết: 30 tiết; thực hành: 0 tiết)

Nội dung		Số tiết
Chương 1	1.1 Tổng quan	2
	1.2 Hiệu suất	
	1.3 Biểu diễn hệ số	
	1.4 Biểu diễn số nguyên	2
	1.5 Các phép tính trên số nguyên	
	1.6 Biểu diễn số với dấu chấm động	2
	1.7 Biểu diễn kí tự	
Chương 2	2.1 Giới thiệu	2
	2.2 Toán tử	
	2.3 Toán hạng	2
	2.4 Giải mã ngôn ngữ máy	2
Chương 3	3.1 Giới thiệu	2
	3.2 Đường dẫn dữ liệu	
	3.3 Tổ chức bộ tính toán và luận lý	2
	3.4 Bộ điều khiển chính	
	3.5 Kỹ thuật ống dẫn	2
Chương 4	4.1 Giới thiệu	2
	4.2 Bộ nhớ cache	
	4.3 Đo lường và cải tiến hiệu suất cache	4
Chương 5	5.1 Giới thiệu	2
	5.2 Đĩa từ	
	5.3 Bộ nhớ flash	
	5.4 Kết nối giữa bộ xử lý, bộ nhớ và thiết bị nhập – xuất	2
	5.5 Giao tiếp thiết bị nhập – xuất với bộ xử lý, bộ nhớ và hệ điều hành	
Ôn tập và kiểm tra		2
Tổng		30

CHƯƠNG 1

ĐẠI CƯƠNG

Mục đích: Giới thiệu tổng quan về lịch sử phát triển của máy tính, cách phân loại máy tính hiện nay. Cách để đánh giá hiệu suất của máy tính. Sau đó trình bày biểu diễn số nguyên không dấu và có dấu, các phép tính cơ bản trên số nguyên này. Trình bày cách biểu diễn số với dấu chấm động, biểu diễn kí tự bằng bảng mã ASCII, bảng mã UNICODE.

1.1 TỔNG QUAN

Sự ra đời của máy vi tính tạo cuộc cách mạng vĩ đại trong lịch sử văn minh nhân loại. Hệ thống máy tính ngày nay có thể giúp chúng ta có những bước phát triển mạnh mẽ, nâng cao năng suất lao động, tri thức cũng như khoa học công nghệ. Một cách tổng thể, lịch sử phát triển của máy tính trải qua các giai đoạn sau:

➤ Thế hệ máy tính thứ nhất (1946 – 1957): thế hệ này được đặc trưng bằng kỹ thuật ống chân không (*vacuum tube*). Chiếc máy vi tính đầu tiên **ENIAC** (Electronic Numerical Integrator And Computer) do Giáo sư Mauchly và học trò của ông Eckert tại đại học pennsylvania bắt đầu thiết kế vào năm 1943 và được hoàn thành vào năm 1946. Đây là một máy vi tính khổng lồ với chiều dài 20 mét, cao 2,8 mét, năng lực tính toán là 5000 phép cộng trong một giây.

Giáo sư toán học John Von Neumann lần đầu đưa ra khái niệm chương trình được lưu trữ (*stored-program concept*) vào năm 1945. Đến năm 1946, ông cùng các đồng nghiệp thiết kế máy tính IAS tại Princeton Institute for Advanced Studies. Máy tính này gồm các thành phần: bộ nhớ dùng để lưu trữ chung cho cả dữ liệu và lệnh, bộ số học – luận lý (ALU) thực hiện phép toán trên dữ liệu nhị phân, bộ điều khiển được dùng để điều khiển thực thi lệnh trong bộ nhớ và hoạt động của hệ thống nhập/xuất. Đây là một ý tưởng nền tảng cho các máy tính hiện đại ngày nay. Máy tính này còn được gọi là máy tính **Von Neumann**.

➤ Thế hệ thứ hai (1958-1964): thế hệ này bắt đầu bằng sự thay thế ống chân không bằng transistor với ưu điểm là nhỏ hơn, giá thành rẻ hơn, tỏa nhiệt ít hơn. Transistor được phát minh bởi công ty Bell Labs vào năm 1947 nhưng đến cuối thập niên 50, máy tính thương mại dùng transistor mới xuất hiện trên thị trường. Ngoài ra, ở thế hệ này các thành phần quan trọng trong máy tính như: bộ số học – luận lý, bộ điều khiển cũng được thiết kế phức tạp hơn. Một số ngôn ngữ cấp cao được dùng để lập trình như: FORTRAN năm 1956, COBOL năm 1959, ALGOL năm 1960) và phần mềm hệ thống (*system software*) trong máy tính cũng được giới thiệu.

➤ Thế hệ thứ ba (1965-1971): Thế hệ này bắt đầu bằng sự xuất hiện của mạch tích hợp (*integrated circuit*). Vi mạch này cho phép đặt một số transistor trên một chip đơn. Các mạch tích hợp với mật độ thấp (SSI: *Small Scale Integration*) có thể chứa vài chục linh kiện và mạch tích hợp với mật độ trung bình (MSI: *Medium Scale Integration*) chứa hàng trăm linh kiện. Do những cải tiến lớn về công nghệ nên các

máy tính của thế hệ này trở nên nhỏ hơn, nhanh hơn và rẻ hơn các máy tính thế hệ trước đó. Hệ điều hành và khả năng đa lập trình cũng đã xuất hiện.

➤ Thế hệ thứ tư (1972 – ngày nay): thế hệ này đánh dấu bằng tiến bộ của kỹ thuật mạch tích hợp. Với sự giới thiệu của mạch tích hợp mật độ cao (LSI: *Large Scale Integration*) chứa hàng ngàn linh kiện được đặt trên một chip đơn, mạch tích hợp mật độ rất cao (VLSI: *Very Large Scale Integration*) có thể đạt hơn 10 ngàn linh kiện trên một chip và mạch tích hợp với mật độ siêu lớn (ULSI: *Ultra Large Scale Integration*) có thể chứa hơn một triệu linh kiện. Đặc biệt thế hệ này có sự xuất hiện của bộ nhớ bán dẫn (*semiconductor memory*) và bộ vi xử lý (*microprocessor*). Máy tính cá nhân PC (*Personal Computer*) đầu tiên đã ra đời như Kenback-1. Các kỹ thuật cải tiến tốc độ xử lý của máy tính không ngừng được phát triển: kỹ thuật ống dẫn, kỹ thuật vô hướng, xử lý song song ...

Ngày nay kỹ thuật máy tính được sử dụng trong nhiều lĩnh vực khác nhau từ các thiết bị nhúng trong ngôi nhà thông minh, điện thoại đến những các siêu máy tính (*supercomputer*). Trong các thiết bị này, nền tảng phần cứng tuy có khác nhau nhưng một cách tổng quan có thể phân loại máy tính thành các loại như sau:

✚ Máy tính cá nhân (*Personal computer - PC*): đây là loại được sử dụng phổ biến nhất. Máy tính cá nhân được thiết kế để sử dụng cho mục đích cá nhân có hiệu suất tương đối tốt với chi phí thấp.

✚ Máy chủ (*Server*): đây là loại máy tính hiện đại bao gồm các loại máy tính lớn (*mainframes, minicomputers* và *supercomputers*) và chỉ được truy cập thông qua mạng. Server thực hiện các luồng công việc lớn như: các ứng dụng tính toán khoa học và kỹ thuật hoặc xử lý rất nhiều công việc như một Web server lớn. Mặc dù không được gọi là supercomputer, nhưng các trung tâm dữ liệu (*datacenter*) trên internet được dùng bởi các công ty lớn như eBay hay Google chứa hàng ngàn bộ vi xử lý với bộ nhớ chính có dung lượng terabyte (1TB = 1024 GB) và hệ thống lưu trữ lên đến petabyte (1PB = 1024 TB). Những hệ thống này bao gồm nhiều cụm (*cluster*) máy tính lớn.

✚ Máy tính nhúng (*Embedded computer*): được ứng dụng rộng lớn nhất trong hầu hết các lĩnh vực. Loại này bao gồm các vi xử lý gắn trong xe hơi, hệ thống xử lý trong điện thoại, các hệ thống xử lý trong video game hay tivi và hệ thống vi xử lý trong máy bay hay tàu thủy hiện đại. Hệ thống nhúng này được thiết kế để chạy một ứng dụng hoặc một tập các ứng dụng có liên quan, được tích hợp với phần cứng và được xem như một hệ thống đơn.

1.2 HIỆU SUẤT (*performance*)

1.2.1 Dẫn nhập

Làm sao để có thể đo đạc, đánh giá hiệu suất (*performance*) và định ra được những yếu tố quyết định đến hiệu suất của 1 máy tính ? Lý do chính để khảo sát về hiệu suất là vì hiệu suất của phần cứng máy tính thường là yếu tố mấu chốt quyết định đến tính hiệu quả trong hoạt động của 1 một hệ thống bao gồm cả phần cứng lẫn phần mềm. Hiệu suất luôn là một thuộc tính quan trọng trong việc lựa chọn, mua bán

các máy tính được cả người bán lẫn người mua quan tâm! Hiệu suất càng được các nhà thiết kế máy tính (trong đó có chúng ta) quan tâm.

Việc đánh giá hiệu suất máy tính không hề đơn giản. Hiệu suất không chỉ có được do các cải tiến phần cứng mà cũng có thể nhờ vào các phần mềm thông minh hay cả hai. Tùy góc độ ứng dụng khác nhau, hiệu suất hoàn toàn có thể được đánh giá theo những phương cách, những chỉ số khác nhau. Ở góc độ nhà thiết kế máy tính (phần cứng/phần mềm), chúng ta cần nắm rõ: Các vấn đề liên quan đến việc đánh giá hiệu suất máy tính, hoạt động của các thành phần khác nhau (phần cứng/phần mềm) và ảnh hưởng của chúng đến hiệu suất. Trong mỗi ứng dụng cụ thể, xác định phương pháp đánh giá hiệu suất phù hợp.

Định nghĩa hiệu suất: trước hết chúng ta xem xét hai khái niệm liên quan

- Thời gian đáp ứng (*response time*) hay thời gian thực thi (*execution time*), là thời gian từ khi bắt đầu đến khi kết thúc chương trình. Thời gian này bao gồm: thời gian truy cập đĩa, thời gian truy cập bộ nhớ, thời gian thực thi CPU,...
- Throughput: là tổng số các chương trình thực thi xong trong một đơn vị thời gian.

Trước tiên, chúng ta đánh giá hiệu suất thông qua thời gian thực thi. Cực đại hóa hiệu suất đồng nghĩa với tối thiểu hóa thời gian thực thi. Quan hệ giữa hiệu suất và thời gian thực thi ở máy tính X sẽ là:

$$\text{hiệu suất}_X = \frac{1}{\text{thời gian thực thi}_X}$$

Ta nói máy tính X có hiệu suất cao hơn máy tính Y n lần đồng nghĩa với máy tính X nhanh hơn máy tính Y n lần.

$$\frac{\text{hiệu suất}_X}{\text{hiệu suất}_Y} = n$$

Thí dụ: nếu máy tính A thực thi chương trình mất 10s và máy tính B thực thi cùng chương trình mất 15s, A nhanh hơn B bao nhiêu lần?

Ta biết rằng A nhanh hơn B n lần nếu:

$$\frac{\text{hiệu suất}_A}{\text{hiệu suất}_B} = \frac{\text{thời gian thực thi}_B}{\text{thời gian thực thi}_A} = n$$

Do đó: $\frac{15}{10} = 1.5$ Vậy máy tính A nhanh hơn máy tính B 1.5 lần

1.2.2 Đo đạc hiệu suất

Thời gian được sử dụng làm thước đo cho hiệu suất máy tính. Tuy nhiên thời gian ở đây được định nghĩa theo nhiều cách khác nhau, tùy theo mục đích đo đạc như: thời gian theo xung nhịp (*clock*), thời gian thực thi (*execution time*), thời gian trôi qua (*elapsed time*). Thời gian thực thi chương trình bao gồm thời gian thực thi bởi CPU lẫn các thiết bị khác (bộ nhớ, đĩa cứng, v.v...). Để đơn giản, chúng ta chỉ giới hạn xem xét đối với CPU mà thôi.

Có thể đo đặc hiệu suất qua thời gian thực thi theo chu kỳ xung nhịp (*clock cycle*) và thời gian chu kỳ xung nhịp (*clock cycle time*) như sau:

$$\text{Thời gian thực thi của một chương trình} = \frac{\text{Số chu kỳ xung nhịp của chương trình}}{\text{tần số xung nhịp}} \times \text{Thời gian chu kỳ xung nhịp}$$

hay:

$$\text{Thời gian thực thi của một chương trình} = \frac{\text{Số chu kỳ xung nhịp của chương trình}}{\text{tần số xung nhịp}}$$

Với tần số xung nhịp (*clock*) (xung nhịp (*clock cycle*)).

Thí dụ: thời gian thực thi chương trình trên máy tính A, tần số 4 GHz, là 10s. Chúng ta muốn thiết kế máy tính B có thể thực thi chương trình trên trong 6s. Để đạt được điều này, cần phải tăng tần số xung nhịp của máy B và vì vậy số chu kỳ xung nhịp thực thi chương trình bị tăng lên 1,2 lần. Hãy xác định tần số xung nhịp của máy B?

Đầu tiên, tính số chu kỳ xung nhịp cần để thực hiện chương trình trên máy A, ta có:

$$\begin{aligned} \text{Thời gian thực thi}_A &= \frac{\text{số chu kỳ xung nhịp}_A}{\text{tần số xung nhịp}_A} \\ 10 \text{ giây} &= \frac{\text{số chu kỳ xung nhịp}_A}{4 \times 10^9 \frac{\text{chu kỳ}}{\text{giây}}} \\ \text{số chu kỳ xung nhịp}_A &= 10 \text{ giây} \times 4 \times 10^9 \frac{\text{chu kỳ}}{\text{giây}} = 40 \times 10^9 \text{ chu kỳ} \end{aligned}$$

Thời gian thực thi trên máy B được tính như sau:

$$\begin{aligned} \text{Thời gian thực thi}_B &= \frac{1.2 \times \text{số chu kỳ xung nhịp}_A}{\text{tần số xung nhịp}_B} \\ 6 \text{ giây} &= \frac{1.2 \times 40 \times 10^9 \text{ chu kỳ}}{\text{tần số xung nhịp}_B} \\ \text{tần số xung nhịp}_B &= \frac{1.2 \times 40 \times 10^9 \text{ chu kỳ}}{6 \text{ giây}} = \frac{8 \times 10^9 \text{ chu kỳ}}{\text{giây}} = 8 \text{GHz} \end{aligned}$$

Do đó, cần tăng gấp đôi tần số xung nhịp của máy B so với máy A để chương trình thực thi trong 6s.

Ngoài ra, ta có thể tính thời gian thực thi chương trình dựa vào chỉ số CPI (*clock cycle per instruction*): số chu kỳ xung nhịp trung bình cần thiết để thực thi một câu lệnh. Khi đó số chu kỳ xung nhịp có thể được tính:

$$\text{Số chu kỳ xung nhịp} = (\text{số lệnh của chương trình}) \times \text{CPI}$$

Thí dụ: xét 2 máy tính A và B có cùng kiến trúc tập lệnh. Máy A có chu kỳ xung nhịp là 250 ps và đạt được CPI là 2,0 khi chạy chương trình P. Máy B có chu kỳ xung nhịp là 500 ps và đạt được CPI bằng 1,2 khi chạy chương trình P. Máy nào thực thi chương trình P nhanh hơn và nhanh hơn bao nhiêu lần ?

Do 2 máy có cùng kiến trúc tập lệnh nên số lệnh thực thi của chương trình P trên 2 máy này bằng nhau. Giả sử gọi số lệnh này là I . Ta có:

$$\text{Số chu kỳ xung nhịp}_A = I \times 2.0$$

$$\text{Số chu kỳ xung nhịp}_B = I \times 1.2$$

Do đó, thời gian thực thi trên mỗi máy:

$$\begin{aligned} \text{Thời gian thực thi}_A &= \text{Số chu kỳ xung nhịp}_A \times \text{thời gian chu kỳ xung nhịp}_A \\ &= I \times 2.0 \times 250 \text{ ps} = 500 \times I \text{ ps} \end{aligned}$$

$$\text{Thời gian thực thi}_B = I \times 1.2 \times 500 \text{ ps} = 600 \times I \text{ ps}$$

Như vậy, máy tính A nhanh hơn là:

$$\frac{\text{hiệu suất}_A}{\text{hiệu suất}_B} = \frac{\text{thời gian thực thi}_B}{\text{thời gian thực thi}_A} = \frac{600 \times I \text{ ps}}{500 \times I \text{ ps}} = 1.2$$

Dựa trên chỉ số CPI, ta có công thức tính thời gian thực thi:

$$\text{Thời gian thực thi} = \text{số lệnh} \times \text{CPI} \times \text{thời gian chu kỳ xung nhịp}$$

hay:

$$\text{thời gian thực thi} = \frac{\text{số lệnh} \times \text{CPI}}{\text{tần số xung nhịp}}$$

Trong trường hợp tập lệnh được phân chia thành nhiều nhóm lệnh, khi đó hiệu suất có thể tính toán theo nhóm lệnh như sau:

$$\text{số chu kỳ xung nhịp} = \sum_{i=1}^n (\text{CPI}_i \times C_i)$$

Thí dụ: xét 1 máy tính có đặc điểm tập lệnh như sau:

	CPI của nhóm lệnh		
	A	B	C
CPI	1	2	3

Khi biên dịch cùng 1 chương trình nguồn bằng 2 trình biên dịch khác nhau, ta được 2 đoạn mã lệnh như sau:

Đoạn mã lệnh	Số lệnh của nhóm lệnh		
	A	B	C
1	2	1	2
2	4	1	1

Đoạn mã nào thực hiện nhiều câu lệnh hơn ? chạy nhanh hơn ? Tính CPI cho từng đoạn mã lệnh ?

Đoạn mã 1 thực hiện: $2 + 1 + 2 = 5$ lệnh, trong khi đoạn mã 2 thực hiện là: $4 + 1 + 1 = 6$ lệnh. Như vậy, đoạn mã 2 thực hiện nhiều lệnh hơn.

Để tính số chu kỳ xung nhịp cho mỗi đoạn mã dựa vào công thức:

$$\text{số chu kỳ xung nhịp} = \sum_{i=1}^n (\text{CPI}_i \times C_i)$$

Do đó:

$$\text{số chu kỳ xung nhịp}_1 = (2 \times 1) + (1 \times 2) + (2 \times 3) = 2 + 2 + 6 = 10 \text{ chu kỳ}$$

$$\text{số chu kỳ xung nhịp}_2 = (4 \times 1) + (1 \times 2) + (1 \times 3) = 4 + 2 + 3 = 9 \text{ chu kỳ}$$

Vì thế đoạn mã 2 thực hiện nhanh hơn. CPI của mỗi đoạn được tính như sau:

$$\text{CPI} = \frac{\text{số chu kỳ xung nhịp}}{\text{số lệnh}}$$

$$\text{CPI}_1 = \frac{\text{số chu kỳ xung nhịp}_1}{\text{số lệnh}_1} = \frac{10}{5} = 2$$

$$\text{CPI}_2 = \frac{\text{số chu kỳ xung nhịp}_2}{\text{số lệnh}_2} = \frac{9}{6} = 1.5$$

Một chỉ số khác cũng được dùng để đánh giá hiệu suất, đó là MIPS (*million instructions per second*) được xác định như sau:

$$\text{MIPS} = \frac{\text{số lệnh}}{\text{thời gian thực thi} \times 10^6}$$

Thí dụ: cũng với thí dụ vừa rồi, xét bảng số liệu sau:

Mã lệnh	Số lệnh (đơn vị: tỷ) của mỗi nhóm lệnh		
	A	B	C
Trình biên dịch 1	5	1	1
Trình biên dịch 2	10	1	1

giả sử máy trên có tần số xung nhịp là 4 GHz. Cho biết trình biên dịch nào thực thi nhanh hơn nếu tính theo thời gian thực thi? nếu tính theo MIPS?

Đầu tiên, tính thời gian thực thi đối với mỗi trình biên dịch theo công thức sau:

$$\text{thời gian thực thi} = \frac{\text{số chu kỳ xung nhịp}}{\text{tần số xung nhịp}}$$

với:

$$\text{số chu kỳ xung nhịp} = \sum_{i=1}^n (\text{CPI}_i \times C_i)$$

$$\text{số chu kỳ xung nhịp}_1 = (5 \times 1 + 1 \times 2 + 1 \times 3) \times 10^9 = 10 \times 10^9$$

$$\text{số chu kỳ xung nhịp}_2 = (10 \times 1 + 1 \times 2 + 1 \times 3) \times 10^9 = 15 \times 10^9$$

Vì vậy thời gian thực thi của từng compiler là:

$$\text{thời gian thực thi}_1 = \frac{10 \times 10^9}{4 \times 10^9} = 2.5 \text{ giây}$$

$$\text{thời gian thực thi}_2 = \frac{15 \times 10^9}{4 \times 10^9} = 3.75 \text{ giây}$$

Dựa vào thời gian thực thi thì ta thấy trình biên dịch 1 tạo ra chương trình thực thi nhanh hơn trình biên dịch 2.

Bây giờ ta tính theo MIPS dùng công thức:

$$\text{MIPS} = \frac{\text{số lệnh}}{\text{thời gian thực thi} \times 10^6}$$

$$\text{MIPS}_1 = \frac{(5 + 1 + 1) \times 10^9}{2.5 \times 10^6} = 2800$$

$$\text{MIPS}_2 = \frac{(10 + 1 + 1) \times 10^9}{3.75 \times 10^6} = 3200$$

1.3 BIỂU DIỄN HỆ SỐ

Một số có dạng tổng quát $(d_{k-1}d_{k-2}\dots d_1d_0)_r$ và có giá trị:

$$V = d_{k-1} \times r^{k-1} + d_{k-2} \times r^{k-2} + \dots + d_1 \times r + d_0$$

Với: k là số chữ số, r: hệ số (r = 2: hệ nhị phân, r = 10: hệ thập phân,...), d là giá trị của chữ số.

Ví dụ: biểu diễn số $(1011)_2$

$$(1011)_2 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1$$

biểu diễn số $(517)_{10}$

$$(517)_{10} = 5 \times 10^2 + 1 \times 10^1 + 7$$

➤ Chuyển từ hệ nhị phân sang hệ thập phân:

Để chuyển một số nhị phân sang số thập phân thì số thập phân tương ứng được xác định bằng $(d_{n-1} \times 2^{n-1}) + \dots + (d_1 \times 2^1) + d_0$

$$\text{Ví dụ: } (10011101)_2 = 2^7 + 2^4 + 2^3 + 2^2 + 1 = 157$$

➤ Chuyển từ hệ thập phân sang hệ nhị phân:

Để đổi số thập phân sang số nhị phân, thực hiện vòng lặp phép chia nguyên số thập phân cho 2 đến khi nào thương số bằng 0 thì kết thúc. Khi đó, các số dư của phép chia chính là số nhị phân cần tìm (ghi kết quả theo thứ tự từ dưới lên trên).

Ví dụ: đổi số 37 sang số nhị phân

Chia	Thương số	Số dư
37/2	18	1
18/2	9	0
9/2	4	1
4/2	2	0
2/2	1	0
1/2	0	1

Kết quả: $37 = (100101)_2$

- Chuyển từ hệ nhị phân sang hệ thập lục phân (*hexadecimal*):

Hệ thập lục phân sử dụng 16 kí số: 0 – 9, A – F. Bảng 1.1 thể hiện số tương ứng giữa số nhị phân, số thập phân và số thập lục phân.

Bảng 1.1: Số nhị phân, thập phân và thập lục phân

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

Để chuyển đổi số nhị phân sang số thập lục phân thì mỗi chữ số thập lục phân sẽ tương ứng với 4 bit của số nhị phân.

Ví dụ: đổi số nhị phân sau sang số thập lục phân:

1110 1011 0001 0110 1010 0111 1001 0100

nhóm 4 bit tương ứng từ phải sang trái tương ứng với các chữ số thập lục phân như sau:

E	B	1	6	A	7	9	4
1110	1011	0001	0110	1010	0111	1001	0100

Kết quả: $(1110\ 1011\ 0001\ 0110\ 1010\ 0111\ 1001\ 0100)_2 = (EB16A794)_{16}$

- Chuyển từ hệ thập lục phân sang hệ thập phân:

Để chuyển một số thập lục phân sang số thập phân thì số thập phân tương ứng được xác định bằng $(d_{n-1} \times 16^{n-1}) + (d_{n-2} \times 16^{n-2}) + \dots + (d_1 \times 16^1) + d_0$

Ví dụ: đổi số $(3BA4)_{16}$ sang số thập phân

$$(3BA4)_{16} = (3 \times 16^3) + (11 \times 16^2) + (10 \times 16^1) + 4 = 15268$$

- Chuyển từ hệ thập phân sang hệ thập lục phân:

Để đổi số thập phân sang số thập lục phân, thực hiện vòng lặp phép chia nguyên số thập phân cho 16 đến khi nào thương số bằng 0 thì kết thúc. Khi đó, các số dư của phép chia chính là số thập lục phân cần tìm (ghi kết quả theo thứ tự từ dưới lên trên).

Ví dụ: đổi số $(422)_{10}$ sang số thập lục phân

Chia	Thương số	Số dư
422/16	26	6
26/16	1	A
1/16	0	1

Kết quả: $(422)_{10} = (1A6)_{16}$

1.4 BIỂU DIỄN SỐ NGUYÊN

1.4.1 Biểu diễn số nguyên không dấu

Tất cả các số cũng như các ký tự trong máy vi tính đều được biểu diễn bằng các chữ số nhị phân. Để biểu diễn các số nguyên không dấu, người ta dùng n bit. Tương ứng với độ dài của số bit được sử dụng, ta có các khoảng giá trị xác định như sau:

Số bit	Khoảng giá trị
n bit	$0 \dots 2^n - 1$
8 bit	$0 \dots 255$
16 bit	$0 \dots 65535$

1.4.2 Biểu diễn số nguyên có dấu

➤ Biểu diễn bằng dấu và trị tuyệt đối (*Sign-Magnitude Representation*)

Để biểu diễn số nguyên có dấu n -bit có dạng $a_{n-1}a_{n-2} \dots a_1a_0$ thì sử dụng bit trái nhất (a_{n-1}) làm bit dấu. Bit dấu bằng 0 thể hiện số dương và bằng 1 thể hiện số âm. Giá trị được thể hiện bằng $n - 1$ bit còn lại. Biểu diễn số +18 và -18 như sau:

$$+18 = 00010010$$

$$-18 = 10010010$$

Số nguyên cần biểu diễn có thể được xác định theo công thức sau:

$$A = \begin{cases} \sum_{i=0}^{n-2} 2^i a_i & \text{if } a_{n-1} = 0 \\ -\sum_{i=0}^{n-2} 2^i a_i & \text{if } a_{n-1} = 1 \end{cases}$$

Phương pháp này gây ra hai khó khăn cho tính toán. Thứ nhất, khi phải thực hiện các phép cộng và phép trừ thì cần phải lưu ý đến dấu và độ lớn của hai số để thực hiện chính xác. Thứ hai, theo phương pháp này thì số 0 có hai cách biểu diễn:

$$+0 = 00000000$$

$$-0 = 10000000$$

Điều này gây ra khó khăn khi phải xử lý các phép tính đối với số 0. Do vậy phương pháp biểu diễn này ít được sử dụng trong bộ phận ALU của vi xử lý.

➤ Biểu diễn bằng bù 2

Giống như phương pháp biểu diễn dấu và trị tuyệt đối, ở phương pháp này cũng dùng bit trái nhất (*leftmost*) để xác định số âm hay số dương. Nếu bit này bằng 0 biểu diễn số dương, ngược lại bit này bằng 1 biểu diễn số âm.

Giả sử biểu diễn số nguyên A bằng n-bit ($a_{n-1}a_{n-2}\dots a_1a_0$), nếu $A \geq 0$ thì bit a_{n-1} bằng 0, các bit còn lại sẽ thể hiện độ lớn của số A. Trong trường hợp này các số được biểu diễn từ 0 đến $2^{n-1}-1$. Đối với số 0 thì chỉ có một cách biểu diễn duy nhất, đó là tất cả các bit đều bằng 0.

Trong trường hợp $A < 0$, để biểu diễn bằng bù 2 trước tiên lấy bù 1 (đảo ngược các bit) của giá trị A. Sau đó lấy kết quả bù 1 vừa tìm được cộng với 1 sẽ được số bù 2. Ví dụ: dùng 8 bit để biểu diễn -20 bằng phương pháp bù 2.

$$\begin{array}{r} 20 = 00010100 \\ \text{lấy bù 1} = 11101011 \\ \quad \quad \quad + \quad \quad \quad 1 \\ \hline 11101100 \text{ (bù 2)} \end{array}$$

Đối với trường hợp số âm thì phương pháp này biểu diễn được các số âm từ -1 đến -2^{n-1} nếu dùng n-bit. Trong trường hợp tổng quát (số dương và số âm) thì công thức để xác định giá trị của số A như sau:

$$A = -2^{n-1}a_{n-1} + \sum_{i=0}^{n-2} 2^i a_i$$

Với bit $a_{n-1} = 0$ nếu biểu diễn số dương và $a_{n-1} = 1$ nếu biểu diễn số âm.

Để thực hiện phép tính $A - B$ chỉ cần lấy bù 2 của B, sau đó cộng kết quả với A. Phương pháp này có ưu điểm là thực hiện các phép cộng và phép trừ tương đối đơn giản. Hơn nữa, chỉ có một biểu diễn duy nhất đối với số 0 nên thường được sử dụng trong ALU của vi xử lý.

➤ Biểu diễn bằng số thừa K:

Trong cách này, biểu diễn của một số dương N có được bằng cách “cộng thêm vào” số thừa K được chọn sao cho tổng của K và một số âm bất kỳ luôn luôn dương. Để biểu diễn số âm -N có được bằng cách lấy $K - N$.

Số thừa K được chọn như sau: K = 127 (nếu dùng 8 bit) hoặc K=1023 (16 bit)

Ví dụ: biểu diễn số +30 và -30 bằng phương pháp số thừa K dùng 8 bit

Biểu diễn số +30: $K+30 = 127+30 = 157$ (10011101)

Biểu diễn số -30: $K-30 = 127-30 = 97$ (01100001)

1.5 CÁC PHÉP TÍNH TRÊN SỐ NGUYÊN

Các phép tính này được thực hiện trên số nguyên được biểu diễn bằng phương pháp bù 2

➤ Phép cộng:

Phép cộng hai số nguyên được biểu diễn bằng phương pháp bù 2 dùng 4 bit thể hiện ở bảng 1.2. Trong đó bốn trường hợp đầu tiên phép cộng cho kết quả đúng, có thể xuất hiện bit nhớ (được tô đậm) nhưng có thể bỏ qua. Hai trường hợp cuối kết quả sai do hiện tượng tràn (*overflow*) xảy ra. Khi hiện tượng này xảy ra thì bộ vi xử lý sẽ phát ra tính hiệu tràn để báo không thể sử dụng kết quả này được. Hiện tượng tràn xảy ra khi thực hiện phép cộng giữa hai số dương hoặc giữa hai số âm mà bit dấu ở kết quả trái ngược với với bit dấu của hai số hạng. Bit nhớ (*carry*) có thể xuất hiện hoặc không trong trường hợp tràn xảy ra.

Bảng 1.2: Phép cộng 2 số được biểu diễn bằng phương pháp bù 2

$\begin{array}{r} 1001 = -7 \\ + 0101 = 5 \\ \hline 1110 = -2 \end{array}$ <p>(a) $(-7) + (+5)$</p>	$\begin{array}{r} 1100 = -4 \\ + 0100 = 4 \\ \hline 10000 = 0 \end{array}$ <p>(b) $(-4) + (+4)$</p>
$\begin{array}{r} 0011 = 3 \\ + 0100 = 4 \\ \hline 0111 = 7 \end{array}$ <p>(c) $(+3) + (+4)$</p>	$\begin{array}{r} 1100 = -4 \\ + 1111 = -1 \\ \hline 11011 = -5 \end{array}$ <p>(d) $(-4) + (-1)$</p>
$\begin{array}{r} 0101 = 5 \\ + 0100 = 4 \\ \hline 1001 = \text{Overflow} \end{array}$ <p>(e) $(+5) + (+4)$</p>	$\begin{array}{r} 1001 = -7 \\ + 1010 = -6 \\ \hline 10011 = \text{Overflow} \end{array}$ <p>(f) $(-7) + (-6)$</p>

➤ **Phép trừ:**

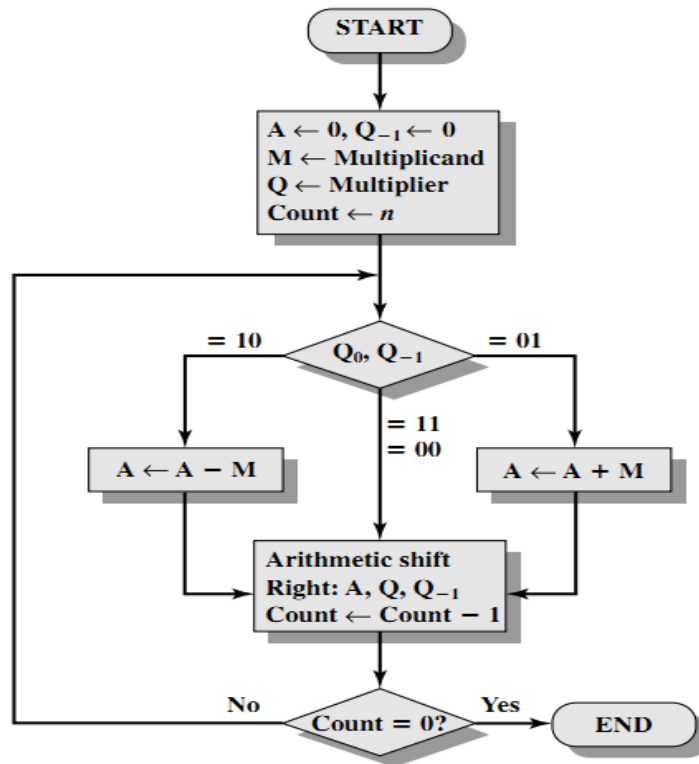
Để thực hiện phép tính $A - B$ chỉ cần lấy bù 2 của B, sau đó cộng kết quả với A. Bảng 1.3 minh họa phép trừ hai số được biểu diễn bằng phương pháp bù 2 dùng 4 bit. Cũng tương tự như phép cộng: bốn trường hợp đầu phép trừ cho kết quả đúng nhưng hai trường hợp cuối kết quả bị sai do hiện tượng tràn xảy ra. Cách xác định hiện tượng tràn xảy ra cũng giống như ở phép cộng.

Bảng 1.3: Phép trừ 2 số (M-S) được biểu diễn bằng bù 2

$\begin{array}{r} 0010 = 2 \\ + 1001 = -7 \\ \hline 1011 = -5 \end{array}$ <p>(a) $M = 2 = 0010$ $S = 7 = 0111$ $-S = 1001$</p>	$\begin{array}{r} 0101 = 5 \\ + 1110 = -2 \\ \hline 10011 = 3 \end{array}$ <p>(b) $M = 5 = 0101$ $S = 2 = 0010$ $-S = 1110$</p>
$\begin{array}{r} 1011 = -5 \\ + 1110 = -2 \\ \hline 11001 = -7 \end{array}$ <p>(c) $M = -5 = 1011$ $S = 2 = 0010$ $-S = 1110$</p>	$\begin{array}{r} 0101 = 5 \\ + 0010 = 2 \\ \hline 0111 = 7 \end{array}$ <p>(d) $M = 5 = 0101$ $S = -2 = 1110$ $-S = 0010$</p>
$\begin{array}{r} 0111 = 7 \\ + 0111 = 7 \\ \hline 1110 = \text{Overflow} \end{array}$ <p>(e) $M = 7 = 0111$ $S = -7 = 1001$ $-S = 0111$</p>	$\begin{array}{r} 1010 = -6 \\ + 1100 = -4 \\ \hline 10110 = \text{Overflow} \end{array}$ <p>(f) $M = -6 = 1010$ $S = 4 = 0100$ $-S = 1100$</p>

➤ **Phép nhân:**

Phép nhân thì phức tạp hơn so với phép cộng và phép trừ. Để nhân hai số nguyên có dấu được biểu diễn bằng bù 2 ta sử dụng giải thuật **Booth**. Giải thuật này được thể hiện ở hình 1.1



Hình 1.1: Giải thuật Booth

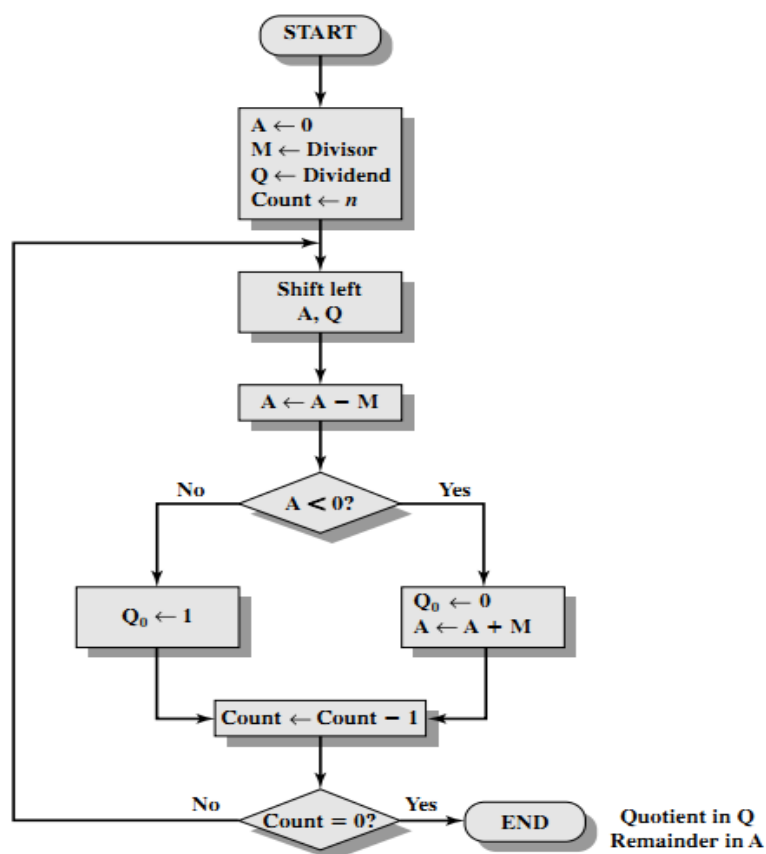
Trong giải thuật này các số bị nhân (*multiplicand*) và số nhân (*multiplier*) được đưa vào các thanh ghi M và Q tương ứng. Q_{-1} là thanh ghi 1-bit ghép vào bên phải của thanh ghi Q. Kết quả của phép nhân nằm trong thanh ghi A và Q. Các thanh ghi A và Q_{-1} được khởi tạo bằng 0. Giá trị n là số bit được dùng để biểu diễn số bị nhân và số nhân. Phép nhân hai số n-bit thì kết quả sẽ có độ dài là 2n bit.

