TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
Hanoi University of Science and Technology

KIẾN TRÚC MÁY TÍNH
Computer Architecture

Nguyễn Kim Khánh
Bộ môn Kỹ thuật máy tính
Viện Công nghệ Thông tin và Truyền thông
Department of Computer Engineering (DCE)
School of Information and Communication Technology (SoICT)

Version: CA2020

Contact Information

- Address: 502-B1
- Mobile: 091-358-5533
- e-mail: khanhnk@soict.hust.edu.vn
  khanh.nguyenkim@hust.edu.vn

Mục tiêu học phần

- Sinh viên được trang bị các kiến thức cơ sở về kiến trúc tập lệnh và tổ chức bên trong của máy tính, cũng như những nguyên tắc cơ bản trong thiết kế máy tính.
- Sau khi học xong học phần này, sinh viên có khả năng:
  - Tìm hiểu kiến trúc tập lệnh của các máy tính cụ thể
  - Lập trình hợp ngữ
  - Đánh giá hiệu năng máy tính và cải thiện hiệu năng của chương trình
  - Khai thác và quản trị hiệu quả các hệ thống máy tính
  - Phân tích và thiết kế máy tính

Tài liệu học tập

- *Bài giảng Kiến trúc máy tính:* **CA2020.pdf**
- *Sách tham khảo:*

[1] William Stallings
Computer Organization and Architecture – 2013, 9th edition

[2] David A. Patterson, John L. Hennessy
Computer Organization and Design – 2012, Revised 4th edition

[3] David Money Harris, Sarah L. Harris
Digital Design and Computer Architecture – 2013, 2nd edition

[4] Andrew S. Tanenbaum
Structured Computer Organization – 2013, 6th edition

- *Phần mềm lập trình hợp ngữ và mô phỏng cho MIPS:*

MARS (MIPS Assembler and Runtime Simulator)

download tại: http://courses.missouristate.edu/KenVollmar/MARS/

Nội dung học phần

Chương 1. Giới thiệu chung

Chương 2. Cơ bản về logic số

Chương 3. Hệ thống máy tính

Chương 4. Số học máy tính

Chương 5. Kiến trúc tập lệnh

Chương 6. Bộ xử lý

Chương 7. Bộ nhớ máy tính

Chương 8. Hệ thống vào-ra

Chương 9. Các kiến trúc song song

Content

Chương 1
GIỚI THIỆU CHUNG

Nguyễn Kim Khánh
Trường Đại học Bách khoa Hà Nội

Nội dung của chương 1

1.1. Máy tính và phân loại máy tính

1.2. Khái niệm kiến trúc máy tính

1.3. Sự tiến hóa của công nghệ máy tính

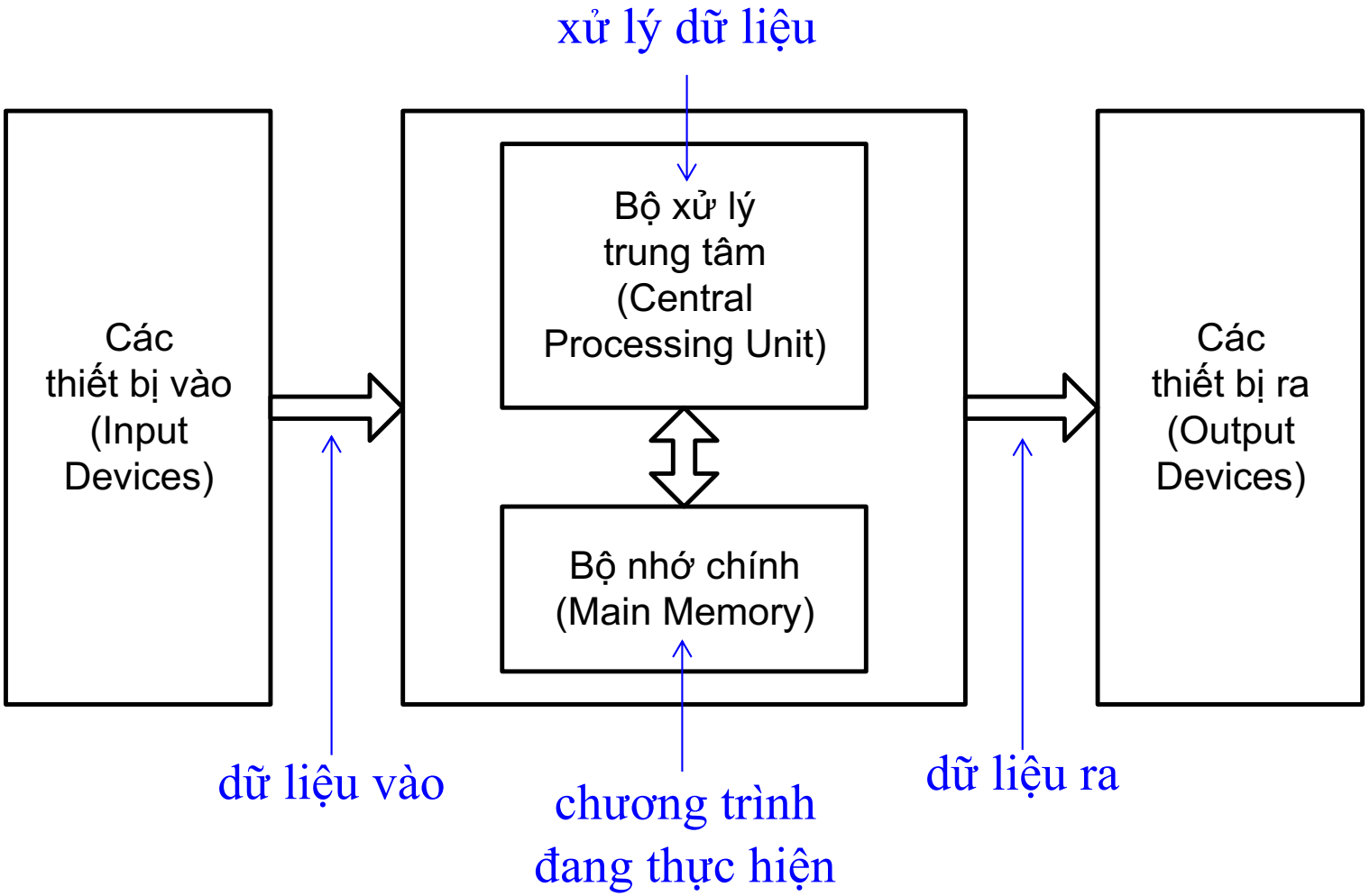
1.4. Hiệu năng máy tính

1.1. Máy tính và phân loại máy tính

- Máy tính (Computer) là thiết bị điện tử thực hiện các công việc sau:
  - Nhận dữ liệu vào,
  - Xử lý dữ liệu theo dãy các lệnh được nhớ sẵn bên trong,
  - Đưa dữ liệu (thông tin) ra.
- Dãy các lệnh nằm trong bộ nhớ để yêu cầu máy tính thực hiện công việc cụ thể gọi là chương trình (program).

→ Máy tính hoạt động theo chương trình

Mô hình đơn giản của máy tính



Phân loại máy tính kỷ nguyên PC

- Máy tính cá nhân (Personal Computers)
  - Desktop computers, Laptop computers
  - Máy tính đa dụng
- Máy chủ (Servers) – máy phục vụ
  - Dùng trong mạng để quản lý và cung cấp các dịch vụ
  - Hiệu năng và độ tin cậy cao
  - Hàng nghìn đến hàng triệu USD
- Siêu máy tính (Supercomputers)
  - Dùng cho tính toán cao cấp trong khoa học và kỹ thuật
  - Hàng triệu đến hàng trăm triệu USD
- Máy tính nhúng (Embedded Computers)
  - Đặt ẩn trong thiết bị khác
  - Được thiết kế chuyên dụng

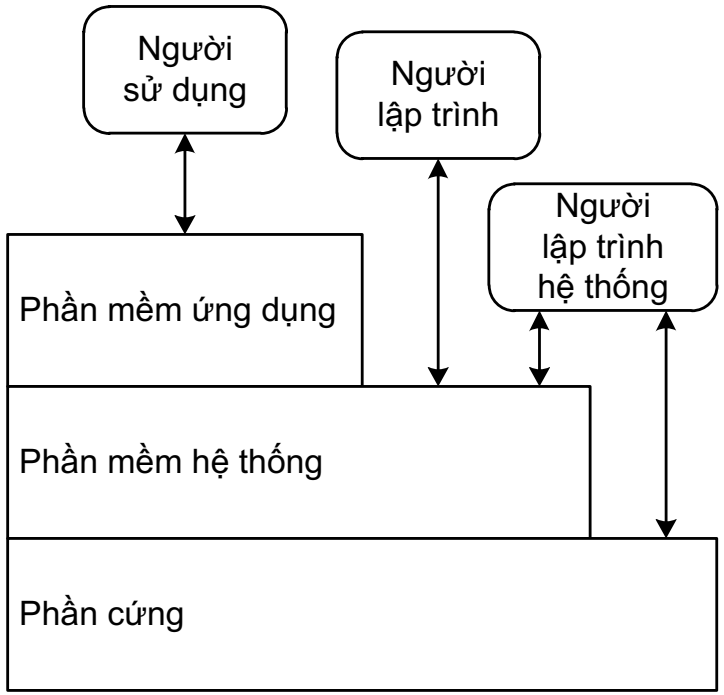
Phân loại máy tính kỷ nguyên sau PC

- Thiết bị di động cá nhân (PMD - Personal Mobile Devices)
  - Smartphones, Tablet
  - Kết nối Internet
- Điện toán đám mây (Cloud Computing)
  - Sử dụng máy tính qui mô lớn (Warehouse Scale Computers), gồm rất nhiều servers kết nối với nhau
  - Cho các công ty thuê một phần để cung cấp dịch vụ phần mềm
  - Software as a Service (SaaS): một phần của phần mềm chạy trên PMD, một phần chạy trên Cloud

1.2. Khái niệm kiến trúc máy tính

- Kiến trúc máy tính bao gồm:
  - Kiến trúc tập lệnh (Instruction Set Architecture): nghiên cứu máy tính theo cách nhìn của người lập trình
  - Tổ chức máy tính (Computer Organization) hay Vi kiến trúc (Microarchitecture): nghiên cứu thiết kế máy tính ở mức cao (thiết kế CPU, hệ thống nhớ, cấu trúc bus, ...)
  - Phần cứng (Hardware): nghiên cứu thiết kế logic chi tiết và công nghệ đóng gói của máy tính.
- Cùng một kiến trúc tập lệnh có thể có nhiều sản phẩm (tổ chức, phần cứng) khác nhau

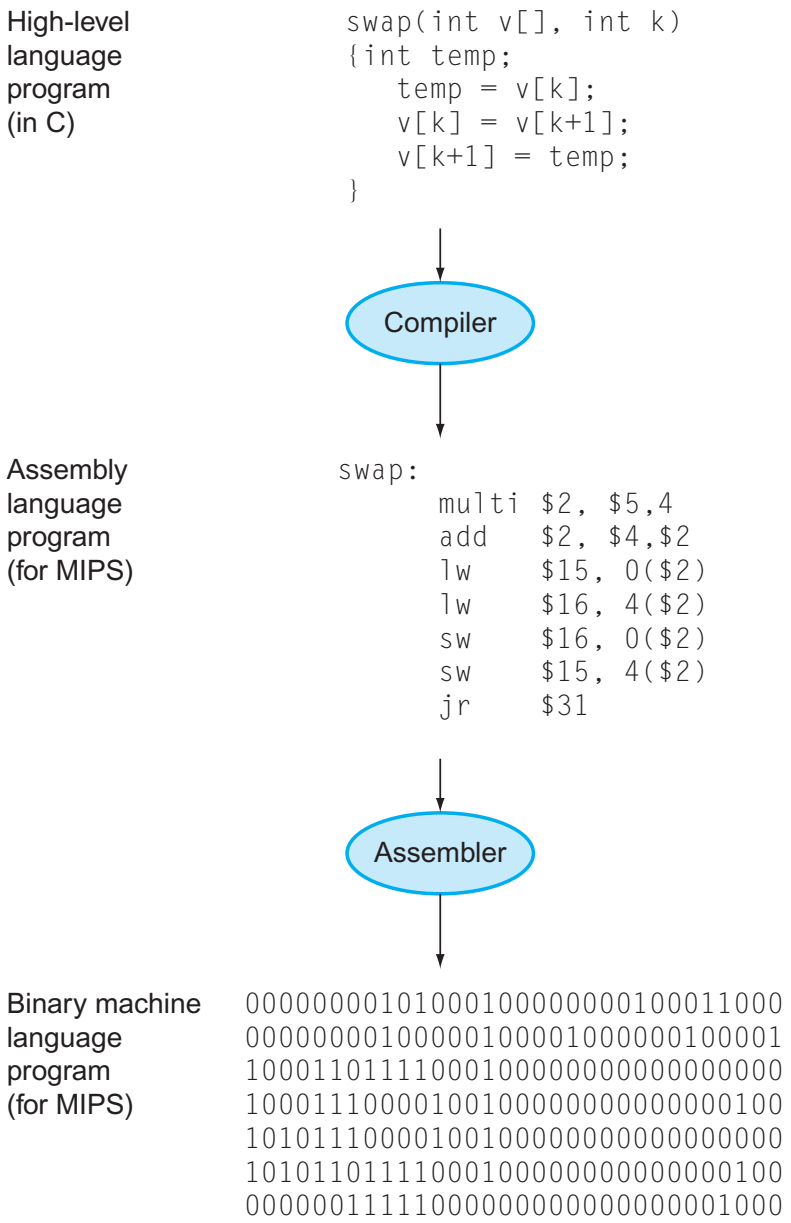
Phân lớp máy tính



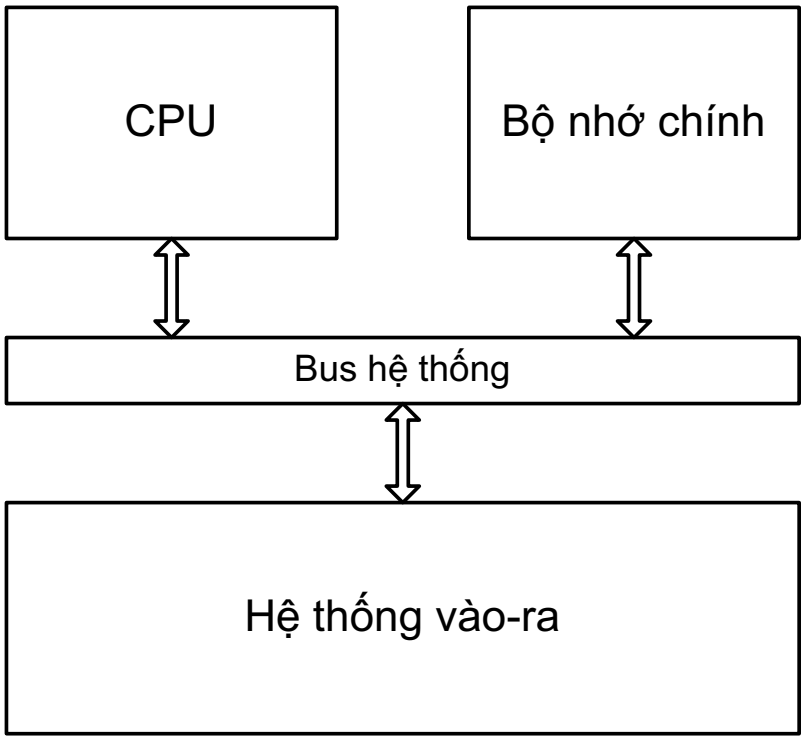
- Phần mềm ứng dụng
  - Được viết theo ngôn ngữ bậc cao
- Phần mềm hệ thống
  - Chương trình dịch (Compiler): dịch mã ngôn ngữ bậc cao thành ngôn ngữ máy
  - Hệ điều hành (Operating System)
    - Lập lịch cho các nhiệm vụ và chia sẻ tài nguyên
    - Quản lý bộ nhớ và lưu trữ
    - Điều khiển vào-ra
- Phần cứng
  - Bộ xử lý, bộ nhớ, mô-đun vào-ra

Các mức của mã chương trình

- Ngôn ngữ bậc cao
  - High-level language – HLL
  - Mức trừu tượng gần với vấn đề cần giải quyết
  - Hiệu quả và linh động
- Hợp ngữ
  - Assembly language
  - Mô tả lệnh dưới dạng text
- Ngôn ngữ máy
  - Machine language
  - Mô tả theo phần cứng
  - Các lệnh và dữ liệu được mã hóa theo nhị phân



Các thành phần cơ bản của máy tính

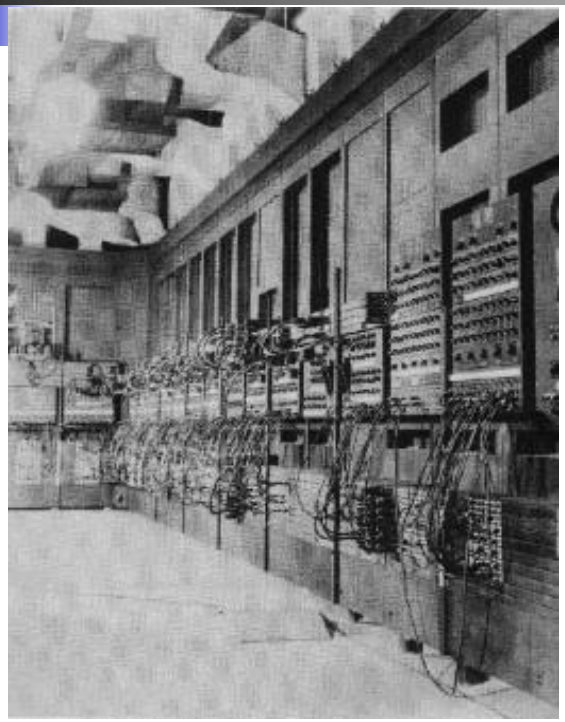


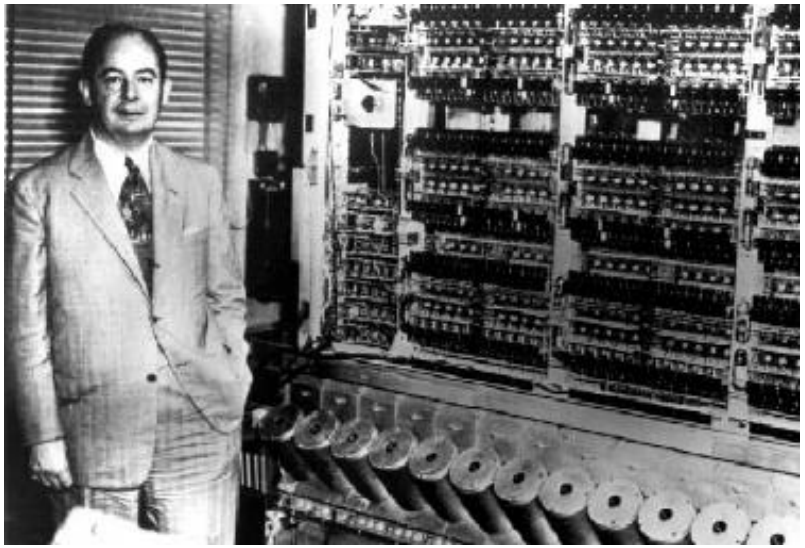
- Giống nhau với tất cả các loại máy tính
- Bộ xử lý trung tâm (Central Processing Unit – CPU)
  - Điều khiển hoạt động của máy tính và xử lý dữ liệu
- Bộ nhớ chính (Main Memory)
  - Chứa các chương trình đang thực hiện
- Hệ thống vào-ra (Input/Output)
  - Trao đổi thông tin giữa máy tính với bên ngoài
- Bus hệ thống (System bus)
  - Kết nối và vận chuyển thông tin

1.3. Sự tiến hóa của công nghệ máy tính

- Máy tính dùng đèn điện tử chân không (1950s)
  - Máy tính ENIAC: máy tính đầu tiên (1946)
  - Máy tính IAS: máy tính von Neumann (1952)
- Máy tính dùng transistors (1960s)
- Máy tính dùng vi mạch SSI, MSI và LSI (1970s)
  - SSI - Small Scale Integration
  - MSI - Medium Scale Integration
  - LSI - Large Scale Integration
- Máy tính dùng vi mạch VLSI (1980s)
  - VLSI - Very Large Scale Integration
- Máy tính dùng vi mạch ULSI (1990s-nay)
  - ULSI - Ultra Large Scale Integration

Máy tính đầu tiên: ENIAC và IAS

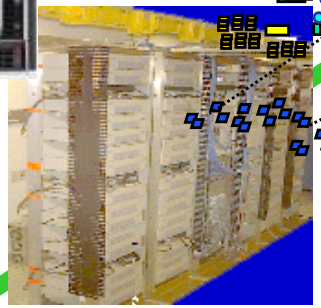


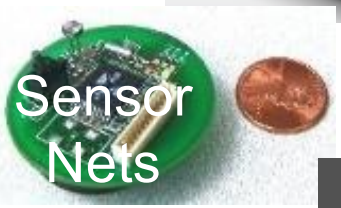


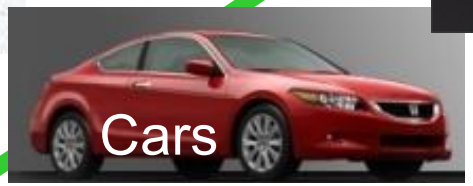
- Electronic Numerical Intergator and Computer
- Dự án của Bộ Quốc phòng Mỹ
- Do John Mauchly ở đại học Pennsylvania thiết kế
- 30 tấn
- Xử lý theo số thập phân

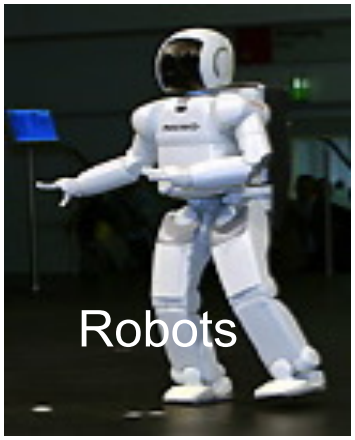
- Thực hiện tại Princeton Institute for Advanced Studies
- Do John von Neumann thiết kế theo ý tưởng "stored program"
- Xử lý theo số nhị phân
- Trở thành mô hình cơ bản của máy tính

Máy tính ngày nay







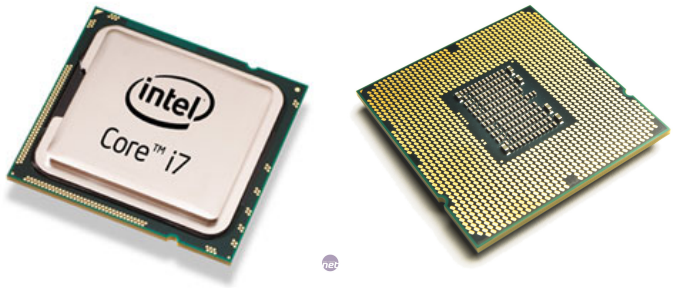


Một số loại vi mạch số điển hình

- Bộ vi xử lý (Microprocessors)
  - Một hoặc một vài CPU được chế tạo trên một chip
- Vi mạch điều khiển tổng hợp (Chipset)
  - Vi mạch thực hiện các chức năng nối ghép các thành phần của máy tính với nhau
- Bộ nhớ bán dẫn (Semiconductor Memory)
  - ROM, RAM, Flash memory
- Hệ thống trên chip (SoC – System on Chip) hay Bộ vi điều khiển (Microcontrollers)
  - Tích hợp các thành phần chính của máy tính trên một chip vi mạch
  - Được sử dụng chủ yếu trên smartphone, tablet và các máy tính nhúng

Sự phát triển của bộ vi xử lý

- 1971: bộ vi xử lý 4-bit Intel 4004
- 1972: các bộ xử lý 8-bit
- 1978: các bộ xử lý 16-bit
  - Máy tính cá nhân IBM-PC ra đời năm 1981
- 1985: các bộ xử lý 32-bit
- 2001: các bộ xử lý 64-bit
- 2006: các bộ xử lý đa lõi (multicores)
  - Nhiều CPU trên 1 chip



1.4. Hiệu năng máy tính

- Định nghĩa hiệu năng P (Performance):

  Hiệu năng = 1/(thời gian thực hiện)

  hay là: ***P = 1/t***

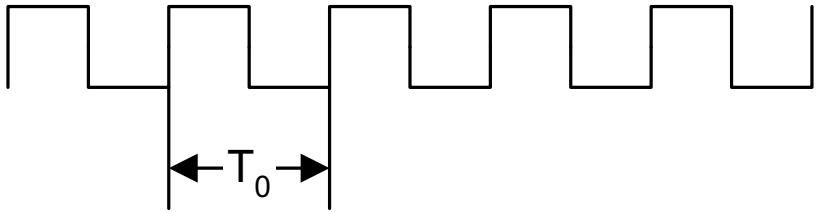
"Máy tính A nhanh hơn máy B k lần"

$$P_A / P_B = t_B / t_A = k$$

- Ví dụ: Thời gian chạy chương trình:
  - 10s trên máy A, 15s trên máy B
  - t_B / t_A = 15s / 10s = 1.5
  - Vậy máy A nhanh hơn máy B 1.5 lần

Tốc độ xung nhịp của CPU

- Về mặt thời gian, CPU hoạt động theo một xung nhịp (clock) có tốc độ xác định



- Chu kỳ xung nhịp T_0 (Clock period): thời gian của một chu kỳ
- Tốc độ xung nhịp f_0 (Clock rate) hay là Tần số xung nhịp: số chu kỳ trong 1s
  - $f_0 = 1/T_0$
- VD: Bộ xử lý có $f_0 = 4\text{GHz} = 4 \times 10^9\text{Hz}$

  $T_0 = 1/(4\text{x}10^9) = 0.25\text{x}10^{-9}\text{s} = 0.25\text{ns}$

Thời gian thực hiện của CPU

- Để đơn giản, ta xét thời gian CPU thực hiện chương trình (CPU time):

Thời gian thực hiện của CPU =

Số chu kỳ xung nhịp x Thời gian một chu kỳ

$$t_{CPU} = n \times T_0 = \frac{n}{f_0}$$

n: số chu kỳ xung nhịp

- Hiệu năng được tăng lên bằng cách:
  - Giảm số chu kỳ xung nhịp n
  - Tăng tốc độ xung nhịp f_0

Ví dụ

- Hai máy tính A và B cùng chạy một chương trình
- Máy tính A:
  - Tốc độ xung nhịp của CPU: f_A = 2GHz
  - Thời gian CPU thực hiện chương trình: t_A = 10s
- Máy tính B:
  - Thời gian CPU thực hiện chương trình: t_B = 6s
  - Số chu kỳ xung nhịp khi chạy chương trình trên máy B (n_B) nhiều hơn 1.2 lần số chu kỳ xung nhịp khi chạy chương trình trên máy A (n_A)
- Hãy xác định tốc độ xung nhịp cần thiết cho máy B (f_B)?

Ví dụ (tiếp)

Ta có: $t = \frac{n}{f}$

Số chu kỳ xung nhịp khi chạy chương trình trên máy A:

$$n_A = t_A \times f_A = 10s \times 2GHz = 20 \times 10^9$$

Số chu kỳ xung nhịp khi chạy chương trình trên máy B:

$$n_B = 1.2 \times n_A = 24 \times 10^9$$

Tốc độ xung nhịp cần thiết cho máy B:

$$f_B = \frac{n_B}{t_B} = \frac{24 \times 10^9}{6} = 4 \times 10^9 Hz = 4GHz$$

Số lệnh và số chu kỳ trên một lệnh

Số chu kỳ xung nhịp của chương trình:

Số chu kỳ = Số lệnh của chương trình x Số chu kỳ trên một lệnh

$$n = IC \times CPI$$

- n - số chu kỳ xung nhịp
- IC - số lệnh của chương trình (Instruction Count)
- CPI - số chu kỳ trên một lệnh (Cycles per Instruction)

Vậy thời gian thực hiện của CPU:

$$t_{CPU} = IC \times CPI \times T_0 = \frac{IC \times CPI}{f_0}$$

Trong trường hợp các lệnh khác nhau có CPI khác nhau, cần tính CPI trung bình

Ví dụ

- Hai máy tính A và B có cùng kiến trúc tập lệnh
- Máy tính A có:
  - Chu kỳ xung nhịp: T_A = 250ps
  - Số chu kỳ/ lệnh trung bình: CPI_A = 2.0
- Máy tính B:
  - Chu kỳ xung nhịp: T_B = 500ps
  - Số chu kỳ/ lệnh trung bình: CPI_B = 1.2
- Hãy xác định máy nào nhanh hơn và nhanh hơn bao nhiêu ?

Ví dụ (tiếp)

Ta có: $t_{CPU} = IC \times CPI_{TB} \times T_0$

Hai máy cùng kiến trúc tập lệnh, vì vậy số lệnh của cùng một chương trình trên hai máy là bằng nhau:

$$IC_A = IC_B = IC$$

Thời gian thực hiện chương trình đó trên máy A và máy B:

$$t_A = IC_A \times CPI_A \times T_A = IC \times 2.0 \times 250\,ps = IC \times 500\,ps$$

$$t_B = IC_B \times CPI_B \times T_B = IC \times 1.2 \times 500\,ps = IC \times 600\,ps$$

Từ đó ta có: $\frac{t_B}{t_A} = \frac{IC \times 600\,ps}{IC \times 500\,ps} = 1.2$

Kết luận: máy A nhanh hơn máy B 1.2 lần

CPI trung bình

- Nếu loại lệnh khác nhau có số chu kỳ khác nhau, ta có tổng số chu kỳ:

$$n = \sum_{i=1}^{K} \left(CPI_i \times IC_i \right)$$

- CPI trung bình:

$$CPI_{TB} = \frac{n}{IC} = \frac{1}{IC} \sum_{i=1}^{K} \left(CPI_i \times IC_i \right)$$

Ví dụ

- Cho bảng chỉ ra các dãy lệnh sử dụng các lệnh thuộc các loại A, B, C. Tính CPI trung bình?

Loại lệnh	A	B	C
CPI theo loại lệnh	1	2	3
IC trong dãy lệnh 1	20	10	20
IC trong dãy lệnh 2	40	10	10

Ví dụ

- Cho bảng chỉ ra các dãy lệnh sử dụng các lệnh thuộc các loại A, B, C. Tính CPI trung bình?

Loại lệnh	A	B	C
CPI theo loại lệnh	1	2	3
IC trong dãy lệnh 1	20	10	20
IC trong dãy lệnh 2	40	10	10

- Dãy lệnh 1: Số lệnh = 50
  - Số chu kỳ =

  = 1x20 + 2x10 + 3x20 = 100
  - CPI_{TB} = 100/50 = 2.0
- Dãy lệnh 2: Số lệnh = 60
  - Số chu kỳ =

  = 1x40 + 2x10 + 3x10 = 90
  - CPI_{TB} = 90/60 = 1.5

Tóm tắt về Hiệu năng

$$\text{CPU Time} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock cycle}}$$

Thời gian CPU = Số lệnh của chương trình x Số chu kỳ/lệnh x Số giây của một chu kỳ

$$t_{CPU} = IC \times CPI \times T_0 = \frac{IC \times CPI}{f_0}$$

- Hiệu năng phụ thuộc vào:
  - Thuật giải
  - Ngôn ngữ lập trình
  - Chương trình dịch
  - Kiến trúc tập lệnh
  - Phần cứng

MIPS như là thước đo hiệu năng

- MIPS: Millions of Instructions Per Second
  (Số triệu lệnh trên 1 giây)

$$\text{MIPS} = \frac{\text{Instruction count}}{\text{Execution time} \times 10^6} = \frac{\text{Instruction count}}{\frac{\text{Instruction count} \times \text{CPI}}{\text{Clock rate}} \times 10^6} = \frac{\text{Clock rate}}{\text{CPI} \times 10^6}$$

$$\text{MIPS} = \frac{f_0}{\text{CPI} \times 10^6} \qquad \text{CPI} = \frac{f_0}{\text{MIPS} \times 10^6}$$

Ví dụ

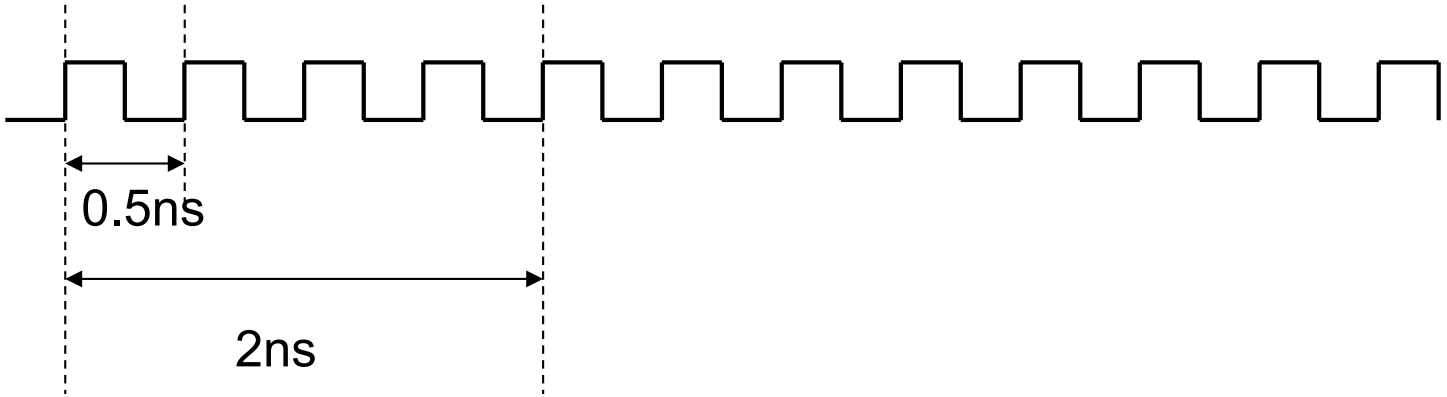
Tính MIPS của bộ xử lý với:

clock rate = 2GHz và CPI = 4

Ví dụ

Tính MIPS của bộ xử lý với:

clock rate = 2GHz và CPI = 4



- Chu kỳ $T_0 = 1/(2x10^9) = 0.5ns$
- CPI = 4 → thời gian thực hiện 1 lệnh = 4 x 0.5ns = 2ns
- Số lệnh thực hiện trong 1s = $(10^9 ns)/(2ns) = 5x10^8$ lệnh
- Vậy bộ xử lý thực hiện được 500 MIPS

Ví dụ

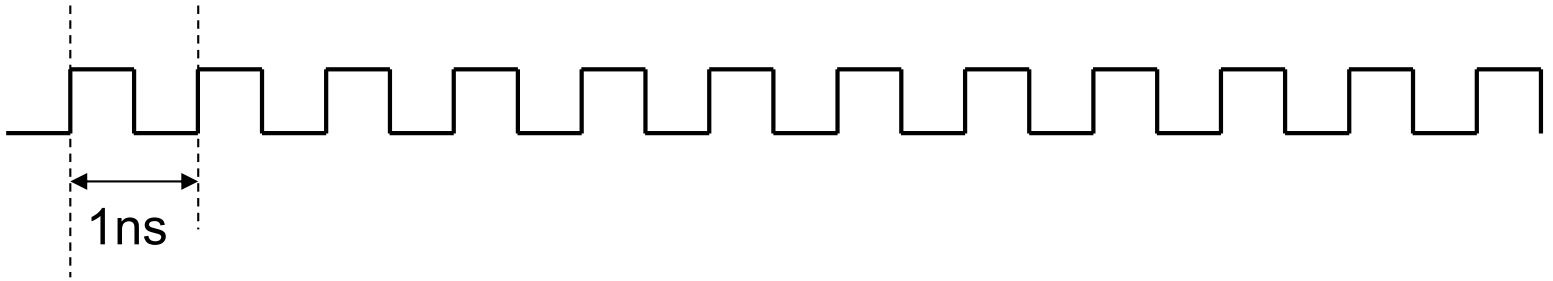
Tính CPI của bộ xử lý với:

clock rate = 1GHz và 400 MIPS

Ví dụ

Tính CPI của bộ xử lý với:

clock rate = 1GHz và 400 MIPS



- Chu kỳ $T_0 = 1/10^9 = 1ns$
- Số lệnh thực hiện trong 1 s là 400MIPS = $4x10^8$ lệnh
- Thời gian thực hiện 1 lệnh = $1/(4x10^8)s = 2.5ns$
- Vậy ta có: CPI = 2.5

MFLOPS

- Sử dụng cho các hệ thống tính toán lớn
- Millions of Floating Point Operations per Second
- Số triệu phép toán số dấu phẩy động trên một giây

$$\text{MFLOPS} = \frac{\text{Executed floating point operations}}{\text{Execution time} \times 10^6}$$

GFLOPS (10^9)

TFLOPS (10^{12})

PFLOPS (10^{15})

Hết chương 1

Chương 2
CƠ BẢN VỀ LOGIC SỐ

Nguyễn Kim Khánh
Trường Đại học Bách khoa Hà Nội

Nội dung học phần

Chương 1. Giới thiệu chung

Chương 2. Cơ bản về logic số

Chương 3. Hệ thống máy tính

Chương 4. Số học máy tính

Chương 5. Kiến trúc tập lệnh

Chương 6. Bộ xử lý

Chương 7. Bộ nhớ máy tính

Chương 8. Hệ thống vào-ra

Chương 9. Các kiến trúc song song

Nội dung của chương 2

2.1. Các hệ đếm cơ bản
2.2. Đại số Boole
2.3. Các cổng logic
2.4. Mạch tổ hợp
2.5. Mạch dãy

2.1. Các hệ đếm cơ bản

- Hệ thập phân (Decimal System)

  → con người sử dụng

- Hệ nhị phân (Binary System)

  → máy tính sử dụng

- Hệ mười sáu (Hexadecimal System)

  → dùng để viết gọn cho số nhị phân

1. Hệ thập phân

- Cơ số 10
- 10 chữ số: 0,1,2,3,4,5,6,7,8,9
- Dùng n chữ số thập phân có thể biểu diễn được 10^n giá trị khác nhau:
  - 00...000 = 0
  - 99...999 = 10^n - 1

Dạng tổng quát của số thập phân

$$A = a_n a_{n-1} \ldots a_1 a_0 , a_{-1} \ldots a_{-m}$$

Giá trị của *A* được hiểu như sau:

$$A = a_n 10^n + a_{n-1} 10^{n-1} + \ldots + a_1 10^1 + a_0 10^0 + a_{-1} 10^{-1} + \ldots + a_{-m} 10^{-m}$$

$$A = \sum_{i=-m}^{n} a_i 10^i$$

Ví dụ số thập phân

$472.38 = 4x10^2 + 7x10^1 + 2x10^0 + 3x10^{-1} + 8x10^{-2}$

- Các chữ số của phần nguyên:
  - 472 : 10 = 47 dư 2 ↑
  - 47 : 10 = 4 dư 7
  - 4 : 10 = 0 dư 4
- Các chữ số của phần lẻ:
  - 0.38 x 10 = 3.8 phần nguyên = 3 ↓
  - 0.8 x 10 = 8.0 phần nguyên = 8

2. Hệ nhị phân

- Cơ số 2
- 2 chữ số nhị phân: 0 và 1
- Chữ số nhị phân được gọi là ***bit*** (*binary digit*)
- *bit* là đơn vị thông tin nhỏ nhất
- Dùng n bit có thể biểu diễn được 2^n giá trị khác nhau:
  - 00...000 = 0
  - 11...111 = 2^n - 1
- Các lệnh của chương trình và dữ liệu trong máy tính đều được mã hóa bằng số nhị phân

Biểu diễn số nhị phân

Số nhị phân				Số thập phân
1-bit	2-bit	3-bit	4-bit	
0	00	000	0000	0
1	01	001	0001	1
	10	010	0010	2
	11	011	0011	3
		100	0100	4
		101	0101	5
		110	0110	6
		111	0111	7
			1000	8
			1001	9
			1010	10
			1011	11
			1100	12
			1101	13
			1110	14
			1111	15

Đơn vị dữ liệu và thông tin trong máy tính

- bit – chữ số nhị phân (binary digit): là đơn vị thông tin nhỏ nhất, cho phép nhận một trong hai giá trị: 0 hoặc 1.
- byte là một tổ hợp 8 bit: có thể biểu diễn được 256 giá trị (2^8)
- Qui ước các đơn vị dữ liệu:
  - KB (Kilobyte) = 2^{10} bytes = 1024 bytes
  - MB (Megabyte) = 2^{10} KB = 2^{20}bytes (~10^6)
  - GB (Gigabyte) = 2^{10} MB = 2^{30}bytes (~10^9)
  - TB (Terabyte) = 2^{10} GB = 2^{40}bytes (~10^{12})
  - PB (Petabyte) = 2^{10} TB = 2^{50}bytes
  - EB (Exabyte) = 2^{10} PB = 2^{60}bytes

Qui ước mới về ký hiệu đơn vị dữ liệu

Theo thập phân			Theo nhị phân		
Đơn vị	Viết tắt	Giá trị	Đơn vị	Viết tắt	Giá trị
kilobyte	KB	10^3	kibibyte	KiB	$2^{10} = 1024$
megabyte	MB	10^6	mebibyte	MiB	2^{20}
gigabyte	GB	10^9	gibibyte	GiB	2^{30}
terabyte	TB	10^{12}	tebibyte	TiB	2^{40}
petabyte	PB	10^{15}	pebibyte	PiB	2^{50}
exabyte	EB	10^{18}	exbibyte	EiB	2^{60}

Dạng tổng quát của số nhị phân

$$A = a_n a_{n-1} \ldots a_1 a_0 , a_{-1} \ldots a_{-m} \qquad \text{với } a_i = 0 \text{ hoặc } 1$$

Giá trị của *A* được tính như sau:

$$A = a_n 2^n + a_{n-1} 2^{n-1} + \ldots + a_1 2^1 + a_0 2^0 + a_{-1} 2^{-1} + \ldots + a_{-m} 2^{-m}$$

$$A = \sum_{i=-m}^{n} a_i 2^i$$

Ví dụ số nhị phân

$$1101001.1011_{(2)} =$$
$$6\ 5\ 4\ 3\ 2\ 1\ 0\ \ -1\ -2\ -3\ -4$$

$$= 2^6 + 2^5 + 2^3 + 2^0 + 2^{-1} + 2^{-3} + 2^{-4}$$

$$= 64 + 32 + 8 + 1 + 0.5 + 0.125 + 0.0625$$

$$= 105.6875_{(10)}$$

Chuyển đổi số nguyên thập phân sang nhị phân

- Phương pháp 1: chia dần cho 2 rồi lấy phần dư
- Phương pháp 2: Phân tích thành tổng của các số 2^i → nhanh hơn

Phương pháp chia dần cho 2

- Ví dụ: chuyển đổi $105_{(10)}$
  - 105 : 2 = 52 dư 1
  - 52 : 2 = 26 dư 0
  - 26 : 2 = 13 dư 0
  - 13 : 2 = 6 dư 1
  - 6 : 2 = 3 dư 0
  - 3 : 2 = 1 dư 1
  - 1 : 2 = 0 dư 1

biểu diễn số dư theo chiều mũi tên

- Kết quả: $105_{(10)} = 1101001_{(2)}$

Phương pháp phân tích thành tổng của các 2^i

- Ví dụ 1: chuyển đổi $105_{(10)}$
  - $105 = 64 + 32 + 8 + 1 = 2^6 + 2^5 + 2^3 + 2^0$

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1
0	1	1	0	1	0	0	1

  - Kết quả: $105_{(10)} = 0110\ 1001_{(2)}$

- Ví dụ 2: $17000_{(10)} = 16384 + 512 + 64 + 32 + 8$

$$= 2^{14} + 2^9 + 2^6 + 2^5 + 2^3$$

$$17000_{(10)} = \underset{15\ 14\ 13\ 12}{0100}\ \underset{11\ 10\ 9\ 8}{0010}\ \underset{7\ 6\ 5\ 4}{0110}\ \underset{3\ 2\ 1\ 0}{1000}_{(2)}$$

Chuyển đổi số lẻ thập phân sang nhị phân

- Ví dụ 1: chuyển đổi $0.6875_{(10)}$
  - 0.6875 x 2 = 1.375 phần nguyên = 1
  - 0.375 x 2 = 0.75 phần nguyên = 0
  - 0.75 x 2 = 1.5 phần nguyên = 1
  - 0.5 x 2 = 1.0 phần nguyên = 1

biểu diễn theo chiều mũi tên

- Kết quả : $0.6875_{(10)} = 0.1011_{(2)}$

Chuyển đổi số lẻ thập phân sang nhị phân (tiếp)

- Ví dụ 2: chuyển đổi $0.81_{(10)}$
  - 0.81 x 2 = 1.62 phần nguyên = 1
  - 0.62 x 2 = 1.24 phần nguyên = 1
  - 0.24 x 2 = 0.48 phần nguyên = 0
  - 0.48 x 2 = 0.96 phần nguyên = 0
  - 0.96 x 2 = 1.92 phần nguyên = 1
  - 0.92 x 2 = 1.84 phần nguyên = 1
  - 0.84 x 2 = 1.68 phần nguyên = 1

- $0.81_{(10)} \approx 0.1100111_{(2)}$

3. Hệ mười sáu (Hexa)

- Cơ số 16
- 16 chữ số: 0,1,2,3,4,5,6,7,8,9, A,B,C,D,E,F
- Dùng để viết gọn cho số nhị phân: cứ một nhóm 4-bit sẽ được thay bằng một chữ số Hexa

Quan hệ giữa số nhị phân và số Hexa

Ví dụ:

- $1011\ 0011_{(2)} = B3_{(16)}$
- $0000\ 0000_{(2)} = 00_{(16)}$

- $0010\ 1101\ 1001\ 1010_{(2)} = 2D9A_{(16)}$
- $1111\ 1111\ 1111\ 1111_{(2)} = FFFF_{(16)}$

4-bit	Số Hexa	Thập 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	A	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15

2.2. Đại số Boole

- Đại số Boole sử dụng các biến logic và phép toán logic
- Biến logic có thể nhận giá trị 1 (TRUE) hoặc 0 (FALSE)
- Các phép toán logic cơ bản: AND, OR và NOT
  - A AND B : $A \bullet B$ hay AB
  - A OR B : $A + B$
  - NOT A : $\overline{A}$
  - Thứ tự ưu tiên: NOT > AND > OR
- Thêm các phép toán logic: NAND, NOR, XOR
  - A NAND B: $\overline{A \bullet B}$
  - A NOR B : $\overline{A + B}$
  - A XOR B: $A \oplus B = A \bullet \overline{B} + \overline{A} \bullet B$

Phép toán đại số Boole với hai biến

A	B	A AND B $A \bullet B$
0	0	0
0	1	0
1	0	0
1	1	1

A	B	A OR B $A + B$
0	0	0
0	1	1
1	0	1
1	1	1

A	NOT A $\overline{A}$
0	1
1	0

NOT là phép toán 1 biến

A	B	A NAND B $\overline{A \bullet B}$
0	0	1
0	1	1
1	0	1
1	1	0

A	B	A NOR B $\overline{A + B}$
0	0	1
0	1	0
1	0	0
1	1	0

A	B	A XOR B $A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

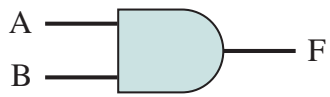
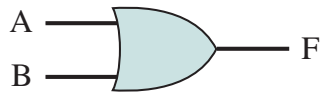
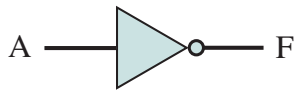
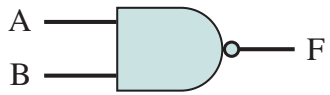
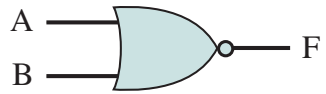
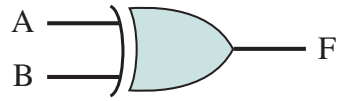
Các đồng nhất thức của đại số Boole

$A \cdot B = B \cdot A$	$A + B = B + A$
$A \cdot (B + C) = (A \cdot B) + (A \cdot C)$	$A + (B \cdot C) = (A + B) \cdot (A + C)$
$1 \cdot A = A$	$0 + A = A$
$A \cdot \overline{A} = 0$	$A + \overline{A} = 1$
$0 \cdot A = 0$	$1 + A = 1$
$A \cdot A = A$	$A + A = A$
$A \cdot (B \cdot C) = (A \cdot B) \cdot C$	$A + (B + C) = (A + B) + C$
$\overline{A \cdot B} = \overline{A} + \overline{B}$ (Định lý De Morgan)	$\overline{A + B} = \overline{A} \cdot \overline{B}$ (Định lý De Morgan)

2.3. Các cổng logic (Logic Gates)

- Thực hiện các hàm logic:
  - NOT, AND, OR, NAND, NOR, XOR
- Cổng logic một đầu vào:
  - Cổng NOT
- Cổng hai đầu vào:
  - AND, OR, XOR, NAND, NOR
- Cổng nhiều đầu vào

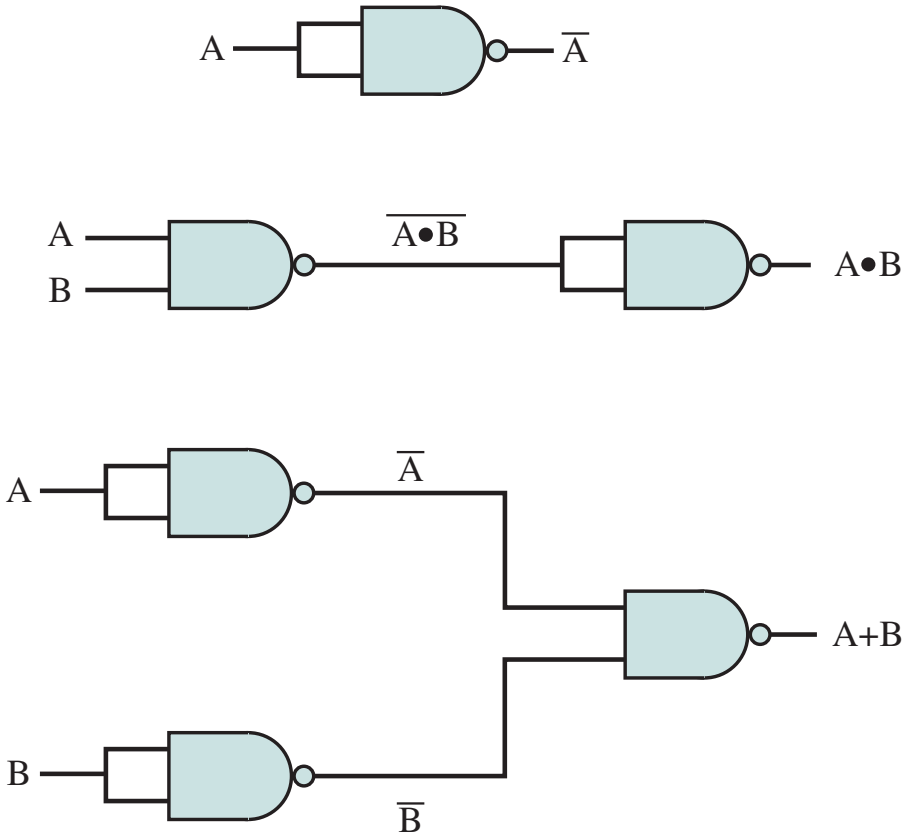
Ký hiệu các cổng logic

Name	Graphical Symbol	Algebraic Function	Truth Table
AND		$F = A \cdot B$ or $F = AB$	A B | F 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1
OR		$F = A + B$	A B | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1
NOT		$F = \overline{A}$ or $F = A'$	A | F 0 | 1 1 | 0
NAND		$F = \overline{AB}$	A B | F 0 0 | 1 0 1 | 1 1 0 | 1 1 1 | 0
NOR		$F = \overline{A + B}$	A B | F 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 0
XOR		$F = A \oplus B$	A B | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 0

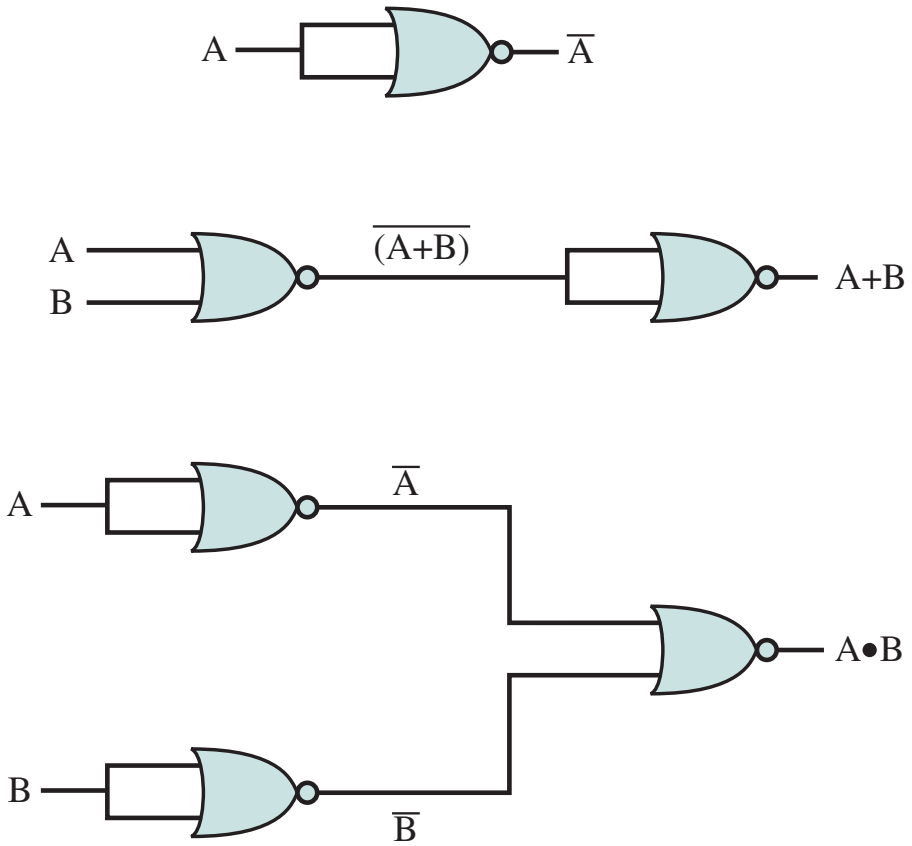
Tập đầy đủ

- Là tập các cổng có thể thực hiện được bất kỳ hàm logic nào từ các cổng của tập đó
- Một số ví dụ về tập đầy đủ:
  - {AND, OR, NOT}
  - {AND, NOT}
  - {OR, NOT}
  - {NAND}
  - {NOR}