Ví dụ phép nhân 7×3 bằng giải thuật Booth như sau:

A	Q	Q_{-1}	M		
0000	0011	0	0111	Khởi tạo	
1001	0011	0	0111	$A \leftarrow A - M$	Lần lặp 1
1100	1001	1	0111	Dịch	
1110	0100	1	0111	Dịch	Lần lặp 2
0101	0100	1	0111	$A \leftarrow A + M$	Lần lặp 3
0010	1010	0	0111	Dịch	
0001	0101	0	0111	Dịch	Lần lặp 4

➤ Phép chia:

Đối với số nguyên dương, phép chia được thực hiện theo giải thuật sau:



Hình 1.2: Giải thuật thực hiện phép chia

Các thanh ghi M chứa số chia (*divisor*) và Q chứa số bị chia (*dividend*). Khởi tạo thanh ghi A chứa 0 và số lần lặp của giải thuật bằng n (n là số bit biểu diễn số chia và số bị chia). Giải thuật kết thúc với thương số chứa trong Q và số dư chứa trong A. Xem ví dụ phép chia hai số dương (7/3) bằng giải thuật này như sau:

A	Q	
0000	0111	Khởi tạo
0000	1110	Dịch
1101		Dùng bù 2 của 0011 để trừ
1101		Trừ
0000	1110	Lưu lại, thiết lập $Q_0 = 0$
0001	1100	Dịch
1101		
1110		Trừ
0001	1100	Lưu lại, thiết lập $Q_0 = 0$
0011	1000	Dịch
1101		
0000	1001	Trừ, thiết lập $Q_0 = 1$
0001	0010	Dịch
1101		
1110		Trừ
0001	0010	Lưu lại, thiết lập $Q_0 = 0$

Một cách tổng quát (phép chia số có dấu) ta có công thức thể hiện quan hệ giữa số bị chia (D), số chia (V), thương số (Q) và số dư (R):

$$D = Q \times V + R$$

Xem ví dụ phép chia có dấu trong các trường hợp sau:

$$D = 7 \quad V = 3 \quad \Rightarrow \quad Q = 2 \quad R = 1$$

$$D = 7 \quad V = -3 \quad \Rightarrow \quad Q = -2 \quad R = 1$$

$$D = -7 \quad V = 3 \quad \Rightarrow \quad Q = -2 \quad R = -1$$

$$D = -7 \quad V = -3 \quad \Rightarrow \quad Q = 2 \quad R = -1$$

Như vậy, giá trị tuyệt đối của Q và R không phụ thuộc vào dấu của D và V nhưng dấu của Q và R có thể xác định như sau:

- dấu (R) = dấu (D)
- dấu (Q) = dấu (D) \times dấu (V)

Như vậy để thực hiện phép chia với số nguyên có dấu, trước tiên ta chuyển tất cả (số chia và số bị chia) về số dương tương ứng. Sau đó thực hiện phép chia các số không dấu theo giải thuật trên. Cuối cùng dựa vào quy tắc xét dấu để lấy kết quả đúng.

1.6 BIỂU DIỄN SỐ VỚI DẤU CHẤM ĐỘNG

Xét số với dấu chấm động 18.625 có thể được viết theo các cách sau: 18.625×10^0 ; 1.8625×10^1 ; 0.18625×10^2 ;...

Số này tương ứng với số nhị phân 10010.101 và cũng được viết theo nhiều cách: 10010.101×2^0 ; 1001.0101×2^1 ; 100.10101×2^2 ; 10.010101×2^3 ; 1.0010101×2^4 ;...

Để thuận tiện trong việc lưu trữ và so sánh giữa các số, số nhị phân sẽ được chuẩn hóa về dạng sau:

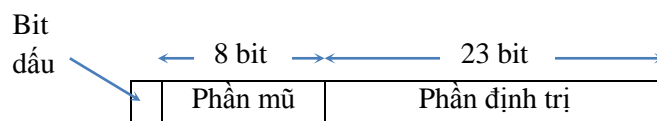
$$\pm 1.bbb..b \times 2^{\pm E}$$

Với b là số nhị phân (0 hoặc 1) và E là phần mũ. Như vậy để lưu trữ số với dấu chấm động ta cần lưu ba thành phần sau:

- Phần dấu (dương hoặc âm)
- Phần định trị
- Phần mũ

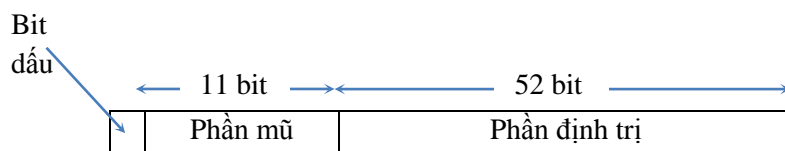
Để lưu trữ ba thành phần trên thì tiêu chuẩn được sử dụng phổ biến trong khoa học máy tính đó là tiêu chuẩn IEEE 754. Tiêu chuẩn này đưa ra việc lưu trữ số với dấu chấm động theo độ chính xác đơn (dùng 32 bit) và độ chính xác kép (64 bit).

❖ Độ chính xác đơn 32 bit được xác định như sau:



Trong đó:

- Bit dấu (1 bit) bằng 0 biểu diễn số dương, bằng 1 biểu diễn số âm
 - 8 bit tiếp theo biểu diễn phần mũ với giá trị của trường này bằng $127 \pm E$
 - 23 bit còn lại thể hiện phần định trị
- ❖ Độ chính xác kép 64 bit được xác định như sau:



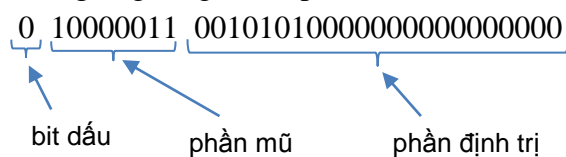
Trong đó:

- Bit dấu (1 bit) bằng 0 biểu diễn số dương, bằng 1 biểu diễn số âm
- 11 bit tiếp theo biểu diễn phần mũ với giá trị của trường này bằng $1023 \pm E$
- 52 bit còn lại thể hiện phần định trị

Ví dụ: hãy biểu diễn 18.625 dưới dạng chuẩn IEEE 754 chính xác đơn?

- Bước 1: chuyển 18.625 sang số nhị phân là 10010.101
- Bước 2: chuẩn hóa thành 1.0010101×2^4
- Bước 3: xác định giá trị 11 bit phần mũ bằng: $127 + 4 = 131$ (10000011) và

điền giá trị tương ứng từng thành phần



1.7 BIỂU DIỄN KÍ TỰ

Trong các hệ thống máy tính khác nhau, có thể sử dụng các bảng mã khác nhau: standard ASCII, extended ASCII, UNICODE,.... Các hệ thống trước đây thường dùng bảng mã ASCII (*American Standard Codes for Information Interchange*) tiêu chuẩn để biểu diễn các chữ, số và một số dấu thường dùng mà ta gọi chung là ký tự. Mỗi ký tự được biểu diễn bởi 7 bit trong một Byte. Với bảng mã extended ASCII thì biểu diễn mỗi ký tự bằng 8 bit. Hiện nay, một trong các bảng mã thông dụng được dùng là Unicode thì mỗi ký tự được mã hoá bởi 2 Byte.

00 NUL	10 DLE	20 SP	30 0	40 @	50 P	60 `	70 p
01 SOH	11 DC1	21 !	31 1	41 A	51 Q	61 a	71 q
02 STX	12 DC2	22 "	32 2	42 B	52 R	62 b	72 r
03 ETX	13 DC3	23 #	33 3	43 C	53 S	63 c	73 s
04 EOT	14 DC4	24 \$	34 4	44 D	54 T	64 d	74 t
05 ENQ	15 NAK	25 %	35 5	45 E	55 U	65 e	75 u
06 ACK	16 SYN	26 &	36 6	46 F	56 V	66 f	76 v
07 BEL	17 ETB	27 '	37 7	47 G	57 W	67 g	77 w
08 BS	18 CAN	28 (38 8	48 H	58 X	68 h	78 x
09 HT	19 EM	29)	39 9	49 I	59 Y	69 i	79 y
0A LF	1A SUB	2A *	3A :	4A J	5A Z	6A j	7A z
0B VT	1B ESC	2B +	3B ;	4B K	5B [6B k	7B {
0C FF	1C FS	2C ´	3C <	4C L	5C \	6C l	7C
0D CR	1D GS	2D -	3D =	4D M	5D]	6D m	7D }
0E SO	1E RS	2E .	3E >	4E N	5E ^	6E n	7E ~
0F SI	1F US	2F /	3F ?	4F O	5F _	6F o	7F DEL

Bảng mã ASCII

1.8 TÓM TẮT

Trong chương này giới thiệu về bốn giai đoạn phát triển của máy tính. Cách tiếp cận để phân loại máy tính hiện nay. Đánh giá hiệu suất của máy tính thông qua thời gian thực thi. Ngoài ra hiệu suất của máy tính còn được đo đạt thông qua chỉ số MIPS. Tiếp theo trình bày công thức tổng quát để biểu diễn một số, các cách để chuyển đổi một số từ hệ nhị phân sang hệ thập phân, hệ thập lục phân và ngược lại. Biểu diễn số nguyên có dấu bằng hai phương pháp khác nhau: phương pháp dấu và trị tuyệt đối, phương pháp bù 2. Trình bày cách thực hiện phép tính cộng và phép trừ các số nguyên có dấu được biểu diễn bằng bù 2. Các giải thuật để thực hiện phép nhân và phép chia các số nguyên có dấu. Biểu diễn số với dấu chấm động theo tiêu chuẩn IEEE 754 với độ chính xác đơn (32 bit) và độ chính xác kép (64 bit). Cuối cùng trình bày cách biểu diễn kí tự bằng bảng mã ASCII và bảng mã UNICODE.

CÂU HỎI VÀ BÀI TẬP CHƯƠNG 1

1. Máy tính được phân thành mấy loại?
2. Định nghĩa hiệu suất máy tính?
3. Công thức biểu diễn một số tổng quát?
4. Cách chuyển một số nhị phân sang số thập phân?
5. Cách chuyển một số thập lục phân sang số thập phân?
6. Hãy tính MIPS của hai máy tính với số liệu như sau:

	Máy tính A	Máy tính B
Số lệnh	10 tỷ	8 tỷ
Tỷ số xung nhịp	4 GHz	4 GHz
CPI	1.0	1.1

7. Cho biết 3 bộ xử lý P1, P2, P3 thực thi cùng tập lệnh với tần số xung nhịp (*clock rate*) và CPI được cho như sau:

Bộ xử lý	Tần số xung nhịp	CPI
P1	3 GHz	1.5
P2	2.5 GHz	1.0
P3	4 GHz	2.2

- a. Hãy tính MIPS của mỗi bộ xử lý.
 - b. Giả sử mỗi bộ xử lý thực thi chương trình trong 10s. Hãy tính số chu kỳ xung nhịp và số lệnh?
8. Cho biết 2 bộ xử lý thực thi cùng tập lệnh với số liệu như sau:

	Tần số xung nhịp	CPI lớp A	CPI lớp B	CPI lớp C	CPI lớp D
P1	2.5 GHz	1	2	3	3
P2	3 GHz	2	2	2	2

Thực thi một chương trình có 10^6 lệnh với 10% lớp A, 20% lớp B, 50% lớp C và 20% lớp D.

- a. Tính số chu kỳ xung nhịp cho mỗi trường hợp?
 - b. Tính CPI tổng (*global CPI*) đối với mỗi trường hợp?
 - c. Bộ xử lý nào thực thi nhanh hơn?
9. Cho số lệnh của một chương trình như sau:

	Lệnh số học	Lệnh store	Lệnh load	Lệnh điều kiện	Tổng
a.	650	100	600	50	1400
b.	750	250	500	500	2000

- a. Giả sử lệnh số học cần 1 chu kỳ, lệnh load và store cần 5 chu kỳ, lệnh branch cần 2 chu kỳ. Tính thời gian thực thi của chương trình với bộ xử lý có tần số 2 GHz?

b. Tính CPI của chương trình?

- 10.** Đổi số 78.625 sang số nhị phân ?
- 11.** Đổi số nhị phân 01011101001_2 sang hệ số 8 (Octal) và hệ số 16 (Hexadecimal)?
- 12.** Dùng cách biểu diễn bằng trị tuyệt đối và dấu, hãy biểu diễn các số + 125, - 45 dùng 1 byte (8 bit)?
- 13.** Dùng cách biểu diễn bằng số bù 2, hãy biểu diễn các số - 127, - 11 dùng 1 byte (8 bit)?
- 14.** Số nhị phân 8 bit $(11001100)_2$, số này tương ứng với số nguyên thập phân có dấu là bao nhiêu nếu được biểu diễn bằng dấu và trị tuyệt đối?
- 15.** Số nhị phân 8 bit $(10010011)_2$, số này tương ứng với số nguyên thập phân có dấu là bao nhiêu nếu được biểu diễn bằng số bù 2?
- 16.** Hãy thực hiện phép tính nhị phân để tính tổng hai số - 25 và 60 với hai số này được biểu diễn bằng bù 2 (dùng 8 bit)?
- 17.** Hãy thực hiện phép tính nhị phân để thực hiện phép trừ: $45 - 11$ với hai số này được biểu diễn bằng bù 2 (dùng 8 bit)?
- 18.** Dùng giải thuật Booth để thực hiện phép nhân -4×2 nếu hai số này được biểu diễn bằng bù 2 (dùng 4 bit). Hãy cho biết kết quả của phép nhân này?
- 19.** Dùng giải thuật chia hai số để thực hiện phép chia $7/2$. Hãy cho biết thương số và số dư? Giả sử hai số này được biểu diễn bằng bù 2 (dùng 4 bit).
- 20.** Hãy biến đổi số thập phân - 25.75 sang số chấm động chuẩn IEEE 754 chính xác đơn (32 bit)?

CHƯƠNG 2

TẬP LỆNH VI XỬ LÝ

Mục đích: Giới thiệu về tập lệnh vi xử lý MIPS bao gồm các lệnh số học – luận lý, lệnh truyền dữ liệu, lệnh rẽ nhánh.... Các kiểu toán hạng khác nhau được sử dụng như: toán hạng thanh ghi, toán hạng bộ nhớ và toán hạng trực tiếp. Ba định dạng lệnh cơ bản của MIPS như: R-format, I-format và J-format. Cách chuyển từ một lệnh vi xử lý MIPS sang mã máy.

2.1 GIỚI THIỆU

Chương này sẽ giới thiệu về tập lệnh của vi xử lý, là ngôn ngữ mà máy có thể hiểu và thực hiện chương trình. Kiến trúc tập lệnh được tìm hiểu ở chương này tập trung vào vi xử lý MIPS (*Microprocessor without Interlocked Pipeline Stages*). Vi xử lý MIPS được phát triển bởi công ty MIPS Technologies vào những năm đầu thập niên 80. Đây là vi xử lý thuộc kiến trúc RISC (*Reduced Instruction Set Computer*). So với kiến trúc CISC (*Complex Instruction Set Computer*) mà đại diện là các dòng vi xử lý x86 của hãng Intel thì kiến trúc RISC có các lệnh đơn giản và số lượng lệnh ít hơn. Vì vậy, RISC thường chạy nhanh hơn RISC do kiến trúc này có thiết kế đơn giản. MIPS là vi xử lý 32 bit, sau đó được mở rộng thành vi xử lý 64 bit.

2.2 TOÁN TỬ (*operation*)

Phép toán cơ bản trong máy tính đó là các phép toán số học, xem biểu diễn của phép toán cộng hai số thể hiện trong kiến trúc MIPS như sau:

ADD a, b, c

phép toán này thể hiện cộng b với c và kết quả lưu vào a. Trong kiến trúc MIPS các phép toán số học như thế này thường bao gồm ba toán hạng: một toán hạng đích dùng để chứa kết quả và hai toán hạng nguồn. Một ví dụ khác, để tính tổng của bốn biến b, c, d, e và kết quả lưu vào a. Ta có chuỗi các lệnh để thực hiện như sau:

ADD a, b, c # tính tổng b + c và lưu vào a

ADD a, a, d # tính tổng của b+c+d và lưu vào a

ADD a, a, e # tính tổng của b+c+d+e và lưu vào a

Như vậy để cộng bốn biến thì ta cần sử dụng đến ba lệnh máy. Kí hiệu # là dùng để chú thích trong MIPS để người đọc dễ hiểu, khi biên dịch máy sẽ bỏ qua các chú thích này.

Trong kiến trúc MIPS sử dụng 32 thanh ghi, các thanh ghi này có cùng độ dài 32 bit. Không gian địa chỉ trong bộ nhớ cũng sử dụng 32 bit. Bảng 2.1 thể hiện các thanh ghi và không gian bộ nhớ.

Bảng 2.1: Các thanh ghi và địa chỉ bộ nhớ trong MIPS

Số	Tên	Ý nghĩa
\$0	\$zero	Hằng số 0
\$1	\$at	Assembler Temporary
\$2-\$3	\$v0-\$v1	Giá trị trả về của hàm hoặc biểu thức
\$4-\$7	\$a0-\$a3	Các tham số của hàm
\$8-\$15	\$t0-\$t7	Thanh ghi tạm (không giữ giá trị trong quá trình gọi hàm)
\$16-\$23	\$s0-\$s7	Thanh ghi lưu trữ (giữ giá trị trong suốt quá trình gọi hàm)
\$24-\$25	\$t8-\$t9	Thanh ghi tạm
\$26-27	\$k0-\$k1	Dự trữ cho nhân HĐH
\$28	\$gp	Con trỏ toàn cục (global pointer)
\$29	\$sp	Con trỏ stack
\$30	\$fp	Con trỏ frame
\$31	\$ra	Địa chỉ trả về

Các lệnh trong kiến trúc MIPS được chia thành 5 loại: lệnh số học, lệnh truyền dữ liệu, lệnh luận lý, lệnh nhảy có điều kiện và lệnh nhảy không điều kiện.

Bảng 2.2: Tập lệnh vi xử lý MIPS

Danh mục	Lệnh	Ví dụ	Ý nghĩa
Lệnh số học	Add	ADD \$s1,\$s2,\$s3	$\$s1 = \$s2 + \$s3$
	Subtract	SUB \$s1,\$s2,\$s3	$\$s1 = \$s2 - \$s3$
	Add immediate	ADDI \$s1,\$s2,20	$\$s1 = \$s2 + 20$
Lệnh truyền dữ liệu	load word	LW \$s1,20(\$s2)	$\$s1 = \text{Memory}[\$s2 + 20]$
	store word	SW \$s1,20(\$s2)	$\text{Memory}[\$s2 + 20] = \$s1$
	load half	LH \$s1,20(\$s2)	$\$s1 = \text{Memory}[\$s2 + 20]$
	store half	SH \$s1,20(\$s2)	$\text{Memory}[\$s2 + 20] = \$s1$
	load byte	LB \$s1,20(\$s2)	$\$s1 = \text{Memory}[\$s2 + 20]$
	store byte	SB \$s1,20(\$s2)	$\text{Memory}[\$s2 + 20] = \$s1$
	load upper immed	LUI \$s1,20	$\$s1 = 20 * 2^{16}$
Lệnh luận lý	and	AND \$s1,\$s2,\$s3	$\$s1 = \$s2 \& \$s3$
	or	OR \$s1,\$s2,\$s3	$\$s1 = \$s2 \mid \$s3$
	nor	NOR \$s1,\$s2,\$s3	$\$s1 = \sim(\$s2 \mid \$s3)$
	and immediate	ANDI \$s1,\$s2,20	$\$s1 = \$s2 \& 20$
	or immediate	ORI \$s1,\$s2,20	$\$s1 = \$s2 \mid 20$
	shift left logical	SLL \$s1,\$s2,10	$\$s1 = \$s2 \ll 10$
	shift right logical	SRL \$s1,\$s2,10	$\$s1 = \$s2 \gg 10$
Lệnh rẽ nhánh điều kiện	branch on equal	BEQ \$s1,\$s2,25	if($\$s1 = \$s2$) PC + 4 + 100
	branch on not equal	BNE \$s1,\$s2,25	if($\$s1 \neq \$s2$) PC + 4 + 100
	set on less than	SLT \$s1,\$s2,\$s3	if($\$s2 < \$s3$) $\$s1 = 1$; else $\$s1 = 0$
Lệnh rẽ nhánh không điều kiện	jump	J 2500	chuyển đến 10000
	jump register	JR \$ra	chuyển đến \$ra
	jump and link	JAL 2500	$\$ra = PC + 4$; chuyển đến 10000

2.3 TOÁN HẠNG (*operand*)

2.3.1 Toán hạng thanh ghi

Số toán hạng trong các lệnh luận lý được giới hạn là ba, các toán hạng này thông thường là các thanh ghi của vi xử lý MIPS. Các thanh ghi này có độ dài là 32 bit. Một từ (word) trong bộ nhớ cũng có độ dài 32 bit.

Ví dụ: viết các lệnh MIPS thể hiện câu lệnh sau

$$f = (g + h) - (i + j)$$

giả sử f , g , h , i , và j lần lượt được lưu trong $\$s0$, $\$s1$, $\$s2$, $\$s3$ và $\$s4$

Để thực hiện các phép tính trên ta dùng hai thanh ghi $\$t0$ và $\$t1$ lưu gửi kết quả trung gian

ADD $\$t0$, $\$s1$, $\$s2$

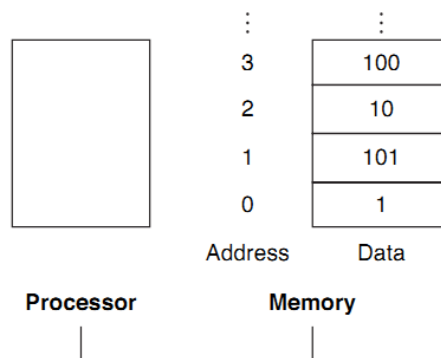
ADD $\$t1$, $\$s3$, $\$s4$

SUB $\$s0$, $\$t0$, $\$t1$

2.3.2 Toán hạng bộ nhớ

Đối với những kiểu dữ liệu đơn thì có thể chứa trong thanh ghi nhưng với những dữ liệu có cấu trúc như mảng thì phải được lưu trong bộ nhớ. Khác với thanh ghi, không gian bộ nhớ lớn nên có thể chứa rất nhiều dữ liệu.

Toán hạng trong các lệnh số học và luận lý chỉ là những thanh ghi. Do đó để thực hiện các phép toán này trên dữ liệu bộ nhớ thì phải có lệnh truyền dữ liệu từ bộ nhớ vào thanh ghi. Lệnh này chính là lệnh *load* có tên cụ thể trong kiến trúc MIPS là lệnh *LW* (*load word*).



Hình 2.1: Địa chỉ ô nhớ và dữ liệu tương ứng

Các phần tử của mảng được lưu liên tiếp trong bộ nhớ, ví dụ một mảng A chứa 100 phần tử (mỗi phần tử là một từ - word). Giả sử địa chỉ của mảng được lưu trong thanh ghi $\$s3$ và các biến g và h lần lượt được lưu trong $\$s1$ và $\$s2$. Ta cần thực hiện phép gán sau:

$$g = h + A[8]$$

Phép toán cộng trên bao gồm một toán hạng thanh ghi và một toán hạng bộ nhớ nhưng lệnh *add* thì chỉ thực hiện trên toán hạng thanh ghi. Do đó, đầu tiên phải thực hiện đưa dữ liệu từ bộ nhớ vào một thanh ghi tạm nào đó. Do mỗi word chiếm 4 byte dữ liệu nên để truy xuất đến $A[8]$ cần phải cộng thêm 32 (8×4) vào địa chỉ mảng (Giả sử chỉ số phần tử đầu tiên của mảng bắt đầu từ 0).

LW \$t0, 32(\$s3) # gán giá trị A[8] vào thanh ghi tạm \$t0
 lệnh này thực hiện lấy nội dung một từ (word) tại địa chỉ [\$s3 + 32] lưu vào thanh ghi \$t0. Thanh ghi chứa địa chỉ mảng \$s3 được gọi là thanh ghi cơ sở (*base register*) và giá trị 32 để xác định phần tử mảng được gọi là độ dời (*offset*). Cuối cùng thực hiện phép cộng và gán kết quả.

ADD \$s1,\$s2,\$t0 # $g = h + A[8]$

Trái ngược với lệnh *load* là lệnh *store*, lệnh này sẽ chép dữ liệu từ thanh ghi vào bộ nhớ. Ví dụ xét một mảng A như ví dụ trên với địa chỉ mảng được lưu trong \$s3 và nội dung của biến h được lưu trong \$s2. Thực hiện phép toán sau đây:

$A[12] = h + A[8]$

Trước tiên lấy nội dung A[8] từ bộ nhớ lưu vào một thanh ghi tạm, sau đó thực hiện phép toán cộng:

LW \$t0, 32(\$s3) # lưu A[8] vào thanh ghi tạm \$t0

ADD \$t0, \$s2, \$t0 # $h + A[8]$ lưu vào \$t0

Cuối cùng lưu kết quả vào phần tử A[12] trong bộ nhớ bằng cách dùng lệnh *store* mà lệnh này thể hiện trong MIPS là SW (*store word*). Để xác định phần tử A[12] của mảng thì độ dời (offset) tương ứng là 48 (4×12).

SW \$t0, 48(\$s3) # lưu $h + A[8]$ vào A[12]


Do số lượng thanh ghi trong vi xử lý có giới hạn nên những dữ liệu nào được dùng thường xuyên sẽ được ưu đưa vào thanh ghi, còn những dữ liệu khác ít sử dụng hơn sẽ được lưu vào bộ nhớ. Hơn nữa, truy cập dữ liệu trong thanh ghi nhanh hơn và tiêu tốn ít năng lượng hơn trong bộ nhớ. Do đó để đảm bảo hiệu quả cao, trình biên dịch phải biết cách sử dụng thanh ghi một cách hiệu quả.

2.3.3 Toán hạng trực tiếp (*immediate operand*)

Toán hạng trực tiếp còn được gọi là toán hạng hằng (*constant operand*) được sử dụng thường xuyên trong các phép toán. Điển hình là sử dụng để trỏ đến phần tử tiếp theo trong một mảng. Theo thống kê thì có hơn một nửa các lệnh có sử dụng toán hạng trực tiếp. ví dụ như khi thực hiện phép cộng với một hằng số ta sử dụng lệnh ADDI (*add immediate*)

ADDI \$s3, \$s3, 4 # $\$s3 = \$s3 + 4$

Toán hạng trực tiếp cũng làm cho lệnh được thực hiện nhanh hơn. Trong MIPS có một thanh ghi đặc biệt, đó là thanh ghi \$zero hay \$0 mà thanh ghi này luôn có giá trị 0. Một ứng dụng của thanh ghi này có thể thay lệnh MOVE bằng lệnh ADD mà có một toán hạng nguồn là \$zero. Ngoài ra, do MIPS có hỗ trợ toán hạng hằng là số âm nên sẽ không có lệnh SUBI (*subtract immediate*).

 Lệnh lui: toán hạng hằng chỉ có độ lớn 16 bit. Do đó, để sử dụng hằng số 32 bit thì thường sử dụng kết hợp với lệnh lui.

LUI register, const # const là số 16 bit

lệnh này ghi giá trị const vào 16 bit cao (16 bit bên trái) của thanh ghi register, còn 16 bit thấp (16 bit bên phải) được thiết lập bằng 0.

Ví dụ: hãy đưa giá trị sau vào thanh ghi \$s0

0000 0000 0011 1101 0000 1001 0000 0000

Để thực hiện điều này, đầu tiên đưa 16 bit cao (giá trị 61) vào 16 bit cao của thanh ghi \$s0 dùng lệnh lui

LUI \$s0, 61 # 0000 0000 0011 1101₂ = 61₁₀

tiếp theo chèn 16 bit thấp (giá trị 2304) vào bằng lệnh ORI

ORI \$s0,\$s0,2304 # 0000 1001 0000 0000₂ = 2304₁₀

2.3.4 Định dạng lệnh

Tất cả lệnh được biểu dạng trong MIPS có cùng độ dài là 32 bit được chia thành các trường (*field*) khác nhau tùy theo từng lệnh. Như chúng ta đã biết trong MIPS có tất cả 32 thanh ghi được đánh số từ 0 đến 31. Những thanh ghi này có chức năng khác nhau nên có thể sử dụng tên thay cho số. Như các thanh ghi từ \$s0 đến \$s7 tương ứng với các thanh ghi từ 16 đến 23 hoặc các thanh ghi từ \$t0 đến \$t7 tương ứng với các thanh ghi từ 8 đến 15. Điều này có nghĩa là \$s0 tương ứng thanh ghi 16, \$s1 tương ứng thanh ghi 17, \$s2 tương ứng thanh ghi 18,...,\$t0 tương ứng với thanh ghi 8,\$t1 tương ứng với thanh ghi 9,v.v....Các số sẽ được sử dụng trong định dạng lệnh. Ví dụ xét lệnh:

ADD \$t0, \$s1, \$s2

Định dạng của lệnh trên được thể hiện như sau:

0	17	18	8	0	32
---	----	----	---	---	----

Định dạng lệnh trên gồm có 6 trường: trường thứ nhất (giá trị 0) và trường cuối cùng (giá trị 32) biểu diễn cho lệnh cộng. Trường thứ hai chứa thanh ghi nguồn đầu tiên (17 = \$s1), trường thứ ba chứa thanh ghi nguồn thứ hai (18 = \$s2). Trường thứ tư chứa thanh ghi đích để nhận kết quả của phép tính tổng (8 = \$t0). Trường thứ 5 không sử dụng nên thiết lập giá trị bằng 0. Hay biểu diễn dưới dạng nhị phân của lệnh trên là:

000000	10001	10010	01000	00000	100000
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

Một cách tổng quát, tập lệnh MIPS gồm có ba định dạng lệnh: R-format, I-format và J-format.

❖ Định dạng R-format:

Một lệnh MIPS có chiều dài 32 bit, với định dạng này thì các bit được phân phối vào 6 trường như sau:

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

- *op*: mã lệnh (opcode) cho biết lệnh được thực hiện
- *rs*: thanh ghi nguồn thứ nhất
- *rt*: thanh ghi nguồn thứ hai
- *rd*: thanh ghi đích dùng chứa kết quả của phép toán

- *shamt*: số bit được dịch shift amount (dùng trong lệnh các lệnh dịch chuyển, đối với các lệnh khác không sử dụng trường này thì được thiết lập giá trị 0)
- *funct*: mã hàm (function code) kết hợp với mã lệnh (opcode) để biết một lệnh cụ thể sẽ được thực hiện.

❖ Định dạng I-format:

Định dạng này có các bit được phân phối như sau:

op	rs	rt	constant or address
6 bits	5 bits	5 bits	16 bits

Trong định dạng này ta có hai thanh ghi rs và rt (mỗi trường chứa 5 bit). Trường cuối cùng chứa 16 bit dùng để chứa giá trị hằng (các lệnh dùng toán hạng hằng như addi) hoặc chứa địa chỉ (các lệnh load, store và các lệnh nhảy có điều kiện). Giá trị của trường này nằm trong giới hạn $\pm 2^{15}$ (32768)

Ví dụ: lw \$t0, 32(\$s3) trong lệnh này trường rs chứa giá trị 19 (của thanh ghi \$s3), còn rt chứa giá trị 8 (của thanh ghi \$t0). Trường cuối cùng 16 bit chứa giá trị 32. Trong trường hợp này thanh ghi rt đóng vai trò là thanh ghi đích để chứa kết quả của lệnh load.

Trong hai định dạng R-format và I-format: ba trường đầu tiên giống nhau, trường thứ tư trong I-format là bao gồm tổng số bit của ba trường còn lại trong định dạng R-format. Để vi xử lý MIPS phân biệt được định dạng nào thì trường *op* được sử dụng để nhận dạng. Dưới đây là một số giá trị của các định dạng trên:

Instruction	Format	op	rs	rt	rd	shamt	funct	address
add	R	0	reg	reg	reg	0	32 _{ten}	n.a.
sub (subtract)	R	0	reg	reg	reg	0	34 _{ten}	n.a.
add immediate	I	8 _{ten}	reg	reg	n.a.	n.a.	n.a.	constant
lw (load word)	I	35 _{ten}	reg	reg	n.a.	n.a.	n.a.	address
sw (store word)	I	43 _{ten}	reg	reg	n.a.	n.a.	n.a.	address

Chú ý hai lệnh add và sub có mã lệnh *op* giống nhau (0) nhưng giá trị hàm *funct* khác nhau. Trong bảng trên *n.a* (not applicable) có nghĩa là không sử dụng trường này.

Xem một ví dụ khác về mảng như sau: giả sử \$t1 chứa địa chỉ của mảng A, giá trị h chứa trong thanh ghi \$s2, hãy biểu diễn các lệnh hợp ngữ tương ứng với phát biểu:

$$A[300] = h + A[300]$$

Các lệnh hợp ngữ tương ứng với phát biểu trên như sau:

LW \$t0,1200(\$t1)

ADD \$t0,\$s2,\$t0

SW \$t0,1200(\$t1)

Hay biểu diễn lệnh mã máy là:

op	rs	rt	rd	address/ shamt	funct
35	9	8	1200		
0	18	8	8	0	32
43	9	8	1200		

Lệnh lw được xác định bằng giá trị 35 trong trường *op*, giá trị 9 (\$t1) trong trường *rs* và giá trị 8 (\$t0) trong trường *rt*. Giá trị offset để chọn phần tử A[300] là 1200 ($1200 = 300 \times 4$). Tương tự với hai lệnh add và sw còn lại được biểu diễn bằng các giá trị tương ứng trong các trường. Tương ứng ta có biểu diễn dạng nhị phân của ba lệnh trên:

100011	01001	01000	0000 0100 1011 0000		
000000	10010	01000	01000	00000	100000
101011	01001	01000	0000 0100 1011 0000		

Bảng sau đây thể hiện một số lệnh thông dụng trong MIPS

Lệnh	Định dạng	Ví dụ						Chú thích
ADD	R	0	18	19	17	0	32	ADD \$s1,\$s2,\$s3
SUB	R	0	18	19	17	0	34	SUB \$s1,\$s2,\$s3
ADDI	I	8	18	17	100			ADDI \$s1,\$s2,100
LW	I	35	18	17	100			LW \$s1,100(\$s2)
SW	I	43	18	17	100			SW \$s1,100(\$s2)
Kích thước		6bit	5bit	5 bit	5bit	5 bit	6 bit	Tất cả lệnh MIPS đều có độ dài 32 bit
R-format	R	op	rs	rt	rd	shamt	funct	Định dạng lệnh số học
I-format	I	op	rs	rt	address			Định dạng lệnh truyền dữ liệu

Các lệnh nhảy có điều kiện cũng có định dạng I-format, xem ví dụ sau đây:

BNE \$s0,\$s1,Exit

lệnh này có định dạng bao gồm có 4 trường: trường *op* (6 bit) có giá trị 5, trường thứ hai 5 bit chứa giá trị 16 (\$s0), trường thứ ba 5 bit chứa giá trị 17 (\$s1) và trường cuối cùng 16 bit chứa địa chỉ nhảy đến.

5	16	17	Exit
6 bits	5 bits	5 bits	16 bits

❖ Định dạng J-format:

Lệnh nhảy không điều kiện sử dụng định dạng này, ví dụ:

J 10000

câu lệnh này có định dạng J-format đơn giản nhất so với các định dạng trên gồm một trường *op* (6 bit) có giá trị 2 và trường địa chỉ (26 bit) chứa giá trị 10000:

2	10000
6 bits	26 bits

Bảng 2.3 trình bày tóm tắt cả 3 định dạng lệnh của tập lệnh MIPS với số bit được phân bố tương ứng với từng trường trong mỗi định dạng lệnh khác nhau.

Bảng 2.3: Định dạng lệnh MIPS

Tên	Các trường						Chú thích
Kích thước	6 bit	5 bit	5 bit	5 bit	5 bit	6 bit	Tất cả lệnh MIPS đều có độ dài 32 bit
R-format	op	rs	rt	rd	shamt	funct	Định dạng lệnh số học
I-format	op	rs	rt	địa chỉ / hằng số			Định dạng lệnh truyền dữ liệu, lệnh nhảy có điều kiện, hằng số
J-format	op	địa chỉ					Định dạng lệnh nhảy không điều kiện

2.3.5 Lệnh luận lý (*logical operation*)

Các lệnh luận lý thực hiện trên từng bit bao gồm các lệnh sau:

- Lệnh dịch trái SLL (*shift left logical*): các bit 0 được thêm vào bên phải khi thực hiện lệnh dịch.
- Lệnh dịch phải SRL (*shift right logical*): các bit 0 được thêm vào bên trái khi thực hiện lệnh dịch.

Giả sử thanh ghi \$s0 chứa giá trị 9

0000 0000 0000 0000 0000 0000 1001_{two} = 9_{ten}

Sau khi thực hiện lệnh dịch trái 4 bit:

SLL \$t2,\$s0,4

thì kết quả thanh ghi \$t2 chứa giá trị là 144.

0000 0000 0000 0000 0000 0000 1001 0000_{two} = 144_{ten}

Lệnh dịch này được biểu diễn trong định dạng R-format như sau:

op	rs	rt	rd	shamt	funct
0	0	16	10	4	0

Với hai trường op và funct chứa giá trị 0, trường thanh ghi rs không sử dụng nên thiết lập bằng 0, trường rt chứa giá trị 16 (\$s0) và rd chứa giá trị 10 (\$t2). Ngoài ra trường shamt (*shift amount*) chứa giá trị 4 là số bit để dịch trái.

Khi dịch trái i bit thì kết quả thu được tương đương với nhân 2^i , như với ví dụ trên thì dịch trái 4 bit nên tương đương với nhân cho 2^4 hay 16 ($9 \times 16 = 144$).

- Lệnh AND: thực hiện phép toán and với hai toán hạng nguồn theo từng bit và kết quả lưu vào toán hạng đích.

Giả sử thanh ghi \$t2 chứa giá trị:

0000 0000 0000 0000 0000 1101 1100 0000_{two}

và thanh ghi \$t1 chứa giá trị:

```
0000 0000 0000 0000 0011 1100 0000 0000two
```

sau đó thực hiện lệnh and sau:

```
AND $t0,$t1,$t2
```

kết quả thanh ghi \$t0 chứa giá trị:

```
0000 0000 0000 0000 0000 1100 0000 0000two
```

- Lệnh OR: tương tự như lệnh and nhưng thực hiện phép toán or trên từng bit.

Ví dụ: các thanh ghi \$t1 và \$t2 chứa giá trị như trên, sau khi thực hiện phép toán

```
OR $t0,$t1,$t2
```

thanh ghi \$t0 sẽ chứa kết quả là:

```
0000 0000 0000 0000 0011 1101 1100 0000two
```

- Lệnh NOT và lệnh NOR: lệnh not thực hiện đảo bit (0 thành 1, 1 thành 0) trên một toán hạng, còn có lệnh NOR (not or) thực hiện đảo ngược của lệnh OR.

Ngoài ra, khi một toán hạng nguồn chứa giá trị hằng số thì các lệnh ANDI (*and immediate*) và ORI (*or immediate*) được sử dụng.

2.3.6 Các lệnh rẽ nhánh

➤ Câu lệnh điều kiện IF:

Các câu lệnh điều kiện dựa vào kết quả so sánh của biểu thức luận lý để quyết định chuyển điều khiển đến một nhãn (label) được chỉ định. Các lệnh này bao gồm:

```
BEQ register1,register2,L1
```

lệnh BEQ (*branch if equal*) nhảy điều khiển đến nhãn L1 nếu register1 = register2

```
BNE register1,register2,L1
```

lệnh BNE (*branch if not equal*) nhảy điều khiển đến nhãn L1 nếu register1 ≠ register2

Các lệnh này thường được sử dụng để thể hiện câu lệnh if-then-else của ngôn ngữ cấp cao.

Ví dụ hãy biểu diễn các lệnh hợp ngữ MIPS tương ứng với phát biểu if của ngôn ngữ C:

```
If (i == j) f = g + h; else f = g - h
```

với các biến f,g,h,i và j lần lượt được lưu trong các thanh ghi từ \$s0 đến \$s4.