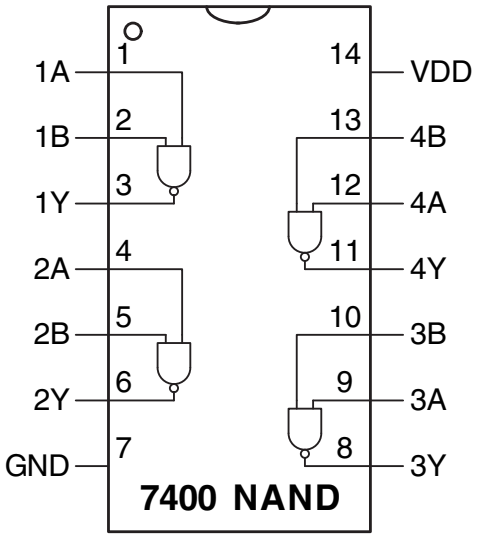
Sử dụng cổng NAND

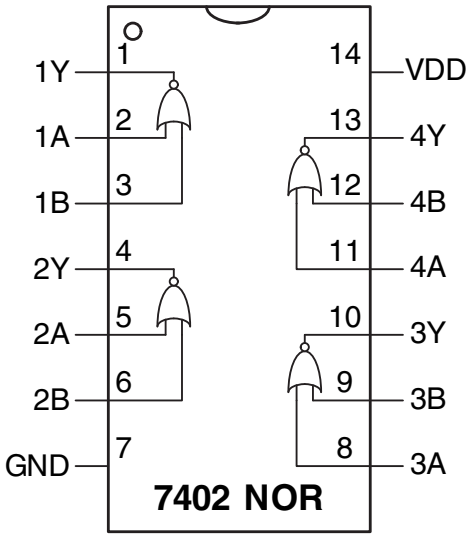


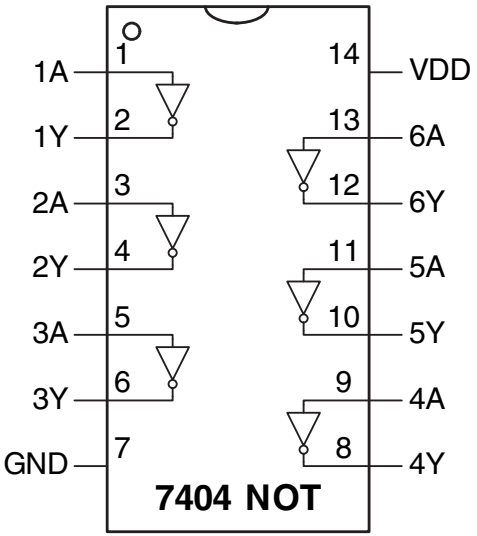
Sử dụng cổng NOR

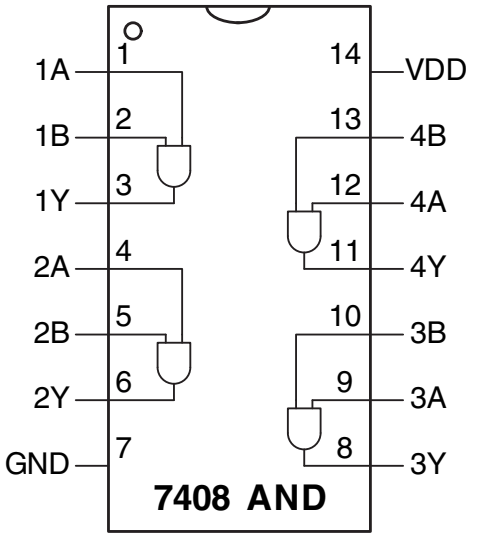


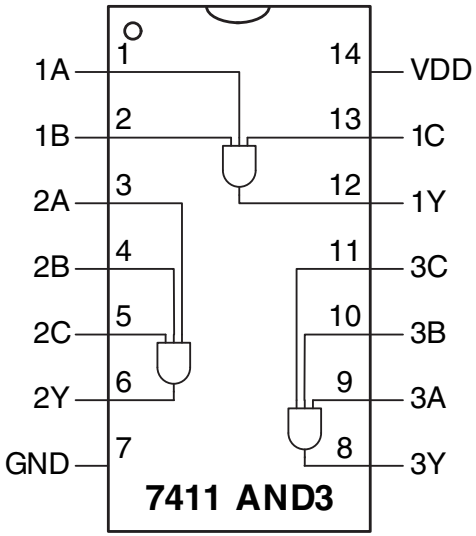
Một số vi mạch logic

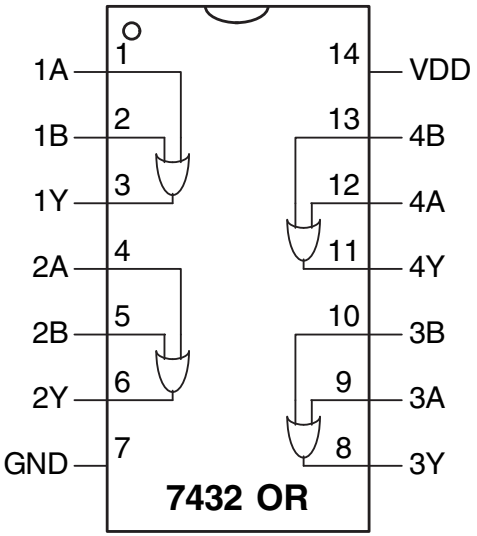












2.4. Mạch tổ hợp

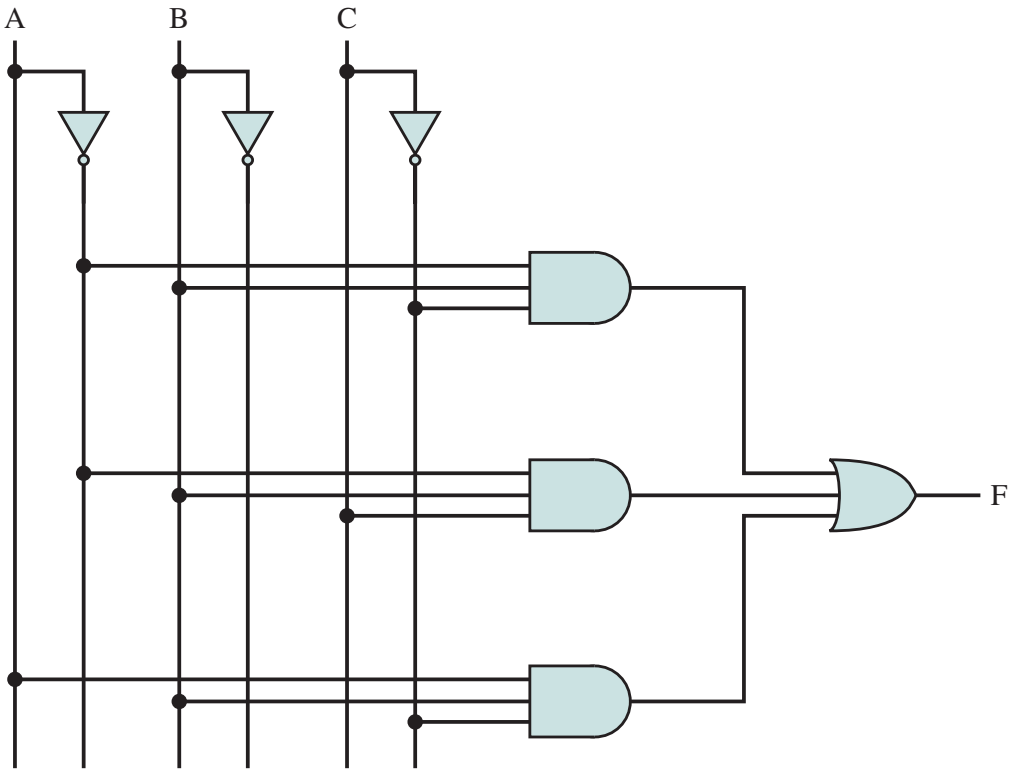
- Mạch logic là mạch bao gồm:
  - Các đầu vào (Inputs)
  - Các đầu ra (Outputs)
  - Đặc tả chức năng (Functional specification)
  - Đặc tả thời gian (Timing specification)
- Các kiểu mạch logic:
  - Mạch tổ hợp (Combinational Circuits)
    - Mạch không nhớ
    - Đầu ra được xác định bởi các giá trị hiện tại của đầu vào
  - Mạch dãy (Sequential Circuits)
    - Mạch có nhớ
    - Đầu ra được xác định bởi các giá trị trước đó và giá trị hiện tại của đầu vào

Mạch tổ hợp

- Mạch tổ hợp là mạch logic trong đó đầu ra chỉ phụ thuộc đầu vào ở thời điểm hiện tại
- Là mạch không nhớ và được thực hiện bằng các cổng logic
- Mạch tổ hợp có thể được định nghĩa theo ba cách:
  - Bảng thật (True Table)
  - Dạng sơ đồ
  - Phương trình Boole

Ví dụ

Đầu vào			Đầu ra
A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

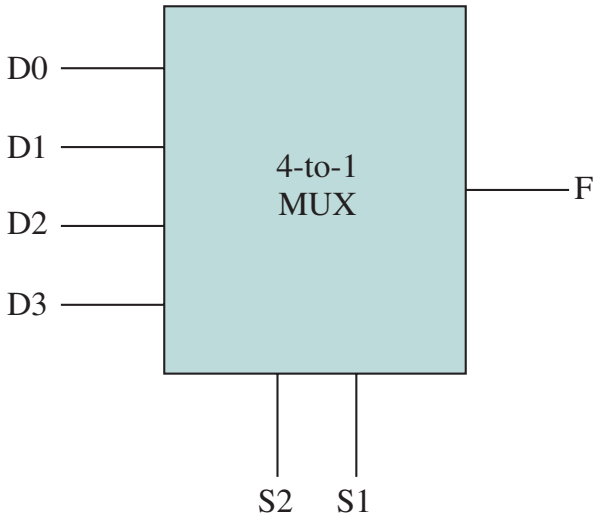


$$F = \overline{A}B\overline{C} + \overline{A}BC + AB\overline{C}$$

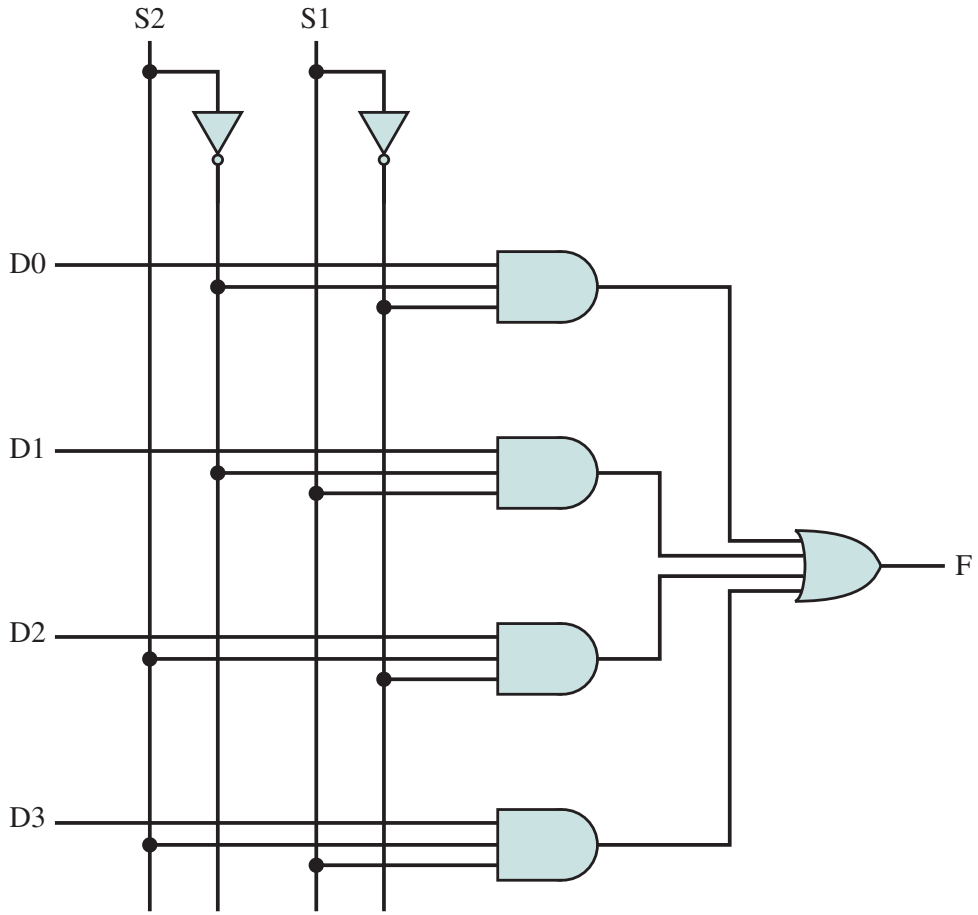
Bộ chọn kênh (Multiplexer - MUX)

- 2^n đầu vào dữ liệu
- n đầu vào chọn
- 1 đầu ra dữ liệu
- Mỗi tổ hợp đầu vào chọn (S) xác định đầu vào dữ liệu nào (D) sẽ được nối với đầu ra (F)

Bộ chọn kênh 4 đầu vào



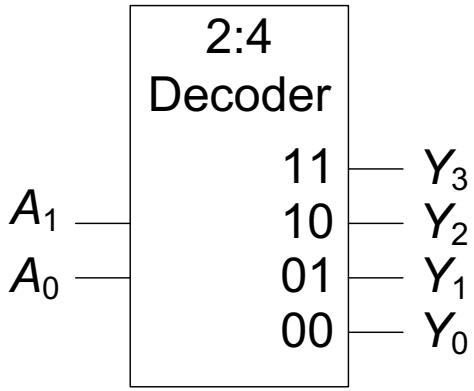
Đầu vào chọn		Đầu ra
S2	S1	F
0	0	D0
0	1	D1
1	0	D2
1	1	D3



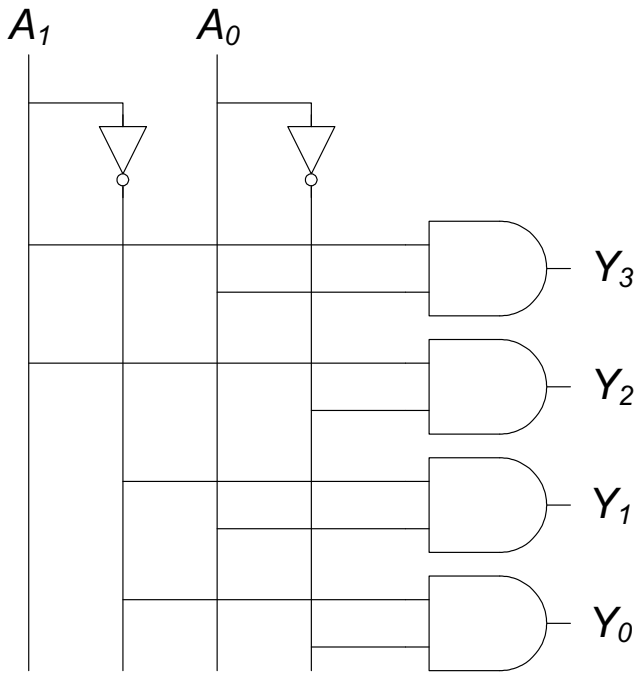
$$F = D0 \bullet \overline{S2} \bullet \overline{S1} + D1 \bullet \overline{S2} \bullet S1 + D2 \bullet S2 \bullet \overline{S1} + D3 \bullet S2 \bullet S1$$

Bộ giải mã (Decoder)

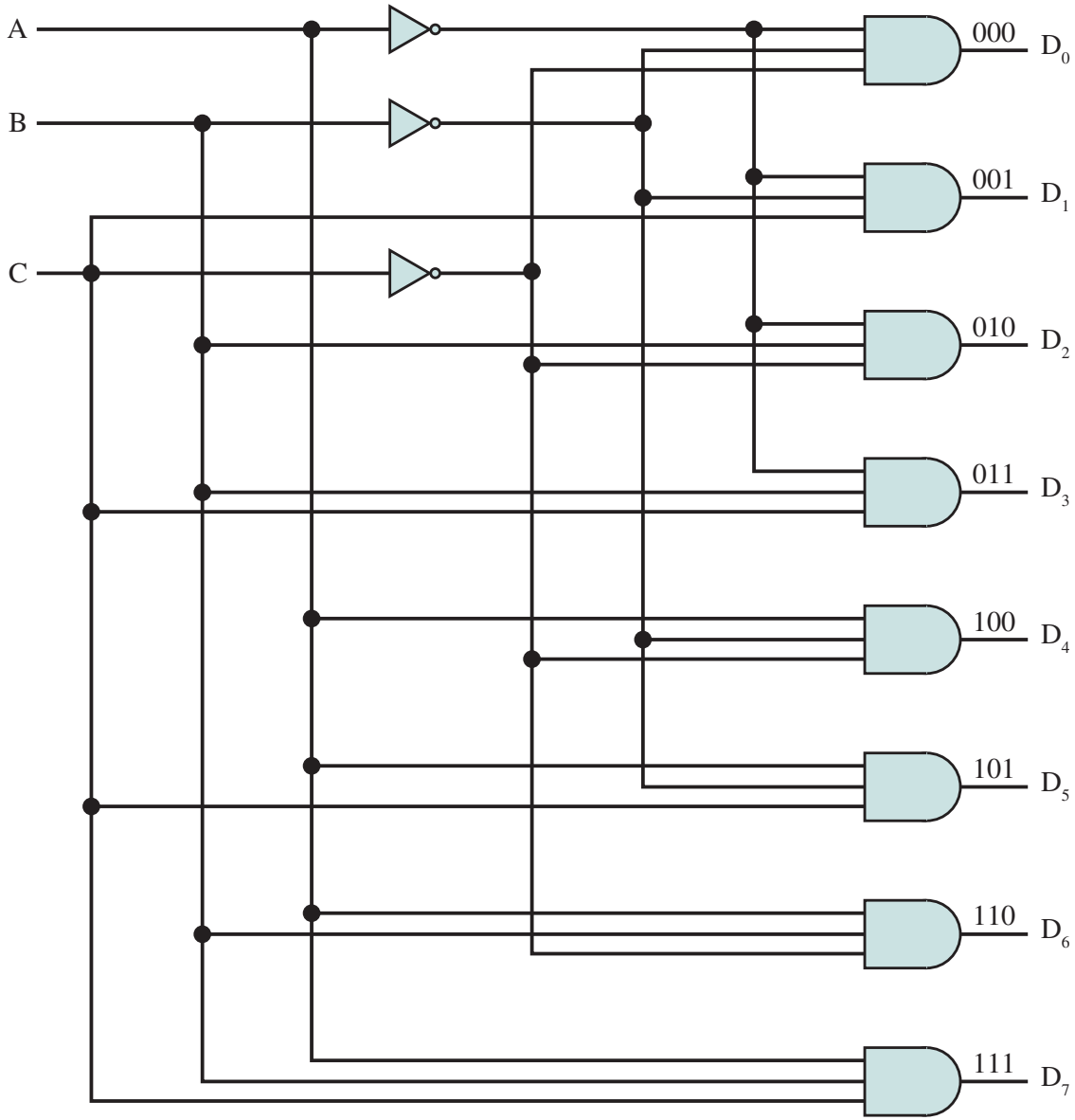
- *N* đầu vào, 2^N đầu ra
- Với một tổ hợp của N đầu vào, chỉ có một đầu ra tích cực (khác với các đầu ra còn lại)
- Ví dụ: Bộ giải mã 2 ra 4



A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0



Thực hiện bộ giải mã 3 ra 8

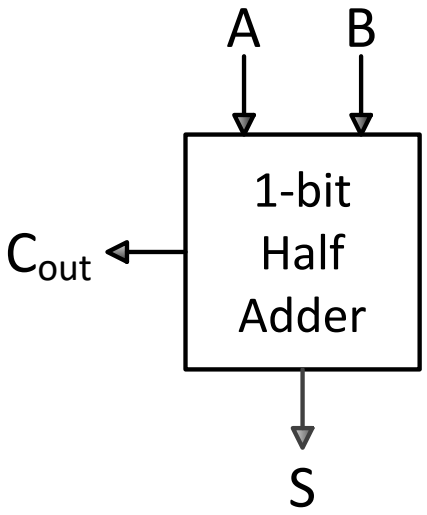


Bộ cộng

- Bộ cộng bán phần 1-bit (Half-adder)
  - Cộng hai bit tạo ra bit tổng và bit nhớ ra
- Bộ cộng toàn phần 1-bit (Full-adder)
  - Cộng 3 bit
  - Cho phép xây dựng bộ cộng N-bit

Bộ cộng bán phần 1-bit

0	0	1	1
+ 0	+ 1	+ 0	+ 1
0	1	1	10

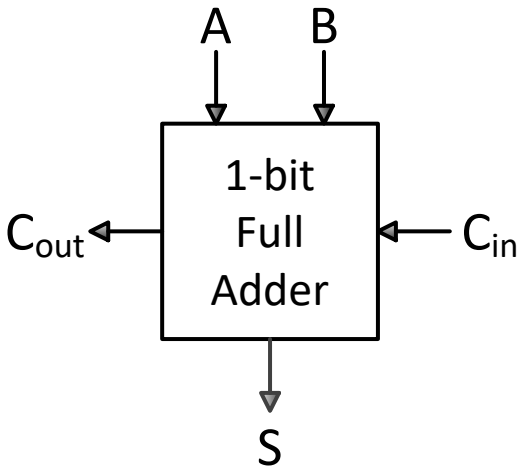


Đầu vào		Đầu ra	
A	B	S	C_{out}
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$$S = A \oplus B$$

$$C_{out} = AB$$

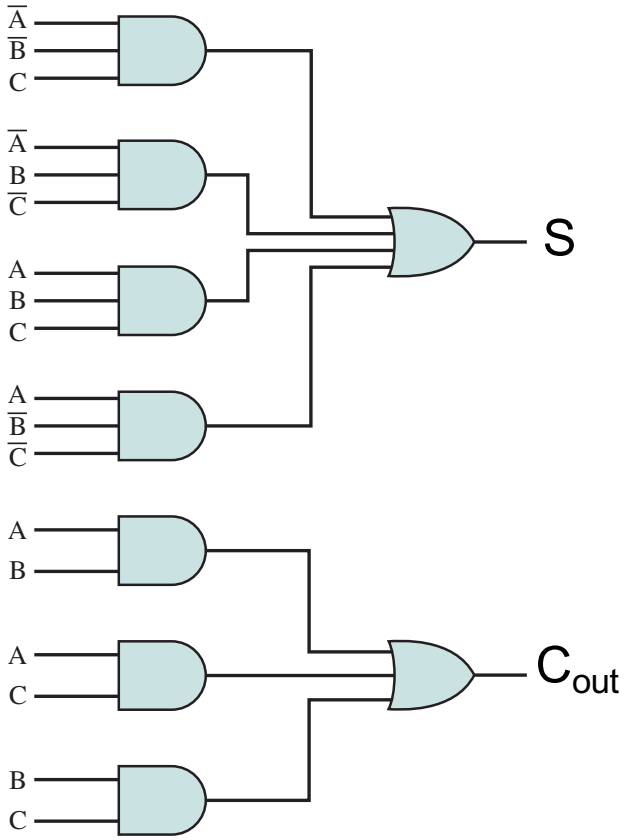
Bộ cộng toàn phần 1-bit



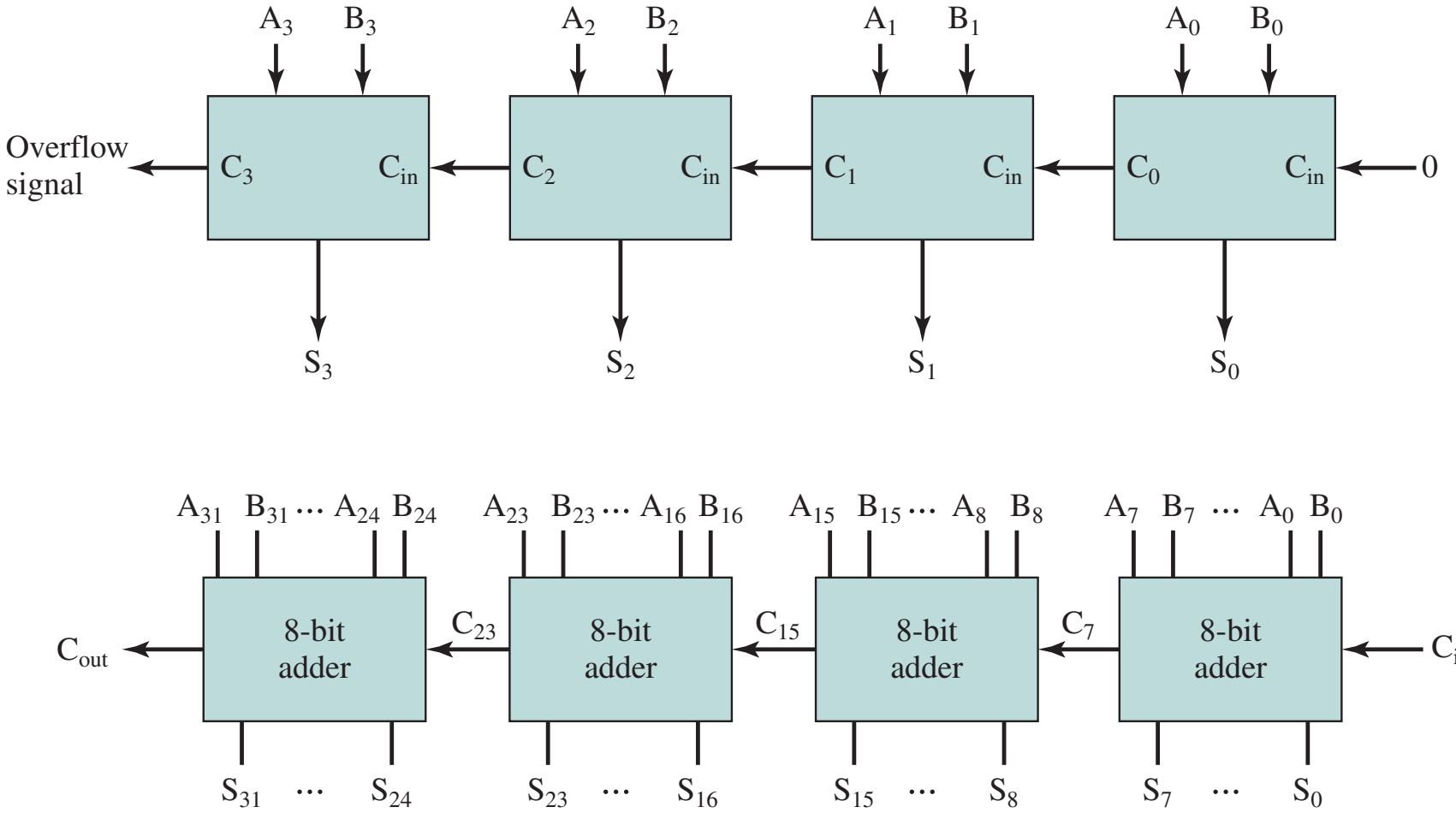
Đầu vào			Đầu ra	
C_{in}	A	B	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$$S = \overline{A}\,\overline{B}C + \overline{A}B\overline{C} + ABC + A\overline{B}\,\overline{C}$$

$$C_{out} = AB + AC + BC$$



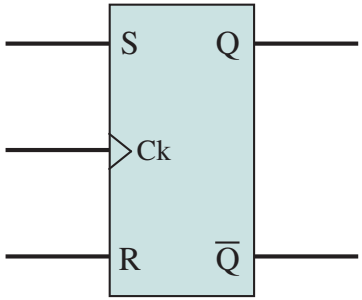
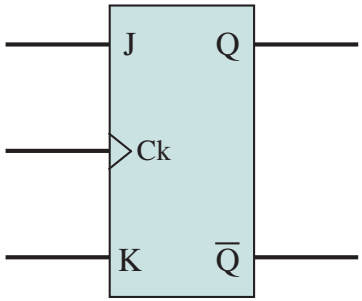
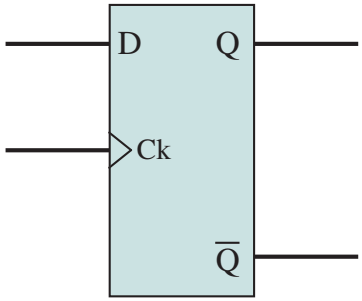
Bộ cộng 4-bit và bộ cộng 32-bit



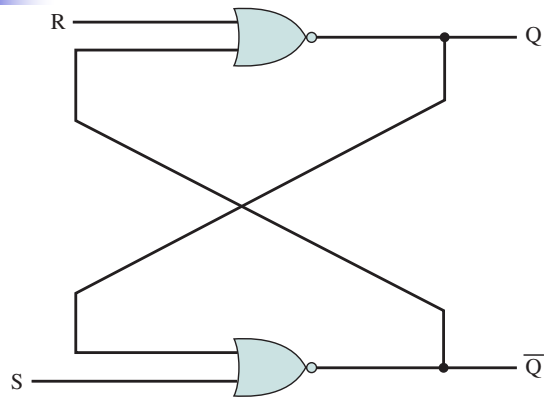
2.5. Mạch dãy

- Mạch dãy là mạch logic trong đó đầu ra phụ thuộc giá trị đầu vào ở thời điểm hiện tại và đầu vào ở thời điểm quá khứ
- Là mạch có nhớ, được thực hiện bằng phần tử nhớ (Latch, Flip-Flop) và có thể kết hợp với các cổng logic

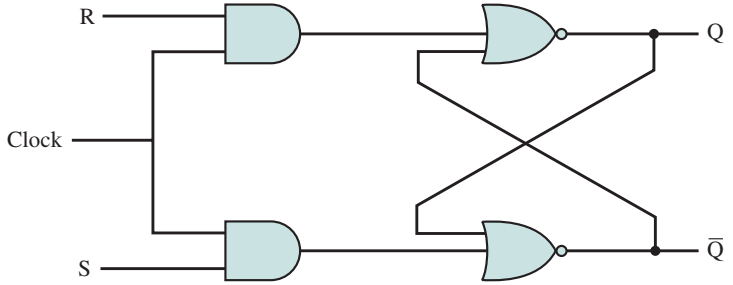
Các Flip-Flop cơ bản

Name	Graphical Symbol	Truth Table
S–R		S R Q_{n+1} 0 0 Q_n 0 1 0 1 0 1 1 1 –
J–K		J K Q_{n+1} 0 0 Q_n 0 1 0 1 0 1 1 1 $\overline{Q}_n$
D		D Q_{n+1} 0 0 1 1