Các lệnh tương ứng như sau:

```
BNE $s3,$s4,Else
```

```
ADD $s0,$s1,$s2
```

```
J Exit
```

```
Else: SUB $s0,$s1,$s2
```

```
Exit: ...
```

➤ Vòng lặp:

Giả sử ta có vòng lặp được thể hiện trong ngôn ngữ C như sau:

```
while (save[i]==k)
```

```
    i +=1;
```

Với các giá trị i, k lần lượt được lưu trong thanh ghi $\$s3, \$s5$ và địa chỉ của mảng *save* (mỗi phần tử mảng là một từ - word gồm 4 byte) được lưu trong thanh ghi $\$s6$. Các lệnh hợp ngữ tương ứng trong MIPS là:

```

Loop: SLL $t1,$s3,2      # thanh ghi tạm $t1 = i*4
      ADD $t1,$t1,$s6     # $t1 chứa địa chỉ của save[i]
      LW $t0,0($t1)       # thanh ghi tạm $t0 = save[i]
      BNE $t0,$s5,Exit    # nhảy đến nhãn Exit nếu save[i] ≠ k
      ADDI $s3,$s3,1      # i = i+1
      J Loop              # lệnh nhảy không điều kiện

Exit: ...

```

Ngoài ra, trong MIPS có các lệnh sau:

SLT register1,register2,register3

lệnh SLT (*set on less then*) sẽ thiết lập thanh ghi register1 = 1 nếu thanh ghi register2 nhỏ hơn thanh ghi register3, thiết lập register1 = 0 nếu ngược lại.

SLTI register1,register2,const

lệnh SLTI (*set on less then immediate*) giống như lệnh slt nhưng thanh ghi thứ 3 được thay bằng hằng số.

```

ví dụ: SLT $t0,$s3,$s4      # $t0 = 1 nếu $s3 < $s4
      SLTI $t0,$s2,10       # $t0 = 1 nếu $s2 < 10

```

Trình biên dịch MIPS dùng tổ hợp của các lệnh SLT, SLTI, BEQ, BNE và giá trị 0 (dùng thanh ghi \$zero) để tạo ra tất cả các điều kiện so sánh: bằng, không bằng, nhỏ hơn, nhỏ hơn hoặc bằng, lớn hơn, lớn hơn hoặc bằng.

2.4 GIẢI MÃ NGÔN NGỮ MÁY

Để hiểu rõ các lệnh mã máy thực hiện lệnh hợp ngữ nào, bảng 2.4 trình bày tập các lệnh dựa trên trường mã lệnh *op* gồm 6 bit (từ bit 26 → bit 31). Bảng này gồm có các dòng được biểu diễn 3 bit (bit 29 → bit 31) và các cột biểu diễn 3 bit (bit 26 → 28). Trong bảng 2.5 trình bày giá trị nhị phân của trường *funct* tương ứng với định dạng R-format

Bảng 2.4: mã lệnh *op* của các lệnh MIPS

op(31:26)								
28-26	0(000)	1(001)	2(010)	3(011)	4(100)	5(101)	6(110)	7(111)
31-29								
0(000)	R-format	Bltz/gez	jump	jump & link	branch eq	branch ne	blez	bgtz
1(001)	add immediate	addiu	set less than imm.	set less than imm. unsigned	andi	ori	xori	load upper immediate
2(010)	TLB	FlPt						
3(011)								
4(100)	load byte	load half	lwl	load word	load byte unsigned	load half unsigned	lwr	
5(101)	store byte	store half	swl	store word			swr	
6(110)	load linked word	lwcl						
7(111)	store cond. word	swcl						

Bảng 2.5: các giá trị của trường funct với R-format

op(31:26)=000000 (R-format), funct(5:0)								
2-0 5-3	0(000)	1(001)	2(010)	3(011)	4(100)	5(101)	6(110)	7(111)
0(000)	shift left logical		shift right logical	sra	sllv		srlv	srav
1(001)	jump register	jalr			syscall	break		
2(010)	mfhi	mthi	mflo	mtlo				
3(011)	mult	multu	div	divu				
4(100)	add	addu	subtract	subu	and	or	xor	not or (nor)
5(101)			set l.t.	set l.t. unsigned				
6(110)								
7(111)								

Giả sử có một lệnh mã máy như sau:

00af8020hex

Lệnh hợp ngữ tương ứng là lệnh gì?

Để giải mã lệnh mã máy trên, đầu tiên chuyển các số hex sang nhị phân để xác định trường *op* có giá trị bao nhiêu.

```
(Bits:31 28 26                                     5  2 0)
      0000 0000 1010 1111 1000 0000 0010 0000
```

trường *op* (6 bit) với các bit 29 → 31 có giá trị 000 và các bit 26 → 28 cũng có giá trị là 000. Do đó đây là một lệnh dạng R-format (xem bảng 2.2). Tiếp theo xác định giá trị của các trường còn lại của định dạng này.

op	rs	rt	rd	shamt	funct
000000	00101	01111	10000	00000	100000

dựa vào trường *funct* với các bit 3 → 5 là 100 và các bit 0 → 2 là 000 nên lệnh này là lệnh *add* (xem bảng 2.3). Trường *rs* có giá trị 5 (\$a1), trường *rt* có giá trị 15 (\$t7) và trường *rd* có giá trị 16 (\$s0) còn trường *shamt* không sử dụng (được thiết lập giá trị 0). Vậy lệnh hợp ngữ tương ứng là: **ADD \$s0,\$a1,\$t7**

2.5 TÓM TẮT

Trong chương này giới thiệu về tập lệnh vi xử lý MIPS. Vi xử lý này có 32 thanh ghi (từ \$0 - \$31) và các thanh ghi này đều có cùng độ dài 32 bit. Ba kiểu toán hạng được sử dụng chủ yếu là : toán hạng thanh ghi, toán hạng bộ nhớ và toán hạng trực tiếp. Tập lệnh của vi xử lý MIPS có thể phân chia thành các loại: lệnh số học – luận lý, lệnh truyền dữ liệu và lệnh rẽ nhánh. Ba định dạng lệnh cơ bản của MIPS như: R-format, I-format và J-format. Định dạng R-format được sử dụng cho các lệnh số học – luận lý và các lệnh dịch (*shift*). Định dạng I-format dùng cho các lệnh truyền dữ liệu giữa bộ nhớ và thanh ghi (lệnh *load* và *store*). Còn định dạng lệnh J-format dùng cho các lệnh rẽ nhánh có điều kiện và không điều kiện. Cuối cùng trình bày cách chuyển từ một lệnh vi xử lý MIPS sang mã máy.

CÂU HỎI VÀ BÀI TẬP CHƯƠNG 2

1. Tập lệnh vi xử lý MIPS chia thành mấy loại?
2. Toán hạng thanh ghi là gì?
3. Toán hạng bộ nhớ là gì?
4. Định dạng lệnh của vi xử lý MIPS phân thành mấy loại?
5. Lệnh luận lý hoạt động như thế nào?
6. Lệnh rẽ nhánh gồm những lệnh nào?
7. Viết các lệnh hợp ngữ MIPS để tính biểu thức $f = (g+h) - (i+j)$, giả sử f, g, h, i, j được lưu lần lượt trong các thanh ghi \$s0, \$s1, \$s2, \$s3, \$s4.
8. Giả sử hai thanh ghi \$s1 = 0xABCD1234 và \$s2 = 0xFFFF0000. Hãy cho biết giá trị thanh ghi \$s0 khi thực hiện các lệnh sau:
and \$s0, \$s1, \$s2
or \$s0, \$s1, \$s2
xor \$s0, \$s1, \$s2
nor \$s0, \$s1, \$s2
9. Giả sử thanh ghi \$s2 = 0xABCD1234, hãy cho biết giá trị của thanh ghi \$s1 khi thực hiện các lệnh:
sll \$s1, \$s2, 8
sra \$s1, \$s2, 4
10. Viết các lệnh hợp ngữ MIPS để thực hiện gán giá trị cho thanh ghi là \$s1 = 0xAC5165D9 (32 bit).
11. Viết các lệnh hợp ngữ MIPS để thể hiện phát biểu IF sau:
if (a = b)
 c = d + e
else
 c = d - e
giả sử a, b, c, d, e lần lượt được lưu tương ứng trong các thanh ghi \$s0, \$s1, \$s2, \$s3, \$s4.
12. Viết các lệnh hợp ngữ MIPS để thể hiện phát biểu IF sau:
if ((\$s1 > 0) && (\$s2 < 0))
 \$s3++;
13. Viết các lệnh hợp ngữ MIPS để thể hiện phát biểu IF sau:
if ((\$s1 > \$s2) || (\$s2 > \$s3))
 \$s4 = 1;
14. Giả sử A là một mảng, mỗi phần tử của A có kích thước một từ (word – 4 byte) và địa chỉ của mảng A được lưu trong thanh ghi \$s0. Hãy viết các lệnh hợp ngữ để tính $A[1] = A[2] + 5$.
15. Giả sử A là một mảng, mỗi phần tử của A có kích thước một từ (word – 4 byte). Hãy viết các lệnh hợp ngữ để thể hiện vòng lặp WHILE như sau:
i = 0;
while (A[i] != k)
 i = i + 1;
với địa chỉ mảng A, i, k được lưu lần lượt trong các thanh ghi \$s0, \$s1, \$s2.

CHƯƠNG 3

TỔ CHỨC BỘ XỬ LÝ

Mục đích: Giới thiệu tổng quan nguyên lý hoạt động của vi xử lý MIPS. Đường dẫn dữ liệu của các loại lệnh, tổ chức của bộ tính toán và luận lý (ALU). Nguyên lý hoạt động của bộ điều khiển và trình bày các đặc điểm cơ bản của kỹ thuật ống dẫn.

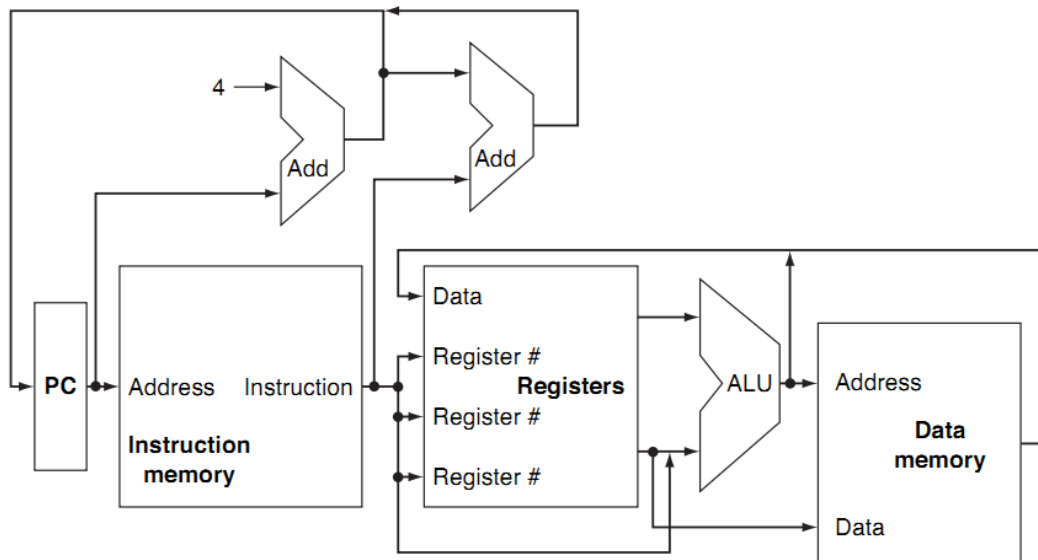
3.1 GIỚI THIỆU

Trong chương này giới thiệu các nguyên lý và kỹ thuật được tổ chức trong bộ xử lý. Cách xây dựng đường dẫn dữ liệu (*datapath*) với các tập lệnh cơ bản của MIPS. Sau đó trình bày các khái niệm cơ bản của kỹ thuật ống dẫn (*pipelining*).

Các lệnh cơ bản được xem xét bao gồm:

- Các lệnh tham chiếu bộ nhớ như: load word (lw) và store word (sw)
- Các lệnh số học và luận lý: add, sub, AND, OR và slt
- Lệnh so sánh bằng beq và lệnh nhảy j.

Sơ đồ tổ chức tổng quát của MIPS như sau:



Hình 3.1: Sơ đồ tổ chức tổng quát của MIPS¹

Các lệnh bắt đầu bằng cách dùng thanh ghi PC (*program counter*) để xác định địa chỉ của lệnh trong bộ nhớ lệnh. Sau khi lệnh được duyệt, các toán hạng của lệnh cũng được xác định bởi các trường của lệnh này. Dựa vào các toán hạng này để tính địa chỉ bộ nhớ (đối với lệnh load và store), tính kết quả của phép toán số học (đối với các lệnh số học – luận lý) hoặc phép toán so sánh (đối với lệnh nhảy). Nếu lệnh đang thực hiện là lệnh số học – luận lý, kết quả phép tính từ ALU phải được ghi vào một thanh ghi. Nếu lệnh đang thực hiện là lệnh load hoặc store, kết quả phép tính từ ALU dùng để xác định địa chỉ bộ nhớ để lưu giá trị của thanh ghi vào bộ nhớ (store) hoặc lấy giá trị từ bộ nhớ đưa vào thanh ghi (load).

¹ Các hình trong chương này được trích từ cuốn sách “*Computer organization and Design: The hardware/software interface*” (fourth edition), tác giả: David A.Patterson & John L.Hennessy.

3.2 ĐƯỜNG DẪN DỮ LIỆU

Đường dẫn dữ liệu gồm có bộ tính toán và luận lý, các mạch dịch, các thanh ghi và các đường nối kết các bộ phận trên. Nhiệm vụ của đường dẫn dữ liệu là đọc các toán hạng từ các thanh ghi, thực hiện các phép tính trên các toán hạng này trong bộ tính toán và luận lý và lưu trữ kết quả vào thanh ghi.

Đường dẫn dữ liệu của các lệnh số học – luận lý và các lệnh bộ nhớ thì tương đối giống nhau. Những điểm khác biệt chính giữa chúng là:

- Các lệnh số học – luận lý dùng ALU với hai toán hạng nhập vào là hai thanh ghi. Trong khi các lệnh bộ nhớ dùng ALU để tính địa chỉ với giá trị nhập vào thứ nhất là một thanh ghi và giá trị nhập vào thứ hai là giá trị độ dời (*offset*) 16 bit cần được mở rộng dấu sang 32 bit.
- Kết quả của ALU lưu vào thanh ghi đích (đối với lệnh số học – luận lý) nhưng kết quả được lưu vào thanh ghi đích được lấy từ bộ nhớ (đối với lệnh load).

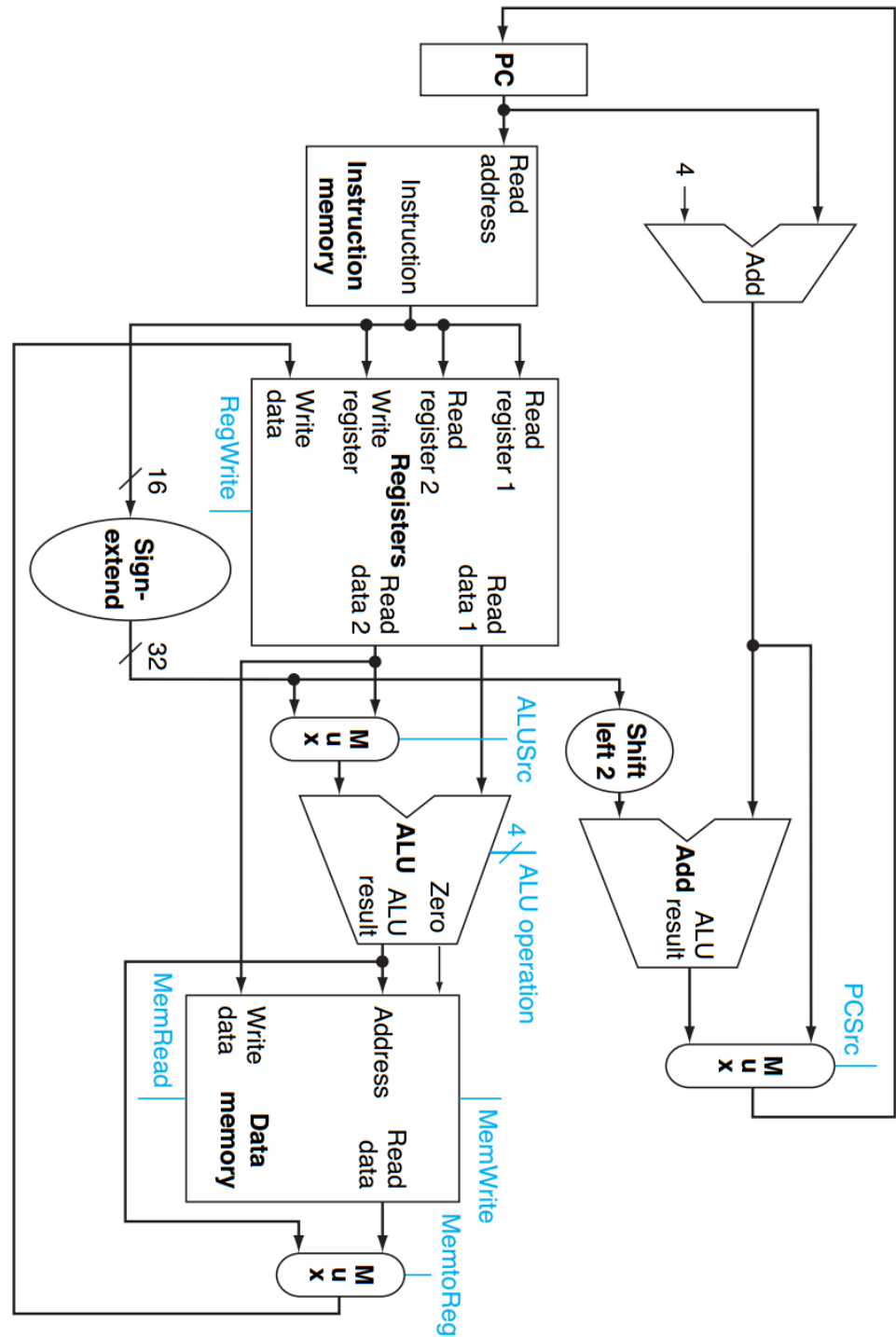
Đường dẫn dữ liệu của các lệnh số học – luận lý, các lệnh bộ nhớ và lệnh so sánh bằng được thể hiện ở hình 3.2. Để thực hiện một lệnh, đầu tiên lệnh đó sẽ được duyệt từ bộ nhớ, lệnh tiếp theo được thực hiện bằng cách tăng thanh ghi đếm chương trình (*program counter* – PC) thêm 4 bởi vì độ dài mỗi lệnh là 4 byte (trong trường hợp thực hiện tuần tự các lệnh).

Trong định dạng R-format (các lệnh số học – luận lý), nội dung hai thanh ghi được đọc, sau đó thực hiện phép toán tương ứng trên hai thanh ghi này và ghi kết quả vào thanh ghi thứ ba. Thí dụ: `add $t1, $t2, $t3` sẽ đọc nội dung của \$t2 và \$t3 thực hiện phép tính cộng trên ALU và lưu kết quả vào \$t1. Các lệnh thuộc nhóm này gồm có: `add`, `sub`, `AND`, `OR` và `sll`. Trong tổ chức ALU thì có hai đầu vào độ dài 32 bit, một đầu ra cũng có độ dài 32 bit. Ngoài ra, ALU còn có tín hiệu 1 bit thể hiện kết quả đầu ra bằng 0 và các tín hiệu điều khiển 4 bit mà sẽ được trình bày ở phần tiếp theo.

Tiếp theo, chúng ta sẽ xem xét các lệnh bộ nhớ bao gồm lệnh load và lệnh store có định dạng như sau: `lw $t1, offset_value($t2)` và `sw $t1, offset_value($t2)`. Các lệnh này tính địa chỉ bộ nhớ bằng cách cộng nội dung thanh ghi cơ sở (*base register*) trong \$t2 với độ dời (*offset*) có dấu 16 bit. Đối với lệnh store thì giá trị lưu trữ vào bộ nhớ được đọc từ thanh ghi \$t1. Trong khi đó lệnh load thì giá trị đọc từ bộ nhớ được ghi vào thanh ghi \$t1.

Ngoài ra cần có một bộ chuyển đổi mở rộng có dấu từ 16 bit của độ dời thành 32 bit tương ứng. Đối với bộ nhớ có các tín hiệu đầu vào là tín hiệu đọc (*MemRead*) dùng để đọc dữ liệu và tín hiệu ghi (*MemWrite*) để ghi dữ liệu vào bộ nhớ.

Đối với lệnh `beq` có ba toán hạng với định dạng `beq $t1, $t2, offset`, hai thanh ghi được so sánh có bằng nhau bằng cách thực hiện phép trừ trong ALU. Kết quả so sánh được xác định bằng tín hiệu ra Zero của ALU. Nếu điều kiện so sánh không thỏa mãn thì lệnh kế tiếp được thực hiện ($PC + 4$). Nếu điều kiện so sánh thỏa mãn thì độ dời 16 bit (mở rộng dấu thành 32 bit) được dịch sang trái 2 bit (tương ứng nhân với 4) được cộng với ($PC + 4$) để thực hiện nhảy đến địa chỉ đích.



Hình 3.2: Đường dẫn dữ liệu của ba loại lệnh cơ bản

Đối với lệnh jump (j) được thực hiện bằng cách thay thế 26 bit của PC bằng 26 bit của lệnh được dịch sang trái 2 bit.

3.3 TỔ CHỨC BỘ TÍNH TOÁN VÀ LUẬN LÝ (ALU)

Bộ tính toán và luận lý (ALU) của bộ xử lý MIPS xác định các phép toán thực hiện dựa vào tín hiệu điều khiển 4-bit:

ALU control lines	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	set on less than
1100	NOR

Tùy thuộc vào từng loại lệnh, ALU sẽ thực hiện một trong các hàm trên. Đối với lệnh load word và store word, ALU thực hiện phép cộng để xác định địa chỉ bộ nhớ. Đối với dạng lệnh R-type, ALU thực hiện một trong năm lệnh (AND, OR, subtract, add hoặc slt) phụ thuộc vào 6-bit thấp của trường funct (function) trong định dạng lệnh. Với lệnh beq, ALU sẽ thực hiện phép toán trừ.

Các tín hiệu điều khiển 4-bit của ALU có thể được tạo ra bằng mạch điều khiển đơn giản có tín hiệu nhập là các bit của trường funct cùng với 2-bit điều khiển đặc biệt (được gọi là ALUOp). ALUOp sẽ xác định phép toán thực hiện: giá trị 00 thực hiện phép cộng cho lệnh load và store, 01 thực hiện phép trừ cho lệnh beq, đối với giá trị 10 thì phép toán được thực hiện phụ thuộc vào trường funct của định dạng lệnh. Đối với giá trị 00 và 01 của ALUOp thì phép toán được thực hiện không phụ thuộc vào giá trị của trường funct (trong trường hợp này các giá trị của trường funct được ghi là X). Các giá trị của ALUOp được tạo ra bởi đơn vị điều khiển chính (CU – Control Unit).

Bảng 3.1: Thiết lập các bit điều khiển ALU dựa vào ALUOp và trường funct

Dạng lệnh	ALUOp	Lệnh	Mã hàm (<i>funct</i>)	Phép toán ALU	Tín hiệu điều khiển ALU (tín hiệu nhập)
LW	00	load word	××××××	add	0010
SW	00	store word	××××××	add	0010
Branch equal	01	branch equal	××××××	subtract	0110
R-type	10	add	100000	add	0010
R-type	10	subtract	100010	subtract	0110
R-type	10	AND	100100	AND	0000
R-type	10	OR	100101	OR	0001
R-type	10	set on less than	101010	set on less than	0111

3.4 BỘ ĐIỀU KHIỂN CHÍNH (*main control unit*)

Để thiết kế mạch điều khiển, chúng ta xét ba định dạng lệnh cơ bản: định dạng lệnh số học và logic (R-format) có opcode = 0, các lệnh này có ba thanh ghi toán hạng rs, rt và rd. Hai thanh ghi rs và rt là các thanh ghi nguồn, còn rd là thanh ghi đích. Phép toán thực hiện trong ALU được xác định bởi trường funct. Định dạng này

thực hiện các lệnh add, sub, AND, OR và slt. Trường shamt được dùng cho các lệnh dịch (shifts). Định dạng lệnh thứ 2 là định dạng cho lệnh load (opcode = 35_{ten}) và lệnh store (opcode = 43_{ten}). Thanh ghi rs là thanh ghi nền, nội dung thanh ghi này cộng với địa chỉ offset 16 bit để xác định được địa chỉ bộ nhớ. Đối với lệnh load, nội dung từ bộ nhớ được chép vào thanh ghi rt. Còn với lệnh store, nội dung từ thanh ghi rt được lưu vào bộ nhớ. Định dạng lệnh thứ 3 là định dạng lệnh nhảy theo điều kiện so sánh bằng (branch equal) có opcode = 4. Trong định dạng này hai thanh ghi rs và rt là các thanh ghi nguồn được dùng để thực hiện phép so sánh. Dựa vào kết quả so sánh này để quyết định: hoặc thực hiện lệnh kế tiếp (PC+4) hoặc cộng nội dung (PC+4) với địa chỉ 16 bit để nhảy đến địa chỉ đích.

Field	0	rs	rt	rd	shamt	funct
Bit positions	31:26	25:21	20:16	15:11	10:6	5:0

a. R-type instruction

Field	35 or 43	rs	rt	address
Bit positions	31:26	25:21	20:16	15:0

b. Load or store instruction

Field	4	rs	rt	address
Bit positions	31:26	25:21	20:16	15:0

c. Branch instruction

Hình 3.3: Ba định dạng lệnh R-type, load và store, branch

Từ ba định dạng trên, chúng ta rút ra một số nhận xét như sau:

- Trường opcode luôn ở vị trí bit 31:26
- Cả ba định dạng thì chỉ đọc hai thanh ghi rs và rt được định vị tại vị trí 25:21 và 20:16
 - Thanh ghi nền (base register) cho định dạng lệnh load và store thì luôn luôn ở vị trí 25:21 (rs)
 - Địa chỉ offset 16 bit cho lệnh nhảy theo điều kiện so sánh bằng, lệnh load và store thì luôn ở vị trí 15:0
 - Thanh ghi đích thì ở một trong hai vị trí: đối với lệnh load thì ở vị trí 20:16 (rt), còn đối với các lệnh R-format thì ở vị trí 15:11 (rd).

Các tín hiệu điều khiển được thể hiện như hình 3.4. Trong hình này thể hiện bảy đường điều khiển 1-bit và một đường điều khiển ALUOp 2-bit. Ngoài ra, còn có thiết bị bộ điều hợp (multiplexor – viết tắt Mux: là thiết bị có hai tín hiệu đầu vào, một tín hiệu đầu ra và một tín hiệu điều khiển. Dựa vào tín hiệu điều khiển mà tín hiệu đầu ra được chọn là một trong hai tín hiệu đầu vào) được dùng trong kết hợp trong hệ thống.

- ALUSrc: giá trị 0 xác định toán hạng thứ hai của ALU lấy từ thanh ghi thứ hai (Read data 2). Ngược lại, giá trị 1 thì toán hạng thứ hai của ALU lấy giá trị mở rộng dấu 16 bit thấp của lệnh.
- PCSrc: giá trị 0 thì PC được thay thế bởi PC+4. Ngược lại, giá trị 1 thì PC được thay thế bởi kết quả tính tổng của ALU để nhảy đến địa chỉ đích.
- MemRead: giá trị 0 không ảnh hưởng. Giá trị 1 thì dữ liệu được đọc từ bộ nhớ.
- MemWrite: giá trị 0 không ảnh hưởng. Giá trị 1 thì dữ liệu được ghi vào bộ nhớ.
- MemtoReg: giá trị 0 xác định dữ liệu để ghi vào thanh ghi được lấy từ kết quả của ALU. Ngược lại, giá trị 1 xác định dữ liệu để ghi vào thanh ghi được đọc từ bộ nhớ.

Trong hình 3.5 thể hiện giá trị của các đường điều khiển của các dạng lệnh khác nhau. Dòng đầu tiên thể hiện của định dạng lệnh R-format (add, sub, AND, OR và slt). Đối với các lệnh này thanh ghi nguồn luôn là rs và rt, còn thanh ghi đích là rd. Hơn nữa dạng lệnh này ghi kết quả vào thanh ghi đích (RegWrite = 1) nhưng không đọc bộ nhớ (MemRead = 0) cũng không ghi vào bộ nhớ (MemWrite = 0). Khi tín hiệu điều khiển Branch = 0 thì giá trị PC được thay thế bởi PC + 4, ngược lại giá trị PC được thay bởi địa chỉ đích đến nếu tín hiệu Zero của ALU được thiết lập bằng 1. Trường ALUOp được thiết lập 10 để xác định định dạng lệnh này (R-format). Dòng thứ 2 và thứ 3 tương ứng với thiết lập của lw và sw. Giá trị của MemRead và MemWrite được thiết lập để truy cập đến bộ nhớ. Các trường ALUSrc và ALUOp được thiết lập để tính toán địa chỉ. Đối với lệnh lw thì các tín hiệu RegDst và RegWrite được thiết lập để kết quả được lưu vào thanh ghi rt. Lệnh beq cũng tương tự như định dạng R-format, giá trị của các thanh ghi rs và rt được gửi tới ALU. Giá trị ALUOp = 01 để ALU thực hiện phép trừ để kiểm tra hai thanh ghi có bằng nhau không. Chú ý khi RegWrite = 0 thì giá trị của MemtoReg (X) và RegDst (X) là không xác định.

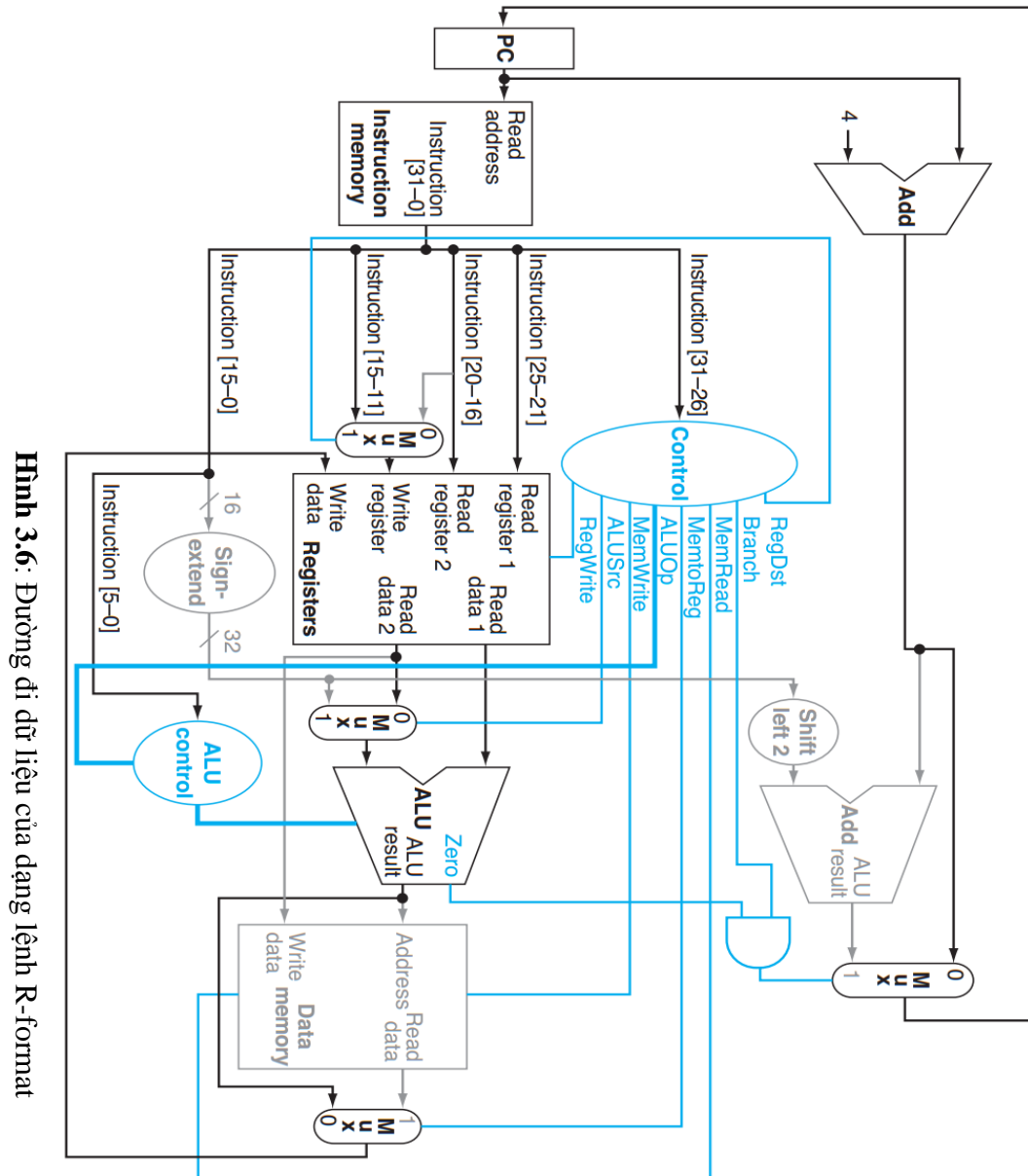
Instruction	RegDst	ALUSrc	Memto-Reg	Reg-Write	Mem-Read	Mem-Write	Branch	ALUOp1	ALUOp0
R-format	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0
sw	X	1	X	0	0	1	0	0	0
beq	X	0	X	0	0	0	1	0	1

Hình 3.5: Các tín hiệu điều khiển của các dạng lệnh

Hình 3.6 thể hiện đường đi dữ liệu cho dạng lệnh R-format, ví dụ lệnh add \$t1,\$t2,\$t3. Các tín hiệu hoạt động được vẽ nổi lên còn các tín hiệu không hoạt động được mờ đi. Lệnh này được thực hiện theo thứ tự như sau:

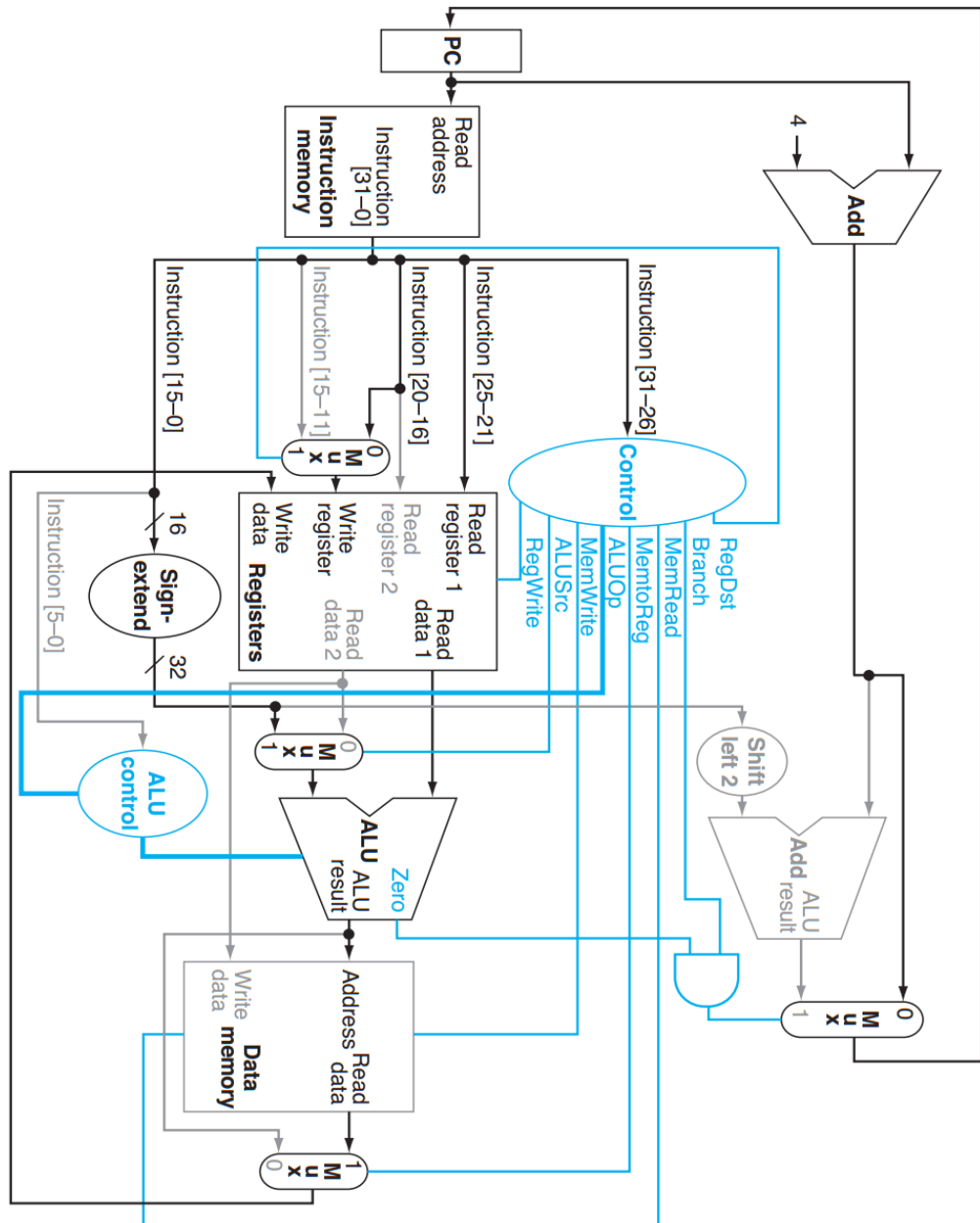
- Lệnh được duyệt, sau đó giá trị PC tăng để chỉ đến lệnh tiếp theo
- Hai thanh ghi \$t2, \$t3 được đọc, bộ phận điều khiển (control unit - CU) sẽ thiết lập các tín hiệu điều khiển tương ứng.

- ALU thực hiện phép toán tương ứng được xác định bởi các bit 5:0 trên các giá trị hai thanh ghi đọc được.
- Kết quả của phép toán ALU được ghi vào thanh ghi đích được xác định bởi các bit 15:11 (trong trường hợp này là thanh ghi \$t1).