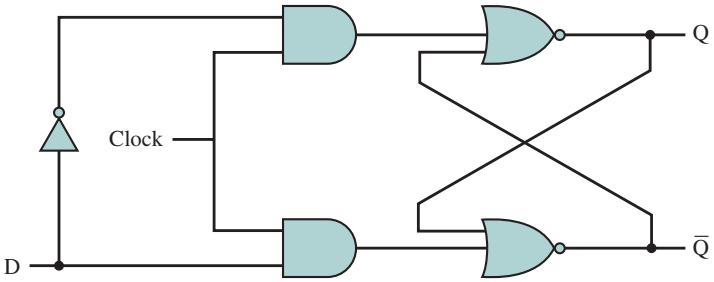
S-R Latch và các Flip-Flop



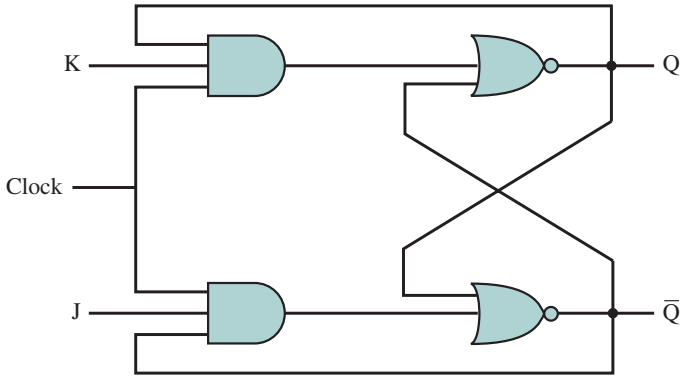
S-R Latch



S-R Flip-Flop

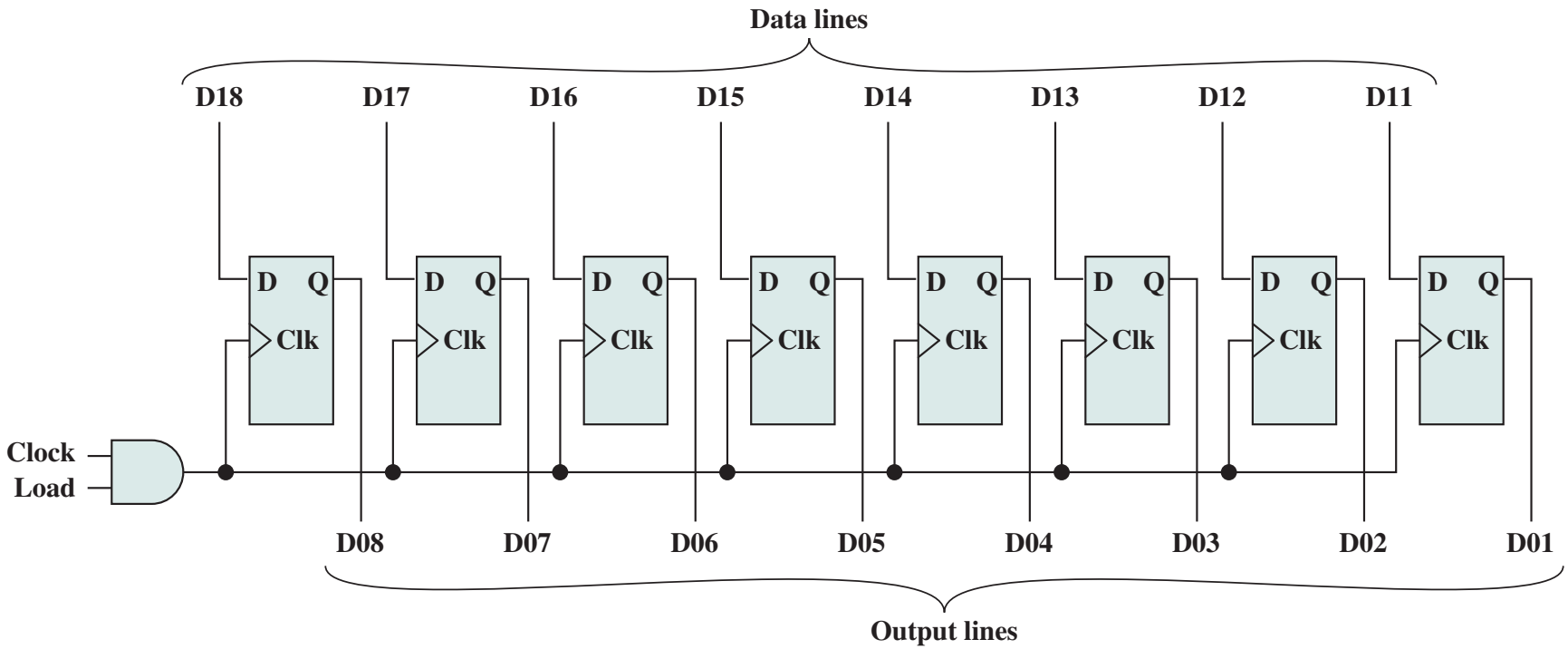


D Flip Flop

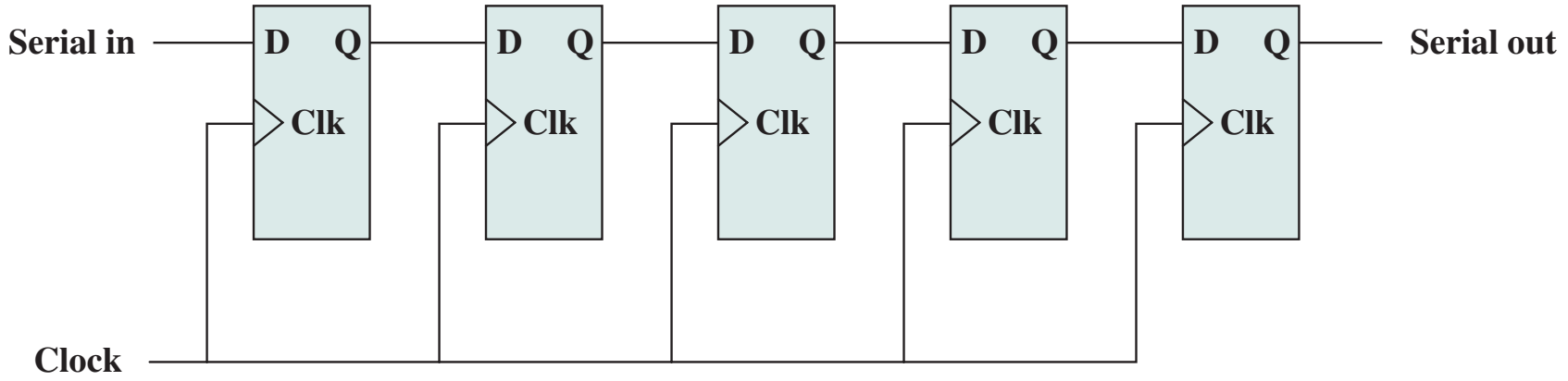


J-K Flip-Flop

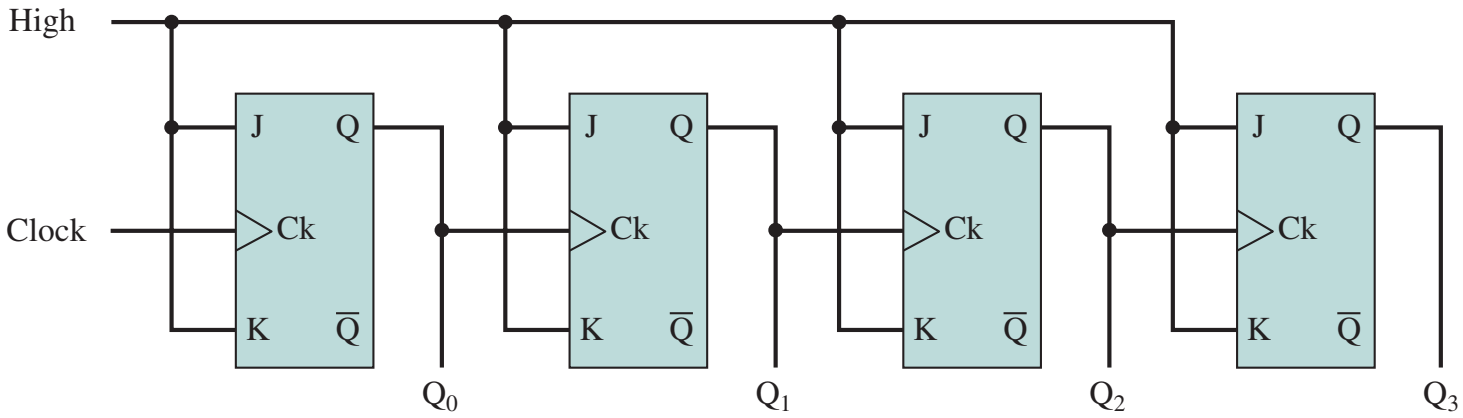
Thanh ghi 8-bit song song



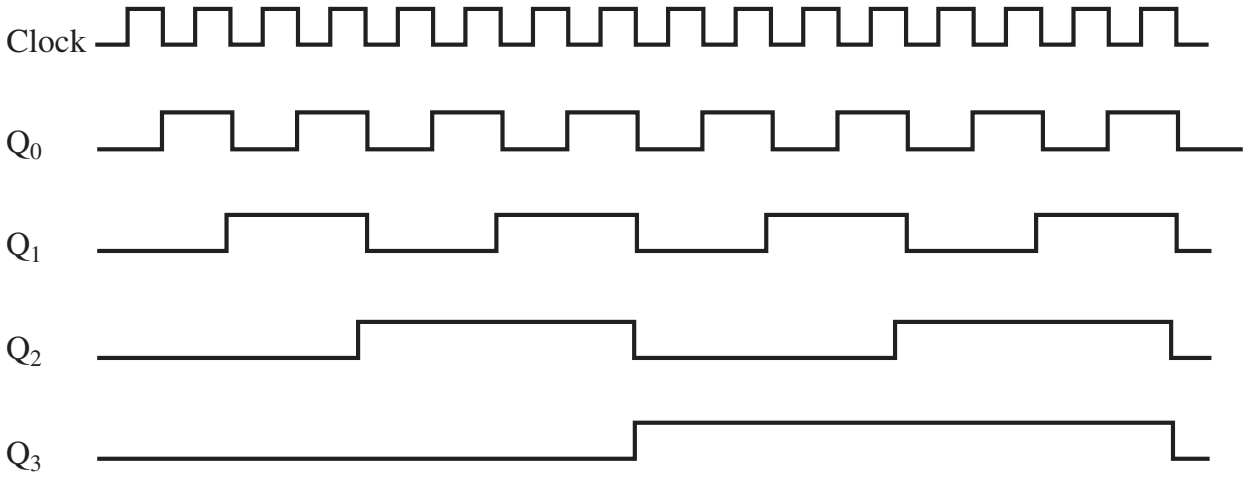
Thanh ghi dịch 5-bit



Bộ đếm 4-bit



(a) Sequential circuit



(b) Timing diagram

Hết chương 2

Chương 3
HỆ THỐNG MÁY TÍNH

Nguyễn Kim Khánh
Trường Đại học Bách khoa Hà Nội

Nội dung học phần

Chương 1. Giới thiệu chung

Chương 2. Cơ bản về logic số

Chương 3. Hệ thống máy tính

Chương 4. Số học máy tính

Chương 5. Kiến trúc tập lệnh

Chương 6. Bộ xử lý

Chương 7. Bộ nhớ máy tính

Chương 8. Hệ thống vào-ra

Chương 9. Các kiến trúc song song

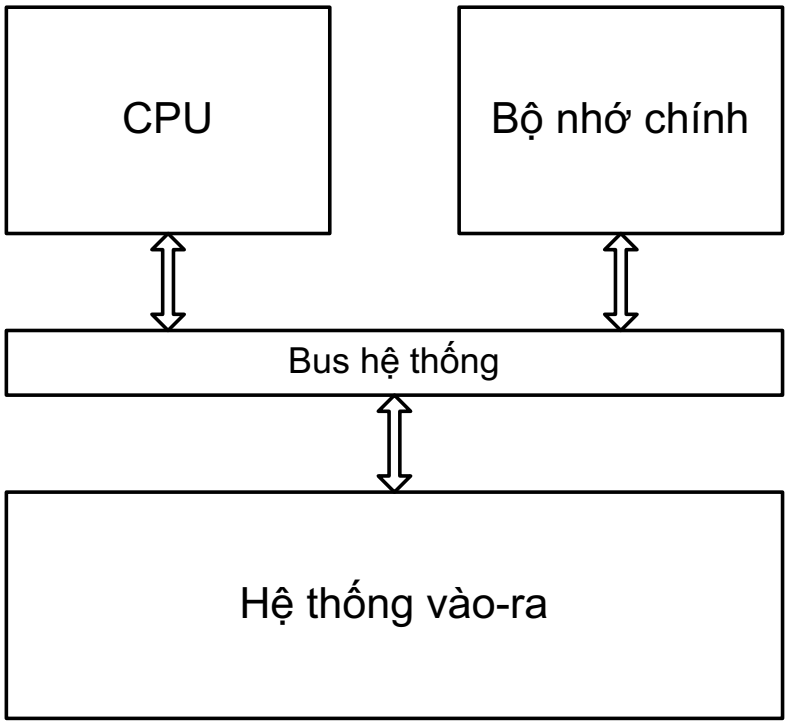
Nội dung của chương 3

3.1. Các thành phần cơ bản của máy tính

3.2. Hoạt động cơ bản của máy tính

3.3. Bus máy tính

3.1. Các thành phần cơ bản của máy tính

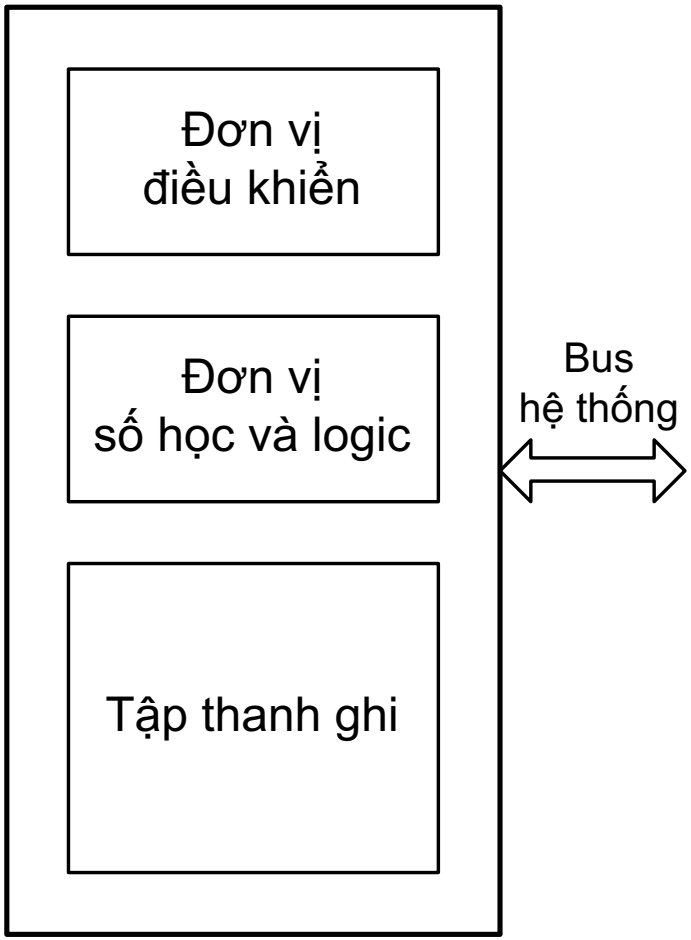


- Bộ xử lý trung tâm (Central Processing Unit – CPU)
  - Điều khiển hoạt động của máy tính và xử lý dữ liệu
- Bộ nhớ chính (Main Memory)
  - Chứa các chương trình đang thực hiện
- Hệ thống vào-ra (Input/Output)
  - Trao đổi thông tin giữa máy tính với bên ngoài
- Bus hệ thống (System bus)
  - Kết nối và vận chuyển thông tin

1. Bộ xử lý trung tâm (CPU)

- Chức năng:
  - điều khiển hoạt động của máy tính
  - xử lý dữ liệu
- Nguyên tắc hoạt động cơ bản:
  - CPU hoạt động theo chương trình nằm trong bộ nhớ chính.
- Là thành phần nhanh nhất trong hệ thống

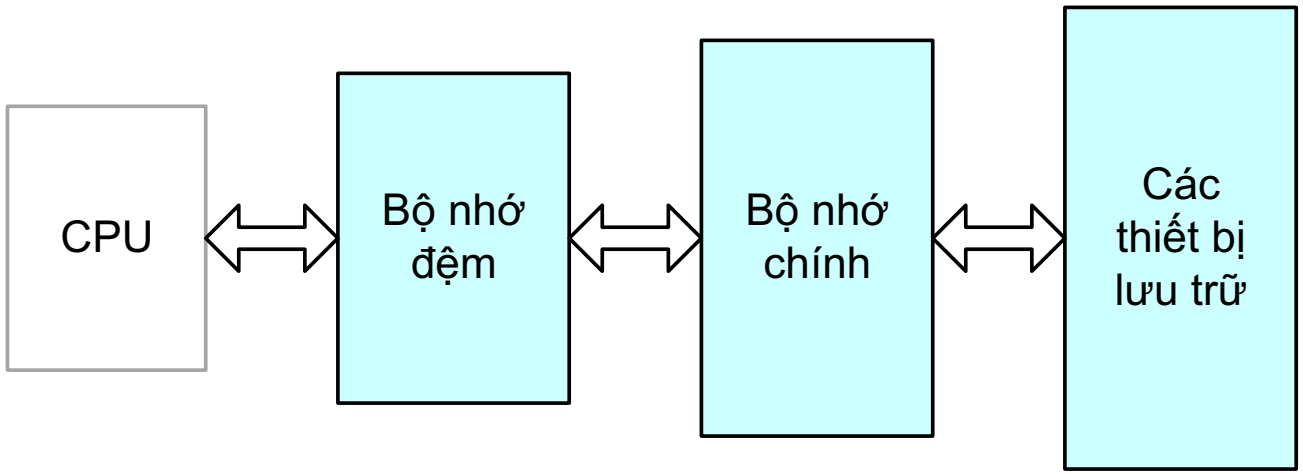
Các thành phần cơ bản của CPU



- Đơn vị điều khiển
  - *Control Unit (CU)*
  - Điều khiển hoạt động của máy tính theo chương trình đã định sẵn
- Đơn vị số học và logic
  - *Arithmetic and Logic Unit (ALU)*
  - Thực hiện các phép toán số học và phép toán logic
- Tập thanh ghi
  - *Register File (RF)*
  - Gồm các thanh ghi chứa các thông tin phục vụ cho hoạt động của CPU

2. Bộ nhớ máy tính

- Chức năng: nhớ chương trình và dữ liệu (dưới dạng nhị phân)
- Các thao tác cơ bản với bộ nhớ:
  - Thao tác ghi (Write)
  - Thao tác đọc (Read)
- Các thành phần chính:
  - Bộ nhớ chính (Main memory)
  - Bộ nhớ đệm (Cache memory)
  - Thiết bị lưu trữ (Storage Devices)



Bộ nhớ chính (Main memory)

- Tồn tại trên mọi máy tính
- Chứa các lệnh và dữ liệu của chương trình đang được thực hiện
- Sử dụng bộ nhớ bán dẫn
- Tổ chức thành các ngăn nhớ được đánh địa chỉ (thường đánh địa chỉ cho từng byte nhớ)
- Nội dung của ngăn nhớ có thể thay đổi, song địa chỉ vật lý của ngăn nhớ luôn cố định
- CPU muốn đọc/ghi ngăn nhớ cần phải biết địa chỉ ngăn nhớ đó

Nội dung	Địa chỉ
0100 1101	00...0000
0101 0101	00...0001
1010 1111	00...0010
0000 1110	00...0011
0111 0100	00...0100
1011 0010	00...0101
0010 1000	00...0110
1110 1111	00...0111
.	
.	
.	
0110 0010	11...1110
0010 0001	11...1111

Bộ nhớ đệm (Cache memory)

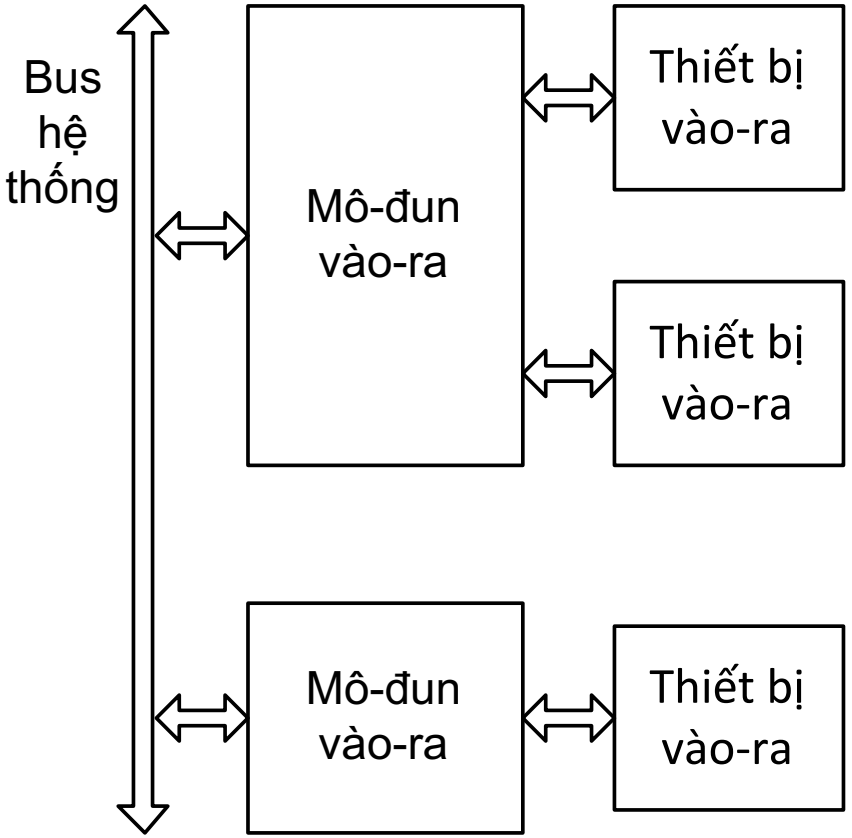
- Bộ nhớ có tốc độ nhanh được đặt đệm giữa CPU và bộ nhớ chính nhằm tăng tốc độ CPU truy cập bộ nhớ
- Dung lượng nhỏ hơn bộ nhớ chính
- Sử dụng bộ nhớ bán dẫn tốc độ nhanh
- Cache thường được chia thành một số mức (L1, L2, L3)
- Cache thường được tích hợp trên cùng chip bộ xử lý
- Cache có thể có hoặc không

Thiết bị lưu trữ (Storage Devices)

- Còn được gọi là bộ nhớ ngoài
- Chức năng và đặc điểm
  - Lưu giữ tài nguyên phần mềm của máy tính
  - Được kết nối với hệ thống dưới dạng các thiết bị vào-ra
  - Dung lượng lớn
  - Tốc độ chậm
- Các loại thiết bị lưu trữ
  - Bộ nhớ từ: ổ đĩa cứng HDD
  - Bộ nhớ bán dẫn: ổ thể rắn SSD, ổ nhớ flash, thẻ nhớ
  - Bộ nhớ quang: CD, DVD

3. Hệ thống vào-ra

- Chức năng: Trao đổi thông tin giữa máy tính với thế giới bên ngoài
- Các thao tác cơ bản:
  - Vào dữ liệu (Input)
  - Ra dữ liệu (Output)
- Các thành phần chính:
  - Các thiết bị vào-ra (IO devices)
  - Các mô-đun vào-ra (IO modules)



Các thiết bị vào-ra

- Còn được gọi là thiết bị ngoại vi (Peripherals)
- Chức năng: chuyển đổi dữ liệu giữa bên trong và bên ngoài máy tính
- Các loại thiết bị vào-ra:
  - Thiết bị vào (Input Devices)
  - Thiết bị ra (Output Devices)
  - Thiết bị lưu trữ (Stotage Devices)
  - Thiết bị truyền thông (Communication Devives)

Mô-đun vào-ra

- Chức năng: nối ghép các thiết bị vào-ra với máy tính
- Mỗi mô-đun vào-ra có một hoặc một vài cổng vào-ra (I/O Port)
- Mỗi cổng vào-ra được đánh một địa chỉ xác định
- Các thiết bị vào-ra được kết nối và trao đổi dữ liệu với máy tính thông qua các cổng vào-ra
- CPU muốn trao đổi dữ liệu với thiết bị vào-ra, cần phải biết địa chỉ của cổng vào-ra tương ứng

3.2. Hoạt động cơ bản của máy tính

- Thực hiện chương trình
- Hoạt động ngắt
- Hoạt động vào-ra

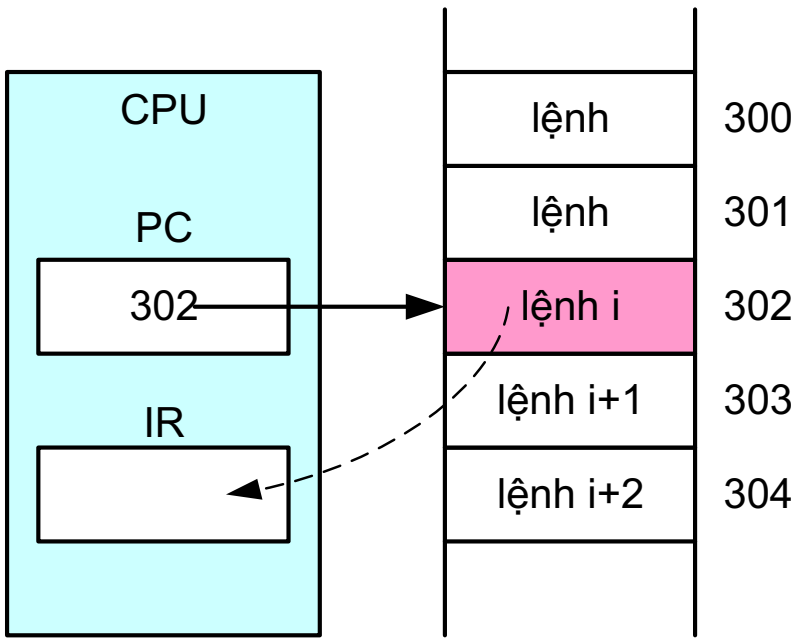
1. Thực hiện chương trình

- Là hoạt động cơ bản của máy tính
- Máy tính lặp đi lặp lại chu trình lệnh gồm hai bước:
  - Nhận lệnh
  - Thực hiện lệnh
- Hoạt động thực hiện chương trình bị dừng nếu:
  - Thực hiện lệnh bị lỗi
  - Gặp lệnh dừng
  - Tắt máy

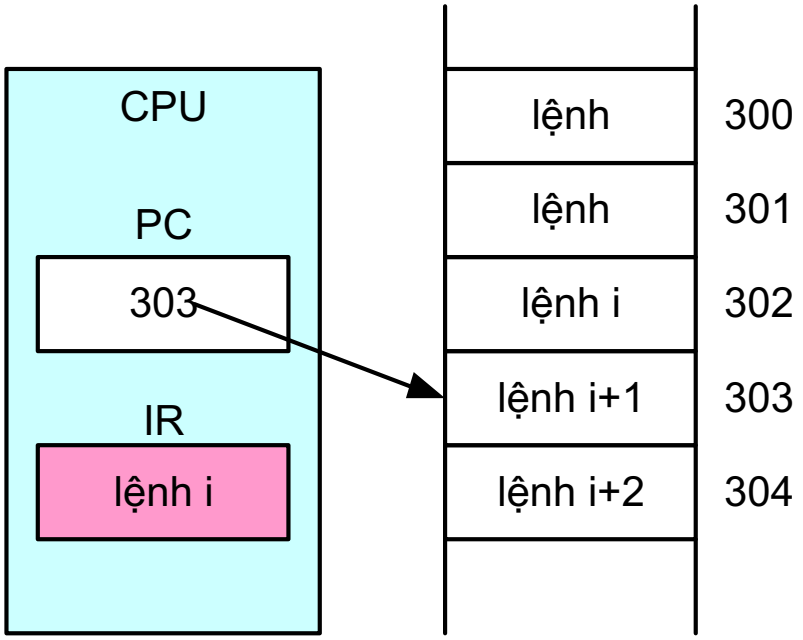
Nhận lệnh

- Bắt đầu mỗi chu trình lệnh, CPU nhận lệnh từ bộ nhớ chính
- Bộ đếm chương trình PC (Program Counter) là thanh ghi của CPU dùng để giữ địa chỉ của lệnh sẽ được nhận vào
- CPU phát ra địa chỉ từ bộ đếm chương trình PC tìm ra ngăn nhớ chứa lệnh
- Lệnh được đọc từ bộ nhớ đưa vào thanh ghi lệnh IR (Instruction Register)
- Sau khi lệnh được nhận vào, nội dung PC tự động tăng để trỏ đến lệnh kế tiếp.

Minh họa quá trình nhận lệnh



Trước khi nhận lệnh i



Sau khi nhận lệnh i

Thực hiện lệnh

- Bộ xử lý giải mã lệnh đã được nhận và phát tín hiệu điều khiển thực hiện thao tác mà lệnh yêu cầu
- Các kiểu thao tác cơ bản của lệnh:
  - Trao đổi dữ liệu giữa CPU với bộ nhớ chính hoặc CPU với mô-đun vào-ra
  - Thực hiện các phép toán số học hoặc phép toán logic với các dữ liệu
  - Chuyển điều khiển trong chương trình: rẽ nhánh hoặc nhảy đến vị trí khác

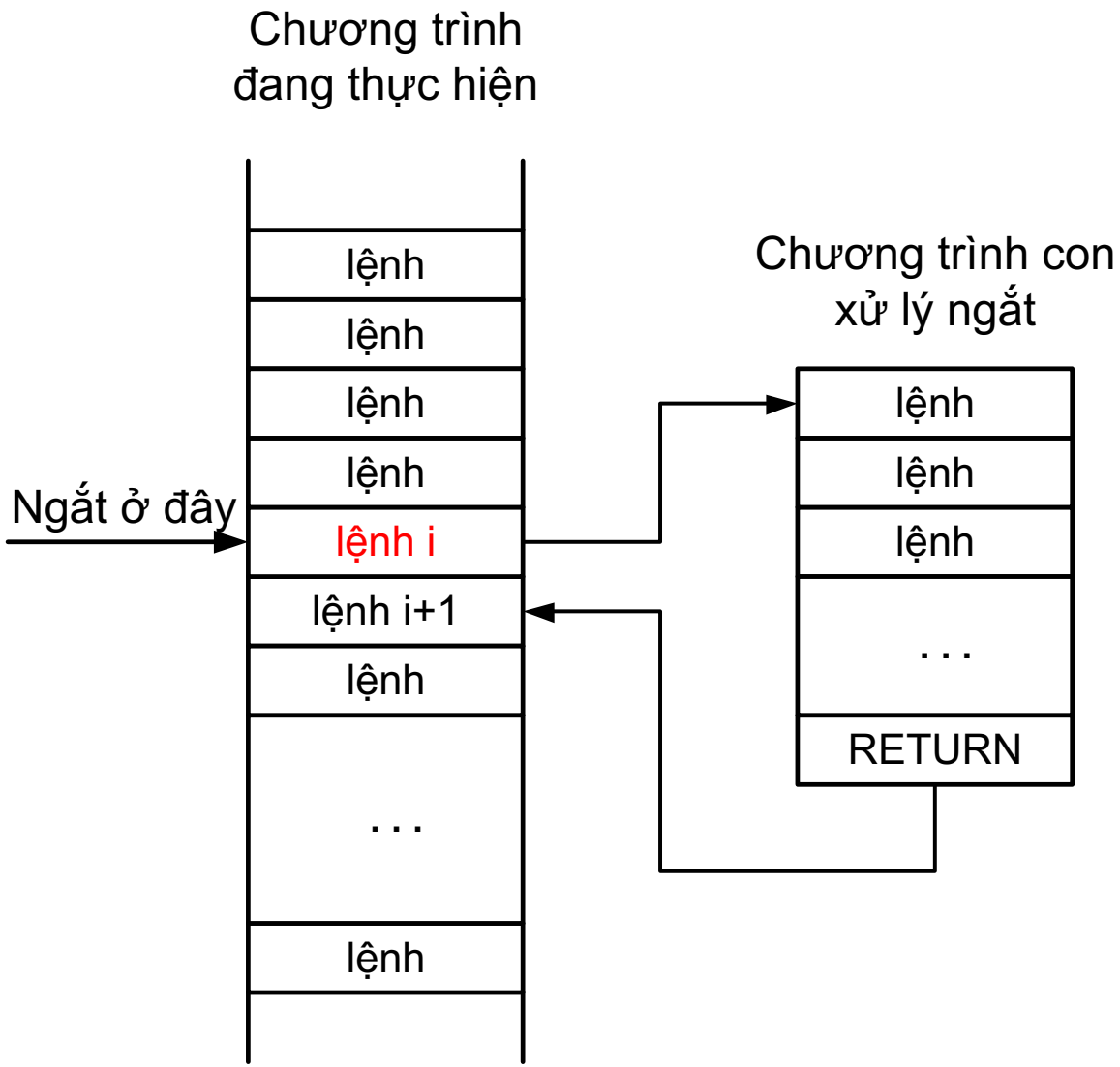
2. Ngắt (Interrupt)

- Khái niệm chung về ngắt: Ngắt là cơ chế cho phép CPU tạm dừng chương trình đang thực hiện để chuyển sang thực hiện một chương trình con có sẵn trong bộ nhớ.
  - Chương trình con xử lý ngắt (Interrupt handlers)
- Các loại ngắt:
  - Biệt lệ (exception): gây ra do lỗi khi thực hiện chương trình (VD: tràn số, mã lệnh sai, ...)
  - Ngắt từ bên ngoài (external interrupt): do thiết bị vào-ra (thông qua mô-đun vào-ra) gửi tín hiệu ngắt đến CPU để yêu cầu trao đổi dữ liệu

Hoạt động với ngắt từ bên ngoài

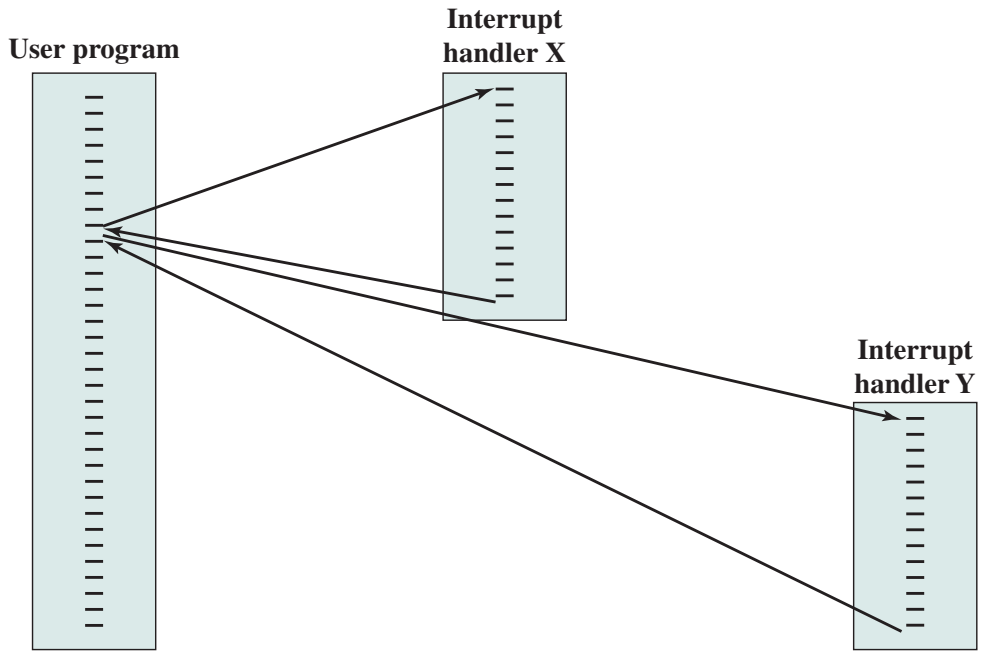
- Sau khi hoàn thành mỗi một lệnh, bộ xử lý kiểm tra tín hiệu ngắt
- Nếu không có ngắt, bộ xử lý nhận lệnh tiếp theo của chương trình hiện tại
- Nếu có tín hiệu ngắt:
  - Tạm dừng (suspend) chương trình đang thực hiện
  - Cất ngữ cảnh (các thông tin liên quan đến chương trình bị ngắt)
  - Thiết lập bộ đếm chương trình PC trỏ đến chương trình con xử lý ngắt tương ứng
  - Chuyển sang thực hiện chương trình con xử lý ngắt
  - Khôi phục ngữ cảnh và trở về tiếp tục thực hiện chương trình đang bị tạm dừng

Hoạt động ngắt (tiếp)



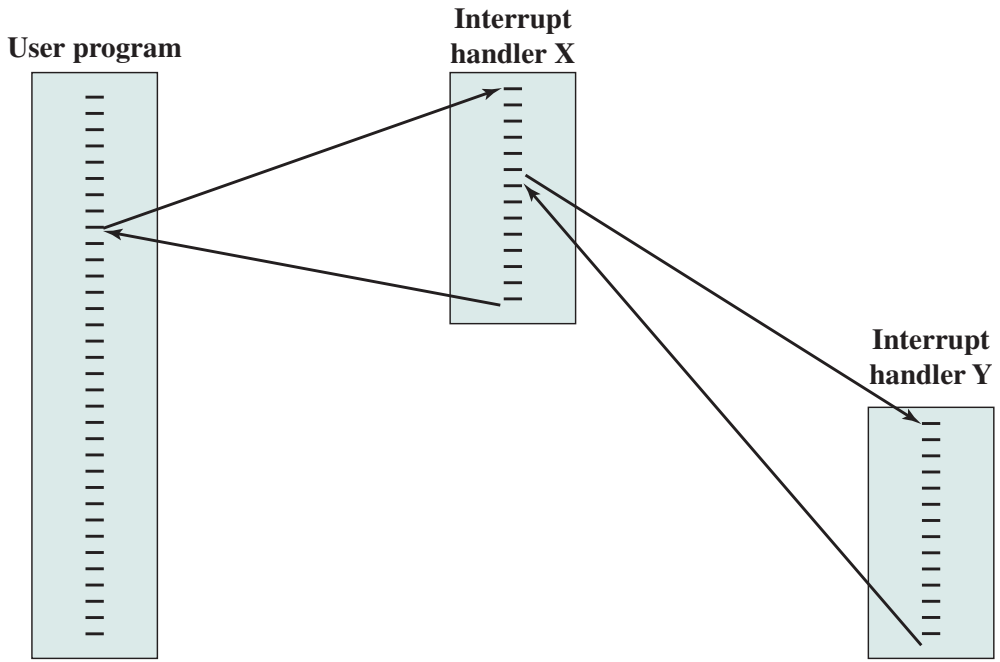
Xử lý với nhiều tín hiệu yêu cầu ngắt

- Xử lý ngắt tuần tự
  - Khi một ngắt đang được thực hiện, các ngắt khác bị cấm (disabled interrupt)
  - Bộ xử lý sẽ bỏ qua các yêu cầu ngắt tiếp theo
  - Các yêu cầu ngắt tiếp theo vẫn đang đợi và được kiểm tra sau khi ngắt hiện tại được xử lý xong
  - Các ngắt được thực hiện tuần tự



Xử lý với nhiều tín hiệu yêu cầu ngắt (tiếp)

- Xử lý ngắt ưu tiên
  - Các ngắt được định nghĩa mức ưu tiên khác nhau
  - Ngắt có mức ưu tiên thấp hơn có thể bị ngắt bởi ngắt có mức ưu tiên cao hơn
  - Xẩy ra ngắt lồng nhau



3. Hoạt động vào-ra

- Hoạt động vào-ra: là hoạt động trao đổi dữ liệu giữa mô-đun vào-ra với bên trong máy tính.
- Các kiểu hoạt động vào-ra:
  - CPU trao đổi dữ liệu với mô-đun vào-ra bởi lệnh vào-ra trong chương trình
  - CPU trao quyền điều khiển cho phép mô-đun vào-ra trao đổi dữ liệu trực tiếp với bộ nhớ chính (DMA - Direct Memory Access).

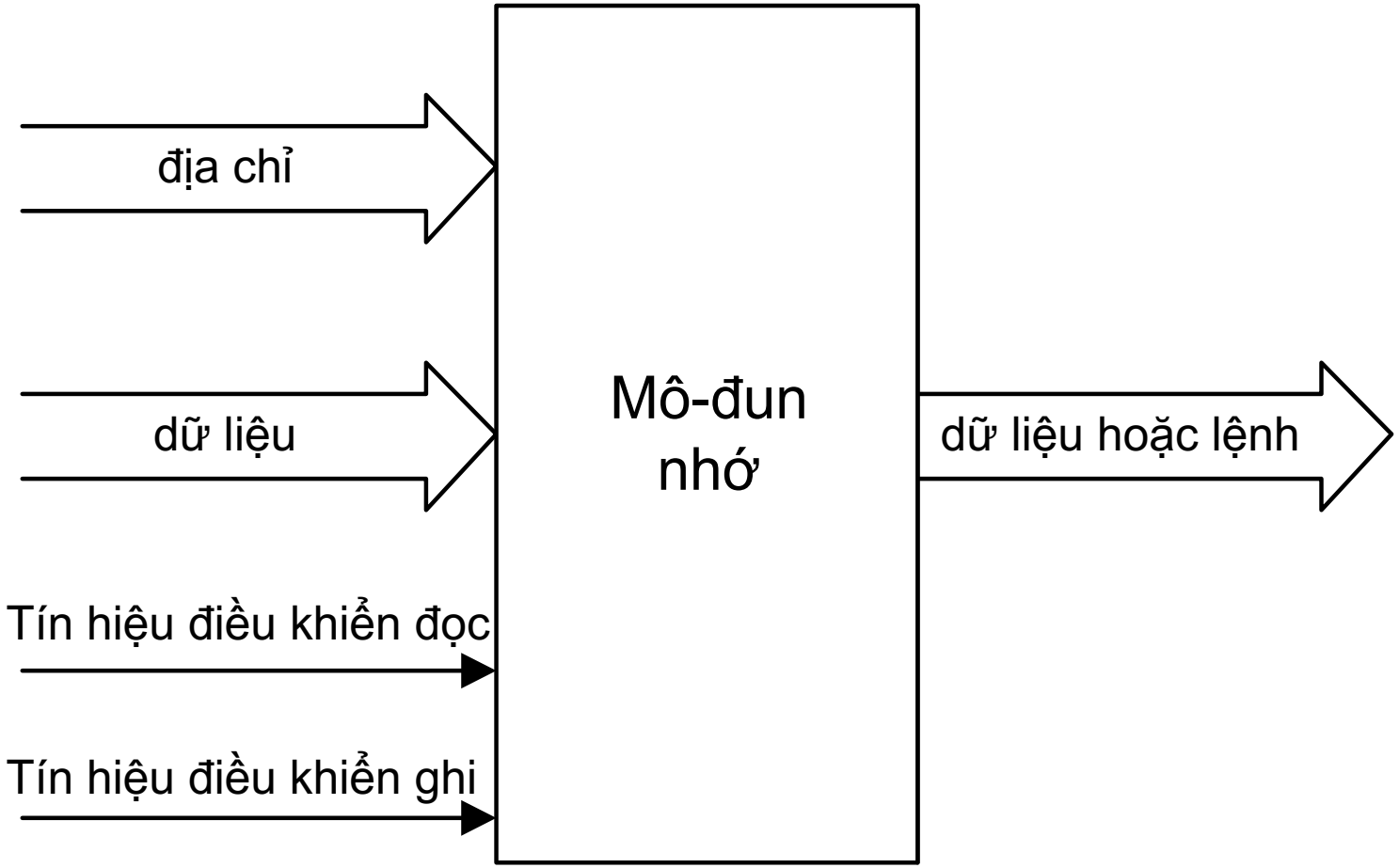
3.3. Bus máy tính

1. Luồng thông tin trong máy tính

- Các mô-đun trong máy tính:
  - CPU
  - Mô-đun nhớ
  - Mô-đun vào-ra

→ cần được kết nối với nhau

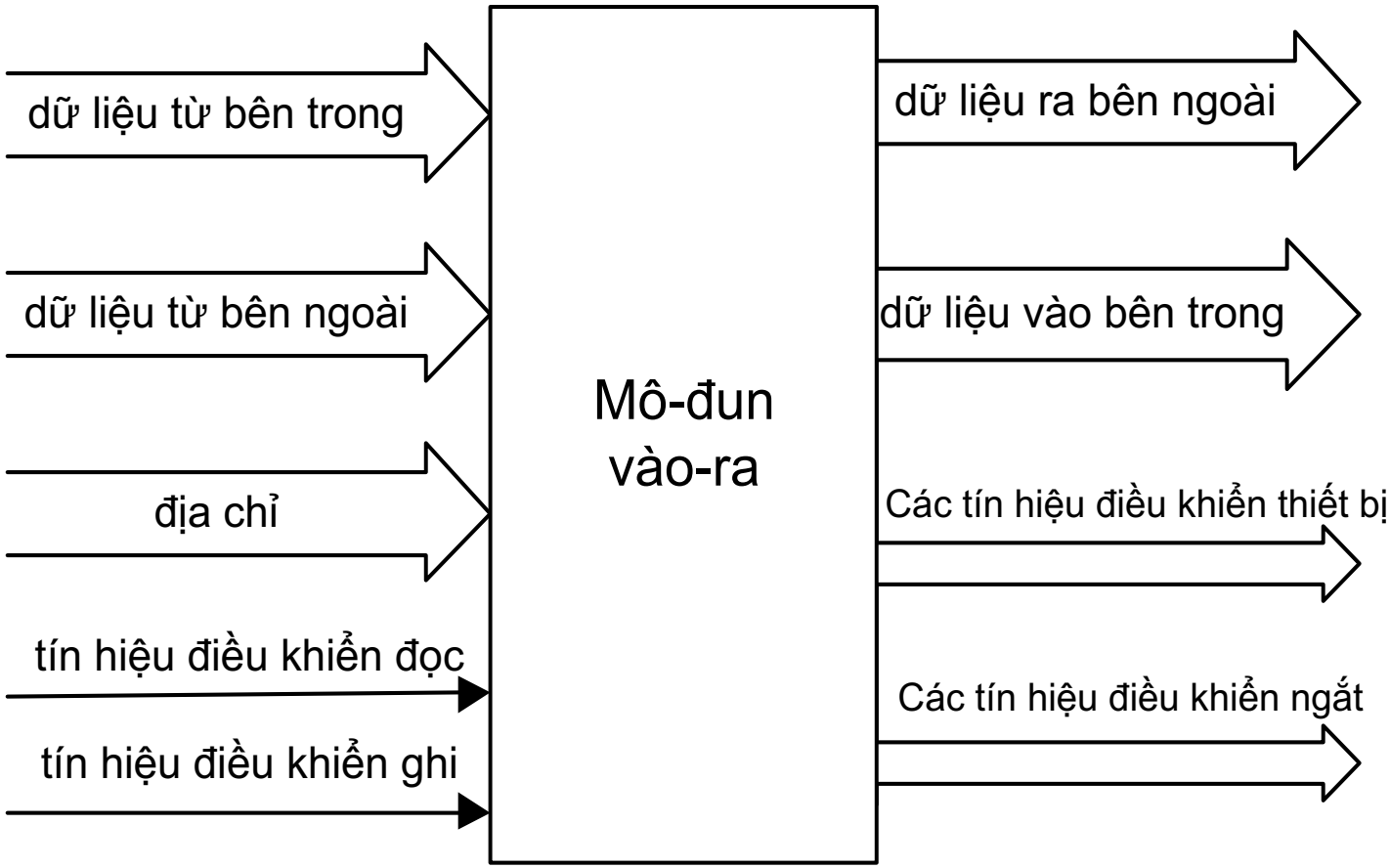
Kết nối mô-đun nhớ



Kết nối mô-đun nhớ (tiếp)

- Địa chỉ đưa đến để xác định ngăn nhớ
- Dữ liệu được đưa đến khi ghi
- Dữ liệu hoặc lệnh được đưa ra khi đọc
  - Bộ nhớ không phân biệt lệnh và dữ liệu
- Nhận các tín hiệu điều khiển:
  - Điều khiển đọc (Read)
  - Điều khiển ghi (Write)

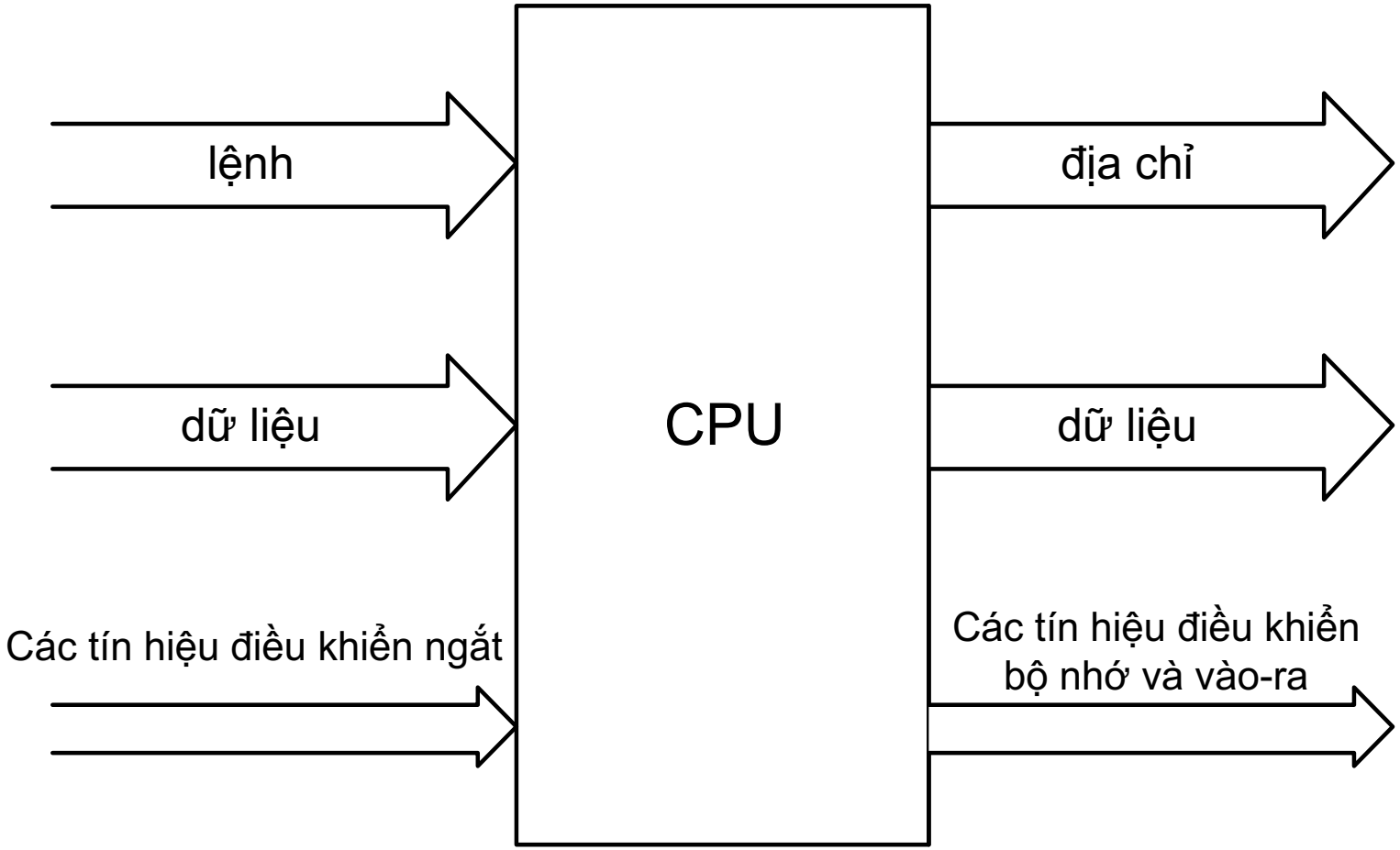
Kết nối mô-đun vào-ra



Kết nối mô-đun vào-ra (tiếp)

- Địa chỉ đưa đến để xác định cổng vào-ra
- Ra dữ liệu (Output)
  - Nhận dữ liệu từ bên trong (CPU hoặc bộ nhớ chính)
  - Đưa dữ liệu ra thiết bị vào-ra
- Vào dữ liệu (Input)
  - Nhận dữ liệu từ thiết bị vào-ra
  - Đưa dữ liệu vào bên trong (CPU hoặc bộ nhớ chính)
- Nhận các tín hiệu điều khiển từ CPU
- Phát các tín hiệu điều khiển đến thiết bị vào-ra
- Phát các tín hiệu ngắt đến CPU

Kết nối CPU


lệnh
dữ liệu
Các tín hiệu điều khiển ngắt
CPU
địa chỉ
dữ liệu
Các tín hiệu điều khiển
bộ nhớ và vào-ra

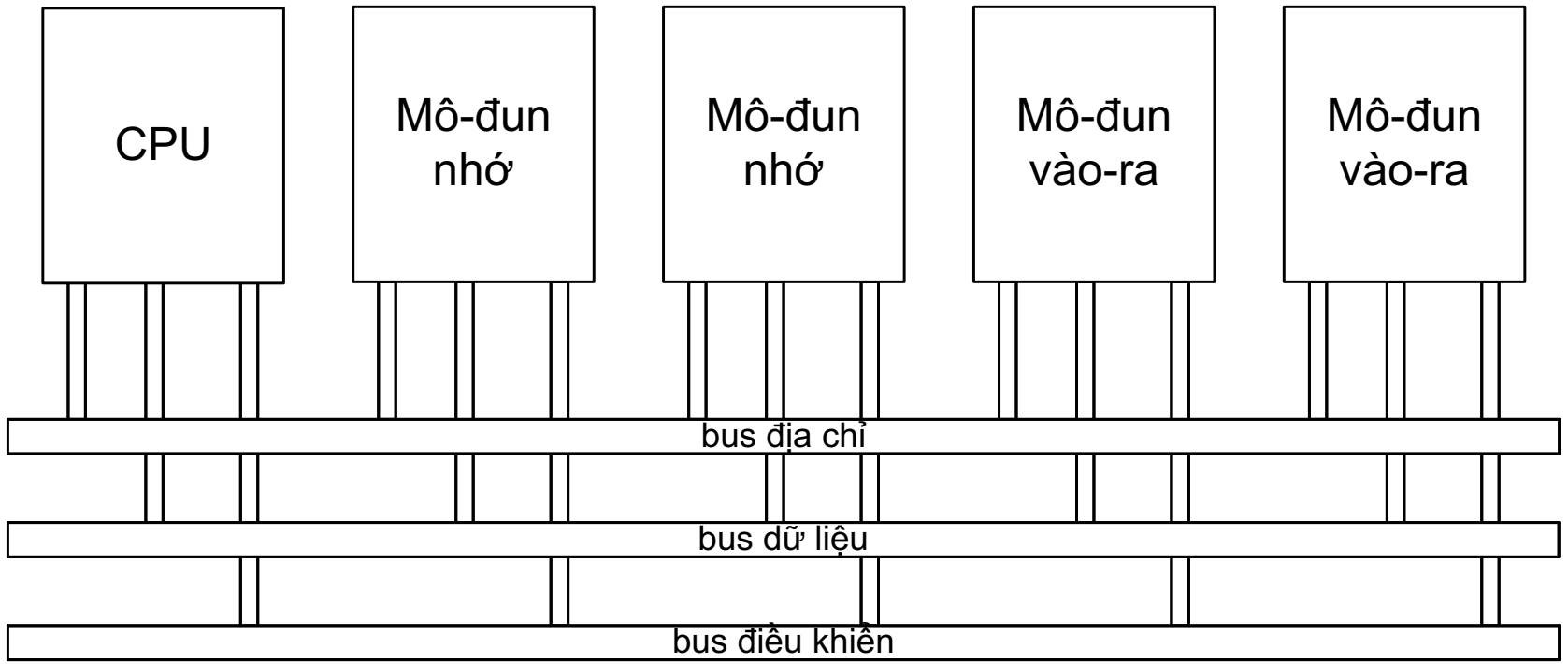
Kết nối CPU (tiếp)

- Phát địa chỉ đến các mô-đun nhớ hay các mô-đun vào-ra
- Đọc lệnh từ bộ nhớ
- Đọc dữ liệu từ bộ nhớ hoặc mô-đun vào-ra
- Đưa dữ liệu ra (sau khi xử lý) đến bộ nhớ hoặc mô-đun vào-ra
- Phát tín hiệu điều khiển đến các mô-đun nhớ và các mô-đun vào-ra
- Nhận các tín hiệu ngắt

2. Cấu trúc bus cơ bản

- Bus: tập hợp các đường kết nối để vận chuyển thông tin giữa các mô-đun của máy tính với nhau.
- Các bus chức năng:
  - Bus địa chỉ (Address bus)
  - Bus dữ liệu (Data bus)
  - Bus điều khiển (Control bus)
- Độ rộng bus: là số đường dây của bus có thể truyền các bit thông tin đồng thời (chỉ dùng cho bus địa chỉ và bus dữ liệu)

Sơ đồ cấu trúc bus cơ bản



Bus địa chỉ

- Chức năng: vận chuyển địa chỉ để xác định vị trí ngăn nhớ hay cổng vào-ra
- Độ rộng bus địa chỉ:
  - N bit: $A_{N-1}, A_{N-2}, \ldots A_2, A_1, A_0$

  → Số lượng địa chỉ tối đa được sử dụng là: 2^N địa chỉ (gọi là không gian địa chỉ)
  - Địa chỉ nhỏ nhất: $00 \ldots 000_{(2)}$
  - Địa chỉ lớn nhất: $11 \ldots 111_{(2)}$
- Ví dụ:
  - Máy tính sử dụng bus địa chỉ 32-bit (A_{31}-A_0), bộ nhớ chính được đánh địa chỉ cho từng byte

  → Có khả năng đánh địa chỉ cho 2^{32} bytes nhớ = 4GiB

Bus dữ liệu

- Chức năng:
  - vận chuyển lệnh từ bộ nhớ đến CPU
  - vận chuyển dữ liệu giữa các thành phần của máy tính với nhau
- Độ rộng bus dữ liệu: số bit được truyền đồng thời
  - M bit: $D_{M-1}, D_{M-2}, \ldots D_2, D_1, D_0$
  - M thường là 8, 16, 32, 64 bit
- Ví dụ:
  - Máy tính có bus dữ liệu kết nối CPU với bộ nhớ là 64-bit

  → Có thể trao đổi 8 byte nhớ ở một thời điểm

Bus điều khiển

- Chức năng: vận chuyển các tín hiệu điều khiển
- Các loại tín hiệu điều khiển:
  - Các tín hiệu điều khiển đọc/ghi
  - Các tín hiệu điều khiển ngắt
  - Các tín hiệu điều khiển bus

Một số tín hiệu điều khiển điển hình

- Các tín hiệu (phát ra từ CPU) điều khiển đọc/ghi:
  - *Memory Read* (MEMR): Tín hiệu điều khiển đọc dữ liệu từ một ngăn nhớ có địa chỉ xác định đưa lên bus dữ liệu.
  - *Memory Write* (MEMW): Tín hiệu điều khiển ghi dữ liệu có sẵn trên bus dữ liệu đến một ngăn nhớ có địa chỉ xác định.
  - *I/O Read* (IOR): Tín hiệu điều khiển đọc dữ liệu từ một cổng vào-ra có địa chỉ xác định đưa lên bus dữ liệu.
  - *I/O Write* (IOW): Tín hiệu điều khiển ghi dữ liệu có sẵn trên bus dữ liệu ra một cổng có địa chỉ xác định.