Đối với lệnh load và store được thực hiện tương tự như R-format. Hình 3.7 thể hiện đường đi dữ liệu của dạng lệnh load, xét lệnh: lw \$t1, offset(\$t2) thực hiện theo các bước sau:

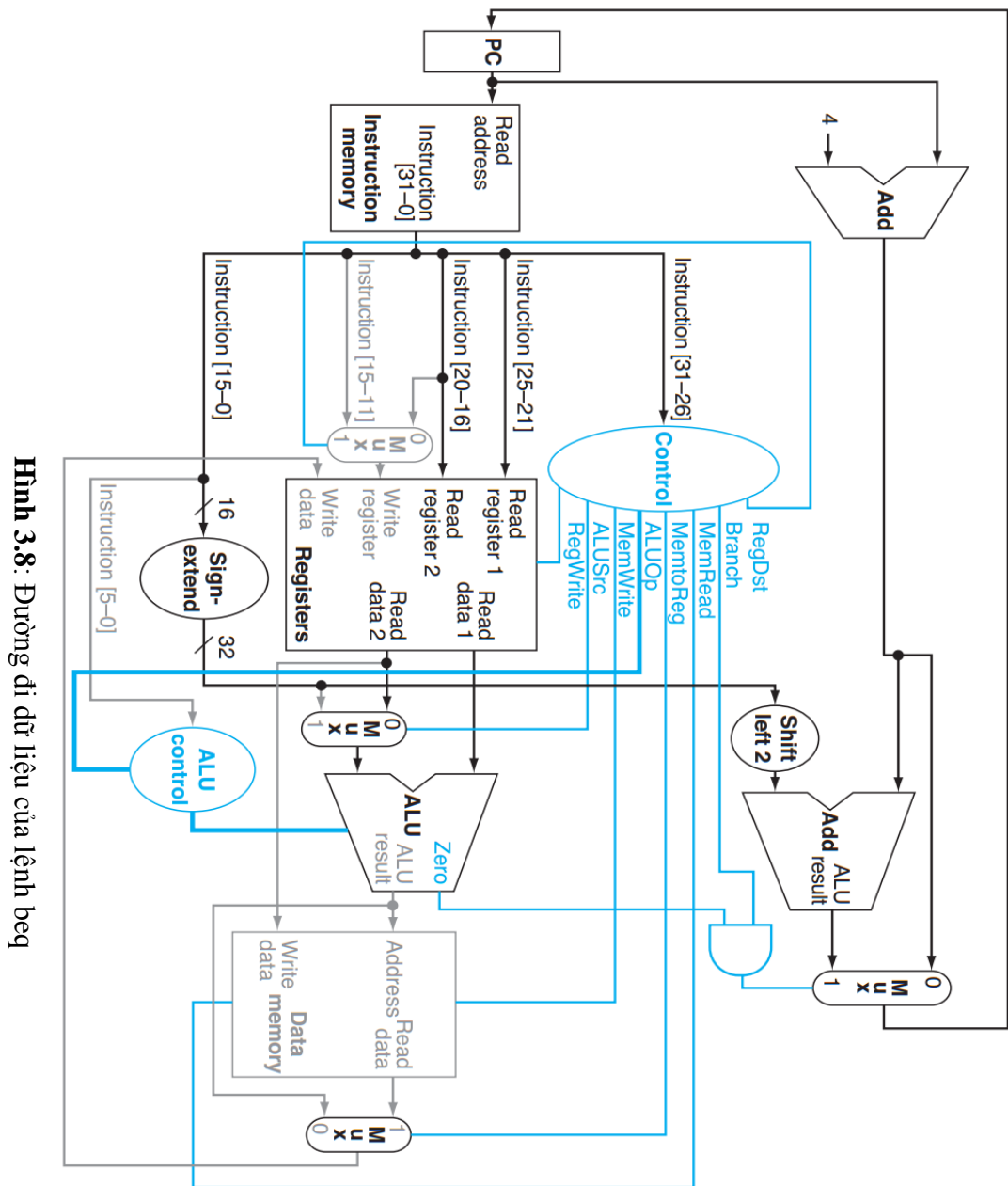
- Lệnh được duyệt từ bộ nhớ, giá trị PC tăng để chỉ đến lệnh kế tiếp.
- Đọc giá trị thanh ghi \$t2.
- Giá trị offset (16 bit) được chuyển sang 32 bit tương ứng. Sau đó ALU tính tổng của giá trị này với giá trị thanh ghi \$t2.
- Giá trị tổng của ALU dùng để xác định địa chỉ bộ nhớ.
- Dữ liệu từ bộ nhớ được đọc và ghi vào thanh ghi đích \$t1 được xác định bởi các bit 20:16.



Hình 3.7: Đường đi dữ liệu của lệnh load

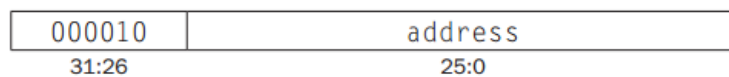
Trong trường hợp định dạng lệnh nhảy có điều kiện như beq cũng được thực hiện tương tự dạng R-format nhưng giá trị đầu ra của ALU được dùng để xác định PC được thay thế bởi PC+4 hoặc nhảy đến địa chỉ đích. Ví dụ xét lệnh: beq \$t1,\$t2,offset. Hình 3.8 sẽ thể hiện các bước để thực thi lệnh này:

- Lệnh được duyệt từ bộ nhớ, giá trị PC tăng thêm 4.
- Đọc giá trị hai thanh ghi \$t1 và \$t2.
- ALU thực hiện phép trừ với hai thanh ghi này. Giá trị offset (16 bit) được chuyển sang 32 bit tương ứng và dịch sang trái 2 bit rồi cộng với PC+4 để xác định địa chỉ đích.
- Kết quả của tín hiệu Zero của ALU sẽ quyết định kết quả được đưa vào PC.

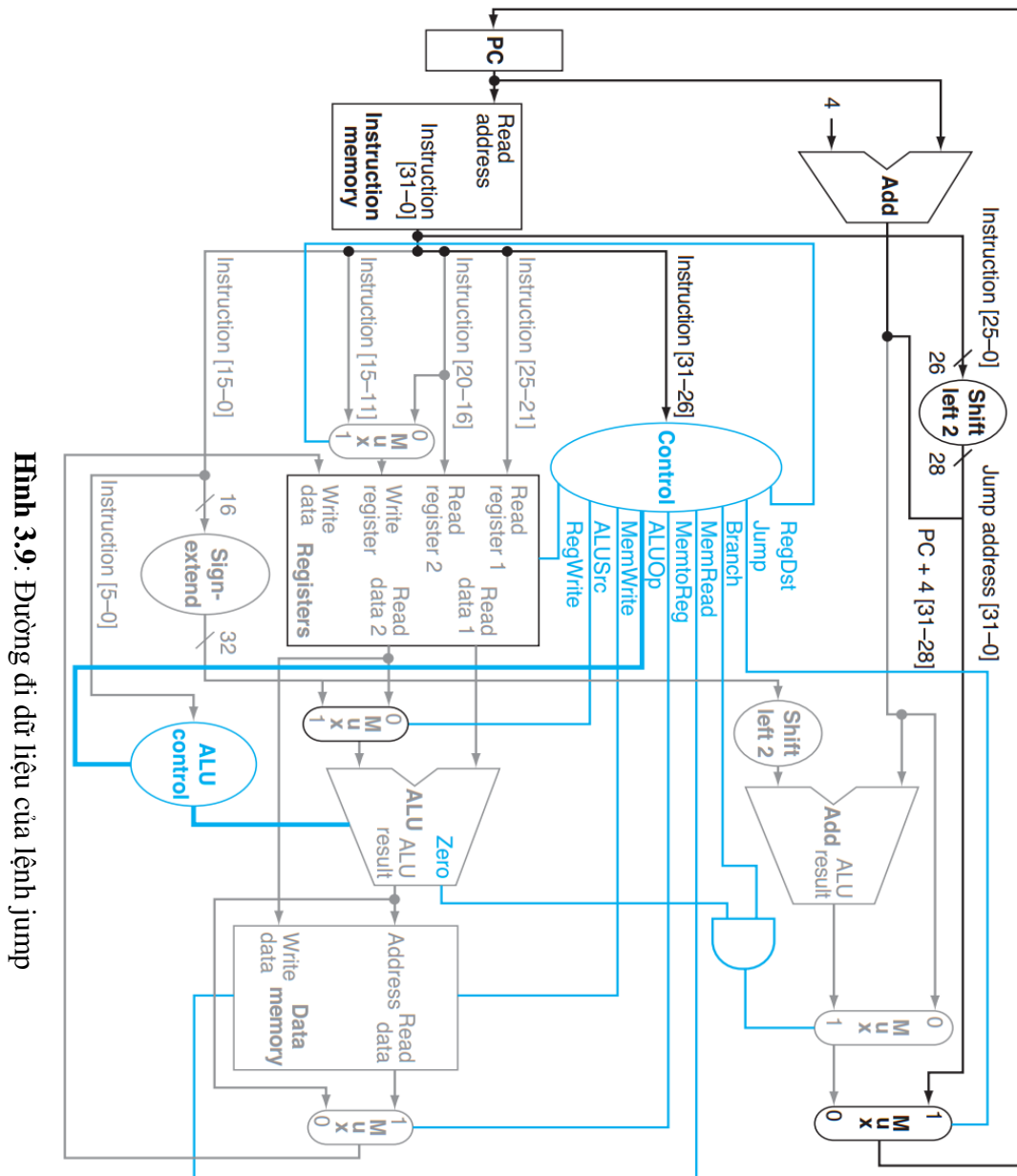


Hình 3.8: Đường đi dữ liệu của lệnh beq

Cuối cùng đối với lệnh nhảy không điều kiện j address, lệnh này có định dạng (opcode=2) như sau:



Hình 3.9 thể hiện đường đi dữ liệu của lệnh này. Một multiplexor được bổ sung vào hệ thống, multiplexor này được điều khiển bởi tín hiệu jump. Giá trị mới được ghi vào PC được xác định như sau: địa chỉ 26 bit của lệnh jump được dịch sang trái 2 bit rồi ghép nối với 4 bit cao của PC+4 để được địa chỉ đích 32 bit cần nhảy đến.



Hình 3.9: Đường đi dữ liệu của lệnh jump

3.5 KỸ THUẬT ỚNG DẪN (PIPELINE)

3.5.1 Tổng quan

Kỹ thuật ống dẫn là kỹ thuật mà nhiều lệnh được thực hiện theo dạng nạp chồng (overlap). Kỹ thuật này được sử dụng phổ biến trong các kiến trúc CPU hiện nay.

Như các phần trên đã trình bày, một lệnh MIPS được thực hiện trong 5 bước:

1. Duyệt lệnh từ bộ nhớ
2. Giải mã lệnh, đồng thời đọc các thanh ghi
3. Thực thi lệnh hoặc tính địa chỉ
4. Truy cập toán hạng trong bộ nhớ
5. Ghi kết quả vào thanh ghi

Để đơn giản trong kỹ thuật ống dẫn, chúng ta chỉ xét đến các lệnh: load word (lw), store word (sw), add (add), subtract (sub), AND (and), OR (or), set less than (slt) và beq.

Giả sử thời gian thực hiện từng thành phần như sau: thao tác truy cập bộ nhớ cần 200 ps, các phép toán ALU cần 200 ps và 100 ps để đọc hoặc ghi vào thanh ghi. Thời gian thực hiện mỗi lệnh được thể hiện như bảng 3.2. Trong hình này thì lệnh lw có thời gian thực hiện lớn nhất (800 ps).

Bảng 3.2: Tổng thời gian thực hiện của mỗi lệnh

Lệnh	Duyệt lệnh	Đọc thanh ghi	Thao tác ALU	Truy cập bộ nhớ	Ghi thanh ghi	Tổng
Load word (lw)	200 ps	100 ps	200 ps	200 ps	100 ps	800ps
Store word (sw)	200 ps	100 ps	200 ps	200 ps		700ps
R-format (add, sub,AND,OR)	200 ps	100 ps	200 ps		100 ps	600ps
Branch (beq)	200 ps	100 ps	200 ps			500ps

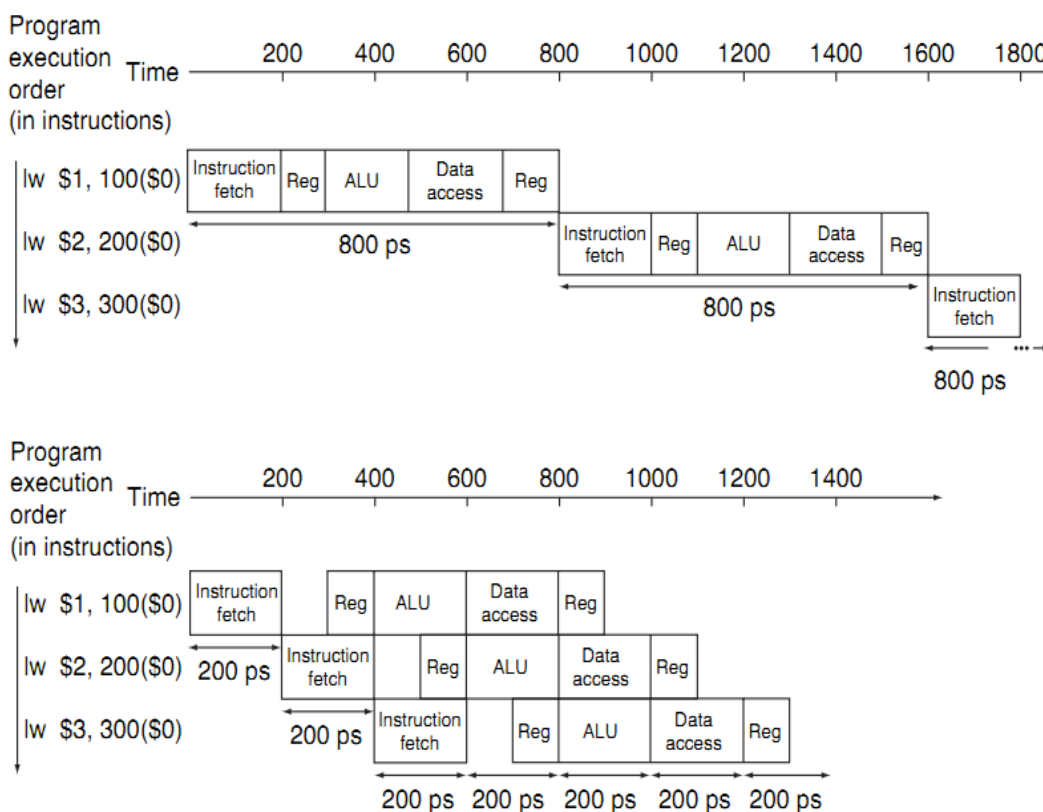
Giả sử chúng ta xét ví dụ với ba lệnh như sau:

LW \$1,100(\$0)

LW \$2,200(\$0)

LW \$3,300(\$0)

Các lệnh này được thực hiện theo hai cách: tuần tự và theo kỹ thuật ống dẫn. Hình 3.10 thể hiện so sánh giữa hai cách này. Ở phần trên của hình cho thấy ba lệnh được thực hiện tuần tự có tổng thời gian là $800 \times 3 = 2400$ ps, thời gian giữa mỗi lệnh là 800 ps. Trong khi ở phần dưới của hình cho thấy nếu ba lệnh này được thực hiện trong kỹ thuật ống dẫn thì chỉ cần 1400 ps, thời gian giữa mỗi lệnh chỉ là 200 ps.



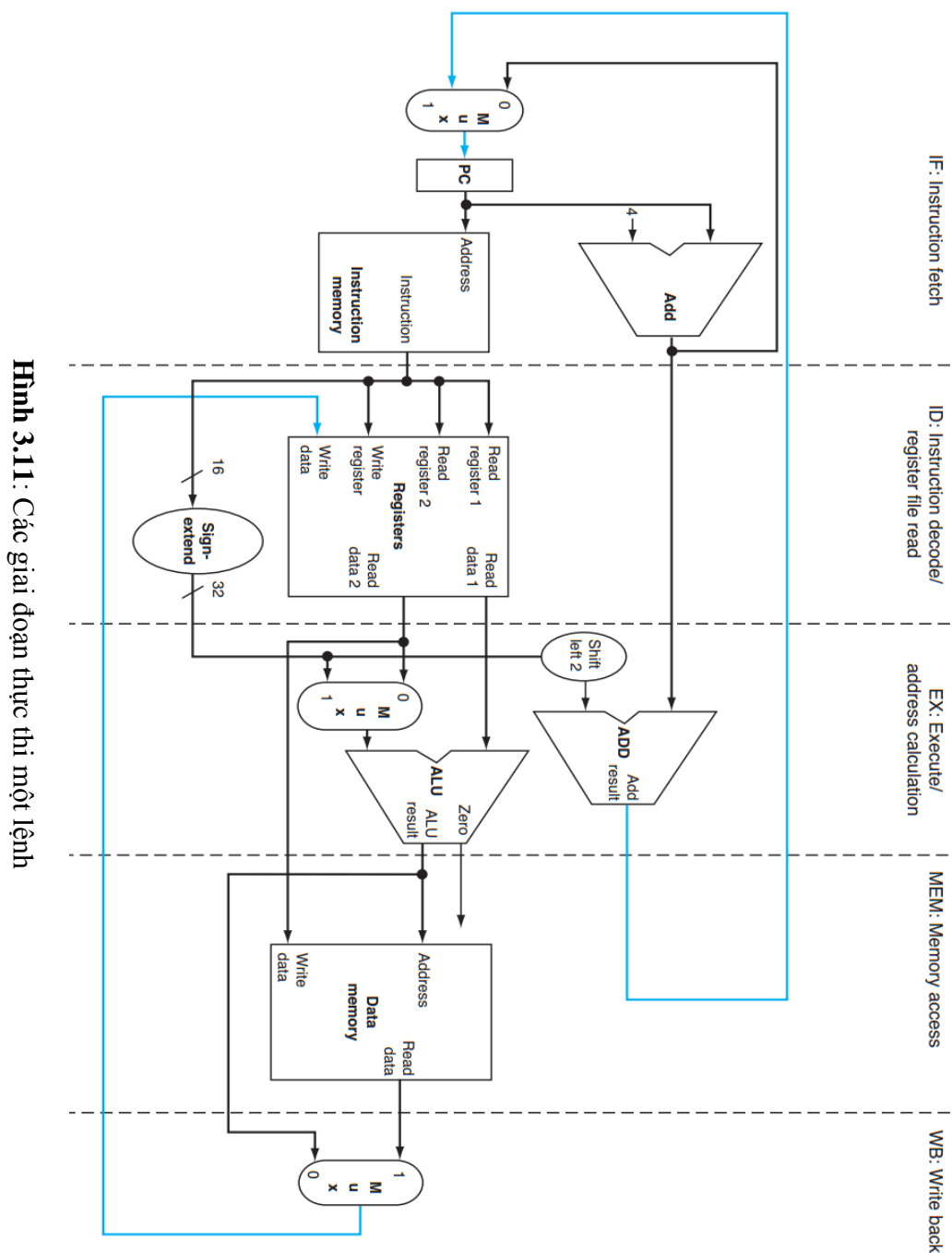
Hình 3.10: Thực hiện tuần tự so với kỹ thuật ống dẫn

3.5.2 Đường đi dữ liệu của kỹ thuật ống dẫn (*pipelined datapath*)

Một lệnh được thực hiện bao gồm 5 giai đoạn:

- Duyệt lệnh (IF)
- Giải mã lệnh (ID) và đọc các thanh ghi
- Thực thi lệnh (EX)
- Truy cập bộ nhớ (MEM)
- Ghi kết quả trở lại thanh ghi (WB)

Các giai đoạn này được thể hiện như hình 3.11 theo chiều từ trái sang phải. Ngoài trừ ở giai đoạn WB giá trị được ghi vào thanh ghi ngược trở lại giai đoạn ID và thao tác cập nhật giá trị PC đến giá trị kế tiếp hoặc địa chỉ của lệnh nhảy ở giai đoạn MEM.



Hình 3.11: Các giai đoạn thực thi một lệnh

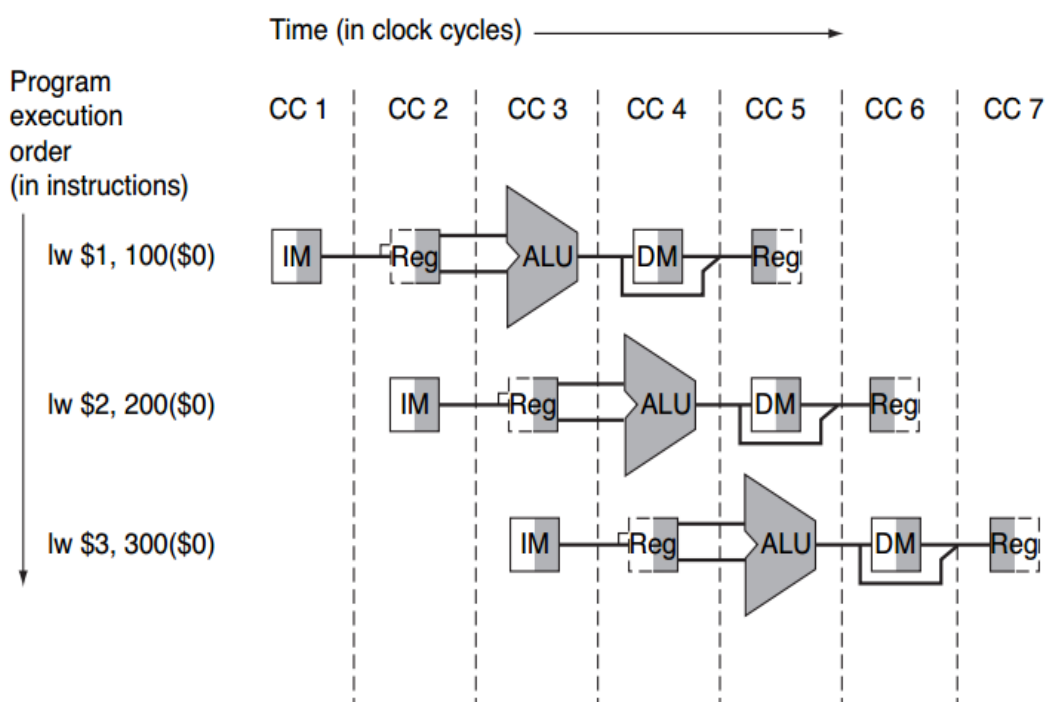
Giả sử chúng ta có ba lệnh được thực thi theo kỹ thuật ống dẫn:

lw \$1,100(\$0)

lw \$2,200(\$0)

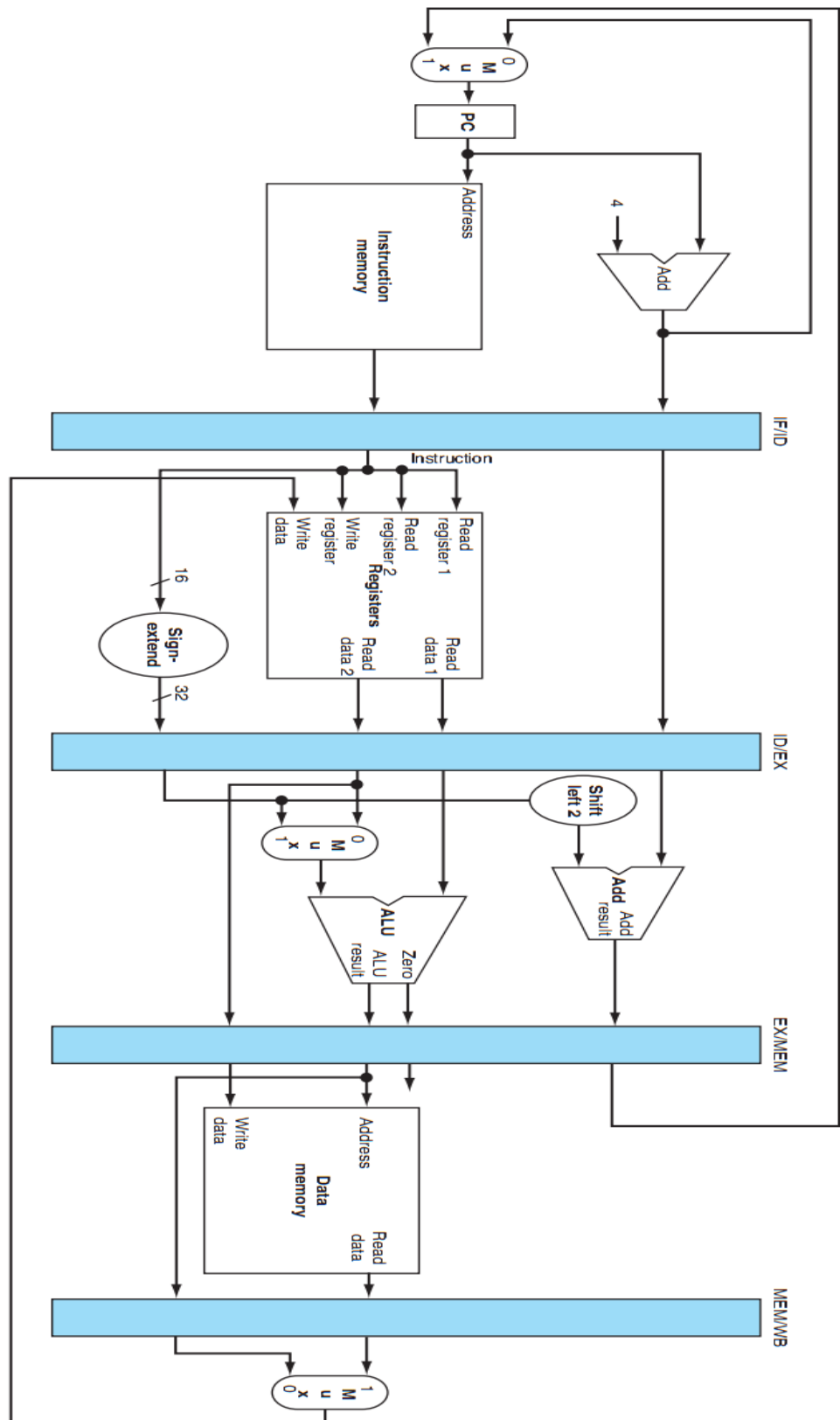
lw \$3,300(\$0)

Hình 3.12 thể hiện các lệnh này trong kỹ thuật ống dẫn. Trong hình này các giai đoạn của mỗi lệnh được thể hiện bằng tên tương ứng: IM (instruction memory), Reg (Register), ALU (Algorithm Logic Unit) và DM (Data Memory). Với quy ước phân nửa bên trái thể hiện quá trình ghi, trong khi phân nửa bên phải thể hiện quá trình đọc. Giai đoạn nào sử dụng quá trình ghi hay đọc thì được tô đậm tương ứng, không sử dụng thì bỏ trống. Như giai đoạn thực thi EX thì biểu tượng Reg được tô đen bên phải, bỏ trống bên trái thể hiện ở giai đoạn này có quá trình đọc thanh ghi nhưng không có quá trình ghi thanh ghi.



Hình 3.12: Ba lệnh được thực thi trong kỹ thuật ống dẫn

Để lưu lại giá trị ở từng giai đoạn của mỗi lệnh, trong kỹ thuật ống dẫn bổ sung thêm các thanh ghi nằm giữa các giai đoạn như hình 3.13 (các thanh ghi này còn được gọi là thanh ghi ống dẫn – pipelined register). Các thanh ghi này được đặt tên bằng cách kết hợp tên giữa hai giai đoạn kề. Ví dụ thanh ghi ống dẫn giữa giai đoạn IF và ID được đặt tên là IF/ID. Thanh ghi IF/ID có độ dài 64 bit (32 bit lưu lệnh được duyệt từ bộ nhớ và 32 bit lưu địa chỉ PC kế tiếp), các thanh ghi ống dẫn còn lại ID/EX, EX/MEM và MEM/WB có độ dài tương ứng là 128 bit, 97 bit và 64 bit.

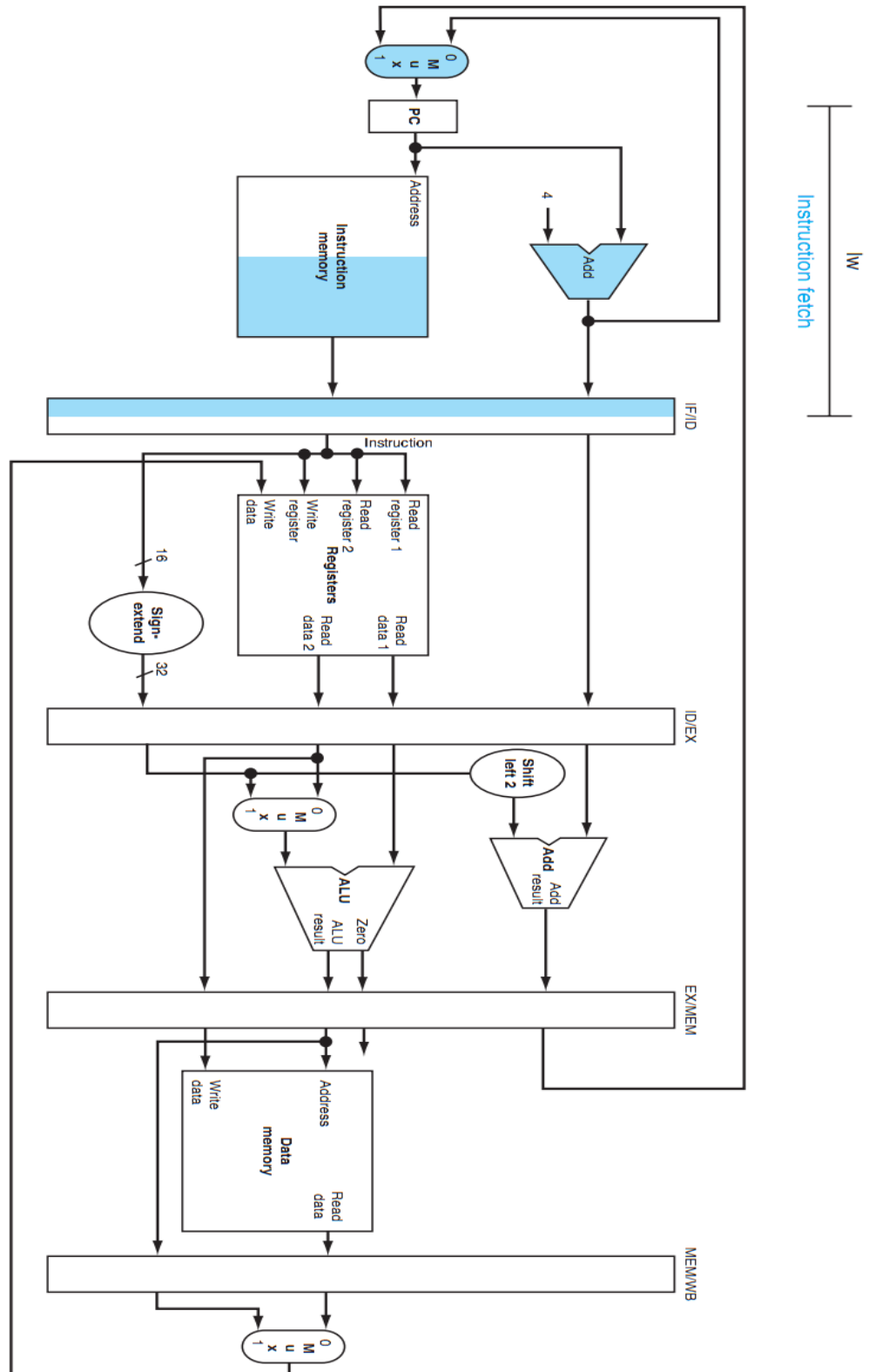


Hình 3.13: Các thành ghi ống dẫn

Để hiểu rõ cách thức hoạt động trong kỹ thuật ống dẫn, chúng ta xét lệnh lw được thực hiện trong kỹ thuật ống dẫn thông qua năm giai đoạn như sau:

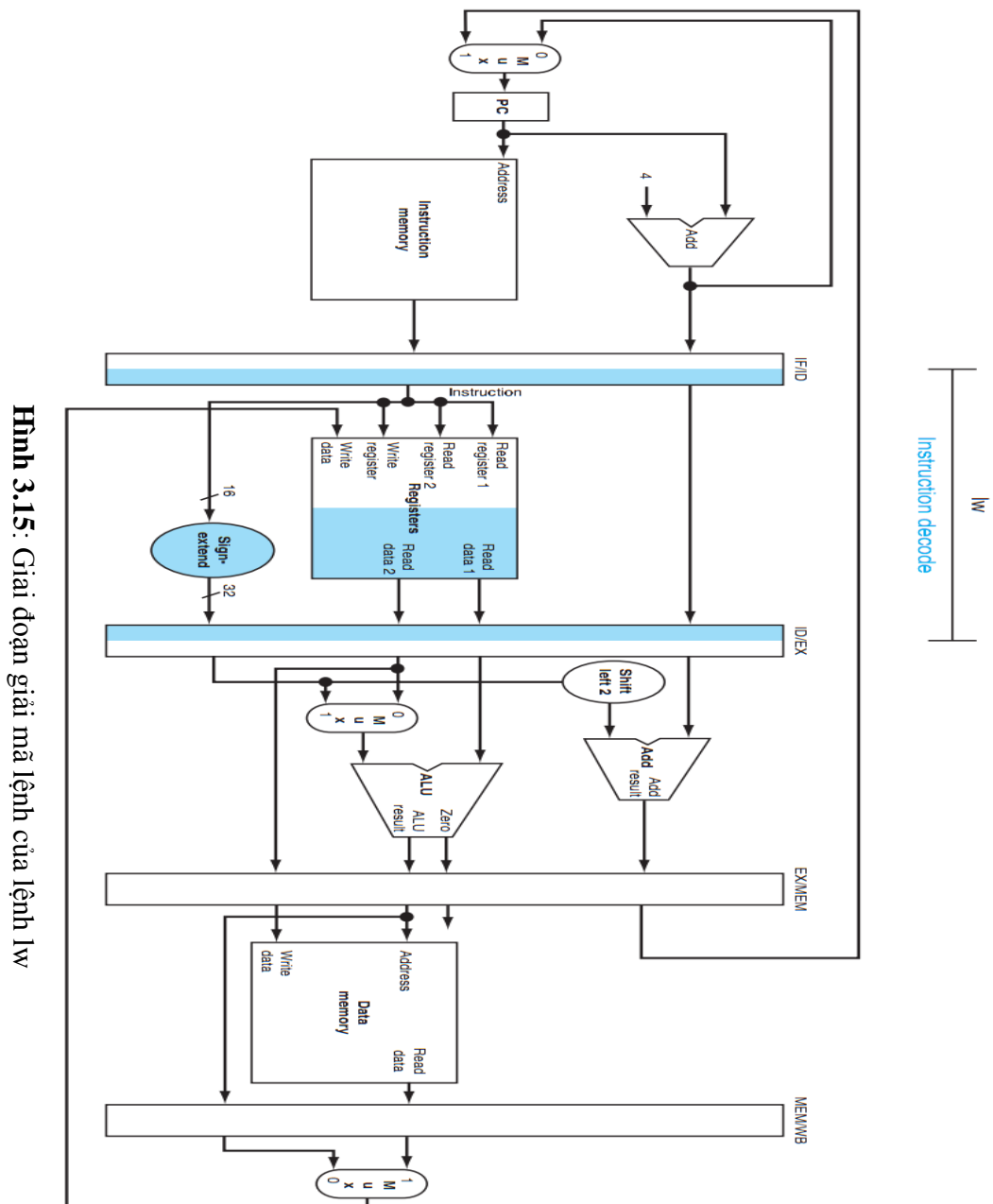
- Duyệt lệnh: hình 3.14 thể hiện lệnh được đọc từ bộ nhớ dùng địa chỉ chứa trong PC, sau đó lệnh đọc được được lưu vào thành ghi ống dẫn IF/ID. Địa chỉ PC tăng lên 4 và được lưu trở lại PC để chỉ đến lệnh kế tiếp. Đồng thời kết quả này cũng

được lưu vào thanh ghi ống dẫn IF/ID để xử lý trong trường hợp một lệnh nhảy tương ứng. Máy tính không thể biết được loại lệnh sẽ được duyệt tiếp theo nên phải lưu lại các thông tin cần thiết vào thanh ghi ống dẫn để xử lý cho bất kỳ loại lệnh nào. Trong giai đoạn này lệnh được đọc từ bộ nhớ lệnh nên phân nửa cuối của bộ nhớ này được tô đậm. Đối với thanh ghi ống dẫn IF/ID được ghi dữ liệu vào nên phân nửa đầu của thanh ghi ống dẫn này được tô đậm.

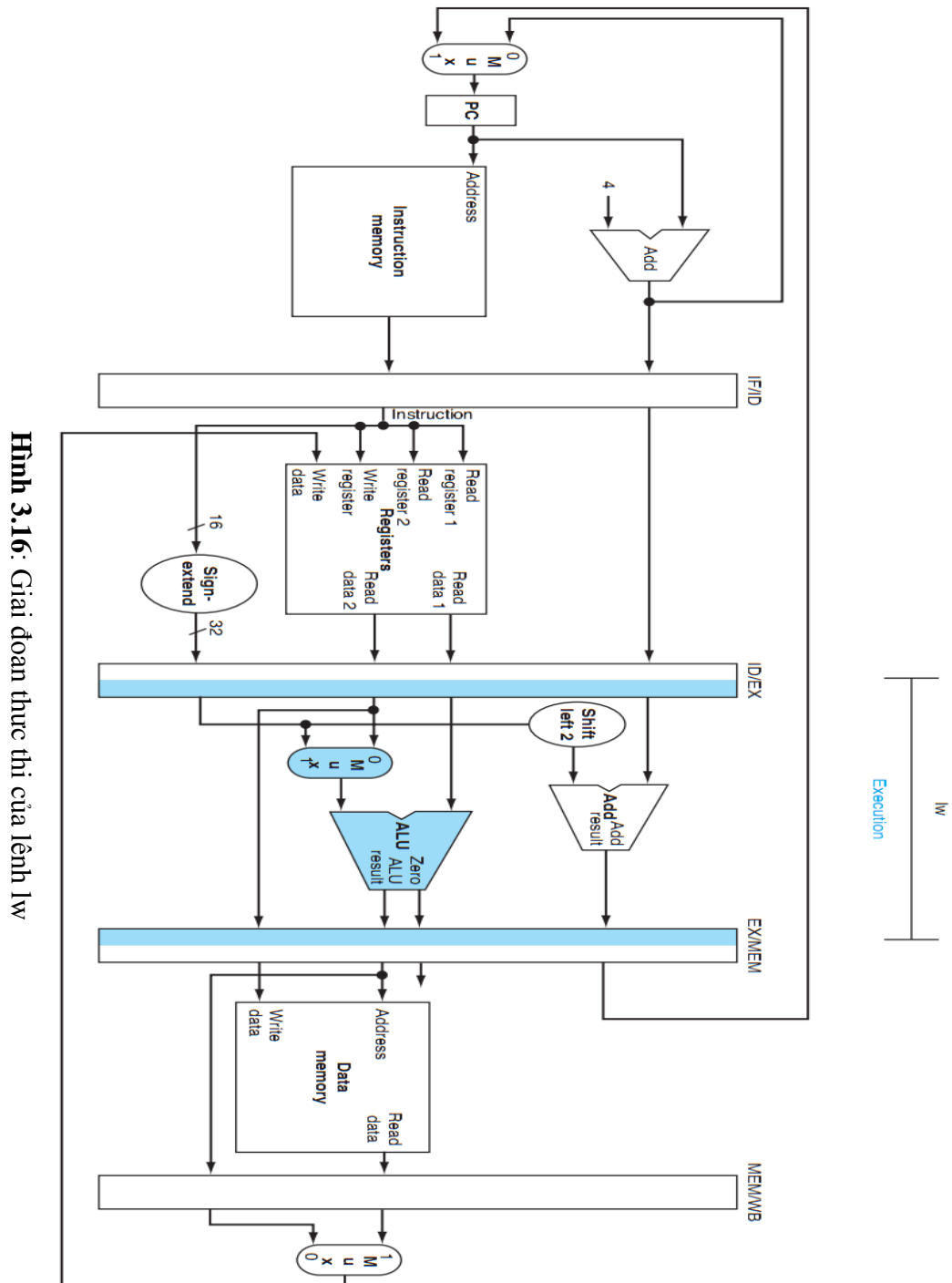


Hình 3.14: Giai đoạn duyệt lệnh của lệnh lw

– Giải mã lệnh và đọc các thanh ghi: Trong hình 3.15 địa chỉ offset 16 bit được đọc từ IF/ID và được mở rộng thành 32 bit tương ứng. Đồng thời xác định các thanh ghi để đọc dữ liệu. Dữ liệu được đọc từ thanh ghi cùng với địa chỉ offset 32 bit và giá trị PC được ghi vào thanh ghi ống dẫn ID/EX. Mặc dù lệnh lw chỉ đọc một thanh ghi nguồn (Read data 1) ở giai đoạn này, bộ xử lý không biết được loại lệnh đang được giải mã. Vì thế, giá trị hằng 16 bit (được mở rộng thành 32 bit) và đọc cả hai thanh ghi được lưu vào thanh ghi ống dẫn ID/EX. Chúng ta không cần cả ba toán hạng, nhưng để đơn giản trong điều khiển thì giữ lại cả ba thành phần này. Trong giai đoạn này lệnh được đọc từ thanh ghi ống dẫn IF/ID nên phân nửa cuối của thanh ghi này được tô đậm. Đối với thanh ghi ống dẫn ID/EX được ghi dữ liệu vào nên phân nửa đầu của thanh ghi ống dẫn này được tô đậm. Trong khi đó thanh ghi nguồn (Read data 1) trong tập thanh ghi được đọc nên phân nửa cuối của tập thanh ghi này được tô đậm.