Một số tín hiệu điều khiển điển hình (tiếp)

- Các tín hiệu điều khiển ngắt:
  - *Interrupt Request* (INTR): Tín hiệu từ bộ điều khiển vào-ra gửi đến yêu cầu ngắt CPU để trao đổi vào-ra. Tín hiệu INTR có thể bị che.
  - *Interrupt Acknowledge* (INTA): Tín hiệu phát ra từ CPU báo cho bộ điều khiển vào-ra biết CPU chấp nhận ngắt để trao đổi vào-ra.
  - *Non Maskable Interrupt* (NMI): tín hiệu ngắt không che được gửi đến ngắt CPU.
  - *Reset*: Tín hiệu từ bên ngoài gửi đến CPU và các thành phần khác để khởi động lại máy tính.

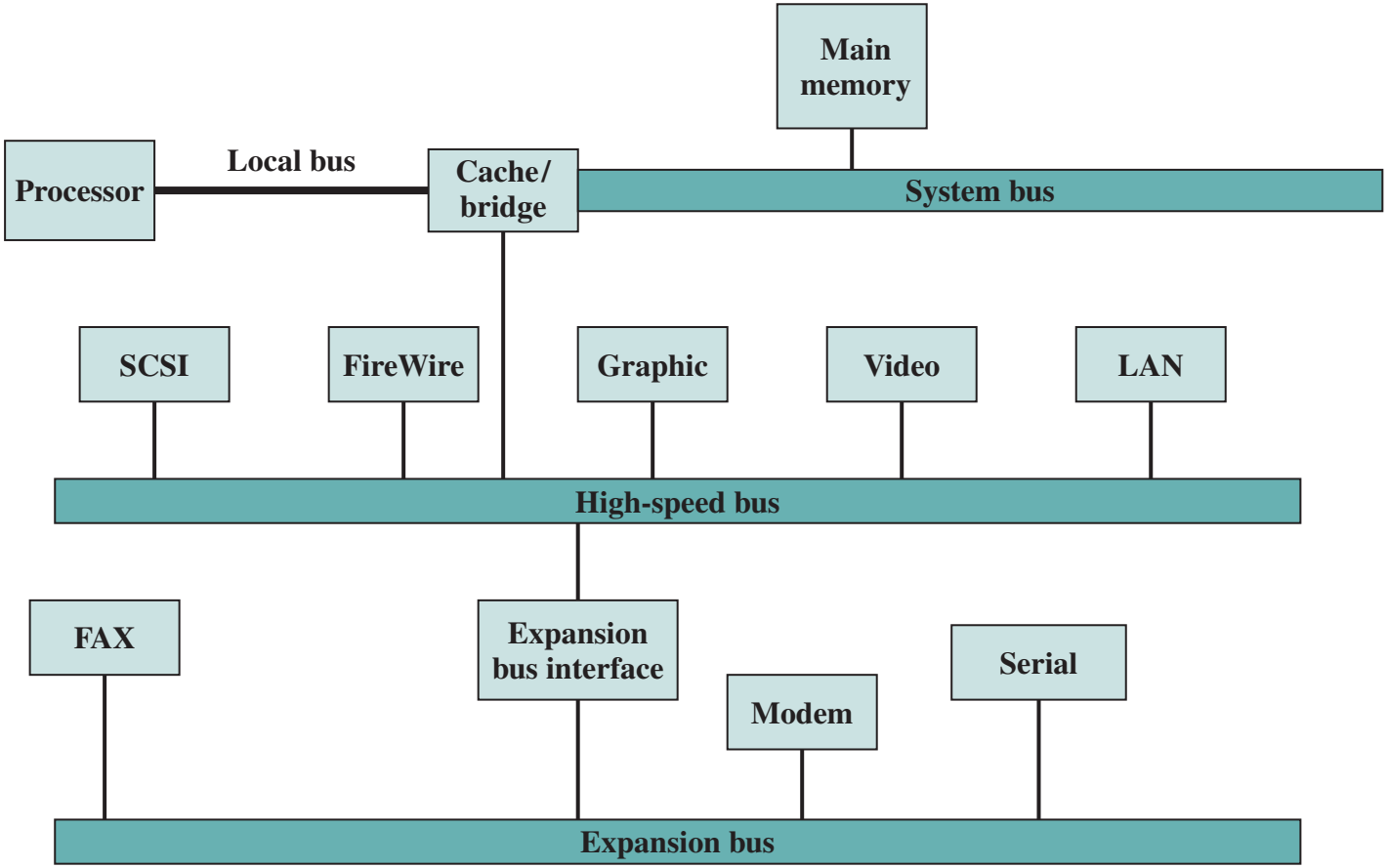
Một số tín hiệu điều khiển điển hình (tiếp)

- Các tín hiệu điều khiển bus:
  - *Bus Request* (BRQ) : Tín hiệu từ mô-đun vào-ra gửi đến yêu cầu CPU chuyển nhượng quyền sử dụng bus.
  - *Bus Grant* (BGT): Tín hiệu phát ra từ CPU chấp nhận chuyển nhượng quyền sử dụng bus cho mô-đun vào-ra.
  - *Lock/ Unlock*: Tín hiệu *cấm/cho-phép* xin chuyển nhượng bus.

3. Phân cấp bus

- Đơn bus: Tất cả các mô-đun kết nối vào bus chung
  - Bus chỉ phục vụ được một yêu cầu trao đổi dữ liệu tại một thời điểm → độ trễ lớn
  - Bus phải có tốc độ bằng tốc độ bus của mô-đun nhanh nhất trong hệ thống
- Đa bus: Phân cấp thành nhiều bus cho các mô-đun khác nhau và có tốc độ khác nhau
  - Bus của bộ xử lý
  - Bus của RAM
  - Các bus vào-ra

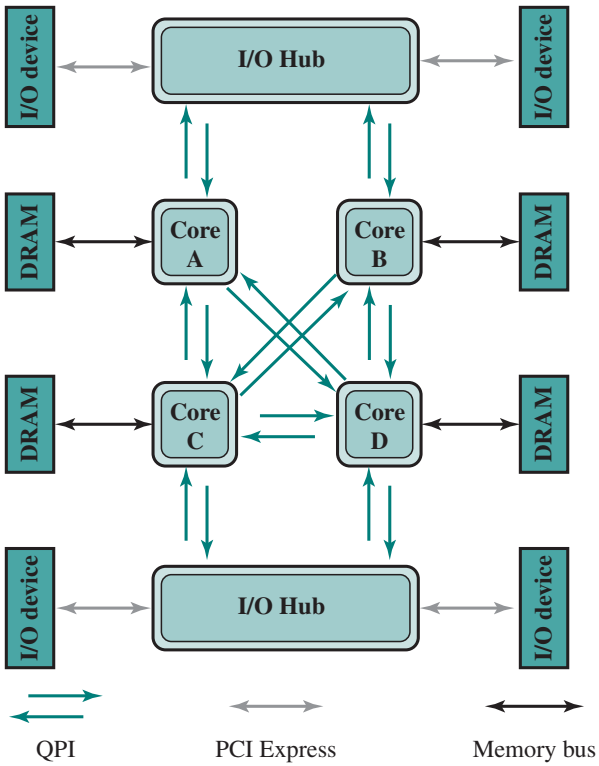
Phân cấp bus


Main memory
Processor
Local bus
Cache/ bridge
System bus
SCSI
FireWire
Graphic
Video
LAN
High-speed bus
FAX
Expansion bus interface
Modem
Serial
Expansion bus

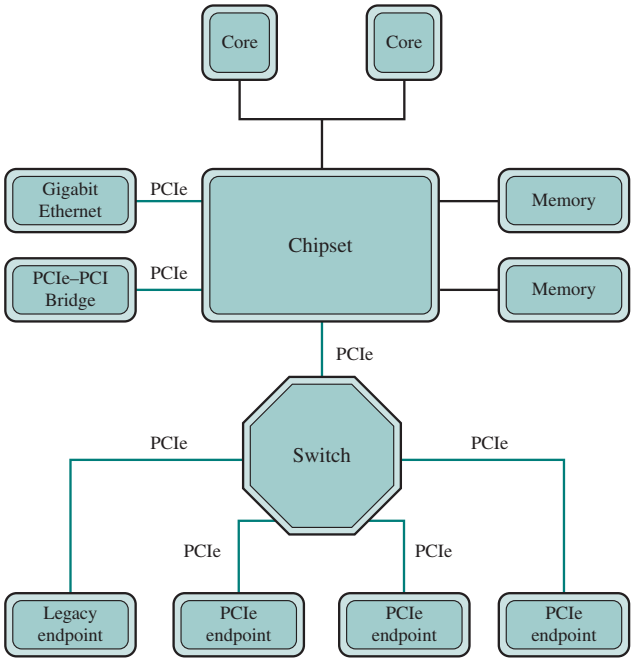
4. Kết nối điểm-điểm

- Point-to-point connection
- Khắc phục nhược điểm của bus dùng chung (shared bus)

Kết nối QPI



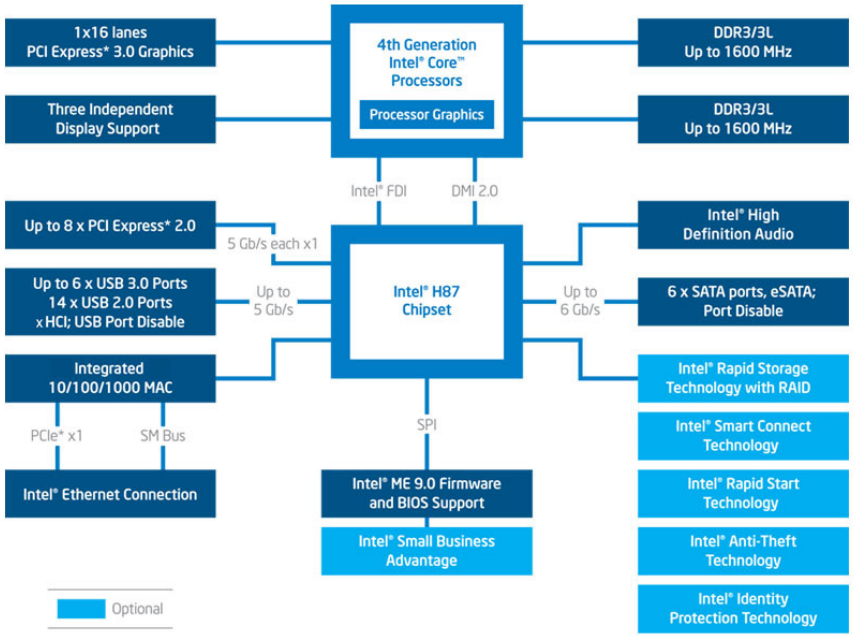
Kết nối PCIe

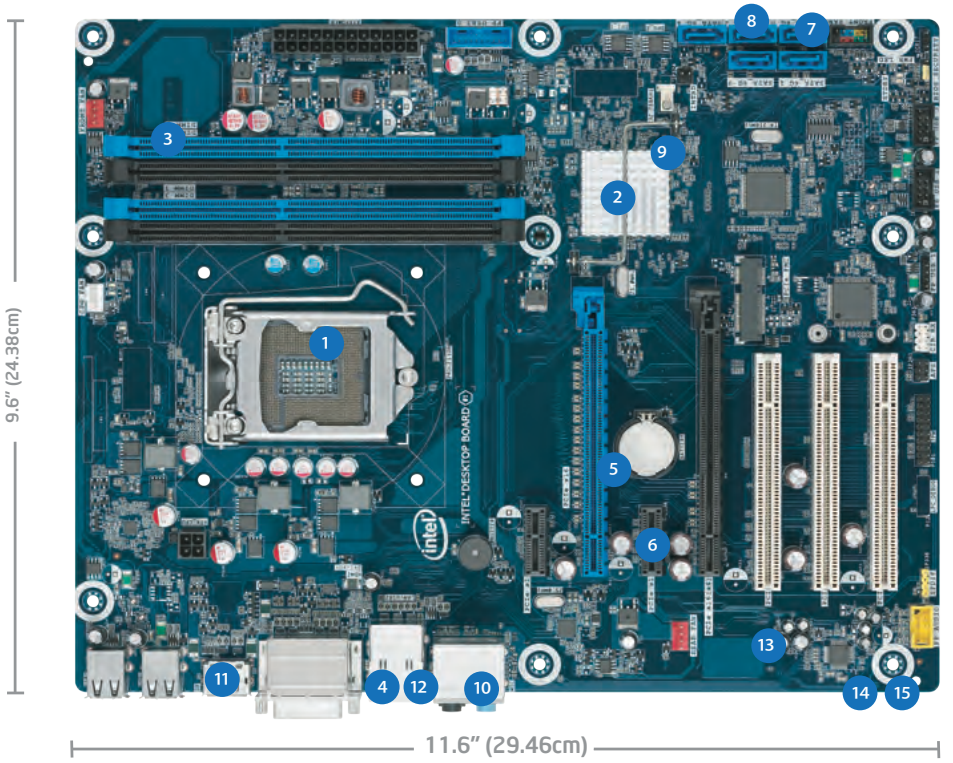


Một số bus điển hình trong máy tính

- QPI (Quick Path Interconnect)
- PCI bus (Peripheral Component Interconnect): bus vào-ra đa năng
- PCIe: (PCI express) kết nối điểm-điểm đa năng tốc độ cao
- SATA (Serial Advanced Technology Attachment): Bus kết nối với ổ đĩa cứng hoặc ổ đĩa CD/DVD
- USB (Universal Serial Bus): Bus nối tiếp đa năng

Ví dụ bus trong máy tính Intel





HDMI®
HIGH-DEFINITION MULTIMEDIA INTERFACE

Hết chương 3

Chương 4
SỐ HỌC MÁY TÍNH

Nguyễn Kim Khánh
Trường Đại học Bách khoa Hà Nội

Nội dung học phần

Chương 1. Giới thiệu chung

Chương 2. Cơ bản về logic số

Chương 3. Hệ thống máy tính

Chương 4. Số học máy tính

Chương 5. Kiến trúc tập lệnh

Chương 6. Bộ xử lý

Chương 7. Bộ nhớ máy tính

Chương 8. Hệ thống vào-ra

Chương 9. Các kiến trúc song song

Nội dung chương 4

4.1. Biểu diễn số nguyên

4.2. Phép cộng và phép trừ số nguyên

4.3. Phép nhân và phép chia số nguyên

4.4. Số dấu phẩy động

4.1. Biểu diễn số nguyên

- Số nguyên không dấu (Unsigned Integer)
- Số nguyên có dấu (Signed Integer)

1. Biểu diễn số nguyên không dấu

- Nguyên tắc tổng quát: Dùng n bit biểu diễn số nguyên không dấu A:

$$a_{n-1}a_{n-2}\ldots a_2a_1a_0$$

Giá trị của A được tính như sau:

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

Dải biểu diễn của A: $[0, 2^n - 1]$

Ví dụ 1

- Biểu diễn các số nguyên không dấu sau đây bằng 8-bit:

A = 41 ; B = 150

Giải:

A = 41 = 32 + 8 + 1 = $2^5 + 2^3 + 2^0$

41 = 0010 1001

B = 150 = 128 + 16 + 4 + 2 = $2^7 + 2^4 + 2^2 + 2^1$

150 = 1001 0110

Ví dụ 2

- Cho các số nguyên không dấu M, N được biểu diễn bằng 8-bit như sau:
  - M = 0001 0010
  - N = 1011 1001

  Xác định giá trị của chúng ?

Giải:

- M = 0001 0010 = $2^4 + 2^1 = 16 + 2 = 18$
- N = 1011 1001 = $2^7 + 2^5 + 2^4 + 2^3 + 2^0$

  $= 128 + 32 + 16 + 8 + 1 = 185$

Với n = 8 bit

Biểu diễn được các giá trị từ 0 đến 255 ($2^8 - 1$)

Chú ý:

1111 1111

\+ 0000 0001

1 0000 0000

có *nhớ ra ngoài*

(Carry out)

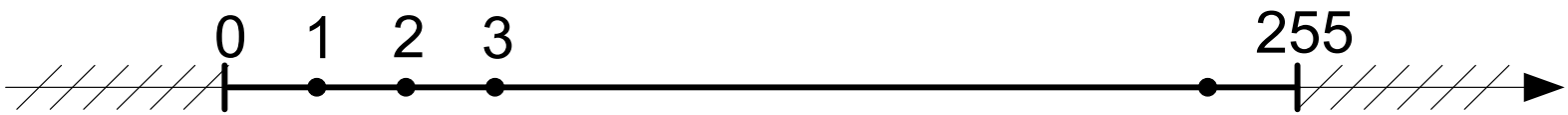
255 + 1 = 0 ???

do vượt ra khỏi dải biểu diễn

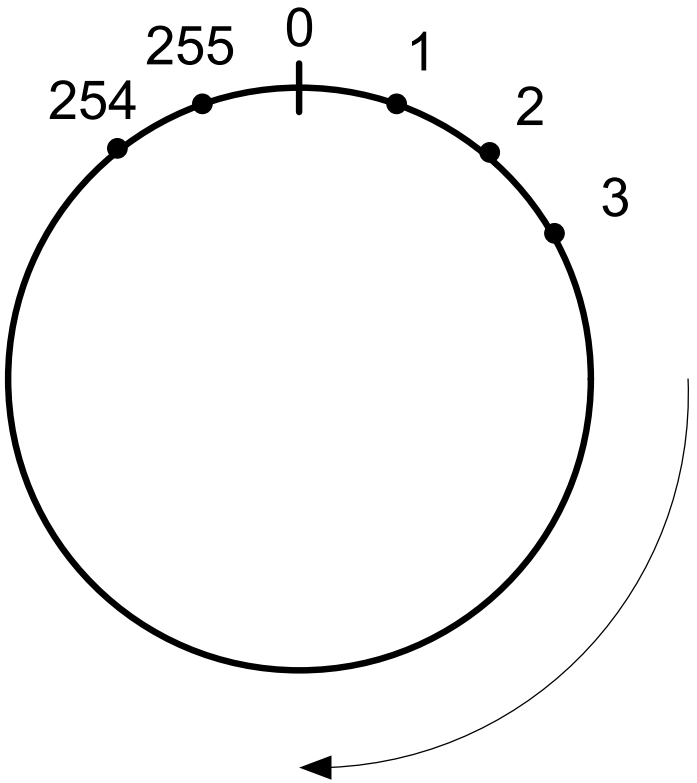
Biểu diễn nhị phân	Giá trị thập phân
0000 0000	0
0000 0001	1
0000 0010	2
0000 0011	3
0000 0100	4
...	
1111 1110	254
1111 1111	255

Trục số học với n = 8 bit

Trục số học:



Trục số học máy tính:



Với n = 16 bit, 32 bit, 64 bit

- n= 16 bit: dải biểu diễn từ 0 đến 65535 ($2^{16} - 1$)
  - 0000 0000 0000 0000 = 0
  - ...
  - 0000 0000 1111 1111 = 255
  - 0000 0001 0000 0000 = 256
  - ...
  - 1111 1111 1111 1111 = 65535
- n= 32 bit: dải biểu diễn từ 0 đến $2^{32} - 1$
- n= 64 bit: dải biểu diễn từ 0 đến $2^{64} - 1$

2. Biểu diễn số nguyên có dấu

Số bù một và Số bù hai

- Định nghĩa: Cho một số nhị phân A được biểu diễn bằng n bit, ta có:
  - Số bù một của A = $(2^n-1) - A$
  - Số bù hai của A = $2^n - A$
- Số bù hai của A = (Số bù một của A) +1

Ví dụ

Với n = 8 bit, cho A = 0010 0101

- Số bù một của A được tính như sau:

  1111 1111 $(2^8 - 1)$

  \- 0010 0101 (A)

  1101 1010

  → đảo các bit của A

- Số bù hai của A được tính như sau:

  1 0000 0000 (2^8)

  \- 0010 0101 (A)

  1101 1011

  → thực hiện khó khăn

Quy tắc tìm Số bù một và Số bù hai

- Số bù một của A = đảo giá trị các bit của A
- (Số bù hai của A) = (Số bù một của A) + 1
- Ví dụ:
  - Cho A = 0010 0101
  - Số bù một của A = 1101 1010
  - \+ 1
  - Số bù hai của A = 1101 1011
- Nhận xét:

A = 0010 0101

Số bù hai của A = + 1101 1011

1 0000 0000 = 0

(bỏ qua bit nhớ ra ngoài)

→ Số bù hai của A = -A

Biểu diễn số nguyên có dấu theo mã bù hai

Nguyên tắc tổng quát: Dùng n bit biểu diễn số nguyên có dấu A:

$$a_{n-1}a_{n-2}\ldots a_2a_1a_0$$

- Với A là số dương: bit $a_{n-1} = 0$, các bit còn lại biểu diễn độ lớn như số không dấu
- Với A là số âm: được biểu diễn bởi số bù hai của số dương tương ứng, vì vậy bit $a_{n-1} = 1$

Ví dụ

- Biểu diễn các số nguyên có dấu sau đây bằng 8-bit:

A = + 58 ; B = - 80

Giải:

A = + 58 = 0011 1010

B = - 80

Ta có: + 80 = 0101 0000

Số bù một = 1010 1111

\+ 1

Số bù hai = 1011 0000

Vậy: B = - 80 = 1011 0000

Xác định giá trị của số dương

- Dạng tổng quát của số dương:

$$0a_{n-2} \ldots a_2 a_1 a_0$$

- Giá trị của số dương:

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

- Dải biểu diễn cho số dương: [0, +(2^{n-1} - 1)]

Xác định giá trị của số âm

- Dạng tổng quát của số âm:

$$1a_{n-2}\ldots a_2 a_1 a_0$$

- Giá trị của số âm:

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

Chứng minh?

- Dải biểu diễn cho số âm: $[-2^{n-1}, -1]$

Công thức xác định giá trị số âm

$$A = 1\, a_{n-2}\, a_{n-3}\, \ldots\, a_2\, a_1\, a_0$$

$$-A = 0\, \overline{a_{n-2}}\, \overline{a_{n-3}} \ldots \overline{a_2}\, \overline{a_1}\, \overline{a_0} + 1$$

$$= 11 \ldots 111 - a_{n-2}\, a_{n-3}\, \ldots\, a_2\, a_1\, a_0 + 1$$

$$= (2^{n-1} - 1) - \left(\sum_{i=0}^{n-2} a_i 2^i \right) + 1$$

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

Công thức tổng quát cho số nguyên có dấu

- Dạng tổng quát của số nguyên có dấu A:

$$a_{n-1}a_{n-2}\ldots a_2a_1a_0$$

- Giá trị của A được xác định như sau:

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

- Dải biểu diễn: $[-(2^{n-1}), +(2^{n-1}-1)]$

Ví dụ

- Hãy xác định giá trị của các số nguyên có dấu được biểu diễn theo mã bù hai với 8-bit như dưới đây:
  - P = 0110 0010
  - Q = 1101 1011

Giải:

- P = 0110 0010 = $2^6 + 2^5 + 2^1 = 64 + 32 + 2 = +98$
- Q = 1101 1011 = $-2^7 + 2^6 + 2^4 + 2^3 + 2^1 + 2^0$
  $= -128 + 64 + 16 + 8 + 2 + 1 = -37$

Với n = 8 bit

- Biểu diễn được các giá trị từ -2^7 đến $+2^7-1$
  - -128 đến +127
  - Chỉ có một giá trị 0
  - Không biểu diễn cho giá trị +128

Chú ý:

+127 + 1 = -128

(-128)+(-1) = +127

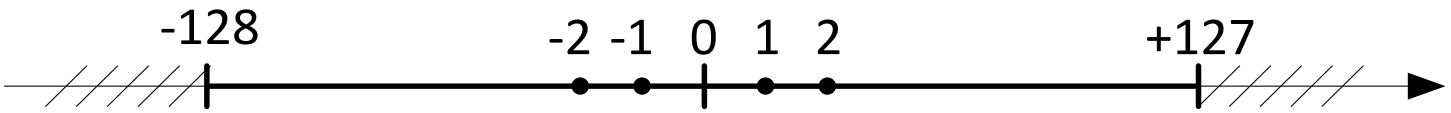
có tràn xảy ra (Overflow)

(do vượt ra khỏi dải biểu diễn)

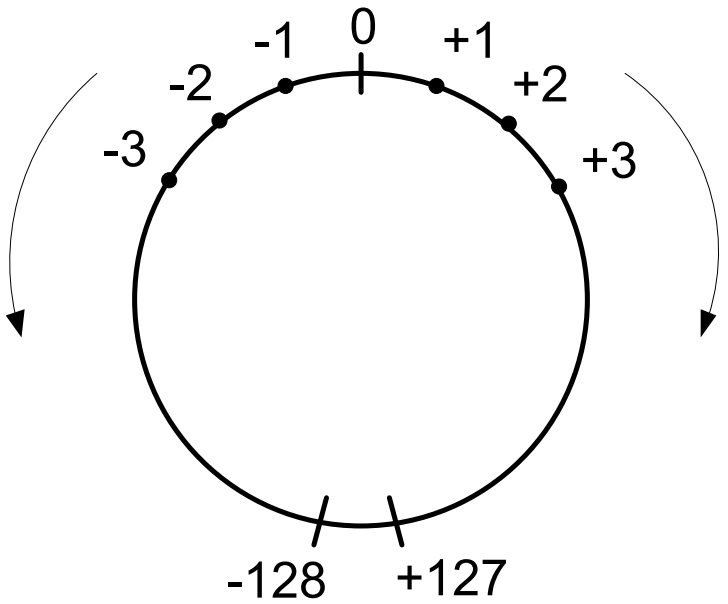
Giá trị thập phân	Biểu diễn bù hai
0	0000 0000
+1	0000 0001
+2	0000 0010
	...
+126	0111 1110
+127	0111 1111
-128	1000 0000
-127	1000 0001
	...
-2	1111 1110
-1	1111 1111

Trục số học số nguyên có dấu với n = 8 bit

- Trục số học:



- Trục số học máy tính:



Với n = 16 bit, 32 bit, 64 bit

- Với n = 16bit: biểu diễn từ -2^{15} đến $2^{15}-1$
  - 0000 0000 0000 0000 = 0
  - 0000 0000 0000 0001 = +1
  - ...
  - 0111 1111 1111 1111 = +32767 ($2^{15}-1$)
  - 1000 0000 0000 0000 = -32768 (-2^{15})
  - 1000 0000 0000 0001 = -32767
  - ...
  - 1111 1111 1111 1111 = -1
- Với n = 32bit: biểu diễn từ -2^{31} đến $2^{31}-1$
- Với n = 64bit: biểu diễn từ -2^{63} đến $2^{63}-1$

Mở rộng bit cho số nguyên

- Mở rộng theo số không dấu (Zero-extended): thêm các bit 0 vào bên trái
- Mở rộng theo số có dấu (Sign-extended):
  - Số dương:

    +19 = 0001 0011 (8bit)

    +19 = 0000 0000 0001 0011 (16bit)

    → thêm các bit 0 vào bên trái
  - Số âm:

    - 19 = 1110 1101 (8bit)

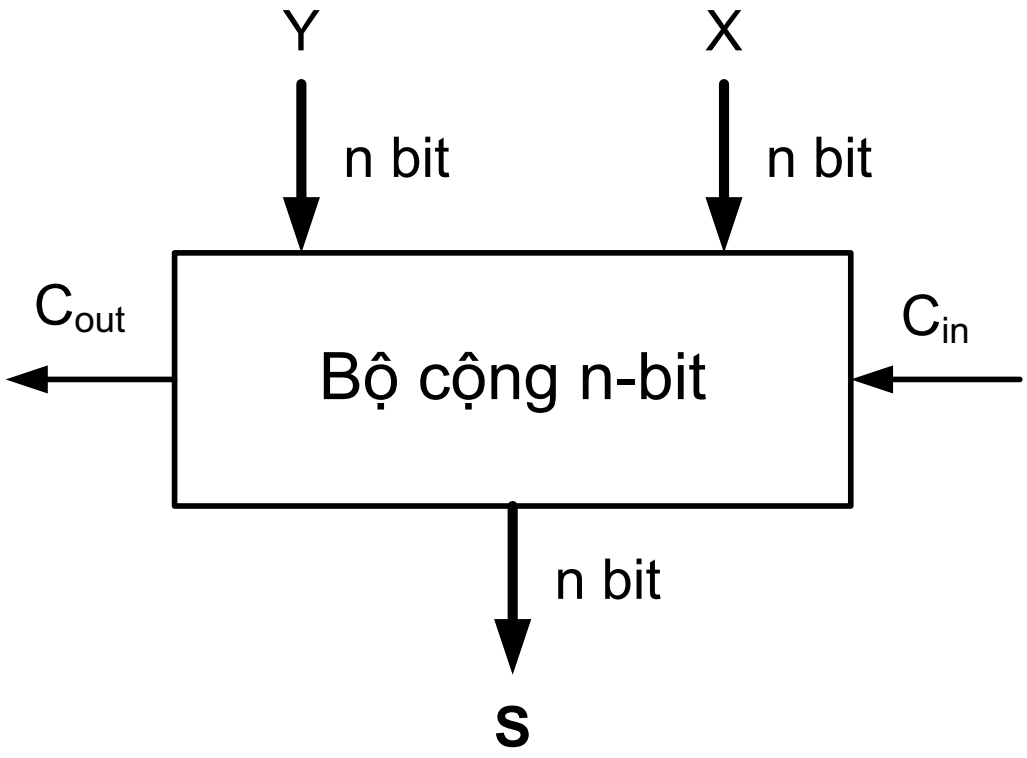
    - 19 = 1111 1111 1110 1101 (16bit)

    → thêm các bit 1 vào bên trái

4.2. Thực hiện phép cộng/trừ với số nguyên

1. Phép cộng số nguyên không dấu

Bộ cộng n-bit



Nguyên tắc cộng số nguyên không dấu

- Khi cộng hai số nguyên không dấu n-bit, kết quả nhận được là n-bit:
  - Nếu $C_{out} = 0$ → nhận được kết quả đúng
  - Nếu $C_{out} = 1$ → nhận được kết quả sai, do có *nhớ ra ngoài (Carry Out)*
- Hiện tượng *nhớ ra ngoài* xảy ra khi: tổng > $(2^n - 1)$

Ví dụ cộng số nguyên không dấu

- 57 = 0011 1001
  + 34 = + 0010 0010
  91 0101 1011 = 64+16+8+2+1=91 → đúng

- 209 = 1101 0001
  + 73 = + 0100 1001
  282 1 0001 1010
  kết quả = 0001 1010 =16+8+2=26 → sai
  do có *nhớ ra ngoài* (C_{out}=1)

Để có kết quả đúng, ta thực hiện cộng theo 16-bit:

209 = 0000 0000 1101 0001
\+ 73 = + 0000 0000 0100 1001
0000 0001 0001 1010 = 256+16+8+2 = 282

2. Phép đảo dấu

- Ta có:

+ 37	=	0010 0101		
bù một	=	1101 1010		
		+ 1		
bù hai	=	1101 1011	=	-37

- Lấy bù hai của số âm:

- 37	=	1101 1011		
bù một	=	0010 0100		
		+ 1		
bù hai	=	0010 0101	=	+37

- Kết luận: *Phép đảo dấu số nguyên trong máy tính thực chất là lấy bù hai*

3. Cộng số nguyên có dấu

- Khi cộng hai số nguyên có dấu n-bit, kết quả nhận được là n-bit và *không cần quan tâm đến bit C_{out}*
  - Khi cộng hai số khác dấu thì kết quả luôn luôn đúng
  - Khi cộng hai số cùng dấu, nếu dấu kết quả cùng dấu với các số hạng thì kết quả là đúng
  - Khi cộng hai số cùng dấu, nếu kết quả có dấu ngược lại, khi đó có ***tràn*** (Overflow) xảy ra và kết quả bị sai
- Hiện tượng *tràn* xảy ra khi tổng nằm ngoài dải biểu diễn: $[-(2^{n-1}),+(2^{n-1}-1)]$

Ví dụ cộng số nguyên có dấu không tràn

- (+ 70) = 0100 0110
  + (+ 42) = 0010 1010
  + 112 0111 0000 = +112

- (+ 97) = 0110 0001
  + (- 52) = 1100 1100 (+52=0011 0100)
  + 45 1 0010 1101 = +45

- (- 90) = 1010 0110 (+90=0101 1010)
  + (+36) = 0010 0100
  - 54 1100 1010 = - 54

- (- 74) = 1011 0110 (+74=0100 1010)
  +(- 30) = 1110 0010 (+30=0001 1110)
  -104 1 1001 1000 = -104

Ví dụ cộng số nguyên có dấu bị tràn

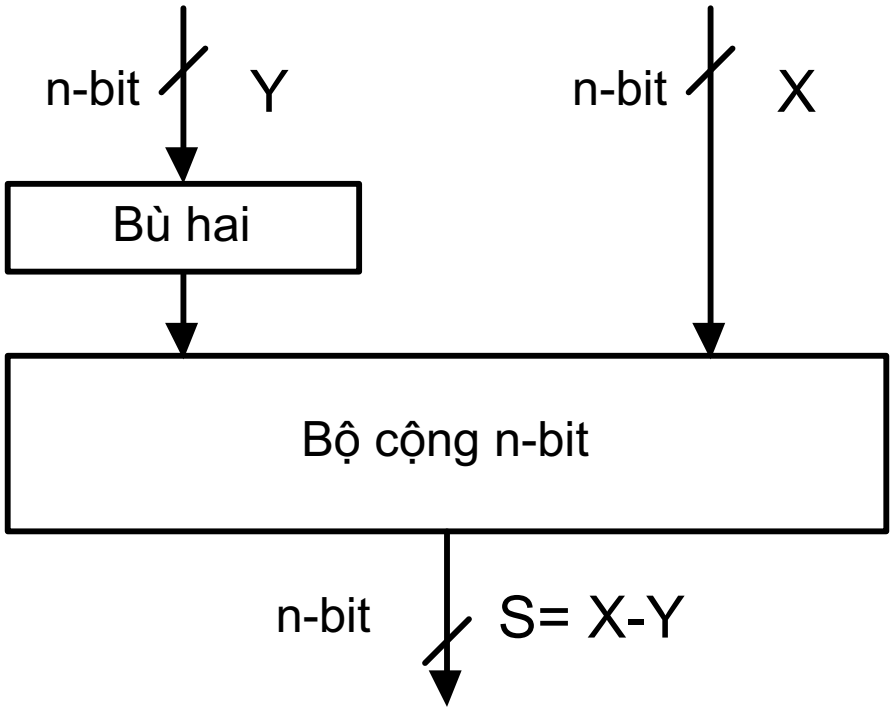
- (+ 75) = 0100 1011
  +(+ 82) = 0101 0010
  +157 1001 1101
  = - 128+16+8+4+1= -99 → sai

- (- 104) = 1001 1000 (+104=0110 1000)
  + (- 43) = 1101 0101 (+ 43 =0010 1011)
  - 147 1 0110 1101
  = 64+32+8+4+1= +109 → sai

- Cả hai ví dụ đều tràn vì tổng nằm ngoài dải biểu diễn [-128, +127]

4. Nguyên tắc thực hiện phép trừ

- Phép trừ hai số nguyên: X-Y = X+(-Y)
- Nguyên tắc: Lấy bù hai của Y để được –Y, rồi cộng với X



4.3. Phép nhân và phép chia số nguyên

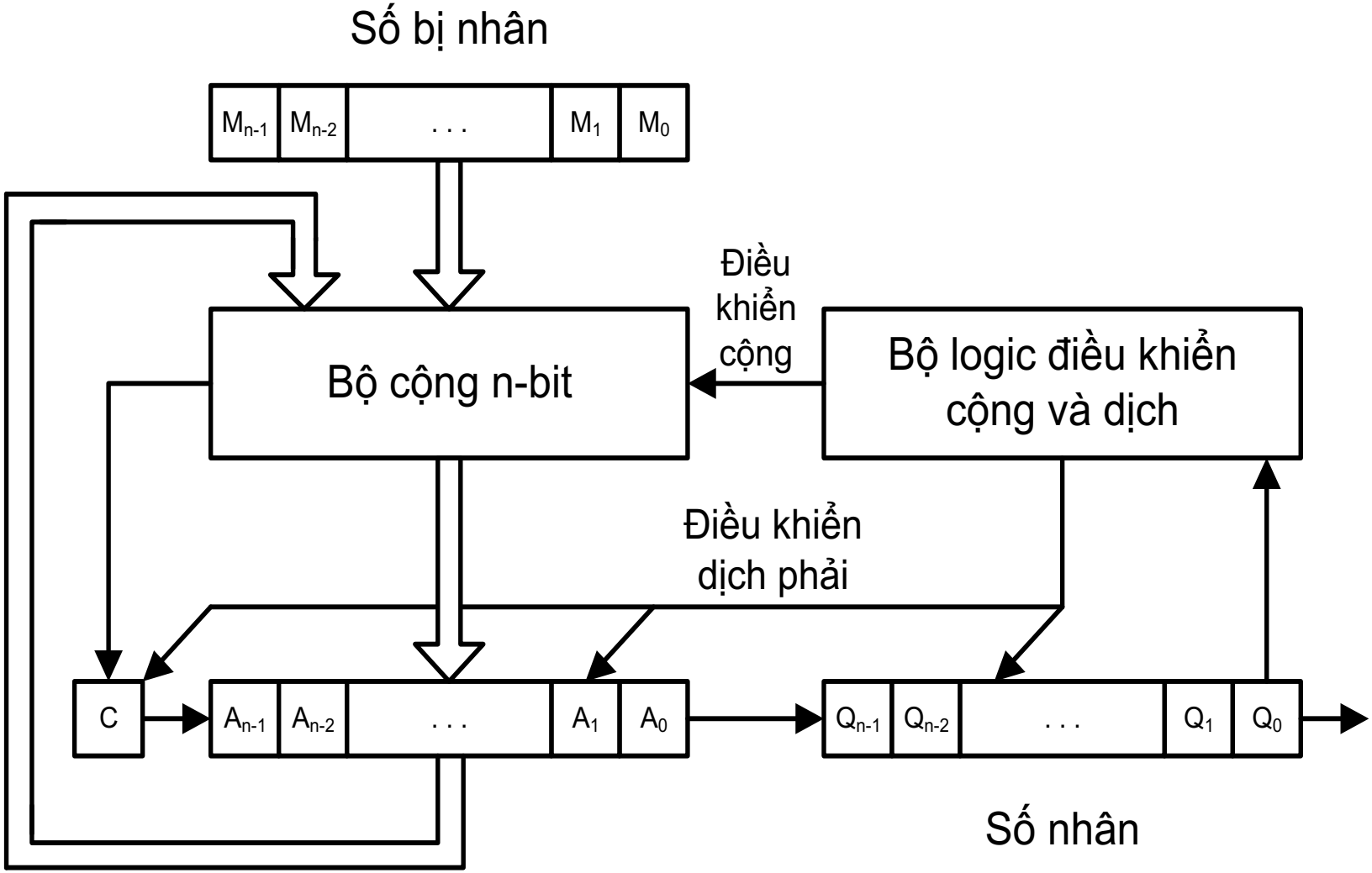
1. Nhân số nguyên không dấu

	1011	Số bị nhân (11)
x	1101	Số nhân (13)
	1011	Các tích riêng phần
	0000	
	1011	
	1011	
	10001111	Tích (143)

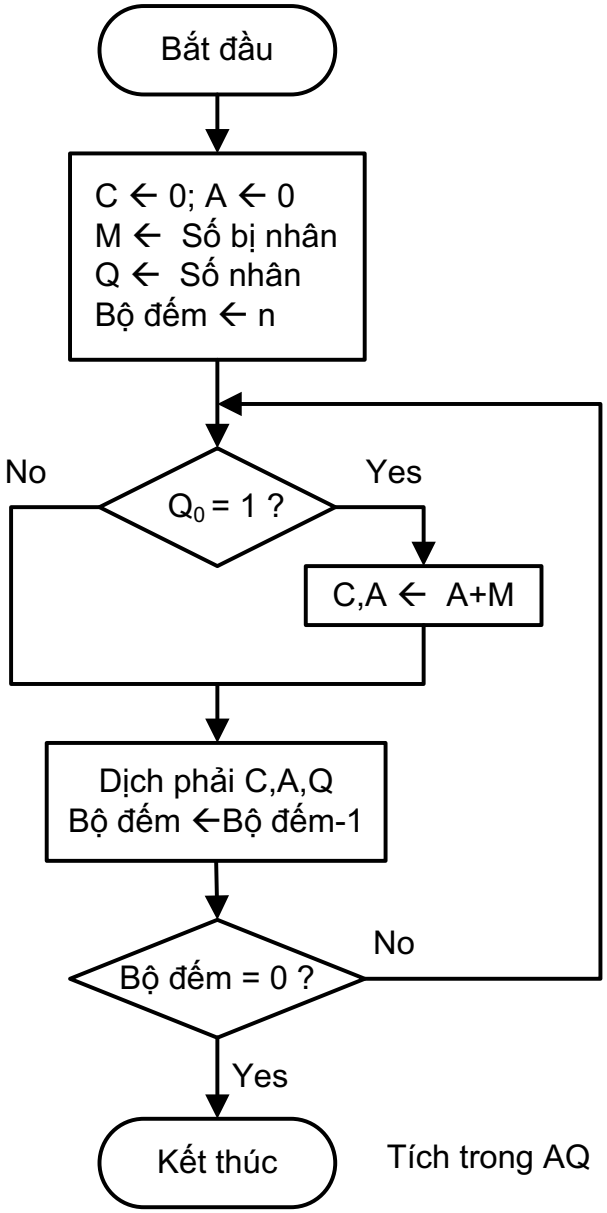
Nhân số nguyên không dấu (tiếp)

- Các tích riêng phần được xác định như sau:
  - Nếu bit của số nhân bằng 0 → tích riêng phần bằng 0
  - Nếu bit của số nhân bằng 1 → tích riêng phần bằng số bị nhân
  - Tích riêng phần tiếp theo được dịch trái một bit so với tích riêng phần trước đó
- Tích bằng tổng các tích riêng phần
- Nhân hai số nguyên n-bit, tích có độ dài 2n bit (không bao giờ tràn)

Bộ nhân số nguyên không dấu



Lưu đồ nhân số nguyên không dấu



Ví dụ nhân số nguyên không dấu

- Số bị nhân M = 1011 (11)
- Số nhân Q = 1101 (13)
- Tích = 1000 1111 (143)

	C	A	Q	
■	0	0000	1101	Các giá trị khởi đầu
		+ 1011		
	0	1011	1101	A ← A + M
■	0	0101	1110	Dịch phải
■	0	0010	1111	Dịch phải
		+ 1011		
	0	1101	1111	A ← A + M
■	0	0110	1111	Dịch phải
		+ 1011		
	1	0001	1111	A ← A + M
■	0	1000	1111	Dịch phải

Ví dụ nhân số nguyên không dấu (tiếp)

- Số bị nhân M = 0110 (6)
- Số nhân Q = 0101 (5)
- Tích = (30)

	C	A	Q	
■	0	0000	0101	Các giá trị khởi đầu
		+ 0110		
	0	0110	0101	A ← A + M
■	0	0011	0010	Dịch phải
■	0	0001	1001	Dịch phải
		+ 0110		
	0	0111	1001	A ← A + M
■	0	0011	1100	Dịch phải
■	0	0001	1110	Dịch phải

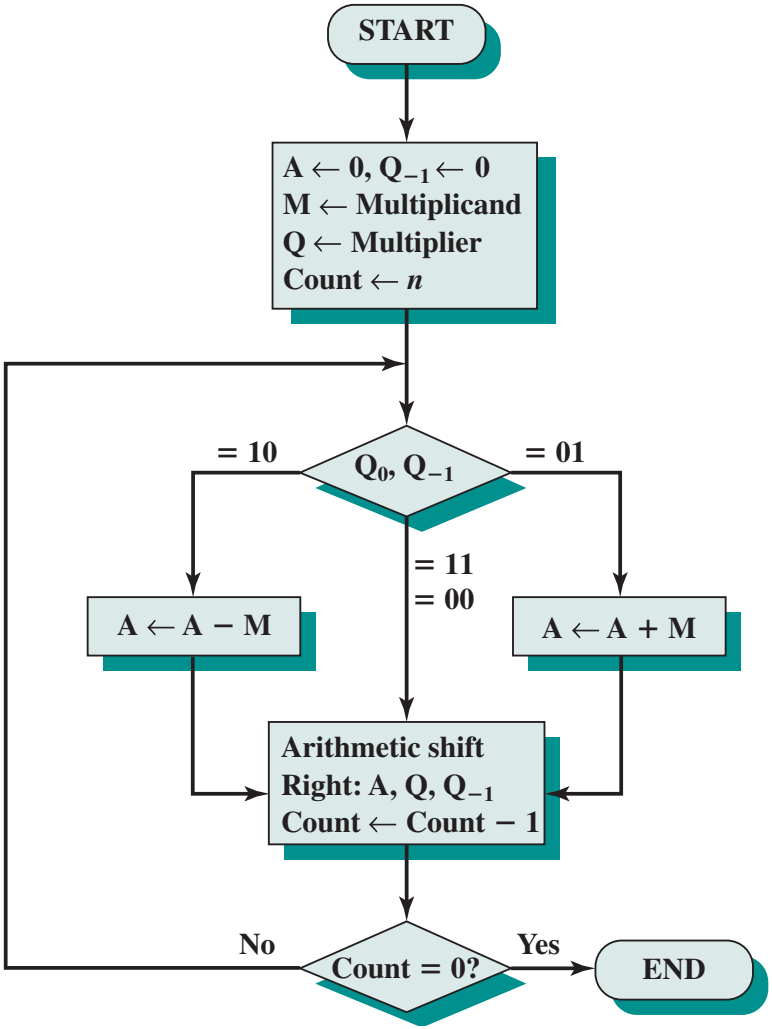
2. Nhân số nguyên có dấu

- Sử dụng thuật giải nhân không dấu
- Sử dụng thuật giải Booth

Sử dụng thuật giải nhân không dấu

- Bước 1. Chuyển đổi số bị nhân và số nhân thành số dương tương ứng
- Bước 2. Nhân hai số dương bằng thuật giải nhân số nguyên không dấu, được tích của hai số dương.
- Bước 3. Hiệu chỉnh dấu của tích:
  - Nếu hai thừa số ban đầu cùng dấu thì giữ nguyên kết quả ở bước 2
  - Nếu hai thừa số ban đầu là khác dấu thì đảo dấu kết quả của bước 2 (lấy bù hai)

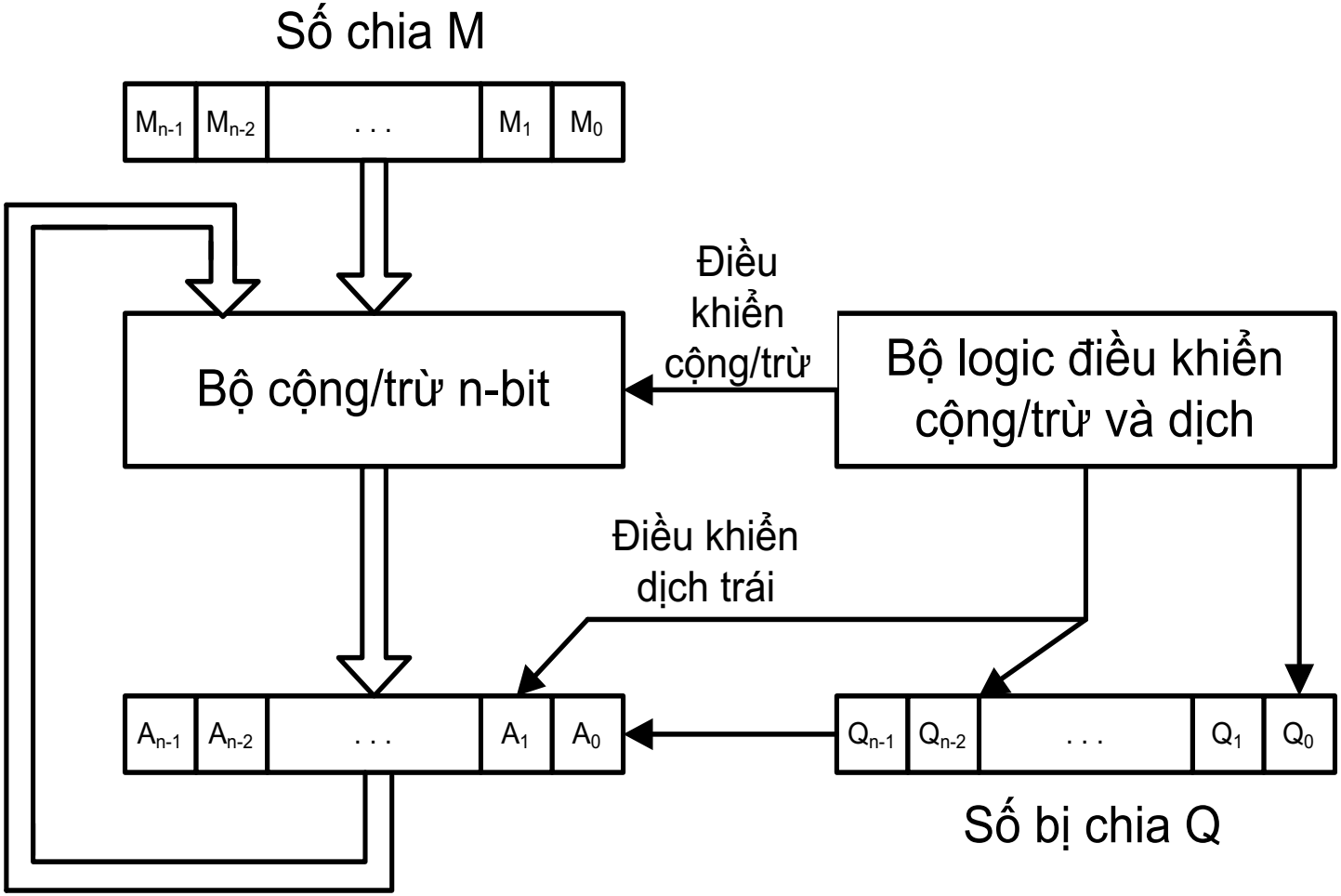
Thuật giải Booth (tham khảo sách COA)



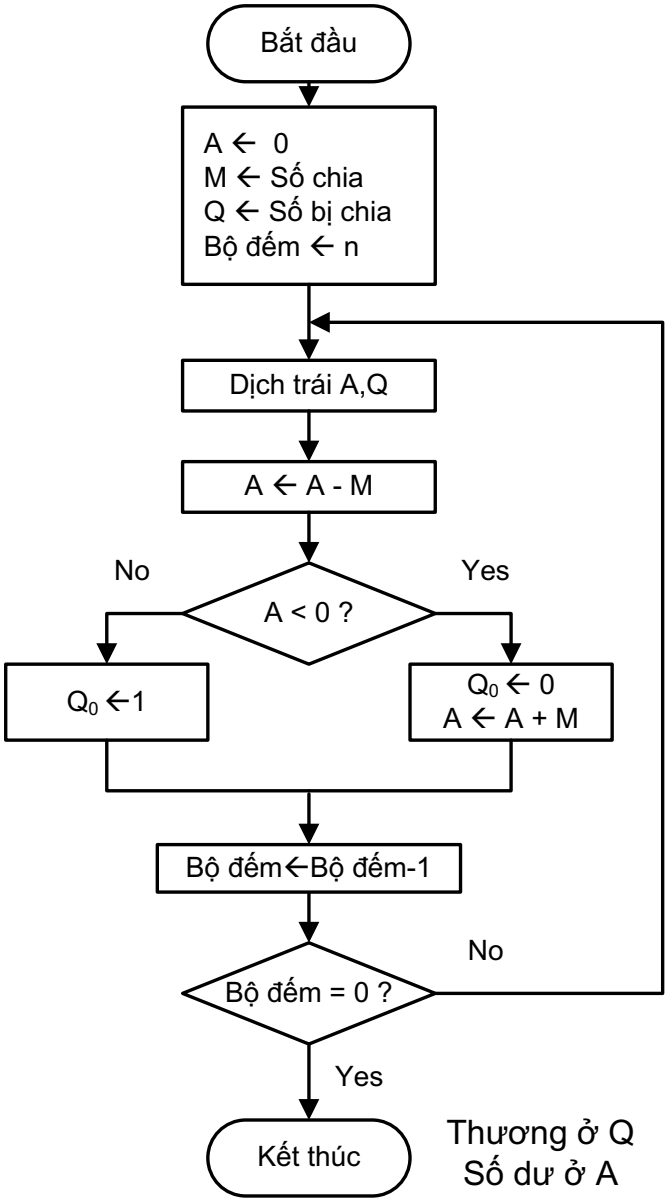
3. Chia số nguyên không dấu

Số bị chia	10010011	\1011	Số chia
	- 1011	00001101	Thương
	001110		
	- 1011		
	001111		
	- 1011		
	100		Phần dư

Bộ chia số nguyên không dấu



Lưu đồ chia số nguyên không dấu



Ví dụ: Q = 1011 (11) M = 0011 (3) → -M = 1101

A	Q		
0000	1011		BĐ = 4
0001	0110	dịch trái	
1101			
1110		A = A – M < 0	
0011			
0001	0110	A = A + M	BĐ = 3
0010	1100	dịch trái	
1101			
1111		A = A – M < 0	
0011			
0010	1100	A = A + M	BĐ = 2
0101	1000	dịch trái	
1101			
0010		A = A – M > 0	
0010	1001		BĐ = 1
0101	0010	dịch trái	
1101			
0010		A = A – M > 0	
0010	0011		BĐ = 0
2	3		

4. Chia số nguyên có dấu

- Bước 1. Chuyển đổi số bị chia và số chia về thành số dương tương ứng.
- Bước 2. Sử dụng thuật giải chia số nguyên không dấu để chia hai số dương, kết quả nhận được là thương Q và phần dư R đều là dương
- Bước 3. Hiệu chỉnh dấu của kết quả như sau:

  (Lưu ý: phép đảo dấu thực chất là thực hiện phép lấy bù hai)

Số bị chia	Số chia	Thương	Số dư
dương	dương	giữ nguyên	giữ nguyên
dương	âm	đảo dấu	giữ nguyên
âm	dương	đảo dấu	đảo dấu
âm	âm	giữ nguyên	đảo dấu

4.4. Số dấu phẩy động

1. Nguyên tắc chung

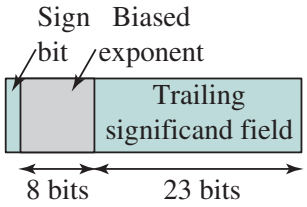
- Floating Point Number → biểu diễn cho số thực
- Tổng quát: một số thực X được biểu diễn theo kiểu số dấu phẩy động như sau:

$$X = \pm M * R^E$$

  - M là phần định trị (Mantissa),
  - R là cơ số (Radix),
  - E là phần mũ (Exponent).