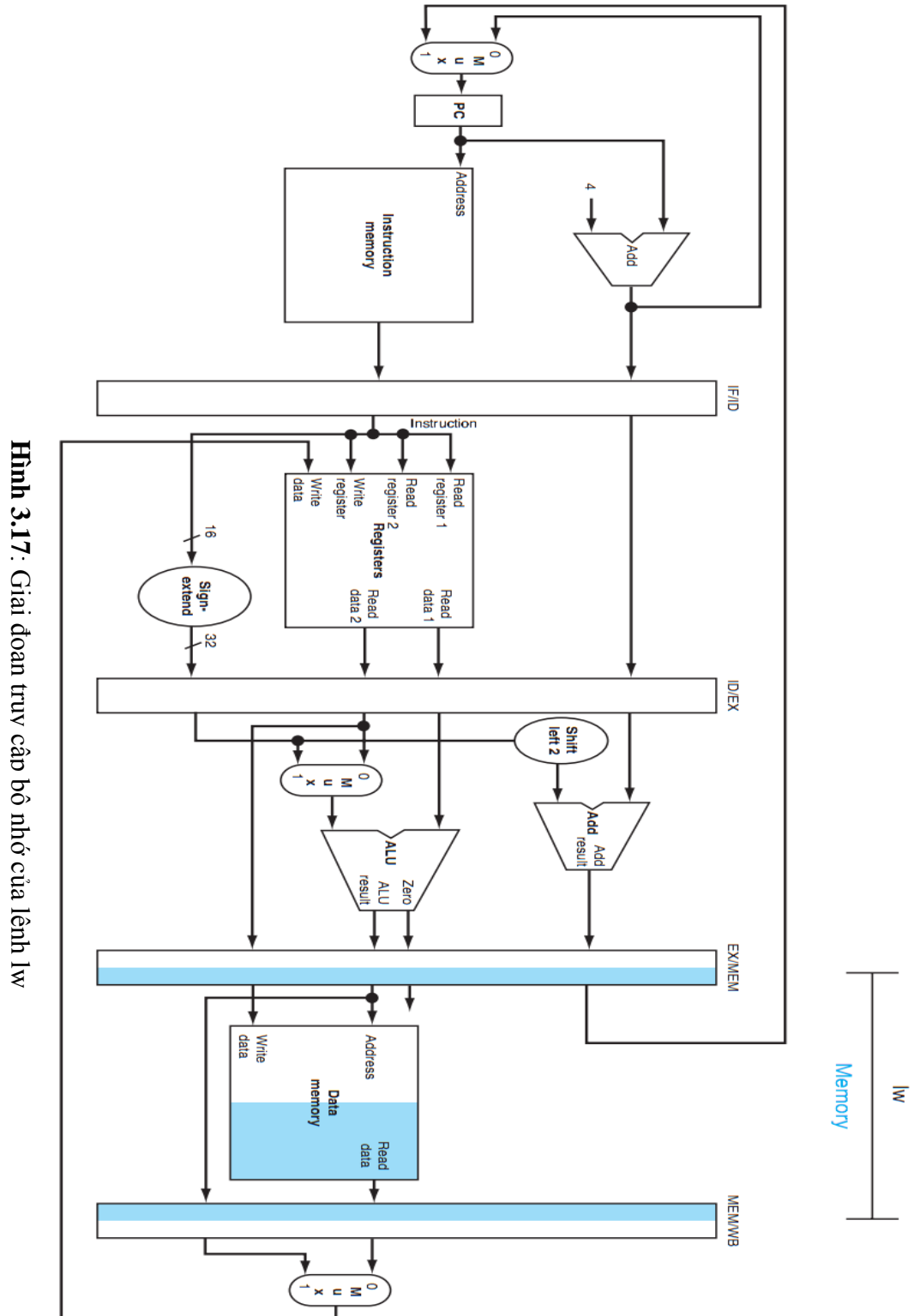


– Thực thi hoặc tính địa chỉ: hình 3.16 thể hiện giai đoạn ba trong kỹ thuật ống dẫn của lệnh lw. Lệnh này đọc nội dung thanh ghi (register 1) để đưa vào thành phần đầu vào thứ nhất của ALU. Còn thành phần đầu vào thứ hai của ALU được lấy từ giá trị địa chỉ offset 32 bit từ thanh ghi ống dẫn ID/EX (trường hợp này giá trị đầu vào của bộ điều hợp (Mux) có giá trị 1). Sau đó ALU thực hiện phép toán cộng để xác định địa chỉ trong bộ nhớ dữ liệu cần truy cập. Kết quả của phép toán cộng trong ALU được ghi vào thanh ghi ống dẫn EX/MEM. Trong giai đoạn này thanh ghi ống dẫn ID/EX được đọc dữ liệu nên phân nửa cuối của thanh ghi này được tô đậm. Sau đó dữ liệu được ghi vào thanh ghi ống dẫn EX/MEM nên phân nửa đầu của thanh ghi ống dẫn này được tô đậm.



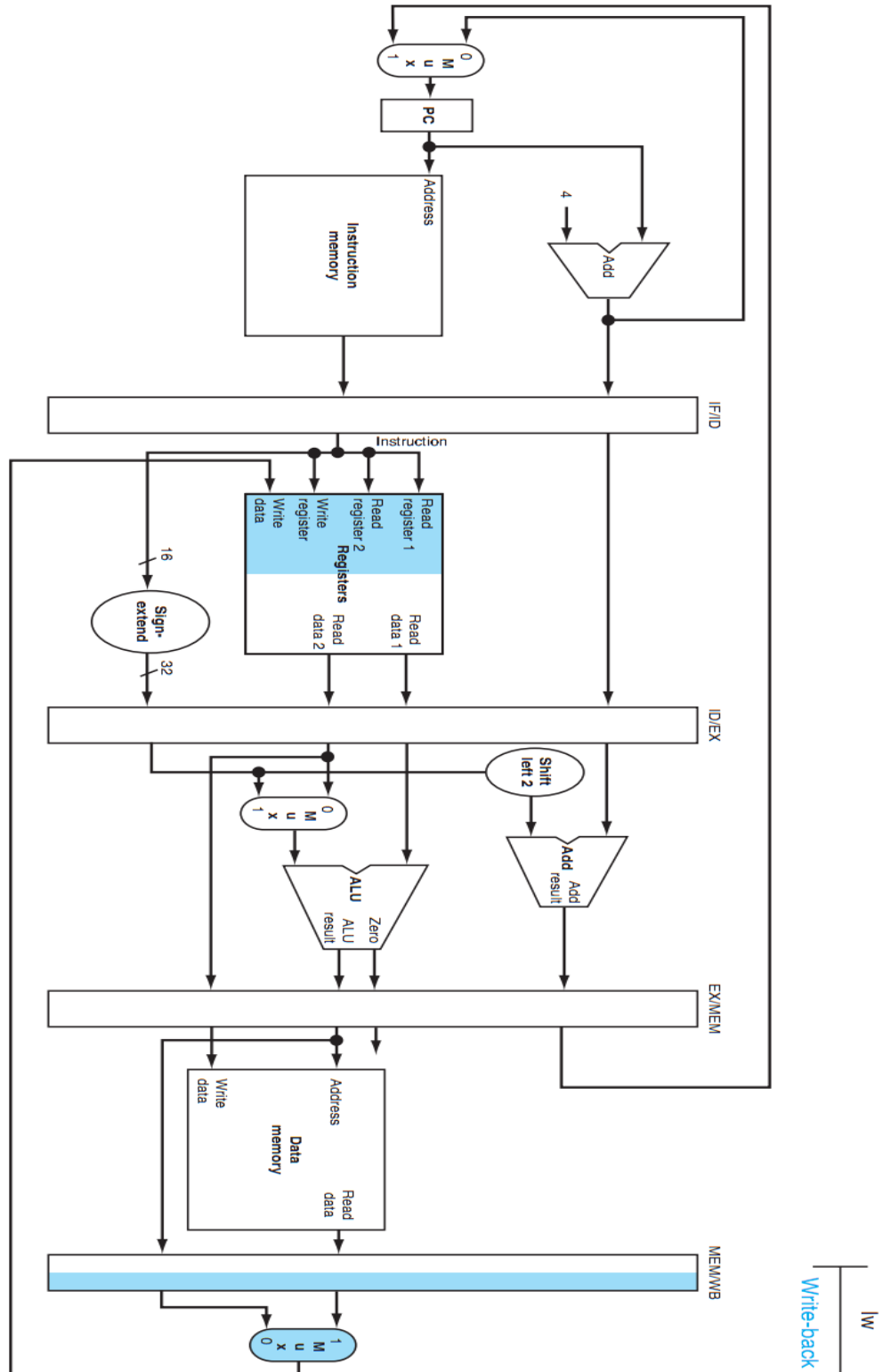
Hình 3.16: Giai đoạn thực thi của lệnh lw

– Truy cập bộ nhớ: hình 3.17 thể hiện giai đoạn bốn trong kỹ thuật ống dẫn của lệnh lw. Địa chỉ 32 bit được đọc từ thanh ghi EX/MEM dùng để xác định nội dung cần đọc trong bộ nhớ dữ liệu. Dữ liệu được đọc từ bộ nhớ sẽ được ghi vào thanh ghi ống dẫn MEM/WB. Trong trường hợp của lệnh lw, dữ liệu ghi vào bộ nhớ (write data) không hoạt động. Tương tự các trường hợp trên, đọc dữ liệu từ thanh ghi ống dẫn EX/MEM nên phân nửa cuối của thanh ghi này được tô đậm và ghi dữ liệu vào thanh ghi ống dẫn MEM/WB nên phân nửa đầu của thanh ghi này được tô đậm.



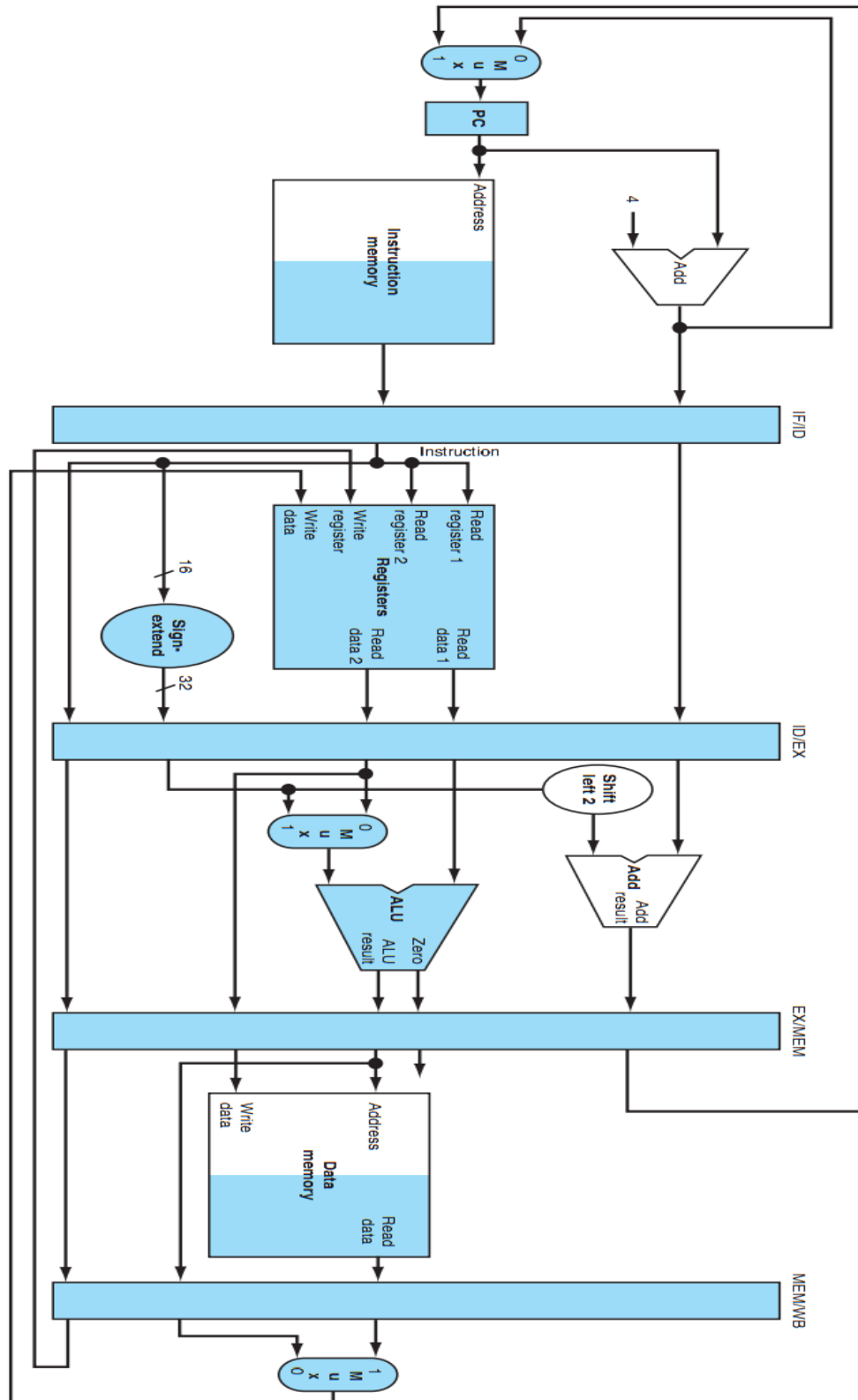
Hình 3.17: Giai đoạn truy cập bộ nhớ của lệnh lw

– Ghi kết quả: hình 3.18 thể hiện giai đoạn cuối cùng trong kỹ thuật ống dẫn của lệnh lw. Dữ liệu được đọc từ thanh ghi ống dẫn MEM/WB và ghi trở lại thanh ghi đích trong tập các thanh ghi (Registers) ở giai đoạn hai. Bộ điều hợp (Mux) trong trường hợp này có giá trị đầu vào tương ứng 1. Do dữ liệu được đọc từ thanh ghi ống dẫn MEM/WB nên phân nửa cuối của thanh ghi này được tô đậm. Trong khi đó, thao tác ghi thanh ghi được thực hiện trong tập thanh ghi nên phân nửa đầu của tập thanh ghi được tô đậm.



Hình 3.18: Giai đoạn ghi dữ liệu vào thanh ghi

Như vậy, đường đi dữ liệu kết hợp cả năm giai đoạn của lệnh lw trong kỹ thuật ống dẫn được thể hiện như hình 3.19. Trong hình này, cả bộ nhớ lệnh (instruction memory) và bộ nhớ dữ liệu (data memory) đều có quá trình đọc dữ liệu nhưng không có quá trình ghi dữ liệu nên phân nửa cuối của các bộ nhớ này được tô đậm. Trong khi đó cả bốn thanh ghi ống dẫn IF/ID, ID/EX, EX/MEM, MEM/WB đều có cả hai quá trình đọc và ghi nên toàn bộ các thanh ghi này được tô đậm.

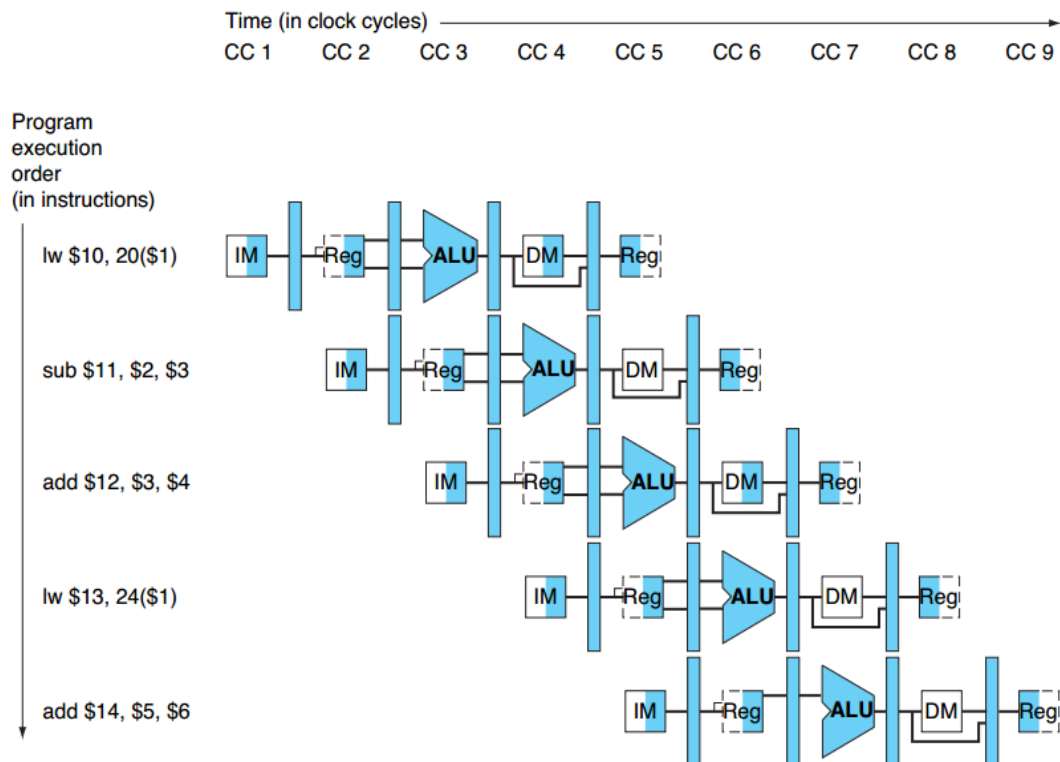


Hình 3.19: Kết hợp các giai đoạn của lệnh lw

Để biểu diễn đường đi dữ liệu trong kỹ thuật ống dẫn thường dùng hai loại giản đồ: giản đồ ống dẫn đa chu kỳ (*multiple-clock-cycle pipeline diagram*) và giản đồ ống dẫn đơn chu kỳ (*single-clock-cycle pipeline diagram*). Hình 3.12 là giản đồ ống dẫn biểu diễn bằng đa chu kỳ. Trong khi các hình từ hình 3.14 đến hình 3.19 là giản đồ ống dẫn biểu diễn bằng đơn chu kỳ. Giản đồ ống dẫn biểu diễn bằng đơn chu kỳ thể hiện trạng thái đường đi dữ liệu trong một chu kỳ. Giản đồ biểu diễn bằng đa chu trình thì thể hiện cách nhìn tổng quan trong kỹ thuật ống dẫn. Xét ví dụ với năm lệnh sau đây được thực hiện trong kỹ thuật ống dẫn:

```
lw $t0, 20($t1)
sub $t1, $t2, $t3
add $t2, $t3, $t4
lw $t3, 24($t1)
add $t4, $t5, $t6
```

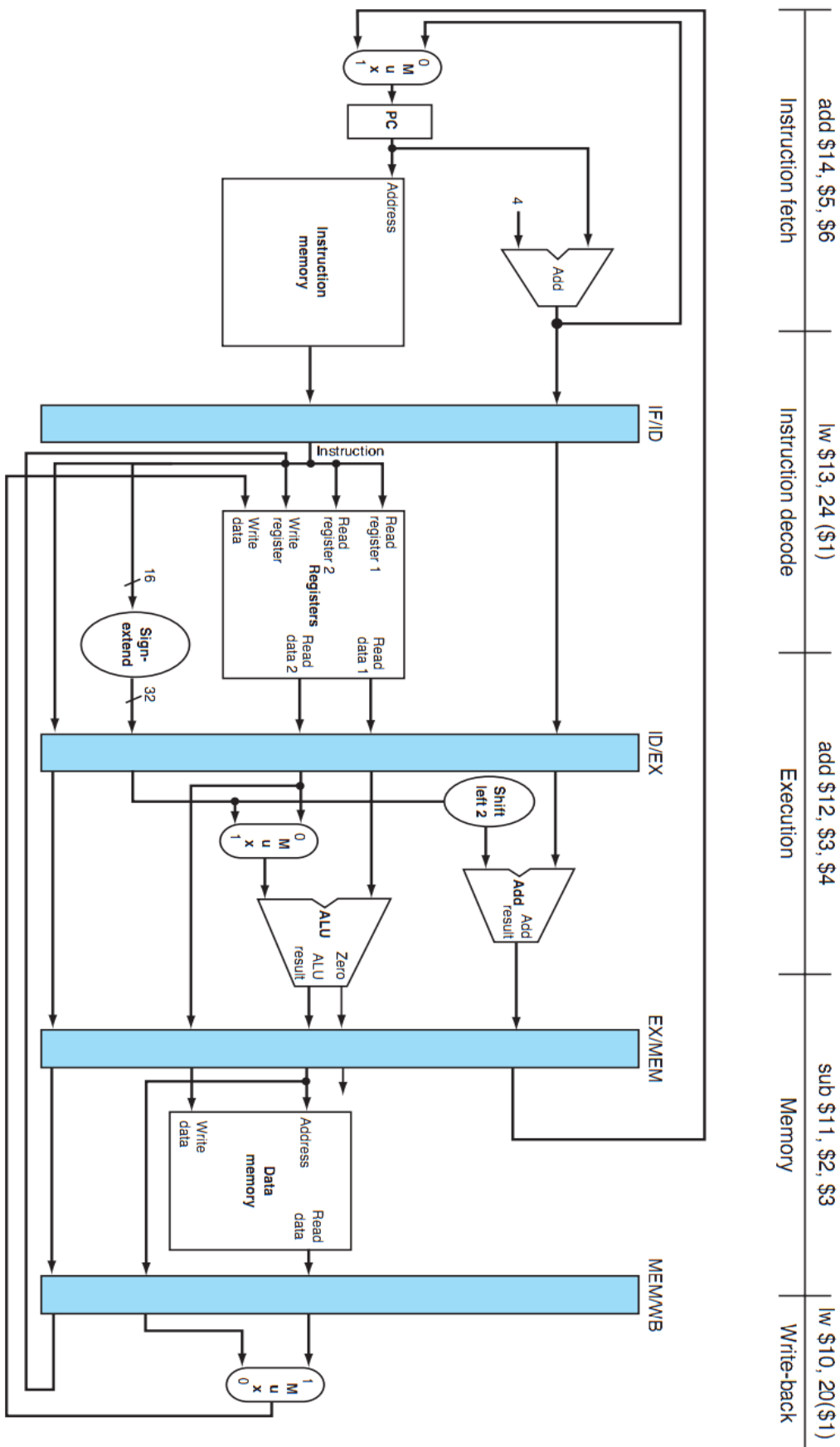
Hình 3.20 thể hiện giản đồ ống dẫn đa chu kỳ của năm lệnh trên. Thời gian theo chiều từ trái sang phải và thứ tự lệnh theo chiều từ trên xuống dưới. Biểu diễn của các giai đoạn ống dẫn được thể hiện mỗi phần theo chiều thẳng đứng, diễn ra trong những chu kỳ xung nhịp tương ứng.



Hình 3.20: Giản đồ ống dẫn biểu diễn bằng đa chu kỳ

Giản đồ ống dẫn đơn chu kỳ thể hiện toàn bộ trạng thái đường dẫn dữ liệu trong một chu kỳ xung nhịp, và tất cả năm lệnh trong kỹ thuật ống dẫn được xác định bằng nhãn ở phía trên các giai đoạn ống dẫn tương ứng của họ. Chúng ta dùng loại giản đồ này để thể hiện chi tiết những gì xảy ra trong kỹ thuật ống dẫn thông qua mỗi chu kỳ xung nhịp. Giản đồ ống dẫn đơn chu kỳ thể hiện một phần thẳng đứng trong giản đồ ống dẫn biểu diễn bằng đa chu kỳ, thể hiện đường dẫn dữ liệu của mỗi lệnh trong kỹ

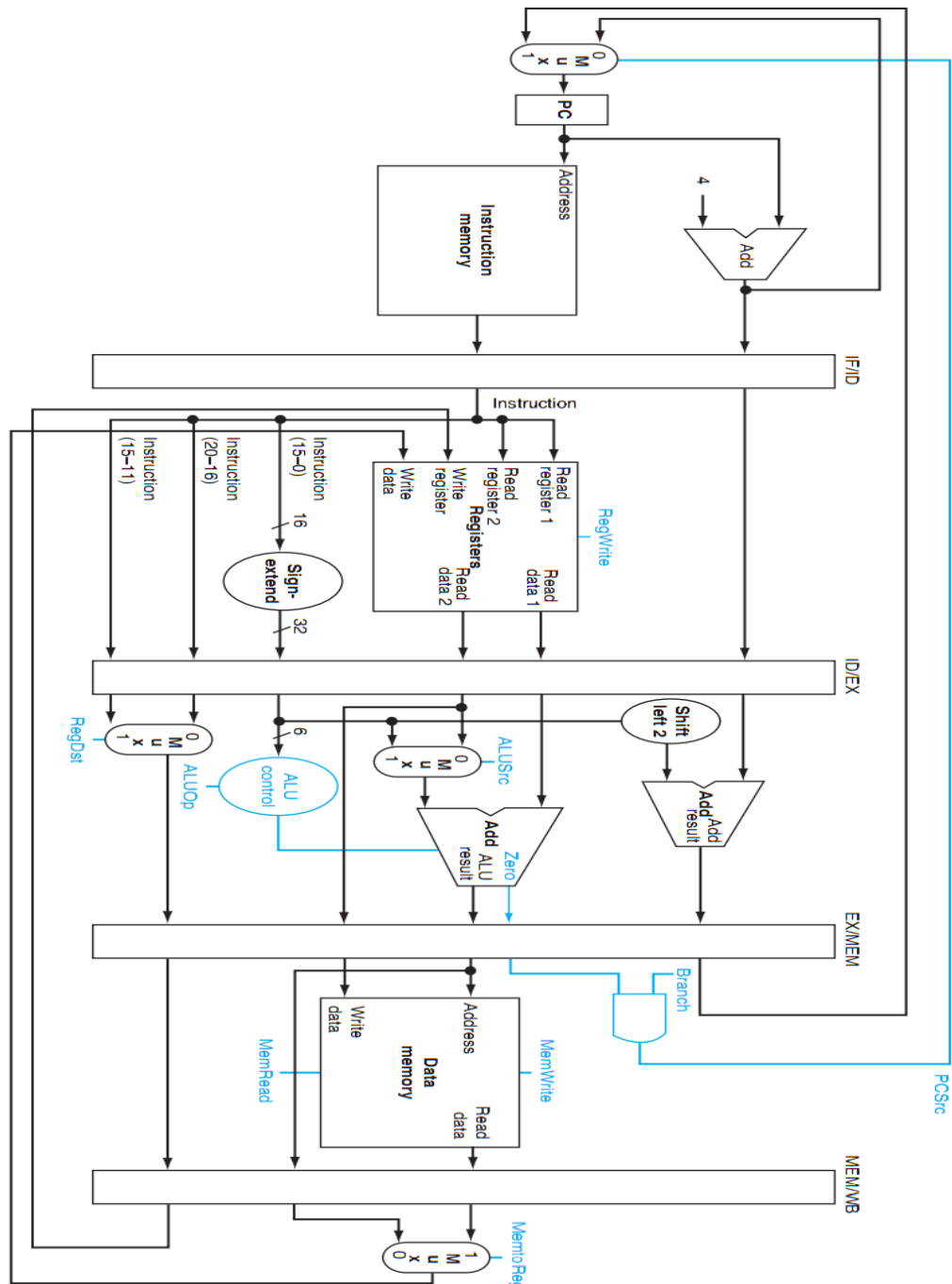
thuật ống dẫn ứng với một chu kỳ xung nhịp cụ thể. Ví dụ, hình 3.21 thể hiện giản đồ ống dẫn đơn chu kỳ tương ứng với chu kỳ thứ 5 (CC5) của hình 3.20



Hình 3.21: Giản đồ ống dẫn đơn chu kỳ tương ứng với chu kỳ 5 của hình 3.20

3.5.3 Điều khiển trong kỹ thuật ống dẫn (pipelined control)

Các đường điều khiển trong kỹ thuật ống dẫn được thể hiện như hình 3.22. Trong đó 6 bit thấp của trường funct (*function code*) trong 16 bit được mở rộng dấu thành 32 bit được lưu lại thành ghi ống dẫn ID/EX dùng làm tín hiệu vào cho ALU control. Đối với bộ điều hợp (Mux) có 2 ngõ vào và một ngõ ra cùng với tín hiệu điều khiển. Nếu tín hiệu điều khiển có giá trị 1 thì ngõ ra sẽ tương ứng với ngõ vào 1 và ngược lại tín hiệu điều khiển có giá trị 0 thì ngõ ra sẽ tương ứng với ngõ vào 0. Tín hiệu PCSrc được thiết lập bởi cổng AND. Tín hiệu được thiết lập (giá trị 1) khi cả hai tín hiệu Branch và Zero đều được thiết lập (giá trị 1), ngược lại tín hiệu này sẽ có giá trị 0. Tín hiệu Branch chỉ được thiết lập cho lệnh beq nên trong trường hợp này PCSrc sẽ có giá trị 1. Các trường hợp khác PCSrc sẽ có giá trị 0. Các trường hợp khác PCSrc sẽ có giá trị 0.



Hình 3.22: Các tín hiệu điều khiển trong kỹ thuật ống dẫn

Dựa vào 2 bit điều khiển ALUOp và 6 bit của trường mã lệnh (*function code*) mà các bit ALU control được thiết lập khác nhau như thể hiện trong bảng 3.3

Bảng 3.3: Thiết lập các bit ALU control

Dạng lệnh	ALUOp	Lệnh	Mã hàm (<i>function</i>)	Thao tác ALU	Tín hiệu điều khiển ALU
LW	00	load word	XXXXXX	add	0010
SW	00	store word	XXXXXX	add	0010
Branch equal	01	branch equal	XXXXXX	subtract	0110
R-type	10	add	100000	add	0010
R-type	10	subtract	100010	subtract	0110
R-type	10	AND	100100	AND	0000
R-type	10	OR	100101	OR	0001
R-type	10	set on less than	101010	set on less than	0111

Các tín hiệu điều khiển có ý nghĩa như sau:

- RegDst: giá trị 0 xác định thanh ghi để ghi được xác định bởi toán hạng rt (bit 20:16). Ngược lại giá trị 1 thì thanh ghi để ghi được xác định bởi toán hạng rd (bit 15:11).
- RegWrite: giá trị 0 không ảnh hưởng. Giá trị 1 thể hiện ghi dữ liệu vào thanh ghi.
- ALUSrc: giá trị 0 xác định toán hạng thứ hai của ALU lấy từ thanh ghi thứ hai (Read data 2). Ngược lại, giá trị 1 thì toán hạng thứ hai của ALU lấy giá trị mở rộng dấu 16 bit thấp của lệnh.
- PCSrc: giá trị 0 thì PC được thay thế bởi PC+4. Ngược lại, giá trị 1 thì PC được thay thế bởi kết quả tính tổng của ALU để nhảy đến địa chỉ đích.
- MemRead: giá trị 0 không ảnh hưởng. Giá trị 1 thì dữ liệu được đọc từ bộ nhớ.
- MemWrite: giá trị 0 không ảnh hưởng. Giá trị 1 thì dữ liệu được ghi vào bộ nhớ.
- MemtoReg: giá trị 0 xác định dữ liệu để ghi vào thanh ghi được lấy từ kết quả của ALU. Ngược lại, giá trị 1 xác định dữ liệu để ghi vào thanh ghi được đọc từ bộ nhớ.

Các tín hiệu điều khiển này cũng có thể được chia theo ba giai đoạn cuối trong kỹ thuật ống dẫn như bảng 3.4:

Bảng 3.4: Các tín hiệu điều khiển tương ứng theo 3 giai đoạn sau cùng

Lệnh	Tín hiệu điều khiển giai đoạn thực thi / tính địa chỉ				Tín hiệu điều khiển giai đoạn truy cập bộ nhớ			Tín hiệu điều khiển giai đoạn ghi lại	
	RegDst	ALUOp1	ALUOp0	ALUSrc	Branch	Mem- Read	Mem- Write	Reg- Write	Mem- toReg
R-format	1	1	0	0	0	0	0	1	0
lw	0	0	0	1	0	1	0	1	1
sw	X	0	0	1	0	0	1	0	X
beq	X	0	1	0	1	0	0	0	X

Cũng giống như mô hình thực thi tuần tự, PC được ghi ở mỗi chu kỳ xung nhịp (clock cycle). Do đó trong kỹ thuật ống dẫn không có thêm tín hiệu điều khiển ghi đối với PC. Tương tự không có tín hiệu điều khiển ghi đối với các thanh ghi ống dẫn (IF/ID, ID/EX, EX/MEM, và MEM/WB) bởi vì các thanh ghi này cũng được ghi ở mỗi chu kỳ xung nhịp.

Để thiết lập điều khiển trong kỹ thuật ống dẫn, chúng ta chỉ cần thiết lập các tín hiệu điều khiển trong mỗi giai đoạn của ống dẫn do các đường điều khiển này chỉ hoạt động trong mỗi giai đoạn tương ứng. Chúng ta có thể chia các tín hiệu điều khiển này vào 5 giai đoạn ống dẫn tương ứng như sau:

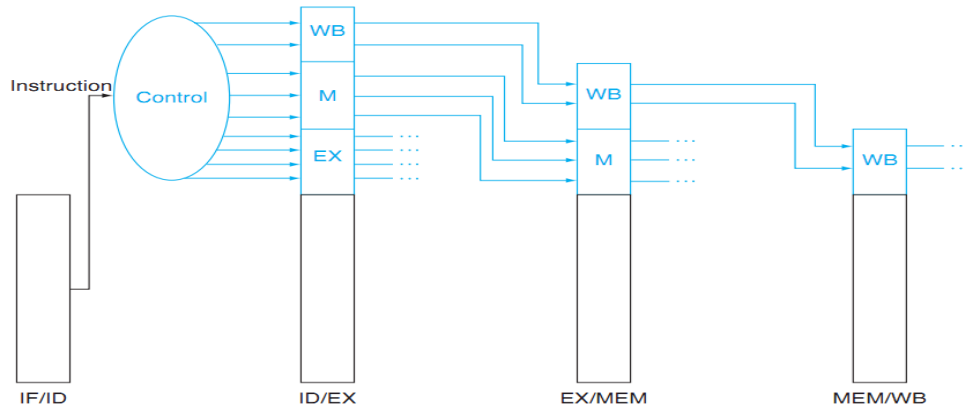
1. Duyệt lệnh (*Instruction fetch*): các tín hiệu điều khiển để đọc lệnh trong bộ nhớ để ghi vào PC thì luôn luôn được thiết lập. Do đó không cần chỉ định tín hiệu điều khiển trong giai đoạn này.

2. Giải mã lệnh/đọc thanh ghi (*Instruction decode/register file read*): giai đoạn này cũng thực hiện tương tự như bước trên nên không có tín hiệu điều khiển nào được bổ sung vào.

3. Thực thi/tính địa chỉ (*Execution/address calculation*): Các tín hiệu được thiết lập bao gồm: RegDst dùng để lựa chọn thanh ghi kết quả (thanh ghi đích), ALUOp để lựa chọn phép toán ALU và ALUSrc để đọc từ Read data 2 hoặc từ mở rộng dấu 32 bit.

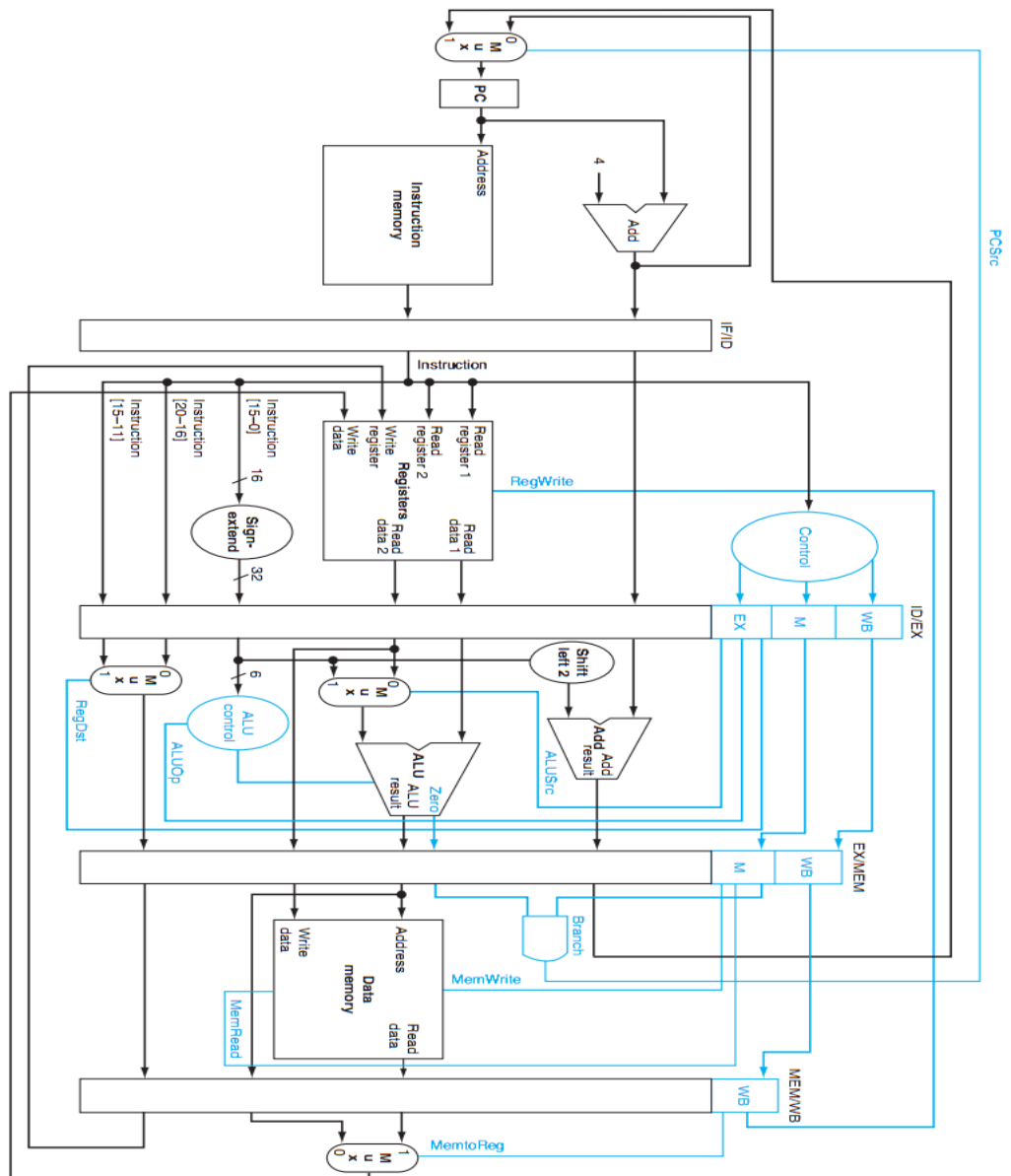
4. Truy cập bộ nhớ (*Memory access*): Các tín hiệu điều khiển được thiết lập trong giai đoạn này gồm có: Branch, MemRead và MemWrite. Các tín hiệu này được thiết lập tương ứng với lệnh beq, load và store. Tín hiệu PCSrc lựa chọn địa chỉ kế tiếp hoặc nhảy đến địa chỉ đích nếu tín hiệu Branch được thiết lập và kết quả của phép toán ALU bằng 0.

5. Ghi kết quả trở lại (*Write-back*): hai tín hiệu điều khiển là MemtoReg để di chuyển kết quả ALU hoặc từ bộ nhớ tới thanh ghi và RegWrite để ghi giá trị đã chọn.



Hình 3.23: Các tín hiệu điều khiển của ba giai đoạn ống dẫn sau cùng.

Vì vậy chín tín hiệu điều khiển có thể được lưu lại trong các thanh ghi ống dẫn như hình 3.23. Bốn tín hiệu được dùng trong giai đoạn EX, năm tín hiệu còn lại được lưu vào thanh ghi EX/MEM. Ba tín hiệu được dùng trong giai đoạn MEM, hai tín hiệu còn lại được lưu vào MEM/WB để sử dụng cho giai đoạn này.



Hình 3.24: Các tín hiệu điều khiển được nối vào các phần tương ứng

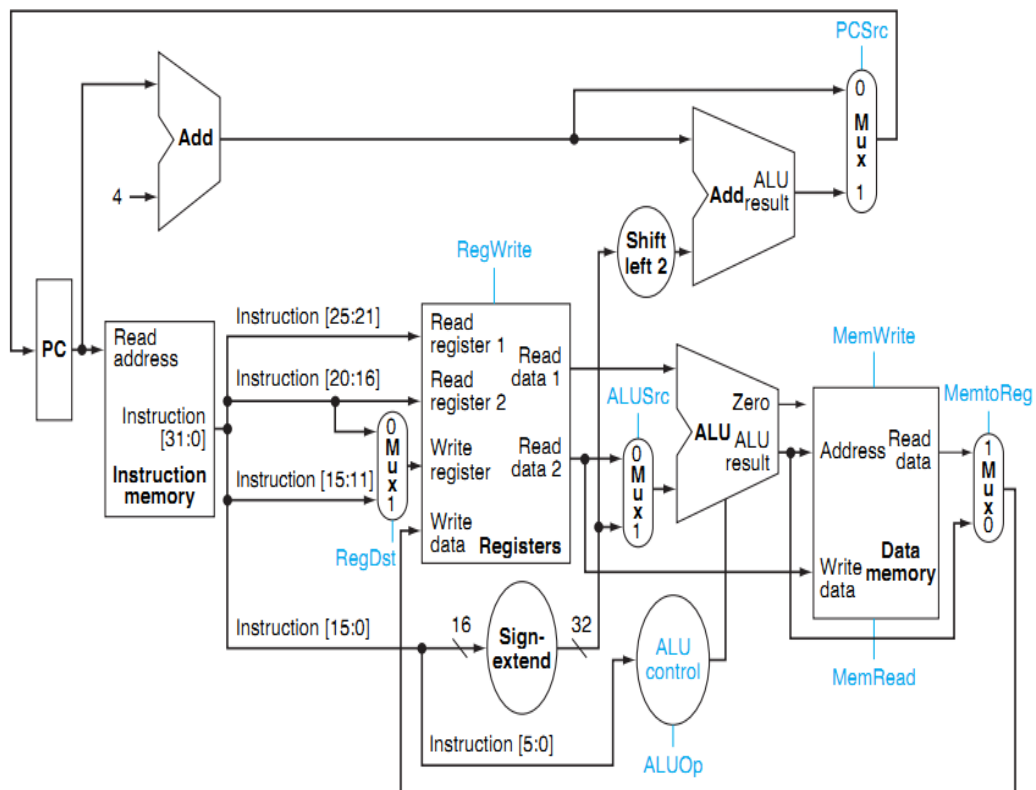
Để điều khiển trong kỹ thuật ống dẫn, chúng ta thiết lập chín tín hiệu điều khiển này cho mỗi giai đoạn của mỗi lệnh, bằng cách mở rộng các thanh ghi ống dẫn để lưu giữ các tín hiệu điều khiển này. Các tín hiệu điều khiển này bắt đầu từ giai đoạn EX nên chúng ta có thể tạo ra các tín hiệu điều khiển này ở giai đoạn giải mã lệnh. Hình 3.24 thể hiện đường dẫn đầy đủ với các thanh ghi ống dẫn và những tín hiệu điều khiển được kết nối tới các giai đoạn tương ứng.

3.6 TÓM TẮT

Trong chương này giới thiệu tổng quan nguyên lý hoạt động của vi xử lý MIPS. Trình bày cơ bản đường dẫn dữ liệu của ba loại lệnh: lệnh số học – luận lý, lệnh truyền dữ liệu và lệnh rẽ nhánh. Tổ chức hoạt động của bộ tính toán và luận lý (ALU) để thực hiện phép toán tùy thuộc vào từng loại lệnh. Dựa vào ba loại lệnh cơ bản, thiết lập các tín hiệu điều khiển của bộ điều khiển chính. Kỹ thuật ống dẫn là kỹ thuật mà nhiều lệnh được thực hiện theo dạng nạp chồng. Ưu điểm của kỹ thuật ống dẫn là thực thi lệnh nhanh hơn nhiều lần so với thực hiện lệnh theo tuần tự. Để điều khiển trong kỹ thuật ống dẫn, chúng ta sử dụng các thanh ghi ống dẫn.

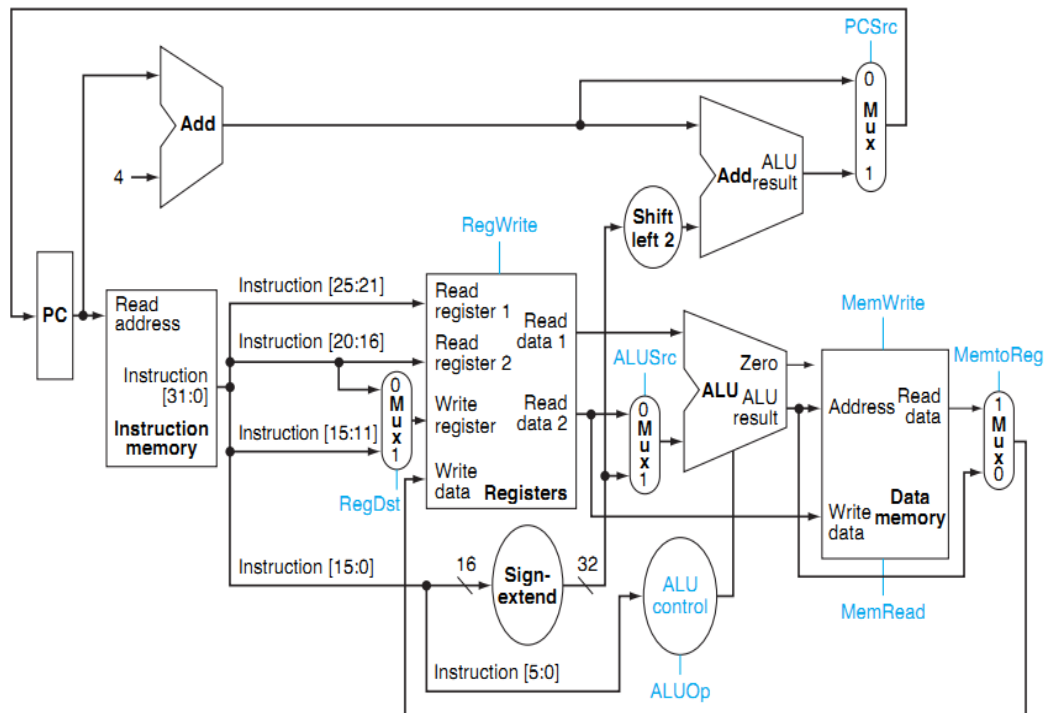
CÂU HỎI VÀ BÀI TẬP CHƯƠNG 3

1. Những điểm khác biệt chính giữa các lệnh số học – luận lý và các lệnh truy cập bộ nhớ?
2. Cách thức hoạt động của bộ tính toán và luận lý?
3. Cách thức hoạt động của bộ điều khiển chính?
4. Ý nghĩa của các tín hiệu điều khiển?
5. Các bước thực hiện lệnh load và store như thế nào?
6. Kỹ thuật ống dẫn là gì?
7. Cho sơ đồ như sau:



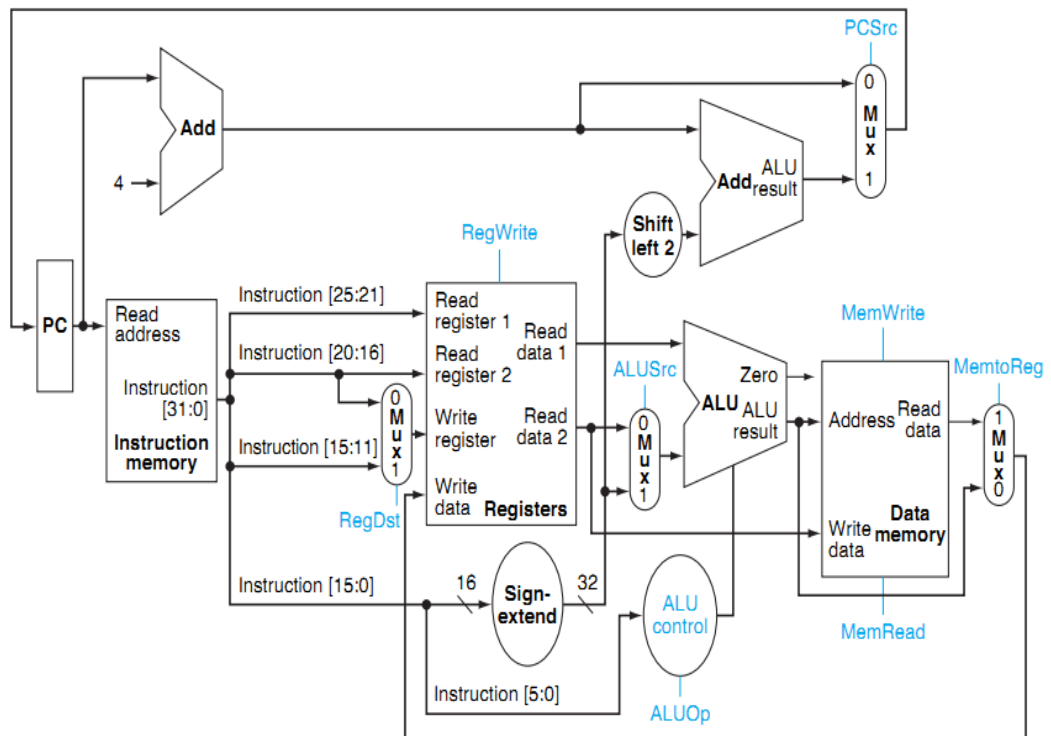
Hãy vẽ sơ đồ đường đi dữ liệu đối với định dạng lệnh **R-format** (chú ý: chỉ vẽ các đường tín hiệu và các thành phần hoạt động, những tín hiệu và thành phần không hoạt động thì không vẽ).

8. Cho sơ đồ như sau:



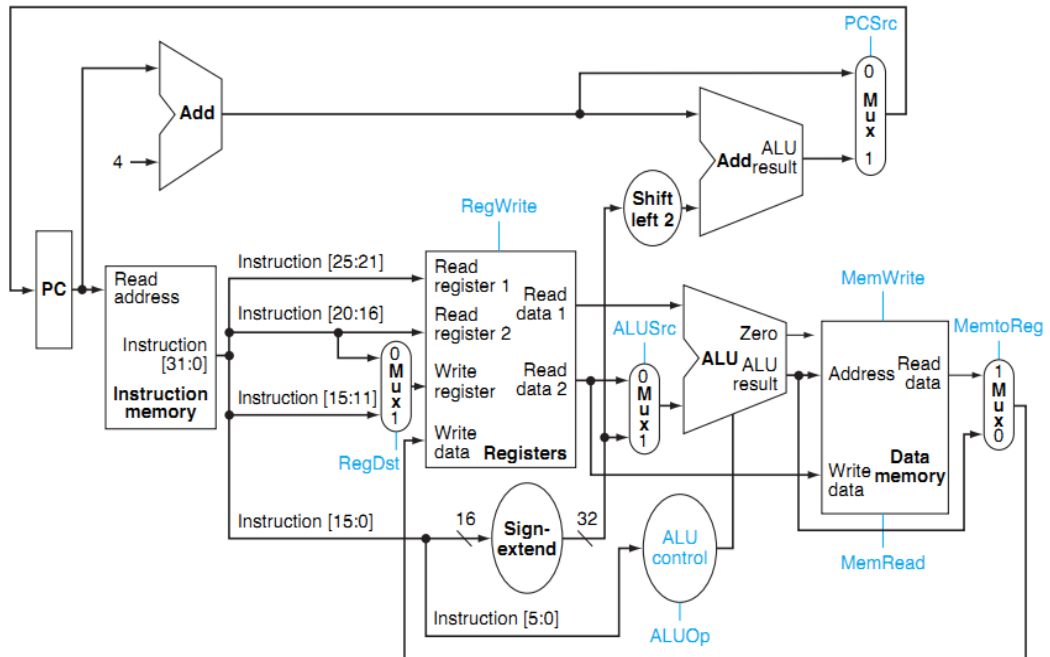
Hãy cho biết các giá trị của các tín hiệu điều khiển (RegDst, RegWrite, ALUSrc, MemWrite, MemRead, MemtoReg và PCSrc) đối với đường đi dữ liệu của định dạng lệnh **R-format**? Giải thích?

9. Cho sơ đồ như sau:



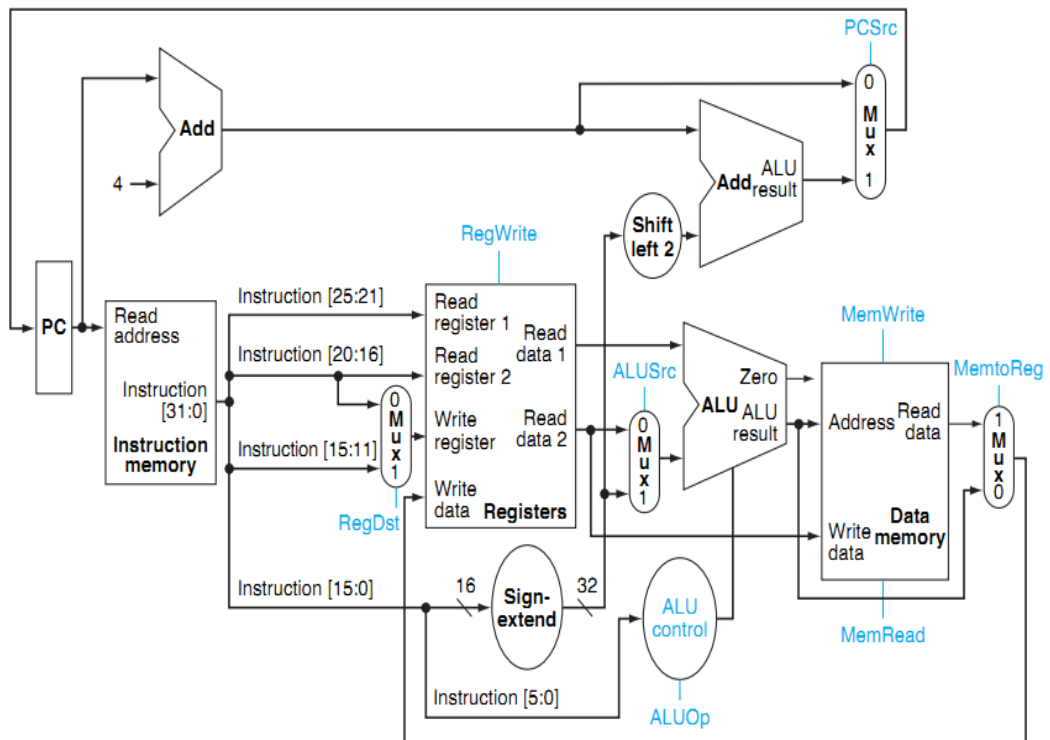
Hãy vẽ sơ đồ đường đi dữ liệu đối với định dạng lệnh **load** (chú ý: chỉ vẽ các đường tín hiệu và các thành phần hoạt động, những tín hiệu và thành phần không hoạt động thì không vẽ).

10. Cho sơ đồ như sau:



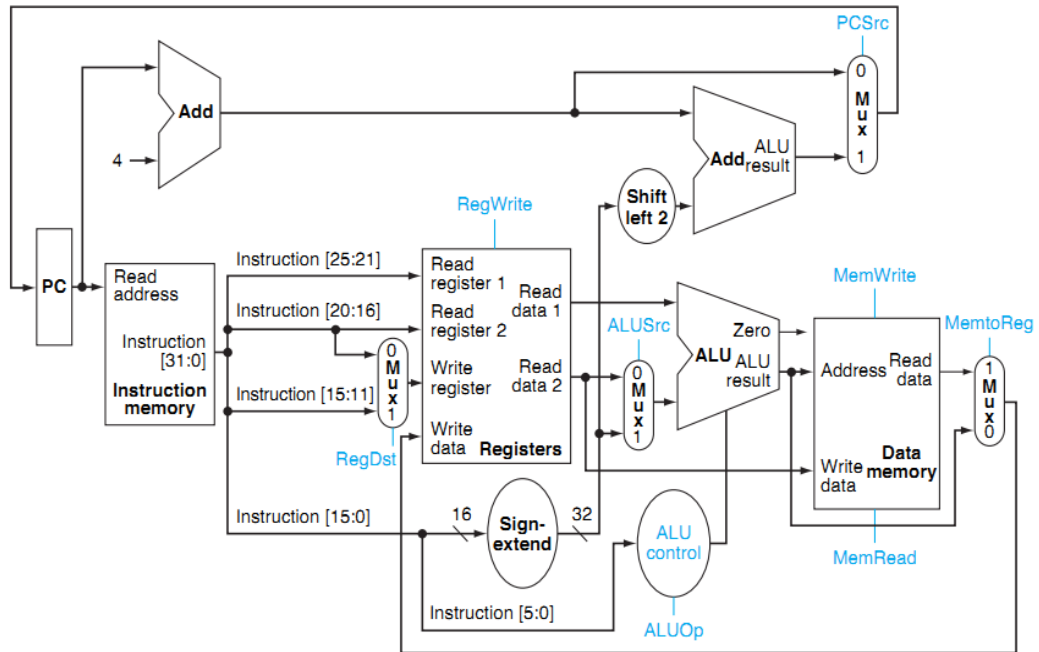
Hãy cho biết các giá trị của các tín hiệu điều khiển (RegDst, RegWrite, ALUSrc, MemWrite, MemRead, MemtoReg và PCSrc) đối với đường đi dữ liệu của định dạng lệnh **load**? Giải thích?

11. Cho sơ đồ như sau:



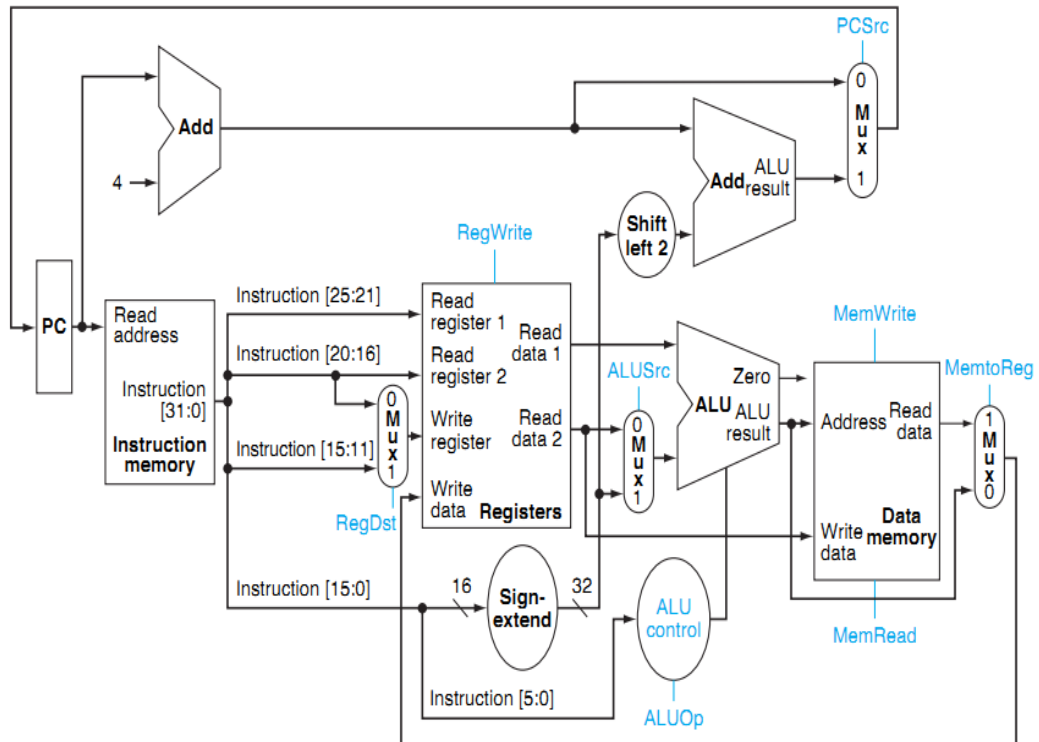
Hãy vẽ sơ đồ đường đi dữ liệu đối với định dạng lệnh **store** (chú ý: chỉ vẽ các đường tín hiệu và các thành phần hoạt động, những tín hiệu và thành phần không hoạt động thì không vẽ).

12. Cho sơ đồ như sau:



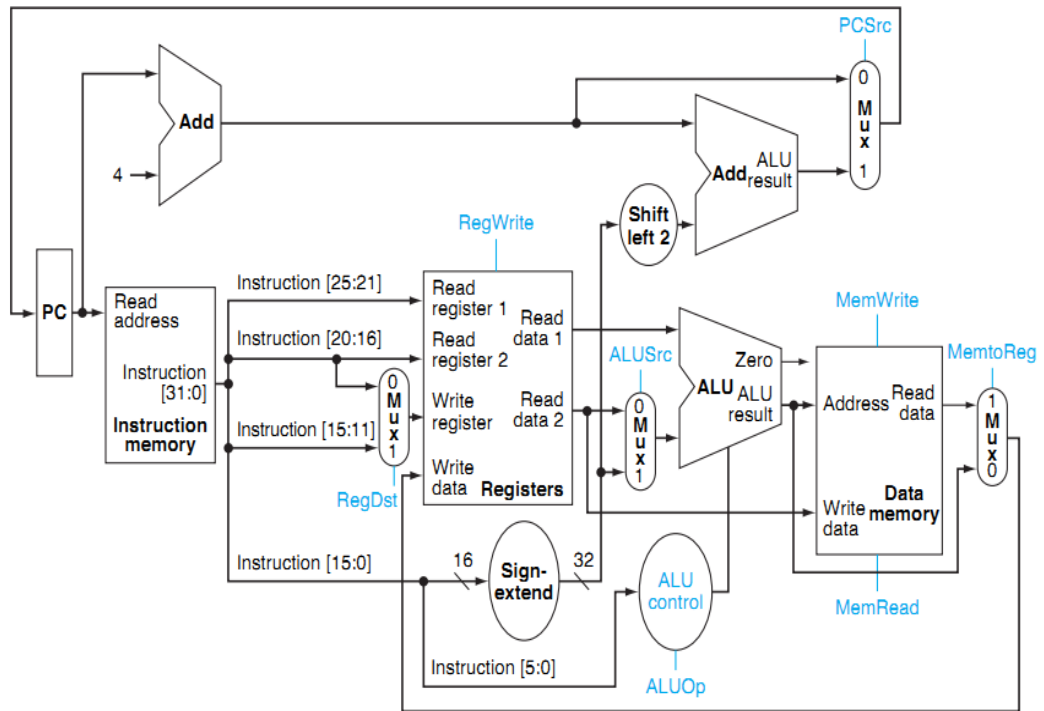
Hãy cho biết các giá trị của các tín hiệu điều khiển (RegDst, RegWrite, ALUSrc, MemWrite, MemRead, MemtoReg và PCSrc) đối với đường đi dữ liệu của định dạng lệnh **store**? Giải thích?

13. Cho sơ đồ như sau:



Hãy vẽ sơ đồ đường đi dữ liệu đối với định dạng lệnh **beq** (chú ý: chỉ vẽ các đường tín hiệu và các thành phần hoạt động, những tín hiệu và thành phần không hoạt động thì không vẽ).

14. Cho sơ đồ như sau:



Hãy cho biết các giá trị của các tín hiệu điều khiển (RegDst, RegWrite, ALUSrc, MemWrite, MemRead, MemtoReg và PCSrc) đối với đường đi dữ liệu của định dạng lệnh **beq**? Giải thích?

CHƯƠNG 4

BỘ NHỚ

Mục đích: Giới thiệu tổng quan về nguyên lý hoạt động của bộ nhớ cache: truy cập bộ nhớ cache, xử lý thất bại, xử lý ghi... Các đại lượng dùng để đo lường bộ nhớ cache và phương pháp để cải tiến hiệu suất của bộ nhớ cache.

4.1 GIỚI THIỆU

Trong chương này chúng ta sẽ tìm hiểu một số khái niệm đặc trưng cho hoạt động của bộ nhớ. Sau đó sẽ tìm hiểu chi tiết về nguyên lý hoạt động của bộ nhớ cache, là bộ nhớ được thiết kế trung gian giữa CPU và bộ nhớ chính (RAM).

Truy cập dữ liệu trong bộ nhớ thường tuân theo nguyên tắc thời gian và nguyên tắc không gian:

- Nguyên tắc thời gian: nếu một khối dữ liệu được tham khảo thì khối dữ liệu này có khuynh hướng chẳng bao lâu sẽ được tham khảo lại lần nữa.
- Nguyên tắc không gian: nếu một khối dữ liệu được tham khảo thì các khối dữ liệu kề với khối này có khuynh hướng chẳng bao lâu sẽ được tham khảo.

Dựa vào thời gian truy cập, dung lượng và giá thành mà bộ nhớ được phân thành cấp bậc bộ nhớ (memory hierarchy) như được thể hiện ở hình 4.1. Bộ nhớ có tốc độ truy xuất nhanh thì có dung lượng nhỏ, giá thành cao. Ngược lại, bộ nhớ có tốc độ truy xuất chậm thì có dung lượng lớn và giá thành thấp.

Tốc độ	Bộ Nhớ	Kích thước	Chi phí (\$/bit)
Nhanh nhất	Register	Nhỏ nhất	Cao nhất
	SRAM		
	DRAM		
Chậm nhất	Đĩa từ	Lớn nhất	Thấp nhất

Hình 4.1: Cấu trúc của cấp bậc bộ nhớ.

Ngày nay, có ba kỹ thuật cơ bản để xây dựng cấp bậc bộ nhớ: thứ nhất, bộ nhớ cache dùng kỹ thuật SRAM (*Static Random Access Memory*). Thứ hai, bộ nhớ chính thì sử dụng kỹ thuật DRAM (*Dynamic Random Access Memory*). Thứ ba, bộ nhớ ngoài thì dùng kỹ thuật đĩa từ (*magnetic disk*). Thời gian truy xuất cũng như giá thành của ba kỹ thuật này được thể hiện trong bảng 4.1 (tham khảo năm 2008).

Bảng 4.1: So sánh giữa ba kỹ thuật trong cấp bậc bộ nhớ

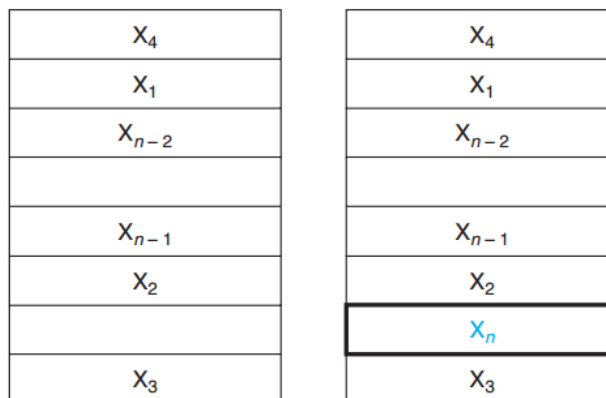
Công nghệ bộ nhớ	Thời gian truy cập	Chi phí (\$/GB năm 2008)
SRAM	0.5 – 2.5 ns	\$2000 – \$5000
DRAM	50 – 70 ns	\$20 – \$75
Đĩa từ	5,000,000 – 20,000,000 ns	\$0.20 – \$2

Dữ liệu được truyền theo dạng khối (*block*). Khi CPU yêu cầu một khối dữ liệu trong cache mà khối này hiện diện trong cache thì yêu cầu này được gọi là một thành công (*hit*). Ngược lại, nếu khối này không có trong cache thì yêu cầu này là một thất bại (*miss*). Tỷ lệ thành công (*hit rate*) là tỷ số của số lần truy cập thành công trên tổng số lần truy cập. Tỷ lệ thất bại (*miss rate*) bằng $1 - \text{tỷ lệ thành công}$, là tỷ số của số lần truy cập thất bại trên tổng số lần truy cập. Ngoài ra, thời gian thành công (*hit time*) là thời gian truy cập trong cache bao gồm thời gian để xác định một yêu cầu truy cập là thành công hay thất bại. Trùng phạt thất bại (*miss penalty*) là thời gian để thay thế khối trong cache bằng khối tương ứng trong bộ nhớ chính cộng với thời gian để phân phối khối này tới CPU. Các đại lượng này được dùng để đo lường hiệu suất của cấp bậc bộ nhớ.

4.2 BỘ NHỚ CACHE

4.2.1 Tổng quan

Cache là bộ nhớ nằm ở cấp cao nhất trong cấp bậc bộ nhớ, đảm nhiệm vai trò giao tiếp giữa CPU và bộ nhớ chính. Trước tiên, chúng ta xét cấu trúc một cache đơn giản như hình 4.2. Cache được khởi tạo chứa các khối X_1, X_2, \dots, X_{n-1} . CPU đưa ra yêu cầu tham khảo đến khối X_n mà khối này không có trong cache (dẫn đến một thất bại cache). Do đó phải tìm khối này trong bộ nhớ, sau đó ghi vào trong cache.

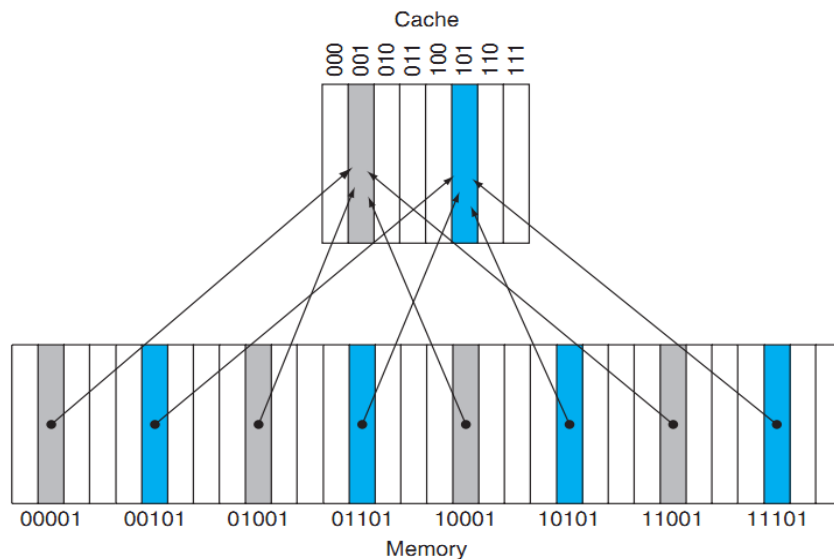


Hình 4.2: Bộ nhớ Cache trước và sau khi CPU tham khảo từ X_n

Như vậy có hai vấn đề cần xem xét: thứ nhất, các khối được lưu trữ như thế nào trong cache. Thứ hai, làm cách nào để tìm khối trong cache. Mô hình đơn giản nhất thường được sử dụng là ánh xạ trực tiếp (*direct mapped*). Mô hình này sẽ ánh xạ địa chỉ bộ nhớ của một khối sang chính xác một vị trí trong cache theo công thức:

(địa chỉ khối trong bộ nhớ) modulo (số khối trong cache)

Số khối trong cache thường được chọn là lũy thừa 2 nên vị trí trong cache chính bằng $\log_2(\text{số mục từ trong cache})$ bit thấp trong địa chỉ bộ nhớ của khối đó. Ví dụ cache có 8 khối thì vị trí trong cache chính bằng 3 ($8=2^3$) bit thấp trong địa chỉ của khối. Hình 4.3 thể hiện cách ánh xạ trực tiếp các địa chỉ 1_{10} (00001_2) và 29_{10} (11101_2) được ánh xạ tương ứng tới vị trí 1_{10} (001_2) và 5_{10} (101_2) trong cache.



Hình 4.3: Ánh xạ trực tiếp từ bộ nhớ sang cache có 8 khối

Như vậy sẽ có nhiều khối bộ nhớ có thể được ánh xạ vào cùng một vị trí trên cache. Làm thế nào để biết được khối nào đang được ánh xạ trong cache. Để giải quyết vấn đề này, chúng ta bổ sung vào các thẻ (*tags*) chứa thông tin về khối đó. Cụ thể các thẻ này chứa các bit cao còn lại trong địa chỉ của khối sau khi đã loại bỏ các bit thấp xác định vị trí khối trong cache. Như hình 4.3 ở trên thì các thẻ này chính là 2 bit cao trong 5 bit địa chỉ khối.

Ngoài ra, khi CPU mới khởi tạo thì các khối trên cache là rỗng, tức các khối này không có ý nghĩa khi được tham khảo đến. Hoặc sau một thời gian hoạt động một số khối trên cache có thể trống. Để một tham khảo đến một khối trên cache là một khối được ánh xạ thực sự bởi một khối trong bộ nhớ. Người ta bổ sung thêm một bit hợp lý (*valid bit*). Khi bit này được thiết lập tức là khối trên cache là một khối được ánh xạ từ một khối trong bộ nhớ. Ngược lại, bit này có giá trị 0 tức là khối trên cache không có ý nghĩa khi được tham khảo đến.

4.2.2 Truy cập bộ nhớ Cache

Để hiểu nguyên tắc vận hành của cache, chúng ta xét ví dụ bộ nhớ cache gồm có 8 khối, được khởi tạo rỗng. CPU lần lượt truy cập 9 địa chỉ bộ nhớ. Các tham khảo này đến cache có thể thành công hoặc thất bại như được thể hiện ở bảng 4.2. Do cache có 8 khối nên dùng 3 (2^3) bit thấp trong địa chỉ để xác định vị trí trong cache.

Bảng 4.2: CPU lần lượt truy cập 9 địa chỉ bộ nhớ

Địa chỉ truy cập (hệ thập phân)	Địa chỉ truy cập (hệ nhị phân)	Thành công / thất bại	Khối được ánh xạ trong cache (tìm thấy hoặc thay thế)
22	10110 ₂	T.bại(4.4b)	$(10110_2 \bmod 8) = 110$
26	11010 ₂	T.công(4.4c)	$(11010_2 \bmod 8) = 010$
22	10110 ₂	Thành công	$(10110_2 \bmod 8) = 110$
26	11010 ₂	Thành công	$(11010_2 \bmod 8) = 010$
16	10000 ₂	T.bại(4.4d)	$(10000_2 \bmod 8) = 000$
3	00011 ₂	T.bại(4.4e)	$(00011_2 \bmod 8) = 011$
16	10000 ₂	Thành công	$(10000_2 \bmod 8) = 000$
18	10010 ₂	T.bại(4.4f)	$(10010_2 \bmod 8) = 010$
16	10000 ₂	Thành công	$(10000_2 \bmod 8) = 000$

Do ban đầu cache được khởi tạo rỗng nên một số tham chiếu địa chỉ đầu không có trong cache nên thất bại (*miss*). Tham chiếu địa chỉ 18 (thứ 8 trong danh sách) xảy ra hiện tượng thay thế khối hiện có tại vị trí này. Khối tại địa chỉ 18 (10010₂) được đặt vào vị trí khối cache số 2 (010₂) thay thế khối ở địa chỉ 26 (11010₂) đang vị trí này.

Địa chỉ	V	Thẻ	Dữ liệu
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	N		
111	N		

a. Trạng thái khởi tạo của cache

Địa chỉ	V	Thẻ	Dữ liệu
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	Y	10	Mem.(10110)
111	N		

b. Xử lý thất bại của địa chỉ (10110)

Địa chỉ	V	Thẻ	Dữ liệu
000	N		
001	N		
010	Y	11	Mem.(11010)
011	N		
100	N		
101	N		
110	Y	10	Mem.(10110)
111	N		

c. Xử lý thất bại của địa chỉ (11010)

Địa chỉ	V	Thẻ	Dữ liệu
000	Y	10	Mem.(10000)
001	N		
010	Y	11	Mem.(11010)
011	N		
100	N		
101	N		
110	Y	10	Mem.(10110)
111	N		

d. Xử lý thất bại của địa chỉ (10000)

Địa chỉ	V	Thẻ	Dữ liệu
000	Y	10	Mem.(10000)
001	N		
010	Y	11	Mem.(11010)
011	Y	00	Mem.(00011)
100	N		
101	N		
110	Y	10	Mem.(10110)
111	N		

e. Xử lý thất bại của địa chỉ (00011)

Địa chỉ	V	Thẻ	Dữ liệu
000	Y	10	Mem.(10000)
001	N		
010	Y	10	Mem.(10010)
011	Y	00	Mem.(00011)
100	N		
101	N		
110	Y	10	Mem.(10110)
111	N		

f. Xử lý thất bại của địa chỉ (10010)

Hình 4.4: Bộ nhớ cache với lần lượt 9 địa chỉ được truy cập

4.2.3 Xử lý thất bại cache

Khi CPU truy cập một khối trong cache, nếu thành công (khối cần truy cập có trong cache) thì CPU thực thi tiếp tục thực thi bình thường. Nhưng khi xảy ra thất bại cache (khối cần truy cập không có trong cache) thì phải tìm khối này trong bộ nhớ và ghi vào lại trong cache. Xử lý thất bại cache được thực hiện kết hợp với bộ điều khiển (CU – *control unit*) của CPU. Để xử lý thất bại cache CPU thực hiện một sự trì hoãn (*stall*) trong một số chu kỳ xung nhịp. Tức là trong lúc đợi dữ liệu từ bộ nhớ, CPU không thể thực hiện các lệnh tiếp theo. Do đó, giá trị của các thanh ghi sẽ được lưu lại.

Đối với lệnh gây ra thất bại cache thì lúc này nội dung thanh ghi lệnh là không hợp lý. Để lấy lệnh thích hợp, chúng ta phải đọc từ bộ nhớ. Do thanh ghi PC

(*program counter*) tự động tăng lệnh vào đầu của chu kỳ thực thi lệnh, nên địa chỉ lệnh gây ra thất bại cache bằng giá trị của thanh ghi PC – 4.

Như vậy, các bước cần thực hiện đối với lệnh gây ra thất bại cache:

1. Gửi giá trị PC ban đầu (PC – 4) tới bộ nhớ
2. Thực hiện thao tác đọc bộ nhớ và chờ bộ nhớ hoàn thành tác vụ.
3. Ghi dữ liệu vào cache, ghi các bit cao của địa chỉ vào trường tag và bật bit hợp lệ (*valid*).
4. Khởi tạo thực thi lệnh ở bước đầu tiên.

4.2.4 Xử lý ghi

Thao tác ghi thì phức tạp hơn. Giả sử chúng ta muốn ghi dữ liệu với lệnh *store*, nếu dữ liệu được ghi vào cache mà không ghi tương ứng vào bộ nhớ. Khi đó sẽ xảy ra tình trạng không nhất quán dữ liệu giữa cache và bộ nhớ. Để giải quyết vấn đề này, các cách ghi được đưa ra gồm có:

- Ghi đồng thời (*write-through*): đây là cách đơn giản nhất để thực hiện nhất quán dữ liệu giữa cache và bộ nhớ. Ghi đồng thời được thực hiện bằng cách dữ liệu được ghi đồng thời vào cache và bộ nhớ tương ứng. Mặc dù cách ghi này đơn giản nhưng hiệu suất lại không cao. Với mỗi thao tác ghi, dữ liệu đều được ghi vào bộ nhớ (cần nhiều thời gian, ít nhất là 100 chu kỳ xung nhịp), do đó làm chậm tốc độ của CPU. Ví dụ với chương trình có 10% là lệnh *store* nếu CPI (trong trường hợp không có thất bại cache) là 1.0. Mỗi lần ghi cần 100 chu kỳ xung nhịp thì CPI sẽ là: $1.0 + 100 \times 10\% = 11$. Do đó làm giảm hiệu suất hơn 10 lần.

- Ghi đệm (*write buffer*): bộ nhớ đệm lưu dữ liệu trong khi chờ ghi vào bộ nhớ. Sau khi dữ liệu được lưu vào cache và bộ nhớ đệm, CPU tiếp tục thực thi. Khi dữ liệu được ghi vào cache thì vị trí trên bộ nhớ đệm được giải phóng. Nếu bộ nhớ đệm đầy trong khi CPU thực hiện thao tác ghi thì CPU phải được trì hoãn (*stall*) cho đến khi có vị trí trống trên bộ nhớ đệm. Nếu tỉ lệ ghi vào bộ nhớ thấp hơn so với tỷ lệ ghi của CPU thì khi đó bộ nhớ đệm sẽ đầy và CPU phải thực hiện trì hoãn. Vấn đề này xuất hiện khi lúc nào đó CPU thực hiện thao tác ghi với tần suất cao. Để tránh vấn đề trì hoãn do bộ nhớ đầy thì có thể tăng kích thước của bộ nhớ đệm.

- Ghi lại (*write-back*): khi thực hiện thao tác ghi, giá trị mới chỉ được ghi vào khối trong cache. Khối thay đổi này sẽ được ghi vào bộ nhớ khi khối này được thay thế. Cách này cải thiện đáng kể hiệu suất hoạt động của hệ thống khi CPU có tốc độ ghi nhanh hơn đáp ứng thao tác ghi của bộ nhớ. Nhưng vấn đề cài đặt sẽ phức tạp hơn so với ghi đồng thời.

4.2.5 Thiết kế bộ nhớ hỗ trợ cache

Như chúng ta đã biết bộ nhớ chính được xây dựng dựa trên kỹ thuật DRAM. CPU thì được kết nối với bộ nhớ chính thông qua bus, nhưng xung nhịp của bus thì thường thấp hơn xung nhịp của CPU. Do đó tốc độ xung nhịp của bus làm ảnh hưởng đến thời gian trừng phạt thất bại. Để giảm trừng phạt thất bại thì có thể tăng băng thông (*bandwidth*) từ bộ nhớ tới cache. Để hiểu được ảnh hưởng của tổ chức bộ nhớ khác nhau, chúng ta xem xét một vài khái niệm về thời gian truy cập bộ nhớ, giả sử:

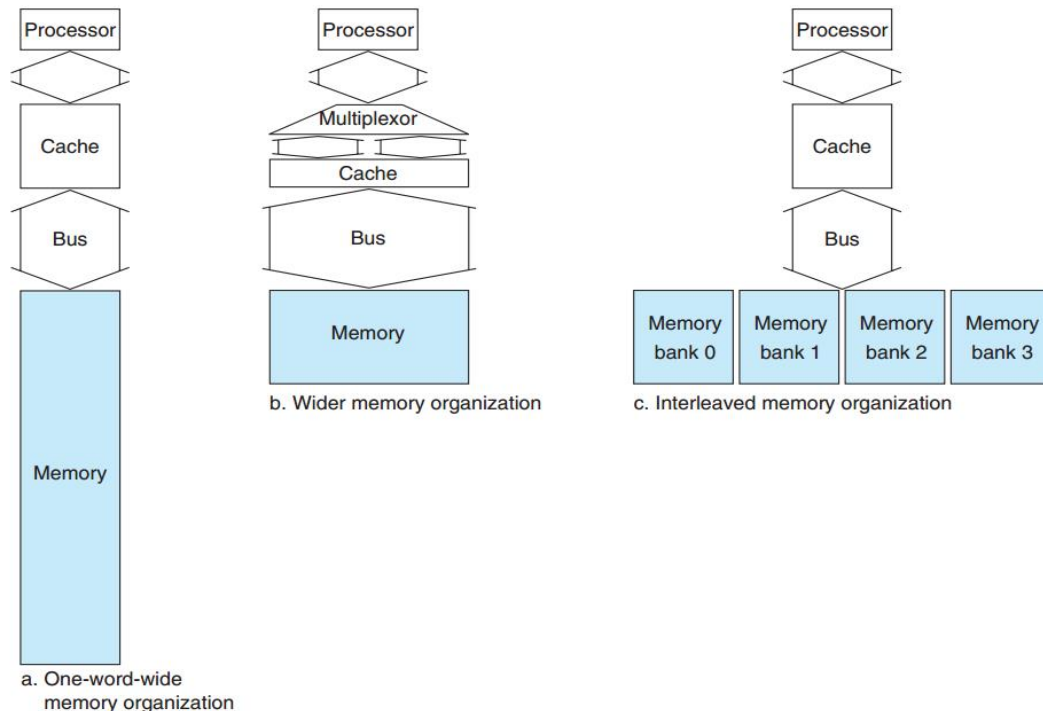
- Cần 1 chu kỳ xung nhịp bus bộ nhớ để gửi địa chỉ
- 15 chu kỳ xung nhịp bus bộ nhớ cho mỗi lần khởi tạo truy cập vào DRAM
- 1 chu kỳ xung nhịp bus bộ nhớ để truyền một từ (*word*) dữ liệu

Nếu chúng ta có khối cache gồm 4 từ và băng thông là 1 từ thì trừng phạt thất bại sẽ là $1 + 4 \times 15 + 4 \times 1 = 65$ chu kỳ xung nhịp bus bộ nhớ. Do đó số byte được truyền trên một chu kỳ xung nhịp bus đối với một thất bại sẽ là:

$$\frac{4 \times 4}{65} = 0.25$$

Có 3 cách để thiết kế hệ thống bộ nhớ như được trình bày trong hình 4.5. Cách thứ nhất là thiết kế hệ thống bộ nhớ với cache có độ rộng bus là 1 từ. Như vậy các truy cập vào bộ nhớ được thực hiện tuần tự. Cách thứ hai tăng băng thông bộ nhớ bằng cách mở rộng bộ nhớ và bus, cho phép truy cập nhiều từ của khối. Cách thứ ba tăng băng thông bằng cách mở rộng bộ nhớ nhưng không mở rộng độ lớn của bus. Chúng ta có thể thấy ở cách thứ hai và ba sẽ cải thiện thời gian trừng phạt thất bại so với cách thứ nhất.

Khi tăng độ rộng của bộ nhớ và của bus thì sẽ làm tăng băng thông bộ nhớ tương ứng, làm giảm thời gian truy cập và thời gian truyền. Với bộ nhớ có độ rộng hai từ, trừng phạt thất bại giảm từ 65 chu kỳ xung nhịp bus bộ nhớ ở cách tổ chức thứ nhất xuống còn $1 + 2 \times 15 + 2 \times 1 = 33$ chu kỳ xung nhịp bus bộ nhớ. Băng thông tương ứng sẽ bằng 0.48 (tăng gần gấp đôi so với cách thứ nhất) byte trên chu kỳ xung nhịp bus.



Hình 4.5: Ba cách tổ chức bộ nhớ khác nhau

Thay vì tăng độ rộng của bộ nhớ và bus rộng hơn thì ở cách tổ chức thứ ba lại chia bộ nhớ thành các dãy (*bank*) để có thể đọc hoặc ghi nhiều từ trong một lần truy cập so với chỉ một từ trong mỗi lần truy cập ở cách thứ nhất. Mỗi dãy có độ rộng một từ

bằng với độ rộng của bus và cache. Nhưng khi gửi địa chỉ tới nhiều dãy sẽ cho phép đọc đồng thời. Ví dụ, với bộ nhớ chia thành bốn dãy thì thời gian để truy cập khối gồm có bốn từ bao gồm 1 chu kỳ để truyền địa chỉ và yêu cầu đọc tới các dãy này, 15 chu kỳ để truy cập bốn dãy này và 4 chu kỳ để truyền bốn từ tới cache. Như vậy, trừng phạt thất bại sẽ là $1 + (1 \times 15) + 4 \times 1 = 20$ chu kỳ xung nhịp bus bộ nhớ. Băng thông sẽ là 0.80 byte trên chu kỳ xung nhịp, tăng hơn ba lần so cách tổ chức với bộ nhớ và bus một từ.

4.3 ĐO LƯỜNG VÀ CẢI TIẾN HIỆU SUẤT CACHE

Trong phần này, chúng ta sẽ giới thiệu về cách đo lường và phân tích hiệu suất của bộ nhớ cache. Sau đó, trình bày các kỹ thuật để cải tiến nhằm nâng cao hiệu suất cache.

Chúng ta có thể tính thời gian CPU (*CPU time*) dựa vào số chu kỳ xung nhịp thực thi CPU (*CPU execution clock cycles*), số chu kỳ trì hoãn bộ nhớ (*Memory-stall clock cycles*) và thời gian chu kỳ xung nhịp (*clock cycle time*). Giả sử rằng thời gian truy cập thành công cache là một phần của thời gian thực thi chương trình. Như vậy:

$$\text{Thời gian CPU} = \left(\frac{\text{Số chu kỳ xung nhịp thực thi CPU}}{\text{nhịp thực thi CPU}} + \frac{\text{Số chu kỳ trì hoãn bộ nhớ}}{\text{hoãn bộ nhớ}} \right) \times \text{Thời gian chu kỳ xung nhịp}$$

Thời gian trì hoãn do bộ nhớ chính là thất bại cache. Để đơn giản chúng ta giả sử xem xét với mô hình bộ nhớ đơn giản. Trong hệ thống CPU thực thi số chu kỳ trì hoãn bộ nhớ có thể phức tạp hơn. Số chu kỳ trì hoãn bộ nhớ này được định nghĩa là số chu kỳ trì hoãn đọc (*read-stall cycles*) và số chu kỳ trì hoãn ghi (*write-stall cycles*):

$$\begin{aligned} \text{Số chu kỳ trì hoãn bộ nhớ} &= \text{số chu kỳ trì hoãn đọc} + \text{số chu kỳ trì hoãn ghi} \end{aligned}$$

Số chu kỳ trì hoãn đọc được xác định dựa vào số lần đọc (*reads*), chương trình (*program*), trừng phạt thất bại đọc (*read miss penalty*) và tỷ lệ thất bại đọc (*read miss rate*):

$$\text{Số chu kỳ trì hoãn đọc} = \frac{\text{số lần đọc}}{\text{chương trình}} \times \text{tỷ lệ thất bại đọc} \times \text{trừng phạt thất bại đọc}$$

Đối với số chu kỳ trì hoãn ghi thì được xác định phức tạp hơn so với đọc. Theo mô hình ghi đồng thời (*write-through*) thì số chu kỳ này được tính dựa vào: thất bại ghi (thao tác này đòi hỏi xác định khối trước khi ghi) và thời gian trì hoãn ghi đệm (*write buffer stalls*) do bộ nhớ đệm đầy khi thực hiện thao tác ghi. Do đó:

$$\begin{aligned} \text{Số chu kỳ trì hoãn ghi} &= \left(\frac{\text{số lần ghi}}{\text{chương trình}} \times \text{tỷ lệ thất bại ghi} \times \text{trừng phạt thất bại ghi} \right) \\ &+ \text{thời gian trì hoãn ghi đệm} \end{aligned}$$

Trong hệ thống với độ rộng bộ đệm ghi hợp lý (bốn từ hoặc lớn hơn) và tần số ghi trung bình của chương trình không vượt quá khả năng ghi vào bộ nhớ thì thời gian trì hoãn ghi đệm là tương đối nhỏ và có thể bỏ qua.

Trong hệ thống bộ nhớ tổ chức theo mô hình ghi đồng thời thì thời gian trừng phạt ghi và đọc là như nhau (thời gian để tìm khối từ bộ nhớ). Giả sử bỏ qua thời gian trì hoãn ghi đệm (*write buffer stalls*), chúng ta có thể gộp chung tỷ lệ thất bại (*miss rate*) và trừng phạt thất bại (*miss penalty*) của thao tác đọc và ghi thì ta được công thức tính số chu kỳ trì hoãn bộ nhớ như sau:

$$\text{Số chu kỳ trì hoãn bộ nhớ} = \frac{\text{số lần truy cập bộ nhớ}}{\text{chương trình}} \times \text{tỷ lệ thất bại} \times \text{trừng phạt thất bại}$$

hay:

$$\text{Số chu kỳ trì hoãn bộ nhớ} = \frac{\text{số lệnh}}{\text{chương trình}} \times \frac{\text{số lần thất bại}}{\text{lệnh}} \times \text{trừng phạt thất bại}$$

Xét một ví dụ cụ thể như sau: xét một hệ thống tỷ lệ thất bại (*miss rate*) của cache lệnh (*instruction cache*) là 2% và của cache dữ liệu (*data cache*) là 4%. Giả sử CPU trong trường hợp không có trì hoãn bộ nhớ có CPI là 2 và trừng phạt thất bại cho thao tác đọc hoặc ghi là 100 chu kỳ. Hãy xác định CPU nhanh hơn bao nhiêu lần trong trường hợp cache hoàn hảo (*perfect cache* – trường hợp CPU không có trì hoãn do bộ nhớ) so với trường hợp có trì hoãn do bộ nhớ. Giả sử tần số của lệnh *load* và *store* là 36%.

Gọi tổng số lệnh là I (*instruction count*), số chu kỳ trì hoãn bộ nhớ của cache lệnh (*Instruction miss cycles*) sẽ là:

$$I \times 2\% \times 100 = 2.00 \times I$$

Do tần số của các lệnh *load* và *store* là 36% nên số chu kỳ trừng phạt bộ nhớ của các lệnh tham khảo dữ liệu (*data miss cycles*) là:

$$I \times 36\% \times 4\% \times 100 = 1.44 \times I$$

Tổng số chu kỳ trì hoãn do bộ nhớ sẽ là $2.0 I + 1.44 I = 3.44 I$, có nghĩa là nhiều hơn ba chu kỳ trì hoãn bộ nhớ trên mỗi lệnh. Do đó CPI trong trường hợp bao gồm trì hoãn do bộ nhớ sẽ là $2 + 3.44 = 5.44$. Tỷ số thời gian thực thi CPU là:

$$\begin{aligned} \frac{\text{Thời gian thực thi (trì hoãn)}}{\text{Thời gian thực thi (cache hoàn hảo)}} &= \frac{I \times \text{CPI}_{\text{trì hoãn}} \times \text{chu kỳ xung nhịp}}{I \times \text{CPI}_{\text{hoàn hảo}} \times \text{chu kỳ xung nhịp}} \\ &= \frac{\text{CPI}_{\text{trì hoãn}}}{\text{CPI}_{\text{hoàn hảo}}} = \frac{5.44}{2} \end{aligned}$$

Nói cách khác, hiệu suất với cache hoàn hảo thì tốt hơn: $\frac{5.44}{2} = 2.72$

Ngoài ra, để đánh giá hiệu suất cache người ta còn dùng thời gian truy cập bộ nhớ trung bình (*average memory access time* – AMAT). Đại lượng này được định nghĩa bằng công thức sau:

$$\text{AMAT} = \text{Thời gian truy cập thành công} + \text{Tỷ lệ thất bại} \times \text{Trừng phạt thất bại}$$

Để hiểu rõ hơn ta xét ví dụ sau: một hệ thống với CPU có chu kỳ xung nhịp là 1ns, trừng phạt thất bại là 20 chu kỳ xung nhịp, tỷ lệ thất bại là 0.05 thất bại trên một lệnh và thời gian truy cập cache (bao gồm xác định thành công) là 1 chu kỳ xung nhịp. Giả sử trừng phạt thất bại đọc và ghi là như nhau. Hãy tính AMAT.

Thời gian truy cập bộ nhớ trung bình sẽ là:

$$\begin{aligned} \text{AMAT} &= \frac{\text{Thời gian truy cập thành công}}{\text{cập thành công}} + \text{Tỷ lệ thất bại} \times \text{Trừng phạt thất bại} \\ &= 1 + 0.05 \times 20 = 2 \text{ chu kỳ xung nhịp} \end{aligned}$$

hay 2 ns.

4.3.1 Thay thế khối

Trong trường hợp thất bại cache, chúng ta có ba mô hình để thay thế khối trong cache là: ánh xạ trực tiếp (*direct mapped*), kết hợp hoàn toàn (*fully associative*) và kết hợp theo tập hợp (*set associative*).

– Mô hình ánh xạ trực tiếp: mỗi khối được đặt chính xác vào một vị trí trong cache theo công thức sau:

$$(\text{block number}) \bmod (\text{number of blocks in the cache})$$

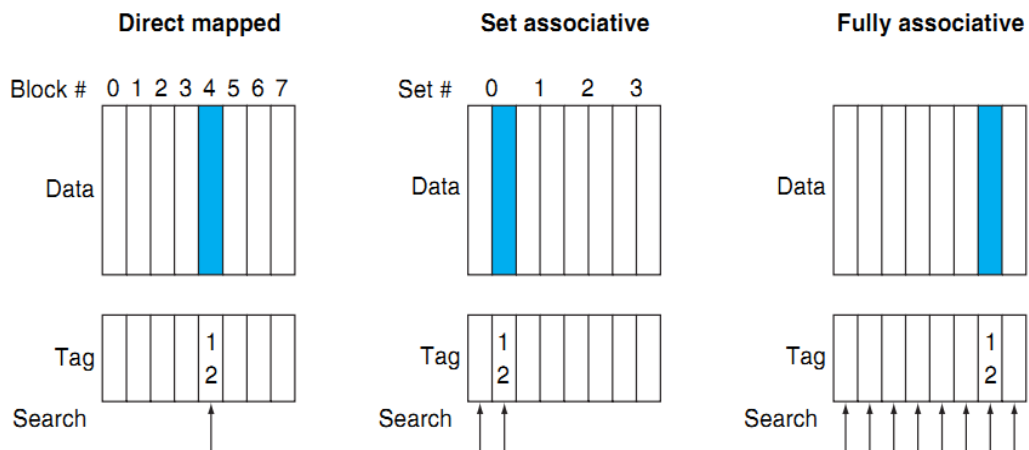
– Kết hợp hoàn toàn: mỗi khối được đặt vào bất kỳ vị trí nào trong cache. Do đó để tìm khối trong cache phải thực hiện song song so sánh thẻ (*tag*) với tất cả các vị trí trong cache. Điều này làm tăng thêm chi phí. Vì vậy, mô hình này chỉ thích hợp với cache có số khối nhỏ.

– Kết hợp theo tập hợp: mô hình này sẽ chia cache thành nhiều tập hợp, mỗi tập hợp có n phần tử bằng nhau, được gọi là kết hợp theo tập hợp có n phần tử (*n-way set associative*). Khối sẽ được đặt vào một trong những vị trí của một tập hợp xác định theo công thức:

$$(\text{số khối}) \bmod (\text{số tập hợp trong cache})$$

Do đó khi xác định khối trong trường hợp này, chúng ta chỉ cần so sánh thẻ (*tag*) với các phần tử trong một tập hợp xác định.

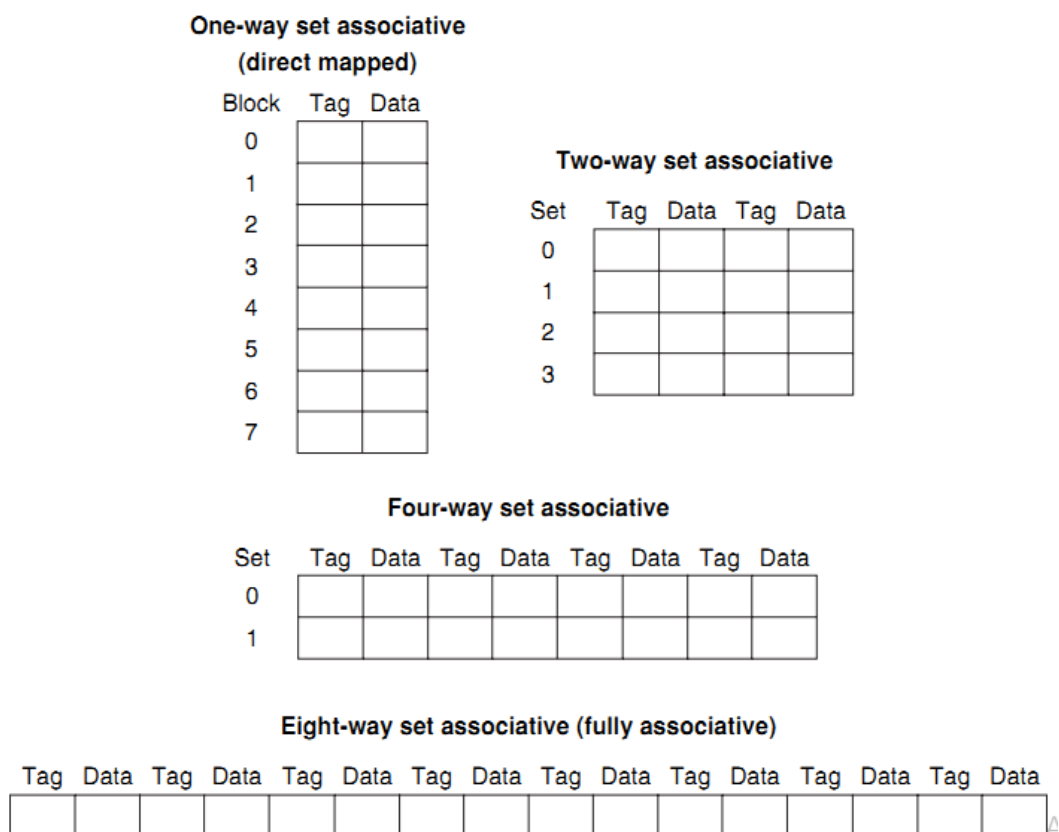
Để hiểu rõ hơn, chúng ta xem ví dụ bộ nhớ cache có 8 khối, khối 12 trong bộ nhớ được đặt vào cache tương ứng theo ba cách như thể hiện trong hình 4.6.



Hình 4.6: Khối 12 được đặt vào cache tương ứng với ba mô hình

Trong mô hình ánh xạ trực tiếp, khối 12 được đặt vào chính xác một vị trí trong cache được xác định theo công thức $(12 \bmod 8) = 4$. Ở mô hình kết hợp theo tập hợp, trong trường hợp này chia thành bốn tập hợp, mỗi tập hợp có hai phần tử. Khối 12 sẽ được đặt vào một trong hai vị trí trong tập hợp có chỉ số là $(12 \bmod 4) = 0$. Còn ở mô hình kết hợp hoàn toàn thì khối 12 có thể đặt bất kỳ vị trí nào trong 8 vị trí trong cache.

Như vậy, mô hình ánh xạ trực tiếp và kết hợp hoàn toàn là một trường hợp đặc biệt của mô hình kết hợp theo tập hợp. Ánh xạ trực tiếp chính là trường hợp kết hợp theo tập hợp một phần tử, tức là chia cache thành n tập hợp (n là tổng số khối của cache) và mỗi tập hợp chỉ có một phần tử. Còn kết hợp hoàn toàn được xem là kết hợp theo tập hợp có n phần tử, cache được chia thành một tập hợp duy nhất và tập hợp này có n phần tử. Hình 4.7 thể hiện bộ nhớ cache có 8 khối được chia thành tập hợp có 1,2,4,8 phần tử tương ứng.



Hình 4.7: Cache 8 khối được chia thành tập hợp có 1,2,4,8 phần tử.

Nếu tăng số phần tử trong tập hợp thì có thể giảm số lần thất bại cache nhưng sẽ làm tăng thời gian thành công cache. Để minh họa điều này ta xét ba bộ nhớ cache có bốn khối (mỗi khối một từ). Cache thứ nhất theo kiểu kết hợp đầy đủ, cache thứ hai theo kiểu kết hợp theo tập hợp (mỗi tập hợp có hai phần tử) và cache thứ ba theo kiểu ánh xạ trực tiếp. Hãy xét số lần thành công và thất bại cache tương ứng với ba mô hình nếu lần lượt truy cập các địa chỉ 0,8,0,6,8.

- Trường hợp ánh xạ trực tiếp: cần xác định vị trí khối được đặt trong cache

Địa chỉ	Khối cache
0	$(0 \bmod 4) = 0$
6	$(6 \bmod 4) = 2$
8	$(8 \bmod 4) = 0$

Như vậy khi lần lượt truy cập các địa chỉ trên thì nội dung của các khối trong cache sẽ là:

Địa chỉ bộ nhớ của khối truy cập	Thành công / Thất bại	Nội dung của khối cache sau khi tham khảo			
		0	1	2	3
0	Thất bại	Memory[0]			
8	Thất bại	Memory[8]			
0	Thất bại	Memory[0]			
6	Thất bại	Memory[0]		Memory[6]	
8	Thất bại	Memory[8]		Memory[6]	

Do đó nếu cache được tổ chức theo mô hình ánh xạ trực tiếp thì sẽ có 5 thất bại cache và không có một truy cập nào là thành công cache.

– Trường hợp cache được tổ chức theo kiểu kết hợp theo tập hợp: chia thành hai tập hợp (mỗi tập hợp có hai phần tử), cần xác định chỉ số tập hợp khi một địa chỉ được truy cập:

Địa chỉ	Tập hợp cache
0	$(0 \bmod 2) = 0$
6	$(6 \bmod 2) = 0$
8	$(8 \bmod 2) = 0$

Khi một địa chỉ khối được ánh xạ vào một chỉ số tập hợp xác định thì ta có một trong hai lựa chọn để thay thế khối (một trong hai vị trí phần tử trong một tập hợp). Thông thường ở mô hình này thì chọn chiến lược thay thế khối ít sử dụng gần đây nhất. Dùng chiến lược này thì nội dung của cache sẽ như sau:

Địa chỉ bộ nhớ của khối truy cập	Thành công / Thất bại	Nội dung của khối cache sau khi tham khảo			
		Tập hợp 0	Tập hợp 0	Tập hợp 1	Tập hợp 1
0	Thất bại	Memory[0]			
8	Thất bại	Memory[0]	Memory[8]		
0	Thành công	Memory[0]	Memory[8]		
6	Thất bại	Memory[0]	Memory[6]		
8	Thất bại	Memory[8]	Memory[6]		

Trong trường hợp này ta có bốn thất bại cache và một thành công cache, giảm một thất bại cache so với mô hình ánh xạ trực tiếp.

– Trường hợp cache được tổ chức theo mô hình kết hợp đầy đủ: một tập hợp đơn có bốn phần tử (khối), do đó có thể đặt vào bất cứ vị trí này trong bốn vị trí khối này. Trong trường hợp này cache sẽ có hiệu suất tốt nhất với chỉ có ba thất bại cache và hai thành công cache.

Địa chỉ bộ nhớ của khối truy cập	Thành công / Thất bại	Nội dung của khối cache sau khi tham khảo			
		Khối 0	Khối 1	Khối 2	Khối 3
0	Thất bại	Memory[0]			
8	Thất bại	Memory[0]	Memory[8]		
0	Thành công	Memory[0]	Memory[8]		
6	Thất bại	Memory[0]	Memory[8]	Memory[6]	
8	Thành công	Memory[0]	Memory[8]	Memory[6]	

4.3.2 Xác định khối trong cache

Để xác định khối trong cache, ta dựa vào mô hình thực hiện trong bộ nhớ cache:

– Kết hợp theo tập hợp: mỗi khối trong cache theo mô hình này sẽ bao gồm ba thành phần: thẻ (*tag*), chỉ số (*index*) và độ dời khối (*block offset*). Chỉ số được dùng để lựa chọn tập hợp xác định, thẻ được so sánh với các khối trong tập hợp đã được xác định. Độ dời khối là địa chỉ của dữ liệu cần truy cập trong khối. Khi CPU đưa ra địa chỉ khối cần truy cập thì địa chỉ này sẽ được phân tích thành ba thành phần như trên, dựa vào chỉ số để xác định được tập hợp xác định có chứa khối cần truy cập. Sau đó thẻ của các khối trong tập hợp sẽ được so sánh để xác định khối cần truy cập. Sự so sánh này được thực hiện song song để đảm bảo tốc độ.

Thẻ	Chỉ số	Độ dời khối
-----	--------	-------------

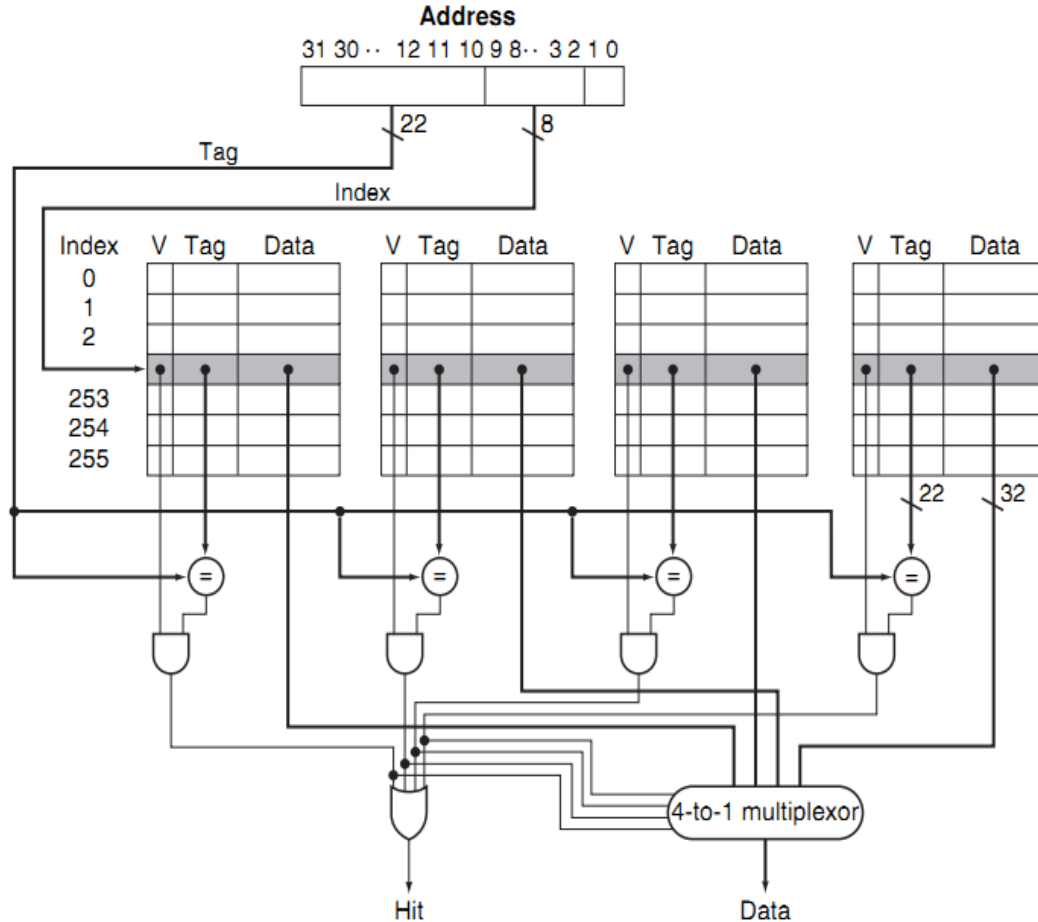
– Ánh xạ trực tiếp: cũng giống như kiểu kết hợp theo tập hợp, mỗi khối trong bộ nhớ cache cũng bao gồm ba phần: thẻ, chỉ số và độ dời khối. Khi CPU đưa ra địa chỉ khối để truy cập thì dựa vào chỉ số để xác định được tập hợp, nhưng tập hợp này chỉ có một phần tử. Do đó chỉ có một sự so sánh đơn thẻ được thực hiện.

Thẻ	Chỉ số	Độ dời khối
-----	--------	-------------

– Kết hợp đầy đủ: trong trường hợp này do chỉ có một tập hợp duy nhất sẽ không có phần chỉ số. Do đó mỗi khối trong cache chỉ có hai thành phần: thẻ và độ dời khối. Khi CPU đưa ra địa chỉ khối để truy cập thì tất cả các thẻ của các khối phải được so sánh song song để xác định khối cần truy cập có trong cache hay không.

Thẻ	Độ dời khối
-----	-------------

Để hiểu rõ hơn cách xác định khối trong bộ nhớ cache, chúng ta xét ví dụ bộ nhớ cache theo mô hình kết hợp theo tập hợp, mỗi tập hợp có bốn phần tử như hình 4.8. Tập hợp được xác định dựa vào chỉ số. Sau đó sẽ có bốn sự so sánh thẻ được thực hiện song song kết hợp với bộ điều hợp 4 ra 1 (*4-to-1 multiplexor*) để chọn một trong bốn khối của tập hợp xác định.



Hình 4.8: Mô hình kết hợp theo tập hợp với tập hợp có bốn phần tử

4.3.3 Thay thế khối

Khi thất bại cache xảy ra thì phải tìm khối trong bộ nhớ chính và sau đó thực hiện thay thế khối này vào bộ nhớ cache. Đối với mô hình ánh xạ trực tiếp thì chỉ có một lựa chọn duy nhất ứng với khối có chỉ số xác định được thay thế. Trong mô hình kết hợp đầy đủ thì có thể lựa chọn một trong tất cả các khối để thay thế. Còn trong mô hình kết hợp theo tập hợp thì có thể lựa chọn một trong các khối trong tập hợp tương ứng với chỉ số để thay thế. Chiến thuật thay thế thường sử dụng là thay thế khối ít được sử dụng gần đây nhất (*least recently used – LRU*). Theo LRU thì khối được thay thế là khối không được sử dụng trong thời gian dài nhất. Chiến thuật LRU được cài đặt bằng cách theo dõi mỗi lần tham khảo đến một phần tử trong tập hợp. Như với tập hợp có hai phần tử thì có thể dùng một bit để theo dõi sự tham khảo đến phần tử trong tập hợp. Bit này được thiết lập để xác định có một tham khảo đến phần tử. Khi số phần tử trong tập hợp tăng lên thì cài đặt LRU sẽ phức tạp hơn.

Xét một ví dụ như sau: giả sử bộ nhớ cache gồm có 4K khối, mỗi khối 4 từ và địa chỉ có kích thước 32 bit. Hãy tìm tổng số tập hợp và tổng số bit của thẻ (*tags*) trong bộ nhớ cache này với các mô hình ánh xạ trực tiếp, kết hợp theo tập hợp có 2 và có 4 phần tử và kết hợp đầy đủ.

Do mỗi khối có 4 từ nên mỗi khối có 16 ($=2^4$) byte, địa chỉ offset sẽ có 4 bit. Còn lại $32 - 4 = 28$ bit dùng cho thẻ (*tag*) và chỉ số (*index*). Bộ nhớ cache theo mô hình ánh xạ trực tiếp sẽ có số tập hợp bằng số khối nên trường chỉ số sẽ dùng 12 bit ($\log_2(4K) = 12$). Tổng số bit của thẻ sẽ bằng $(28 - 12) \times 4K = 16 \times 4K = 64K\text{bit}$.

Khi tăng số phần tử của tập hợp lên gấp đôi thì trường chỉ số sẽ giảm 1 bit đồng thời tăng trường thẻ lên 1 bit. Do đó, nếu kết hợp theo tập hợp có 2 phần tử thì sẽ có 2K tập hợp và tổng số bit của thẻ là $(28 - 11) \times 2 \times 2K = 34 \times 2K = 68K\text{bit}$. Nếu kết hợp theo tập hợp có 4 phần tử thì sẽ có 1K tập hợp và tổng số bit của thẻ là $(28 - 10) \times 4 \times 1K = 72 \times 1K = 72K\text{bit}$. Đối với mô hình kết hợp đầy đủ thì chỉ có 1 tập hợp duy nhất với 4K khối, trường thẻ sẽ là 28 bit và tổng số bit của thẻ sẽ bằng $28 \times 4K \times 1 = 112K\text{bit}$. Như vậy khi tăng số phần tử của tập hợp thì sẽ làm tăng số lượng phép toán so sánh và tăng dung lượng để lưu trữ của thẻ.

4.3.4 Cache nhiều mức (*Multilevel Caches*)

Để giảm trùng phạt thất bại thì một khuynh hướng được sử dụng đó là dùng nhiều mức cache. Hầu hết các bộ xử lý hiện nay đều bổ sung thêm một cache mức 2 (còn gọi là cache thứ cấp – *secondary cache*). Cache mức 2 cũng nằm trên bộ xử lý và được truy cập bất cứ khi nào một thất bại xảy ra ở cache mức 1 (còn gọi là cache sơ cấp – *primary cache*). Nếu cache mức 2 có chứa khối dữ liệu cần truy cập này thì trùng phạt thất bại của cache mức 1 sẽ bằng thời gian truy cập ở cache mức 2. Thời gian này sẽ nhỏ hơn thời gian truy cập vào bộ nhớ chính. Nếu cả hai cache mức 1 và cache mức 2 đều không chứa khối dữ liệu cần truy cập trùng phạt thất bại sẽ là thời gian truy cập vào bộ nhớ chính. Trùng phạt thất bại trong trường hợp này sẽ lớn hơn so với trùng phạt thất bại ở hai mức cache.

Để hiểu rõ hơn hiệu quả của việc sử dụng thêm một cache mức 2, chúng ta hãy xét ví dụ như sau: Giả sử chúng ta có bộ xử lý với CPI cơ sở (*base CPI*) là 1.0, giả sử tất cả truy cập thành công trong cache sơ cấp và có tỷ lệ xung nhịp (*clock rate*) là 4GHz. Giả sử thời gian truy cập bộ nhớ chính là 100 ns, bao gồm tất cả xử lý thất bại. Giả sử tỷ lệ thất bại trên lệnh ở cache sơ cấp là 2%. Hỏi bộ xử lý sẽ nhanh hơn bao nhiêu lần nếu chúng ta bổ sung thêm cache mức 2 có thời gian truy cập là 5 ns cho cả trường hợp thành công cũng như thất bại và cache này đủ lớn để có thể giảm tỷ lệ thất bại của bộ nhớ chính là 0.5%?

Ta có trùng phạt thất bại đối với bộ nhớ chính là:

$$\frac{100 \text{ ns}}{0.25 \frac{\text{ns}}{\text{clock cycle}}} = 400 \text{ clock cycles}$$

Tổng CPI (*total CPI*) trong trường hợp có một mức cache là:

Tổng CPI = CPI cơ sở + số chu kỳ trì hoãn bộ nhớ mỗi lệnh

hay:

$$\text{Tổng CPI} = 1.0 + 2\% \times 400 = 9$$

Trong trường hợp có hai mức cache, một thất bại trong cache mức 1 (cache sơ cấp) thì có thể tìm khối này trong cache mức 2 (cache thứ cấp) hoặc trong bộ nhớ chính. Trừng phạt thất bại đối với một truy cập của cache mức 2 sẽ là:

$$\frac{5 \text{ ns}}{0.25 \frac{\text{ns}}{\text{clock cycle}}} = 20 \text{ clock cycles}$$

Nếu một thất bại được tìm thấy trong cache mức 2 thì khi đó tổng trừng phạt thất bại sẽ là trừng phạt thất bại ở cache mức 2. Nếu thất bại phải cần tìm đến bộ nhớ thì tổng trừng phạt thất bại sẽ bằng tổng thời gian truy cập ở cache mức 2 và thời gian truy cập ở bộ nhớ.

Do đó, đối với trường hợp có hai mức cache thì tổng CPI sẽ bằng tổng số chu kỳ trì hoãn (*stall cycles*) của cả hai mức cache và CPI cơ sở:

$$\begin{aligned} \text{Tổng CPI} &= 1 + \text{số chu kỳ trì hoãn sơ cấp} + \text{số chu kỳ trì hoãn thứ cấp} \\ &= 1 + 2\% \times 20 + 0.5\% \times 400 = 1 + 0.4 + 2.0 = 3.4 \end{aligned}$$

Vì thế, CPU với cache thứ cấp sẽ nhanh hơn:

$$\frac{9.0}{3.4} = 2.6$$

Như vậy với cấu trúc tổ chức cache hai mức thì cache mức sơ cấp có tác dụng làm tối thiểu hóa thời gian truy cập thành công trong khi cache thứ cấp sẽ tập trung đến tỷ lệ thất bại để giảm trừng phạt thời gian truy cập bộ nhớ lâu.

4.4 TÓM TẮT

Chương này giới thiệu nguyên tắc thời gian và nguyên tắc không gian trong truy cập bộ nhớ. Trình bày đặc điểm trong cấp bậc bộ nhớ. Sau đó giới thiệu tổng quan về bộ nhớ cache, truy cập dữ liệu trong bộ nhớ cache, các trường hợp để xử lý thất bại cache. Thao tác ghi dữ liệu vào bộ nhớ cache: ghi đồng thời, ghi đệm, ghi lại. Thiết kế bộ nhớ để vận hành cache tốt hơn. Giới thiệu ba mô hình để thay thế khối: ánh xạ trực tiếp, kết hợp hoàn toàn và kết hợp theo tập hợp. Cách để xác định khối trong bộ nhớ cache dựa vào từng mô hình được thực hiện trong cache. Cuối cùng trình bày cache nhiều mức: cache mức 1 (cache sơ cấp) và cache mức 2 (cache thứ cấp) để đảm bảo hiệu suất vận hành cache tốt hơn.

CÂU HỎI VÀ BÀI TẬP CHƯƠNG 4

1. Nguyên tắc thời gian là gì? Nguyên tắc không gian là gì?
2. Các cấp bộ nhớ được hiểu như thế nào?
3. Mô hình ánh xạ trực tiếp là gì?
4. Mô hình kết hợp theo tập hợp là gì?
5. Cách xác định khối trong bộ nhớ Cache?
6. Chiến thuật LRU là gì?
7. Cho bộ nhớ cache có 4 khối (mỗi khối một từ) theo kiểu ánh xạ trực tiếp. Hãy cho biết nội dung của bộ nhớ cache và số lần thành công hoặc thất bại nếu lần lượt truy cập các địa chỉ 0, 15, 6, 0, 6, 20 với giả sử bộ nhớ cache được khởi tạo ban đầu rỗng (không chứa bất kỳ nội dung nào).
8. Cho bộ nhớ cache có 8 khối (mỗi khối một từ) theo kiểu kết hợp theo tập hợp (mỗi tập hợp có hai phần từ) với chiến lược thay thế khối ít được sử dụng gần đây nhất (*least recently used* – LRU). Hãy cho biết nội dung của bộ nhớ cache và số lần thành công hoặc thất bại nếu lần lượt truy cập các địa chỉ 2, 5, 12, 2, 8, 5 với giả sử bộ nhớ cache được khởi tạo ban đầu rỗng (không chứa bất kỳ nội dung nào).
9. Cho bộ nhớ cache có 4 khối (mỗi khối một từ) theo kết hợp đầy đủ. Hãy cho biết nội dung của bộ nhớ cache và số lần thành công hoặc thất bại nếu lần lượt truy cập các địa chỉ 0, 5, 8, 5, 6, 8 với giả sử bộ nhớ cache được khởi tạo ban đầu rỗng (không chứa bất kỳ nội dung nào).
10. Xét một ví dụ như sau: giả sử bộ nhớ cache gồm có 512 khối, mỗi khối 2 từ và địa chỉ có kích thước 32 bit. Hãy tìm tổng số tập hợp và tổng số bit của thẻ (tags) trong bộ nhớ cache này với mô hình ánh xạ trực tiếp.
11. Xét một ví dụ như sau: giả sử bộ nhớ cache gồm có 2K khối, mỗi khối 4 từ và địa chỉ có kích thước 32 bit. Hãy tìm tổng số tập hợp và tổng số bit của thẻ (tags) trong bộ nhớ cache này với mô hình kết hợp theo tập hợp có 2 phần từ.
12. Xét một ví dụ như sau: giả sử bộ nhớ cache gồm có 4K khối, mỗi khối 8 từ và địa chỉ có kích thước 32 bit. Hãy tìm tổng số tập hợp và tổng số bit của thẻ (tags) trong bộ nhớ cache này với mô hình kết hợp đầy đủ.

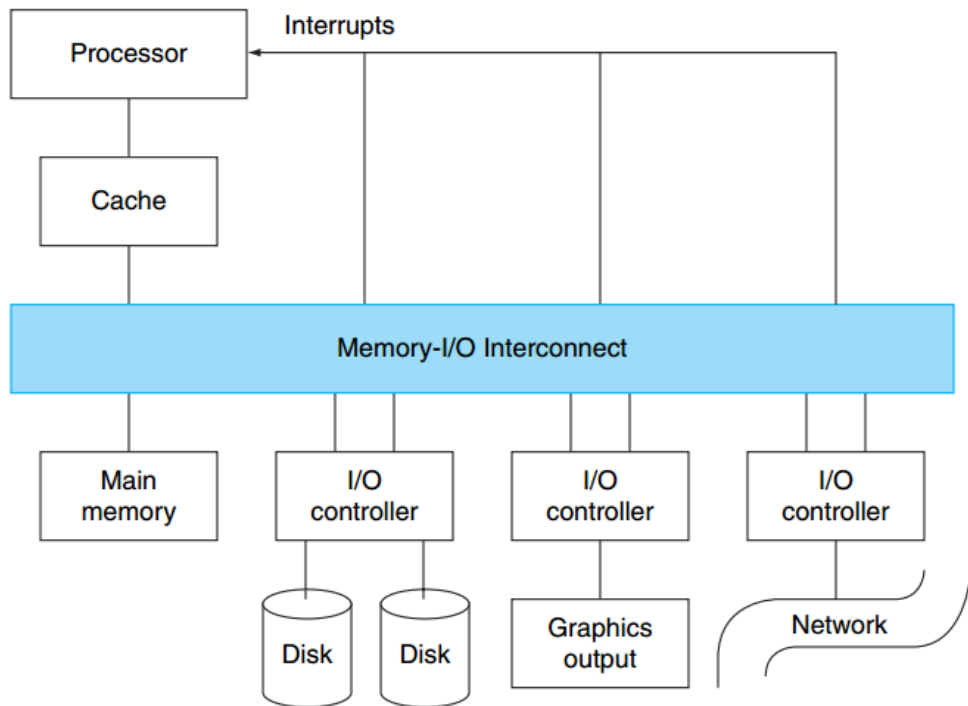
CHƯƠNG 5

HỆ THỐNG LƯU TRỮ VÀ NHẬP – XUẤT

Mục đích: Giới thiệu tổng quan về hệ thống lưu trữ và nhập – xuất trong máy tính. Cấu tạo và nguyên lý hoạt động của đĩa từ và bộ nhớ flash. Cách kết nối giữa bộ xử lý, bộ nhớ và thiết bị nhập – xuất. Giao tiếp thiết bị nhập – xuất với bộ xử lý, bộ nhớ và hệ điều hành.

5.1 GIỚI THIỆU

Một hệ thống nhập xuất điển hình được thể hiện như hình 5.1. Đường kết nối giữa thiết bị nhập/xuất (I/O), bộ xử lý và bộ nhớ thì được gọi là **bus**. Kết nối giữa thiết bị nhập/xuất thông qua các đường kết nối ngắt (*interrupts*) và các đường kết nối bên trong (*interconnect*). Các tiêu chí về độ tin cậy và khả năng mở rộng thì được quan tâm khi thiết kế hệ thống nhập/xuất.



Hình 5.1: Hệ thống nhập/xuất điển hình

5.2 ĐĨA TỪ

Đĩa từ bao gồm nhiều lá đĩa (1 – 4) và đầu đọc/ghi có thể di chuyển để truy cập đĩa. Mỗi lá đĩa có hai mặt được dùng để ghi dữ liệu. Tốc độ quay của lá đĩa từ 5400 đến 15000 rpm và có đường kính từ 1 inch đến 3.5 inch. Mỗi bề mặt đĩa được chia thành các hình tròn đồng tâm, được gọi là rãnh (*track*). Có khoảng 10000 đến 50000 rãnh trên một bề mặt. Mỗi rãnh được chia thành các cung (*sector*) được dùng để lưu dữ liệu. Mỗi rãnh chứa từ 100 đến 500 cung. Thông thường cung có kích thước 512 byte. Tập hợp các rãnh có cùng kích thước thì được gọi là *cylinder*.

Để truy cập dữ liệu, đầu tiên phải di chuyển đầu đọc/ghi đến rãnh có chứa cung mong muốn. Thao tác này được gọi là một tìm kiếm (*seek*), thời gian để đầu đọc/ghi di chuyển đến rãnh đọc gọi là thời gian tìm kiếm (*seek time*). Thông thường thời gian này là khoảng từ 3ms đến 13ms. Sau khi di chuyển đến rãnh thích hợp, đầu đọc/ghi phải đợi lá đĩa quay đến cung chứa dữ liệu cần truy suất. Thời gian này được gọi là độ trễ quay (*rotational delay* hay *rotational latency*). Giá trị này trung bình bằng nửa vòng quay của đĩa. Tức là nếu tốc độ quay của đĩa từ 5400 rpm đến 15000 rpm thì:

$$\text{Average rotational latency} = \frac{0.5 \text{ rotation}}{5400 \text{ RPM}} = \frac{0.5 \text{ rotation}}{5400 \text{ RPM} / \left(60 \frac{\text{seconds}}{\text{minute}}\right)}$$

$$= 0.0056 \text{ seconds} = 5.6 \text{ ms}$$

and

$$\text{Average rotational latency} = \frac{0.5 \text{ rotation}}{15,000 \text{ RPM}} = \frac{0.5 \text{ rotation}}{15,000 \text{ RPM} / \left(60 \frac{\text{seconds}}{\text{minute}}\right)}$$

$$= 0.0020 \text{ seconds} = 2.0 \text{ ms}$$

Thời gian tiếp theo để truy cập đĩa, đó là thời gian truyền (*transfer time*). Đây là thời gian được dùng để truyền dữ liệu. Thời gian này phụ thuộc vào kích thước cung, tốc độ quay và mật độ ghi dữ liệu của rãnh. Tốc độ truyền khoảng từ 70 đến 125 MB/s. Ngày nay, trình điều khiển đĩa có xây dựng sẵn bộ nhớ đệm (*cache*) để lưu tạm thời các cung. Do đó tốc độ truyền từ bộ nhớ đệm có thể tăng lên đến 375 MB/s.

Ngoài ra, trình điều khiển đĩa (*disk controller*) sẽ xử lý các điều khiển chi tiết của đĩa và truyền dữ liệu giữa đĩa và bộ nhớ. Thời gian này được gọi là thời gian điều khiển (*controller time*). Đây là thành phần cuối cùng trong thời gian truy cập đĩa. Như vậy, thời gian truy cập đĩa sẽ bao gồm thời gian tìm kiếm, độ trễ quay, thời gian truyền và thời gian điều khiển.

Ví dụ: tính thời gian truy cập đĩa để đọc hoặc ghi một cung (512 byte) của một đĩa có tốc độ quay 15000 rpm. Cho biết thời gian tìm kiếm là 4 ms, tốc độ truyền 100 MB/s và thời gian điều khiển là 0.2 ms.

Ta có: thời gian truy cập = thời gian tìm kiếm + độ trễ quay + thời gian truyền + thời gian điều khiển. Như vậy, giá trị này bằng:

$$4.0 \text{ ms} + \frac{0.5 \text{ rotation}}{15,000 \text{ RPM}} + \frac{0.5 \text{ KB}}{100 \text{ MB/sec}} + 0.2 \text{ ms} = 4.0 + 2.0 + 0.005 + 0.2 = 6.2 \text{ ms}$$

5.3 BỘ NHỚ FLASH

Bộ nhớ flash là một loại bộ nhớ bán dẫn giống như đĩa từ nhưng có độ trễ (*latency*) nhanh hơn 100 đến 1000 lần so với đĩa. Đây là bộ nhớ nhỏ gọn, tiêu thụ ít năng lượng và có sức chống va chạm cao. Bộ nhớ này được sử dụng phổ biến trong điện thoại, thiết bị số, máy quay phim,... Về giá thành đối với mỗi gigabyte thì cao hơn từ 2 đến 40 lần so với đĩa cứng và thấp hơn 5 đến 10 lần so với DRAM.

Bộ nhớ flash có nguyên lý hoạt động dựa trên EEPROM và được phân thành hai loại: NOR flash có ô nhớ dùng cổng NOR và tương tự là NAND flash. NOR flash dùng để chứa các chương trình khởi động của thiết bị trong khi NAND flash được dùng trong thẻ nhớ usb và có giá thành rẻ hơn NOR flash. Với sự phát triển nhanh về dung lượng cũng như giá thành ngày càng thấp làm cho bộ nhớ flash ngày càng được sử dụng rộng rãi trong nhiều loại thiết bị ngày nay.

5.4 KẾT NỐI GIỮA BỘ XỬ LÝ, BỘ NHỚ VÀ THIẾT BỊ NHẬP/XUẤT

Trong một hệ thống máy tính, các thiết bị như bộ xử lý và bộ nhớ cần được kết nối với nhau cũng như giữa bộ xử lý và thiết bị nhập/xuất. Bộ phận đảm nhận vai trò này chính là bus. Bus bao gồm tập hợp các sợi dây để kết nối các hệ thống con bên trong. Có hai thuận lợi của bus đó là tính linh hoạt và có chi phí thấp. Với một đường kết nối đơn các thiết bị mới có thể được gắn thêm vào. Các thiết bị ngoại vi có thể dễ dàng gắn vào một máy tính này hay máy tính khác có cùng loại bus. Điểm bất lợi chính của bus chính là tạo ra hiện tượng thắt cổ chai (*bottleneck*). Vấn đề này làm hạn chế lưu lượng nhập/xuất. Khi lưu lượng nhập/xuất được truyền trên bus, băng thông của bus sẽ giới hạn mức độ tối đa của lưu lượng nhập/xuất này. Thiết kế một hệ thống bus đáp ứng nhu cầu của bộ xử lý cũng như khả năng kết nối với nhiều loại thiết bị nhập/xuất là một vấn đề khó khăn đối với các nhà thiết kế máy tính.

Bus được chia thành hai loại: loại thứ nhất là bus hệ thống dùng để kết nối giữa bộ xử lý và bộ nhớ, bus này thường ngắn và có tốc độ cao. Loại thứ hai là bus nhập/xuất, loại này thường dài và có thể kết nối với nhiều loại thiết bị và thông thường băng thông dữ liệu (*data bandwidth*) bằng với thiết bị được gắn vào bus. Hơn nữa, tốc độ của bus ngoại vi bị giới hạn bởi các yếu tố vật lý: chiều dài của bus và số thiết bị được gắn vào.

Một giao dịch nhập/xuất (*I/O transaction*) bao gồm có hai phần: gửi địa chỉ và nhận hoặc gửi dữ liệu. Thao tác này được thực hiện dựa trên bộ nhớ, tức là một thao tác nhập (*input*) là nhập dữ liệu từ thiết bị nhập/xuất vào bộ nhớ (bộ xử lý có thể đọc dữ liệu này). Trong khi thao tác xuất (*output*) là xuất dữ liệu tới thiết bị nhập/xuất từ bộ nhớ (bộ xử lý có thể ghi dữ liệu này). Để thống nhất giữa nhà sản xuất máy tính và nhà sản xuất thiết bị ngoại vi, ngành công nghiệp máy tính đưa ra các tiêu chuẩn. Một tiêu chuẩn đưa ra các đặc tả để các thiết bị ngoại vi của các nhà sản xuất khác nhau có thể được gắn vào một máy tính theo chuẩn đó. Một số tiêu chuẩn của thiết bị nhập/xuất phổ biến như: USB, PCI Express (PCIe), Serial ATA (SATA), SCSI,...

Bus có thể là đồng bộ (*synchronous*) tức là bus gồm có tín hiệu xung nhịp (*clock*) trong đường điều khiển và một giao thức (*protocol*) cố định để giao tiếp dựa trên tín hiệu xung nhịp đó. Ví dụ để thực hiện một thao tác đọc từ bộ nhớ, chúng ta có nghi thức để truyền địa chỉ và lệnh đọc ở chu kỳ đầu tiên. Dùng đường điều khiển để chỉ định loại yêu cầu. Sau đó, bộ nhớ sẽ đáp ứng yêu cầu một từ (*word*) dữ liệu ở chu kỳ thứ 5. Loại giao thức này có thể được thực thi bằng máy trạng thái hữu hạn (*finite-state machine*) nhỏ. Tuy nhiên, bus đồng bộ có hai bất lợi chính. Thứ nhất, mỗi thiết bị trên bus phải chạy cùng tần số xung nhịp (*clock rate*). Thứ hai, vấn đề lệch xung

nhip, bus đồng bộ không thể dài nếu bus này có tốc độ nhanh. Do những bất lợi trên nên dẫn đến hình thành bus không đồng bộ (*asynchronous*). Loại bus này không có tín hiệu xung nhịp. Điều này thích hợp để kết nối với nhiều loại thiết bị khác nhau và bus có thể dài hơn mà không lo lắng về vấn đề lệnh xung nhịp và đồng bộ. Để thực hiện việc truyền dữ liệu, bus không đồng bộ dùng giao thức bắt tay (*handshaking protocol*). Giao thức này bao gồm tập hợp nhiều bước để bên gửi và bên nhận xử lý ở bước tiếp theo khi cả hai bên đồng ý với nhau. Các đường điều khiển cũng được bổ sung trong giao thức này.

5.5 GIAO TIẾP THIẾT BỊ NHẬP/XUẤT VỚI BỘ XỬ LÝ, BỘ NHỚ VÀ HỆ ĐIỀU HÀNH

Hệ điều hành đóng vai trò quan trọng trong xử lý các yêu cầu nhập/xuất, như là giao diện giữa phần cứng và chương trình yêu cầu nhập/xuất. Vai trò của hệ điều hành liên quan đến đặc điểm của hệ thống nhập/xuất:

1. Nhiều chương trình dùng bộ xử lý để chia sẻ hệ thống nhập/xuất.
2. Hệ thống nhập/xuất thường dùng ngắt (*interrupt*) để giao tiếp thông tin về thao tác nhập/xuất. Ngắt được truyền tới nhân (*kernel*) và được xử lý bởi hệ điều hành.
3. Điều khiển cấp thấp của thiết bị nhập/xuất thì phức tạp, do bởi phải quản lý tập hợp các sự kiện đồng thời và những yêu cầu cần thực hiện chính xác thường rất chi tiết.

Dựa vào các đặc điểm trên, hệ điều hành cung cấp một số chức năng như sau:

- Hệ điều hành phải đảm bảo chương trình của người dùng chỉ được phép truy cập vào một phần của thiết bị nhập/xuất mà người dùng được phép. Ví dụ hệ điều hành không cho phép chương trình đọc hoặc ghi một tập tin trên đĩa nếu chương trình này không được phép để truy cập tập tin đó.
- Hệ điều hành cung cấp các hàm (*routine*) để xử lý các thao tác cấp thấp của thiết bị nhập/xuất.
- Hệ điều hành phải xử lý các ngắt được tạo ra bởi thiết bị nhập/xuất, cũng như xử lý các ngoại lệ (*exception*) được tạo ra bởi chương trình.
- Đối với tài nguyên nhập/xuất được chia sẻ, hệ điều hành cần cung cấp quyền truy cập như nhau cũng như lập thời biểu các truy cập để đảm bảo nâng cao thông lượng (*throughput*) của hệ thống.

Để thực hiện các chức năng này, hệ điều hành phải giao tiếp với thiết bị nhập/xuất và ngăn chương trình người dùng giao tiếp trực tiếp với thiết bị nhập/xuất. Có ba loại giao tiếp:

1. Hệ điều hành ra lệnh cho thiết bị ngoại vi. Những lệnh này không chỉ là thao tác đọc hoặc ghi mà còn là những thao tác khác được thực hiện trên thiết bị như một thao tác dịch chuyển đầu đọc/ghi đến vị trí thích hợp (*seek*) của đĩa.
2. Thiết bị nhập/xuất sẽ thông báo cho hệ điều hành khi thực hiện xong một thao tác hoặc có một lỗi xảy ra.
3. Dữ liệu được truyền giữa bộ nhớ và thiết bị nhập/xuất.

Trong phần tiếp theo, chúng ta sẽ tìm hiểu kỹ hơn về giao tiếp với thiết bị nhập/xuất.

5.5.1 Ra lệnh cho thiết bị nhập xuất

Để điều khiển thiết bị nhập/xuất, bộ xử lý phải xác định địa chỉ thiết bị và đưa ra một hoặc nhiều từ lệnh. Hai phương pháp được dùng để xác định địa chỉ: thiết bị nhập/xuất được ánh xạ vào bộ nhớ (*memory-mapped I/O*) và tập lệnh nhập/xuất riêng (*special I/O instructions*).

➤ Thiết bị nhập/xuất được ánh xạ vào bộ nhớ: một vùng không gian địa chỉ được gán cho thiết bị nhập/xuất. Đọc và ghi vào vùng địa chỉ này tương ứng với giao tiếp với thiết bị nhập/xuất. Ví dụ thao tác ghi dữ liệu vào thiết bị nhập/xuất, bộ xử lý sẽ đặt địa chỉ và dữ liệu vào bus. Hệ thống bộ nhớ sẽ bỏ qua thao tác này bởi vì địa chỉ chỉ định phần không gian bộ nhớ dành cho thiết bị nhập/xuất. trình điều khiển thiết bị sẽ thực hiện thao tác bằng cách truyền dữ liệu này tới thiết bị nhập/xuất xem như một lệnh.

Đọc hoặc ghi dữ liệu của một chương trình thường yêu cầu một vài thao tác nhập/xuất riêng biệt. Hơn nữa, bộ xử lý phải tham dò trạng thái của thiết bị để xem lệnh đưa ra có thực hiện xong hay chưa. Ví dụ, một máy in đơn giản có hai thanh ghi nhập/xuất: một thanh ghi trạng thái lưu trạng thái của thiết bị và một thanh ghi dữ liệu lưu dữ liệu sẽ được in. Thanh ghi trạng thái chứa một bit đã thực hiện (*done bit*) được thiết lập bởi máy in khi thực hiện in xong một ký tự và một bit lỗi (*error bit*). Bit lỗi này xác định rằng máy in đang kẹt giấy hoặc hết giấy. Mỗi byte dữ liệu cần in được đặt vào thanh ghi dữ liệu. Bộ xử lý phải đợi cho đến khi máy in thiết lập bit đã thực hiện thì mới đưa ký tự tiếp theo để in. Bộ xử lý cũng cần kiểm tra bit lỗi để xem lỗi xảy ra hay không.

➤ Tập lệnh nhập/xuất riêng: những lệnh nhập/xuất này sẽ xác định số thiết bị và từ lệnh. Bộ xử lý giao tiếp với thiết bị thông qua các sợi dây của bus nhập/xuất. Lệnh được truyền trên các đường dữ liệu của bus. Diễn hình của máy tính theo kiểu này đó là Intel $\times 86$ và IBM 370.

5.5.2 Giao tiếp với bộ xử lý

Thiết bị nhập/xuất ghi thông tin vào thanh ghi trạng thái (*status register*). Bộ xử lý định kỳ kiểm tra thông tin của thanh ghi trạng thái để biết thiết bị có sẵn sàng hay không. Quá trình này được gọi là thăm dò (*polling*). Quá trình này có điểm bất lợi là tiêu tốn nhiều thời gian của bộ xử lý do phải thường xuyên thăm dò trạng thái của thiết bị. Hơn nữa, tốc độ của bộ xử lý thì nhanh hơn nhiều so với thiết bị nhập/xuất. Do hạn chế của quá trình thăm dò nên người ta sử dụng ngắt (*interrupt*) để thiết bị nhập/xuất thông báo với bộ xử lý. Một ngắt nhập/xuất (*I/O interrupt*) cũng giống như một ngoại lệ (*exception*) với hai khác biệt quan trọng:

1. Ngắt nhập/xuất thì không đồng bộ với sự thực thi lệnh. Nghĩa là, ngắt không kết hợp với bất kỳ lệnh nào và cũng không cản trở thực thi hoàn thành một lệnh. Điều này khác với một ngoại lệ lỗi trang (*page fault exception*) hay ngoại lệ tràn số

học (*arithmetic overflow*). Đơn vị điều khiển sẽ kiểm tra ngắt nhập/xuất chưa được giải quyết lúc bắt đầu một lệnh mới.

2. Khi một ngắt xảy ra, chúng ta thích truyền thông tin hơn. Như là định danh (*identity*) thiết bị tạo ra ngắt. Hơn nữa, ngắt của mỗi thiết bị nhập/xuất có độ ưu tiên khác nhau. Do đó mức độ khẩn cấp kết hợp với mỗi ngắt của thiết bị sẽ khác nhau.

Để giao tiếp thông tin với bộ xử lý, như là định danh của thiết bị, hệ thống có thể dùng vector ngắt (*vectored interrupt*) hoặc thanh ghi nguyên nhân (*Cause register*). Khi bộ xử lý nhận một ngắt, thiết bị nhập/xuất gửi địa chỉ vector hoặc trường trạng thái (*status field*) đặt vào thanh ghi nguyên nhân. Hệ điều hành biết được định danh của thiết bị gây ra ngắt thì sẽ xem xét tức thì đến thiết bị đó. Mô hình ngắt sẽ loại bỏ yêu cầu của bộ xử lý để thăm dò thiết bị và thay vào đó bộ xử lý sẽ tập trung hơn để thực thi chương trình.

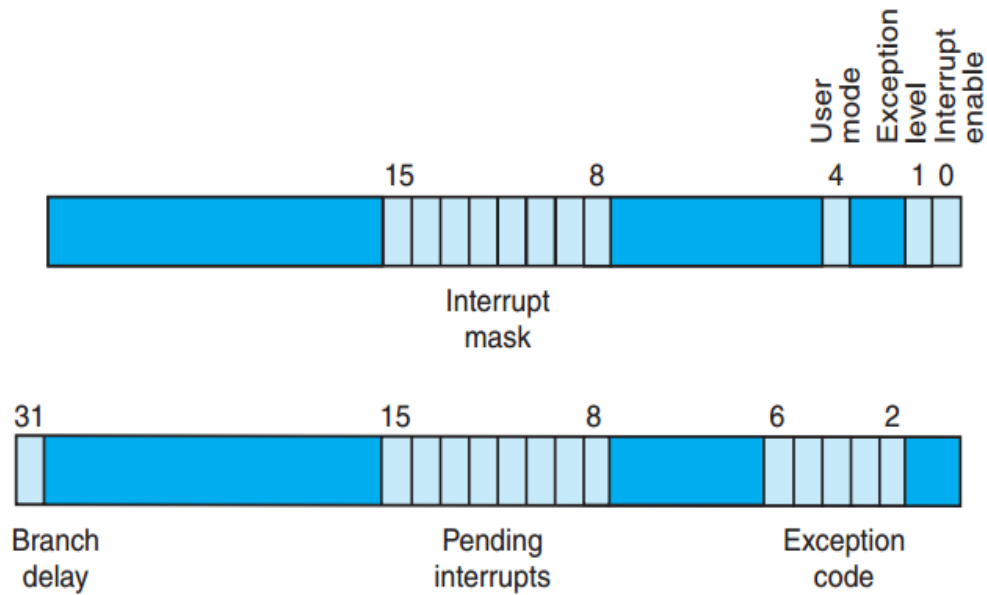
5.5.3 Độ ưu tiên ngắt

Để giải quyết với độ ưu tiên khác nhau của thiết bị nhập/xuất, mô hình ngắt đưa ra nhiều mức ưu tiên. Thí dụ, hệ điều hành UNIX dùng từ 4 đến 6 mức ưu tiên ngắt. Bộ xử lý sẽ xử lý ngắt dựa theo thứ tự của độ ưu tiên của mỗi ngắt. Các ngoại lệ được tạo ra bên trong và các ngắt nhập/xuất đều có độ ưu tiên. Các ngoại lệ có độ ưu tiên cao hơn các ngắt nhập/xuất bên ngoài. Đối với độ ưu tiên của ngắt nhập/xuất cũng có nhiều độ ưu tiên, thiết bị có tốc độ cao thường được kết hợp với độ ưu tiên cao hơn.

Thanh ghi trạng thái xác định tác nhân gây ra ngắt đối với máy tính (hình 5.2). Nếu bit ngắt được gán giá trị 0 thì ngắt không cho phép. Mỗi bit của mặt nạ ngắt (*interrupt mask*) tương ứng với mỗi bit của của trường ngắt chờ xử lý (*pending interrupt*) của thanh ghi nguyên nhân. Để thiết lập ngắt, phải gán giá trị 1 tại vị trí bit tương ứng của mặt nạ ngắt. Hệ điều hành đọc trường mã ngoại lệ (*exception code*) của thanh ghi trạng thái: 0 tương ứng với ngắt đã xảy ra, với giá trị khác cho trường hợp ngoại lệ.

Các bước để xử lý một ngắt:

1. Thực hiện phép toán AND giữa mặt nạ ngắt và trường ngắt chờ xử lý để xác định tác nhân gây ra ngắt.
2. Lựa chọn các ngắt có độ ưu tiên cao hơn. Theo quy ước thì bên trái nhất có độ ưu tiên cao nhất.
3. Lưu mặt nạ ngắt của thanh ghi trạng thái.
4. Thay đổi mặt nạ ngắt để vô hiệu hóa tất cả các ngắt có độ ưu tiên bằng hoặc nhỏ hơn.
5. Lưu lại trạng thái bộ xử lý cần thiết để xử lý ngắt.
6. Để cho phép các ngắt có độ ưu tiên cao hơn, thiết lập bit ngắt của thanh ghi nguyên nhân là 1.
7. Gọi hàm xử lý ngắt tương ứng.
8. Trước khi khôi phục lại trạng thái, thiết lập bit ngắt của thanh ghi nguyên nhân là 0. Điều này cho phép chúng ta khôi phục lại mặt nạ ngắt.



Hình 5.2: Thanh ghi trạng thái và thanh ghi nguyên nhân

5.5.4 Truyền dữ liệu giữa thiết bị và bộ nhớ

Chúng ta có hai phương pháp khác nhau để thiết bị nhập/xuất giao tiếp với bộ xử lý. Đó là phương pháp thăm dò và ngắt nhập/xuất. Cả hai phương pháp này thích hợp với thiết bị nhập/xuất có băng thông thấp. Các thiết bị này thường có giá rẻ với trình điều khiển thiết bị và giao tiếp đơn giản hơn so với các thiết bị nhập/xuất có băng thông cao. Trong cả hai phương pháp, bộ xử lý có vai trò quan trọng để quản lý việc truyền dữ liệu.

Cơ chế luân phiên được sử dụng trong phương pháp dùng ngắt nhập/xuất. Trong trường hợp này hệ điều hành truyền dữ liệu nhỏ (số byte ít) từ thiết bị hoặc đến thiết bị. Do thao tác nhập/xuất sử dụng ngắt nên hệ điều hành có thể thực hiện các công việc khác trong khi dữ liệu được đọc từ thiết bị hoặc ghi vào thiết bị. Khi hệ điều hành nhận một ngắt từ thiết bị, hệ điều hành đọc trạng thái để kiểm tra lỗi. Nếu không có lỗi hệ điều hành tiếp tục truyền dữ liệu tiếp theo. Khi byte cuối cùng của yêu cầu nhập/xuất được truyền và thao tác nhập xuất hoàn thành, hệ điều hành thông báo chương trình. Bộ xử lý và hệ điều hành làm tất cả công việc trong quá trình này: truy cập bộ nhớ và thiết bị khi dữ liệu được truyền.

Ngắt nhập/xuất làm giảm thời gian bộ xử lý chờ đợi mỗi sự kiện nhập/xuất, mặc dù chúng ta dùng phương pháp này để truyền dữ liệu với đĩa cứng thì đầu đọc đĩa cứng không thể chấp nhận bởi vì sẽ dùng phần lớn thời gian bộ xử lý khi đĩa đang được truyền. Đối với các thiết bị băng thông lớn như đĩa cứng, quá trình truyền sẽ bao gồm một khối dữ liệu lớn (hàng trăm đến hàng ngàn byte). Vì vậy, các nhà thiết kế máy tính phát triển một mô hình để giảm tải cho bộ xử lý và dùng trình điều khiển thiết bị để truyền dữ liệu trực tiếp với bộ nhớ mà không có sự tham gia của bộ xử lý. Mô hình này được gọi là truy cập bộ nhớ trực tiếp (DMA - *direct memory access*).

Mô hình ngắt vẫn được sử dụng để thiết bị giao tiếp với bộ xử lý nhưng chỉ khi quá trình truyền nhập/xuất hoàn thành hoặc khi có lỗi xảy ra.

DMA được thực hiện bởi trình điều khiển đặc biệt để truyền dữ liệu giữa thiết bị nhập/xuất và bộ nhớ mà không cần đến bộ xử lý. Trình điều khiển DMA trở thành chủ nhân bus (*master* – đơn vị chiếm giữ bus để thực hiện thao tác truyền dữ liệu) để điều khiển thao tác đọc hoặc ghi giữa thiết bị và bộ nhớ. Một quá trình truyền dữ liệu với DMA bao gồm ba bước:

1. Bộ xử lý thiết lập DMA bằng cách cung cấp định danh (*identity*) của thiết bị, thao tác được thực hiện với thiết bị, địa chỉ bộ nhớ để truyền dữ liệu và số byte dữ liệu cần được truyền.

2. DMA bắt đầu thực hiện thao tác trên thiết bị và điều khiển hệ thống bus. Khi dữ liệu sẵn sàng (từ thiết bị hoặc bộ nhớ), thao tác truyền dữ liệu được thực hiện. DMA cung cấp địa chỉ bộ nhớ để đọc hoặc ghi dữ liệu. Nếu yêu cầu nhiều quá trình truyền dữ liệu, DMA sẽ cung cấp địa chỉ bộ nhớ tiếp theo và khởi tạo quá trình truyền tiếp theo. Như vậy, theo mô hình này DMA sẽ thực hiện toàn bộ quá trình truyền dữ liệu (dung lượng có thể hàng ngàn byte) mà không có sự tham gia của bộ xử lý.

3. Khi quá trình truyền dữ liệu DMA hoàn thành, trình điều khiển DMA sẽ gửi tín hiệu ngắt tới bộ xử lý. Sau đó, bộ xử lý sẽ kiểm tra trình điều khiển DMA và bộ nhớ để xem toàn bộ thao tác có thực hiện thành công hay không.

Một hệ thống có thể có nhiều thiết bị DMA. Xem một hệ thống có một bus bộ nhớ - bộ xử lý và nhiều bus nhập/xuất. Mỗi trình điều khiển bus nhập/xuất thường bao gồm một vi xử lý DMA để điều khiển quá trình truyền dữ liệu giữa bộ nhớ và thiết bị trên bus nhập/xuất này.

Không giống như phương pháp thăm dò hay phương pháp ngắt nhập/xuất, DMA giao tiếp với bộ nhớ mà không dùng thời gian của bộ xử lý khi thực hiện thao tác truyền dữ liệu trên bus nhập/xuất. Bộ xử lý sẽ được trì hoãn khi muốn truy cập bộ nhớ trong khi bộ nhớ đang thực hiện thao tác DMA. Do bộ xử lý sử dụng bộ nhớ cache nên ít truy cập vào bộ nhớ chính. Vì vậy, băng thông bộ nhớ chính được giải phóng để thực hiện thao tác truyền dữ liệu với thiết bị nhập/xuất.

5.5.5 Truy cập bộ nhớ trực tiếp DMA và hệ thống bộ nhớ

Khi DMA được kết hợp vào hệ thống nhập/xuất, mối quan hệ giữa hệ thống bộ nhớ và bộ xử lý đã thay đổi. Khi không có DMA, tất cả các truy cập vào bộ nhớ đều đến từ bộ xử lý và được xử lý thông qua cơ chế dịch địa chỉ và truy cập bộ nhớ cache. Khi có DMA, truy cập vào bộ nhớ nhưng không thông qua cơ chế dịch địa chỉ hoặc cấp bậc bộ nhớ cache. Điều này gây khó khăn cho cả hệ thống bộ nhớ ảo và hệ thống bộ nhớ với cache. Vấn đề này được giải quyết bằng sự kết hợp của kỹ thuật phần cứng và hỗ trợ phần mềm.

Kỹ thuật DMA gây khó khăn cho hệ thống bộ nhớ ảo bởi vì trang có địa chỉ vật lý và địa chỉ ảo. Kỹ thuật này cũng tạo ra khó khăn cho hệ thống với bộ nhớ cache bởi vì có hai bản sao của một thành phần dữ liệu: một ở trong cache và một ở trong bộ nhớ. Bởi vì, DMA yêu cầu truy cập trực tiếp bộ nhớ mà không thông qua bộ nhớ

cache của bộ xử lý nên nội dung bộ nhớ được thấy bởi DMA và bộ xử lý có thể khác nhau. Xem một thao tác đọc đĩa mà DMA lưu trực tiếp vào bộ nhớ. Nếu một số vị trí bộ nhớ DMA ghi vào mà các vị trí này ở trong cache, khi đó bộ xử lý sẽ nhận giá trị cũ khi thực hiện thao tác đọc. Tương tự, đối với thao tác ghi lại của bộ nhớ cache thì DMA có thể đọc giá trị cũ trực tiếp từ bộ nhớ trong khi giá trị mới trong cache chưa được ghi lại.

5.6 TÓM TẮT

Trong chương này giới thiệu tổng quan về hệ thống lưu trữ và nhập/xuất trong máy tính. Cấu tạo của đĩa từ bao gồm nhiều lá đĩa, các đại lượng đặc trưng của đĩa từ như tốc độ quay, thời gian truy cập. Bộ nhớ flash có nguyên lý hoạt động dựa trên EEPROM. Ưu điểm của bộ này là nhỏ gọn, tiêu thụ ít điện năng... nên được sử dụng ngày càng phổ biến. Hệ thống bus dùng để kết nối các thành phần bên trong máy tính. Có hai loại bus: bus hệ thống và bus nhập/xuất. Hệ điều hành đóng vai trò quan trọng trong việc xử lý yêu cầu nhập/xuất như: ra lệnh cho thiết bị nhập/xuất, giao tiếp với bộ xử lý, độ ưu tiên ngắt của thiết bị nhập/xuất và điều khiển truyền dữ liệu giữa thiết bị nhập/xuất và bộ nhớ bằng kỹ thuật DMA.

CÂU HỎI ÔN TẬP CHƯƠNG 5

1. Cấu tạo và nguyên lý hoạt động đĩa từ?
2. Cấu tạo và nguyên lý hoạt động bộ nhớ flash?
3. Bus được chia thành mấy loại?
4. Đặc điểm của bus đồng bộ?
5. Đặc điểm của bus không đồng bộ?
6. Mô hình thiết bị nhập/xuất được ánh xạ vào bộ nhớ là gì?
7. Mô hình dùng tập lệnh nhập/xuất riêng là gì?
8. Kỹ thuật thăm dò hoạt động như thế nào?
9. Các bước để xử lý ngắt?
10. Kỹ thuật DMA hoạt động như thế nào?

TÀI LIỆU THAM KHẢO

- David A.Patterson., & John L.Hennessy. (2007). *Computer Architecture: A Quantitative Approach* (fourth edition). San Francisco: Morgan Kaufmann.
- David A.Patterson., & John L.Hennessy. (2012). *Computer organization and Design: The hardware/software interface* (fourth edition). Waltham: Morgan Kaufmann
- Miles Murdocca., & Vincent Heuring. (k.n.). *Principles of Computer Architecture*. Retrieved from <http://iuisaedu.com>
- Võ Văn Chín., Nguyễn Hồng Vân., & Phạm Hữu Tài. (2003). Giáo trình *Kiến trúc máy tính*. Cần Thơ: Đại học Cần Thơ.
- William Stallings. (2010). *Computer organization and Architecture: Designing for Performance* (eighth edition). New Jersey: Prentice Hall.