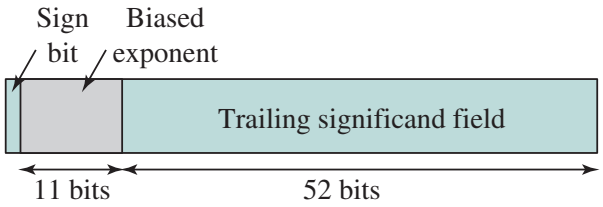
2. Chuẩn IEEE754-2008

- Cơ số R = 2
- Các dạng:
  - Dạng 32-bit



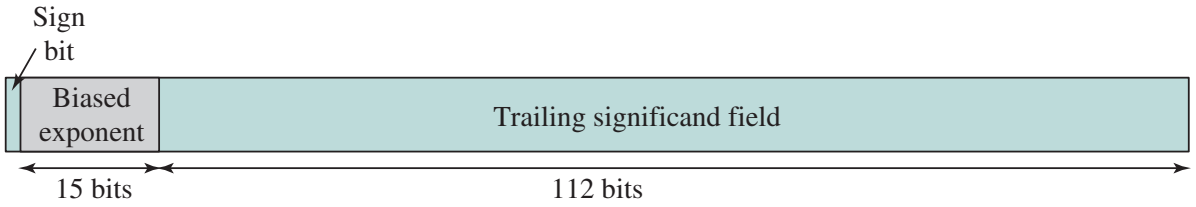
(a) Binary32 format

  - Dạng 64-bit



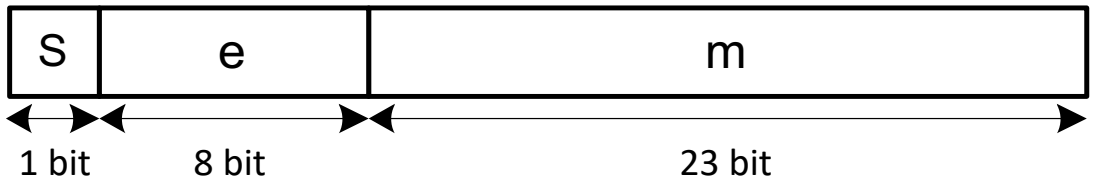
(b) Binary64 format

  - Dạng 128-bit



(c) Binary128 format

Dạng 32-bit



- S là bit dấu:
  - S = 0 → số dương
  - S = 1 → số âm
- e (8 bit) là giá trị dịch chuyển của phần mũ E:
  - e = E + 127 → phần mũ E = e - 127
- m (23 bit) là phần lẻ của phần định trị M:
  - M = 1.m
- Công thức xác định giá trị của số thực:

$$X = (-1)^S * 1.m * 2^{e-127}$$

Ví dụ 1

Xác định giá trị của các số thực được biểu diễn bằng 32-bit sau đây:

- 1100 0001 0101 0110 0000 0000 0000 0000
  - S = 1 → số âm
  - e = $1000\ 0010_{(2)} = 130_{(10)}$ → E = 130 - 127 = 3

  Vậy

  $X = -1.10101100_{(2)} * 2^3 = -1101.011_{(2)} = -13.375_{(10)}$

- 0011 1111 1000 0000 0000 0000 0000 0000 = ?

Ví dụ 2

Biểu diễn số thực $X = 83.75_{(10)}$ về dạng số dấu phẩy động IEEE754 32-bit

Giải:

- $X = 83.75_{(10)} = 1010011.11_{(2)} = 1.01001111 \times 2^6$
- Ta có:
  - $S = 0$ vì đây là số dương
  - $E = e - 127 = 6 \rightarrow e = 127 + 6 = 133_{(10)} = 1000\ 0101_{(2)}$
- Vậy:

$X = 0100\ 0010\ 1010\ 0111\ 1000\ 0000\ 0000\ 0000$

Các qui ước đặc biệt

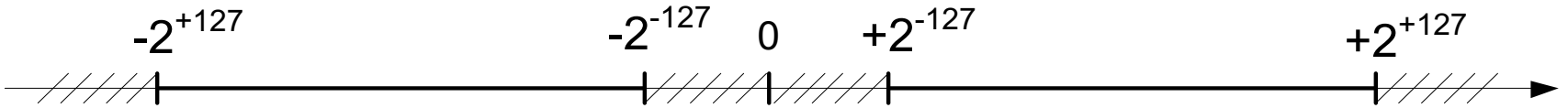
- Các bit của e bằng 0, các bit của m bằng 0, thì X = ± 0
  x000 0000 0000 0000 0000 0000 0000 0000 → X = ± 0

- Các bit của e bằng 1, các bit của m bằng 0, thì X = ± ∞
  x111 1111 1000 0000 0000 0000 0000 0000 → X =± ∞

- Các bit của e bằng 1, còn m có ít nhất một bit bằng 1, thì nó không biểu diễn cho số nào cả (NaN - not a number)

Dải giá trị biểu diễn

- 2^{-127} đến 2^{+127}
- 10^{-38} đến 10^{+38}



Dạng 64-bit

- S là bit dấu
- e (11 bit) là giá trị dịch chuyển của phần mũ E:
  - e = E + 1023 → phần mũ E = e - 1023
- m (52 bit): phần lẻ của phần định trị M
- Giá trị số thực:

$$X = (-1)^S * 1.m * 2^{e-1023}$$

- Dải giá trị biểu diễn: 10^{-308} đến 10^{+308}

Dạng 128-bit

- S là bit dấu
- e (15 bit) là giá trị dịch chuyển của phần mũ E:
  - e = E + 16383 → phần mũ E = e - 16383
- m (112 bit): phần lẻ của phần định trị M
- Giá trị số thực:

$$X = (-1)^S * 1.m * 2^{e-16383}$$

- Dải giá trị biểu diễn: 10^{-4932} đến 10^{+4932}

3. Thực hiện phép toán số dấu phẩy động

- $X1 = M1 * R^{E1}$
- $X2 = M2 * R^{E2}$
- Ta có
  - $X1 * X2 = (M1 * M2) * R^{E1+E2}$
  - $X1 / X2 = (M1 / M2) * R^{E1-E2}$
  - $X1 \pm X2 = (M1 * R^{E1-E2} \pm M2) * R^{E2}$, với $E2 \geq E1$

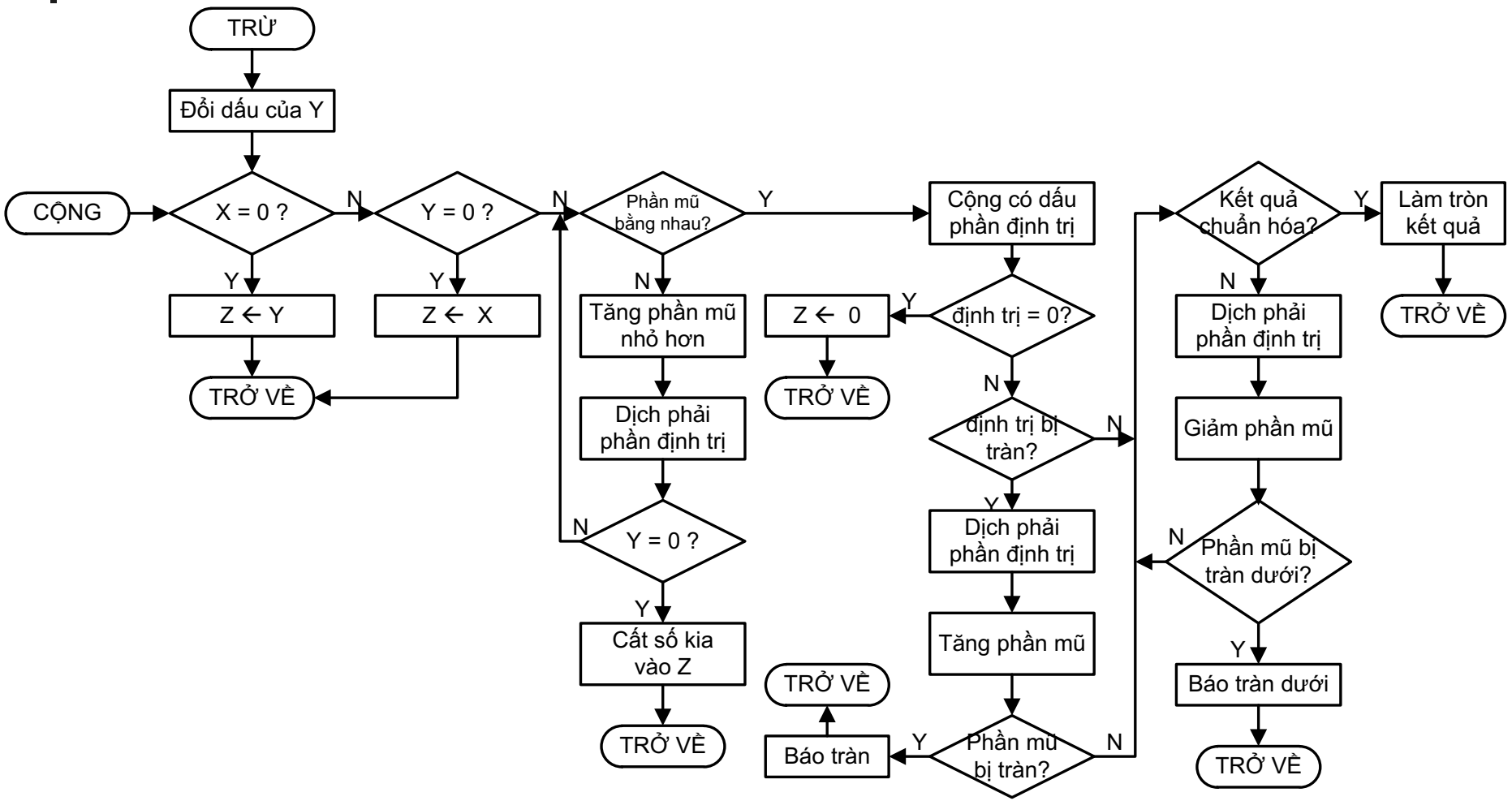
Các khả năng tràn số

- Tràn trên số mũ (Exponent Overflow): mũ dương vượt ra khỏi giá trị cực đại của số mũ dương có thể (→ ∞)
- Tràn dưới số mũ (Exponent Underflow): mũ âm vượt ra khỏi giá trị cực đại của số mũ âm có thể (→ 0)
- Tràn trên phần định trị (Mantissa Overflow): cộng hai phần định trị có cùng dấu, kết quả bị nhớ ra ngoài bit cao nhất
- Tràn dưới phần định trị (Mantissa Underflow): Khi hiệu chỉnh phần định trị, các số bị mất ở bên phải phần định trị

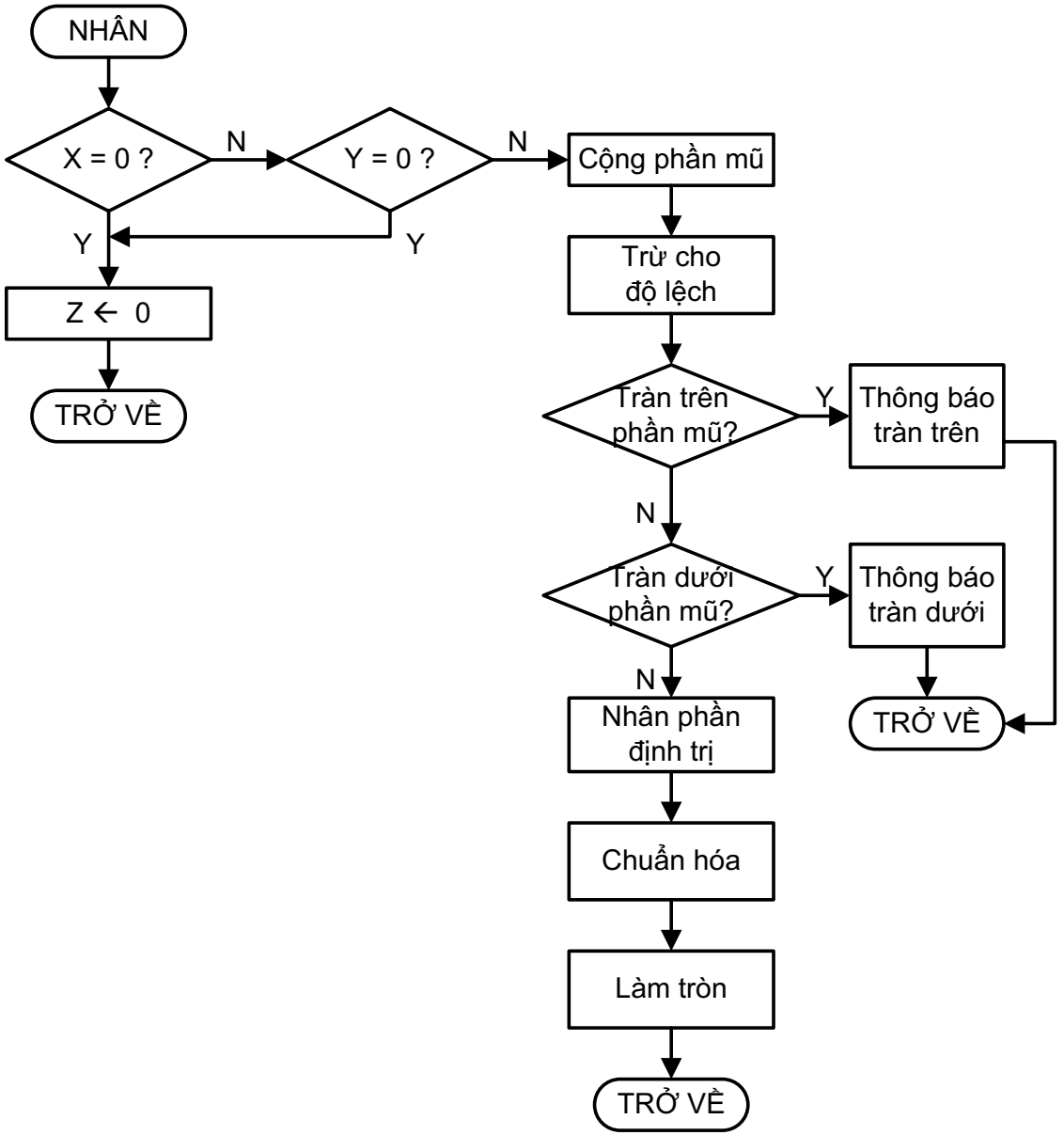
Phép cộng và phép trừ

- Kiểm tra các số hạng có bằng 0 hay không
- Hiệu chỉnh phần định trị
- Cộng hoặc trừ phần định trị
- Chuẩn hoá kết quả

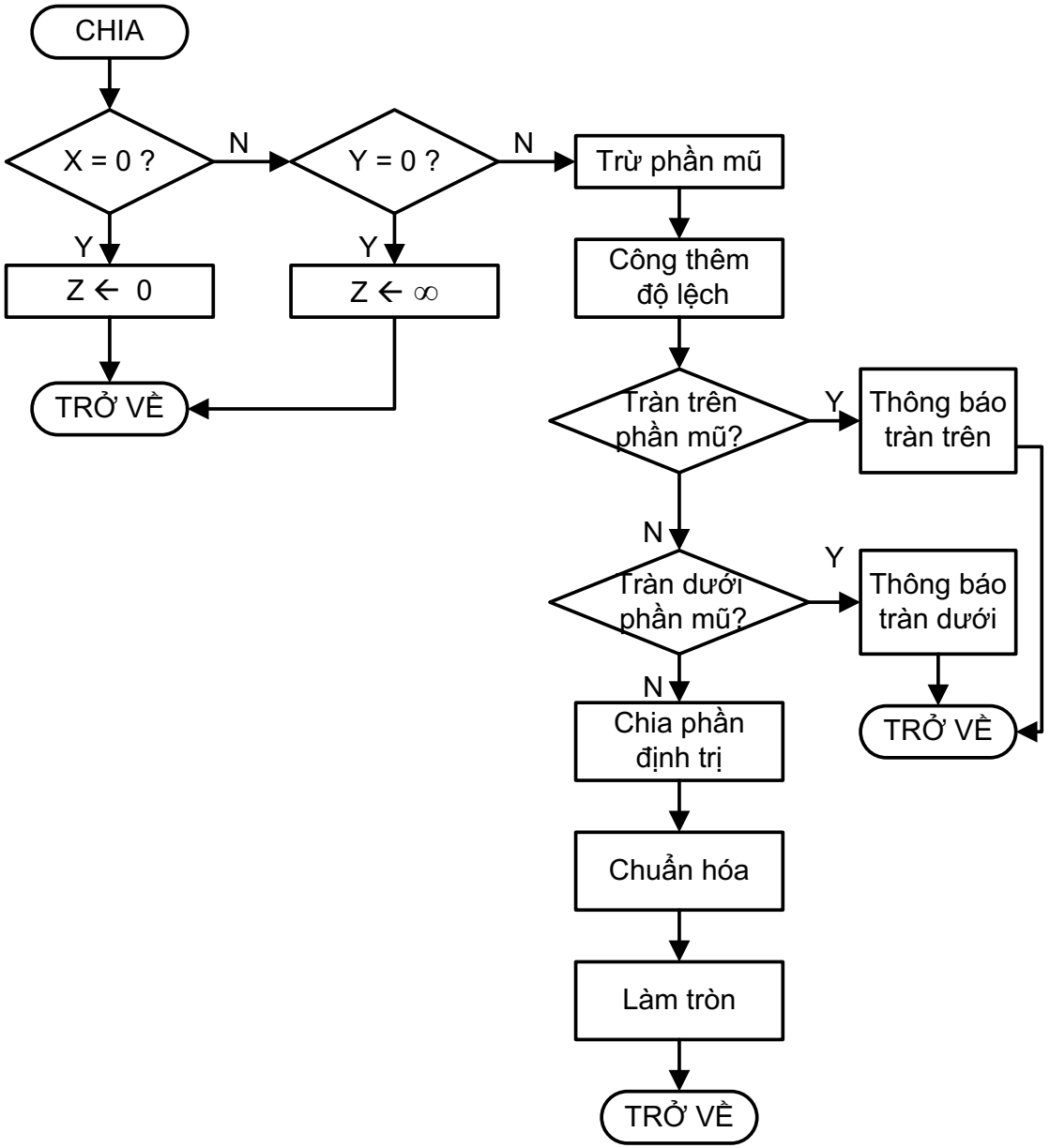
Thuật toán cộng/trừ số dấu phẩy động



Thuật toán nhân số dấu phẩy động



Thuật toán chia số dấu phẩy động



Hết chương 4

Chương 5
KIẾN TRÚC TẬP LỆNH

Nguyễn Kim Khánh
Trường Đại học Bách khoa Hà Nội

Nội dung học phần

Chương 1. Giới thiệu chung

Chương 2. Cơ bản về logic số

Chương 3. Hệ thống máy tính

Chương 4. Số học máy tính

Chương 5. Kiến trúc tập lệnh

Chương 6. Bộ xử lý

Chương 7. Bộ nhớ máy tính

Chương 8. Hệ thống vào-ra

Chương 9. Các kiến trúc song song

Nội dung của chương 5

5.1. Giới thiệu chung về kiến trúc tập lệnh

5.2. Lệnh hợp ngữ và toán hạng

5.3. Mã máy

5.4. Cơ bản về lập trình hợp ngữ

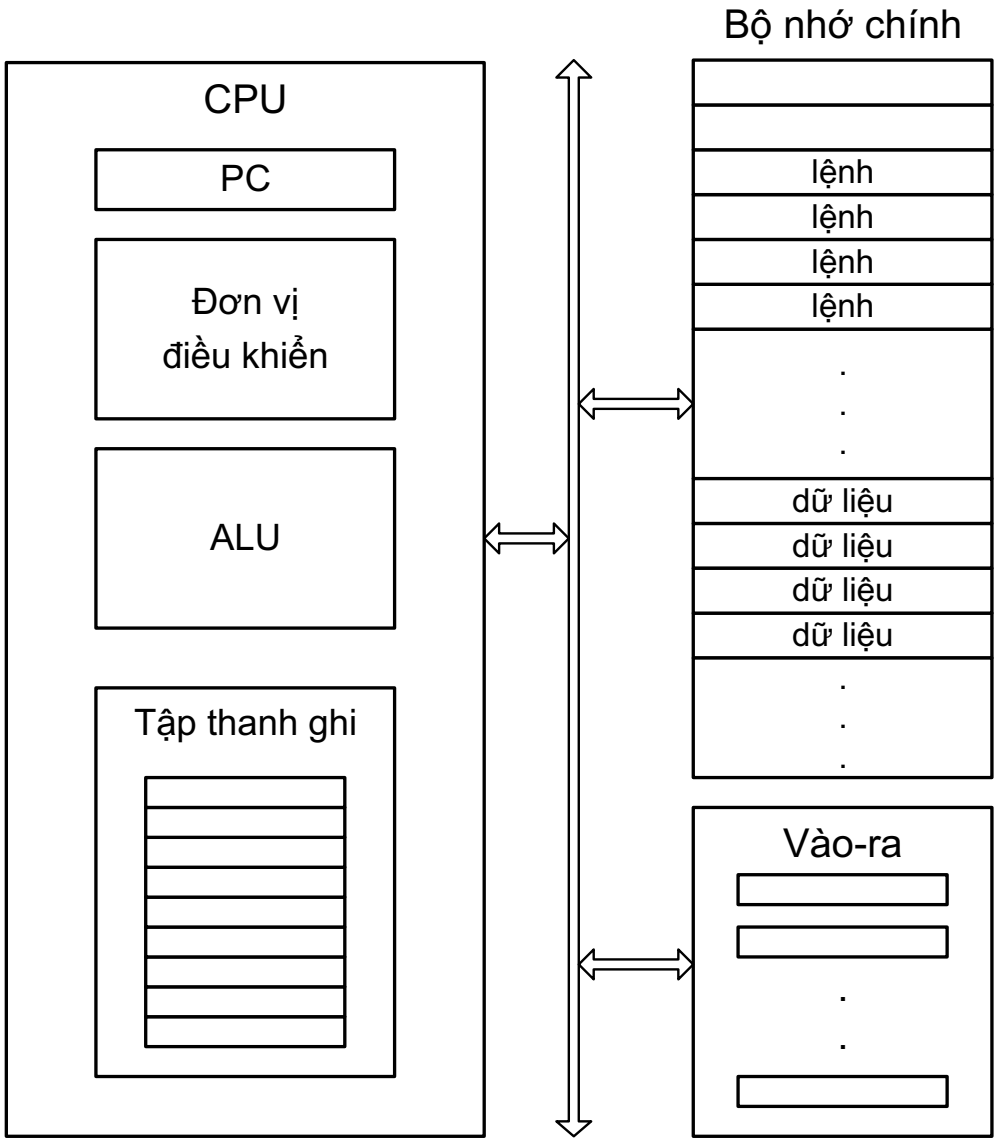
5.5. Các phương pháp định địa chỉ

5.6. Dịch và chạy chương trình hợp ngữ

5.1. Giới thiệu chung về kiến trúc tập lệnh

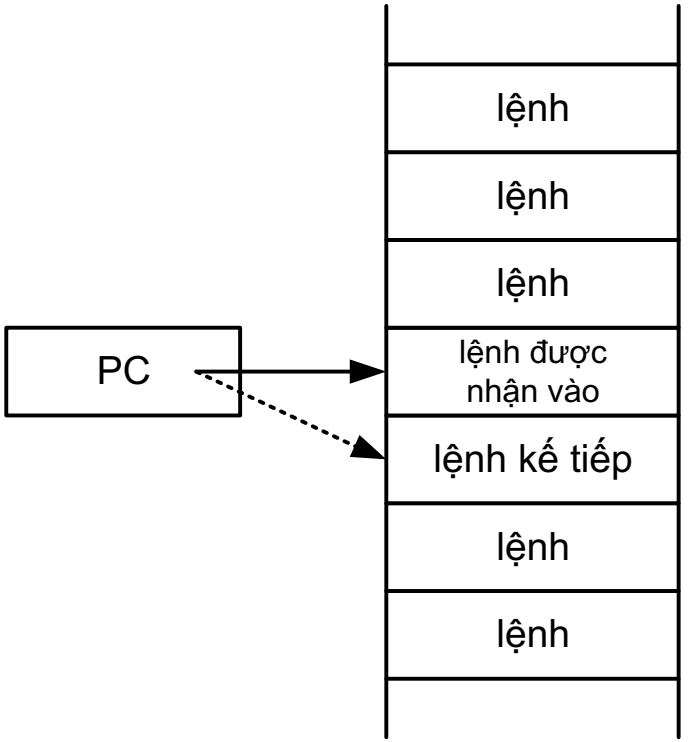
- Kiến trúc tập lệnh (Instruction Set Architecture): cách nhìn máy tính bởi người lập trình
- Vi kiến trúc (Microarchitecture): cách thực hiện kiến trúc tập lệnh bằng phần cứng
- Ngôn ngữ trong máy tính:
  - Hợp ngữ (assembly language):
    - dạng lệnh có thể đọc được bởi con người
    - biểu diễn dạng text
  - Ngôn ngữ máy (machine language):
    - còn gọi là mã máy (machine code)
    - dạng lệnh có thể đọc được bởi máy tính
    - biểu diễn bằng các bit 0 và 1

Mô hình lập trình của máy tính



CPU nhận lệnh từ bộ nhớ

- Bộ đếm chương trình PC (Program Counter) là thanh ghi của CPU giữ địa chỉ của lệnh cần nhận vào để thực hiện
- CPU phát địa chỉ từ PC đến bộ nhớ, lệnh được nhận vào
- Sau khi lệnh được nhận vào, nội dung PC tự động tăng để trỏ sang lệnh kế tiếp
- PC tăng bao nhiêu?
  - Tùy thuộc vào độ dài của lệnh vừa được nhận
  - MIPS: lệnh có độ dài 32-bit, PC tăng 4



Giải mã và thực hiện lệnh

- Bộ xử lý giải mã lệnh đã được nhận và phát các tín hiệu điều khiển thực hiện thao tác mà lệnh yêu cầu
- Các kiểu thao tác chính của lệnh:
  - Trao đổi dữ liệu giữa CPU với bộ nhớ chính hoặc với cổng vào-ra
  - Thực hiện các phép toán số học hoặc phép toán logic với các dữ liệu (được thực hiện bởi ALU)
  - Chuyển điều khiển trong chương trình (rẽ nhánh, nhảy)

CPU đọc/ghi dữ liệu bộ nhớ

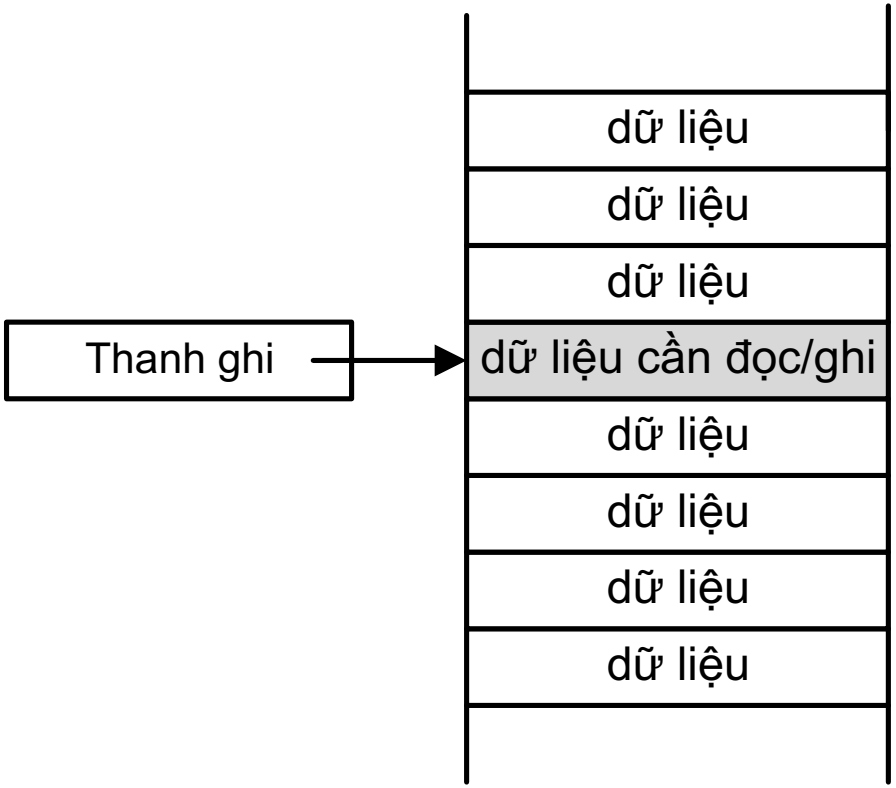
- Với các lệnh trao đổi dữ liệu với bộ nhớ, CPU cần biết và phát ra địa chỉ của ngăn nhớ cần đọc/ghi
- Địa chỉ đó có thể là:
  - Hằng số địa chỉ được cho trực tiếp trong lệnh
  - Giá trị địa chỉ nằm trong thanh ghi con trỏ
  - Địa chỉ = Địa chỉ cơ sở + giá trị dịch chuyển

Hằng số địa chỉ

- Trong lệnh cho hằng số địa chỉ cụ thể
- CPU phát giá trị địa chỉ này đến bộ nhớ để tìm ra ngăn nhớ dữ liệu cần đọc/ghi

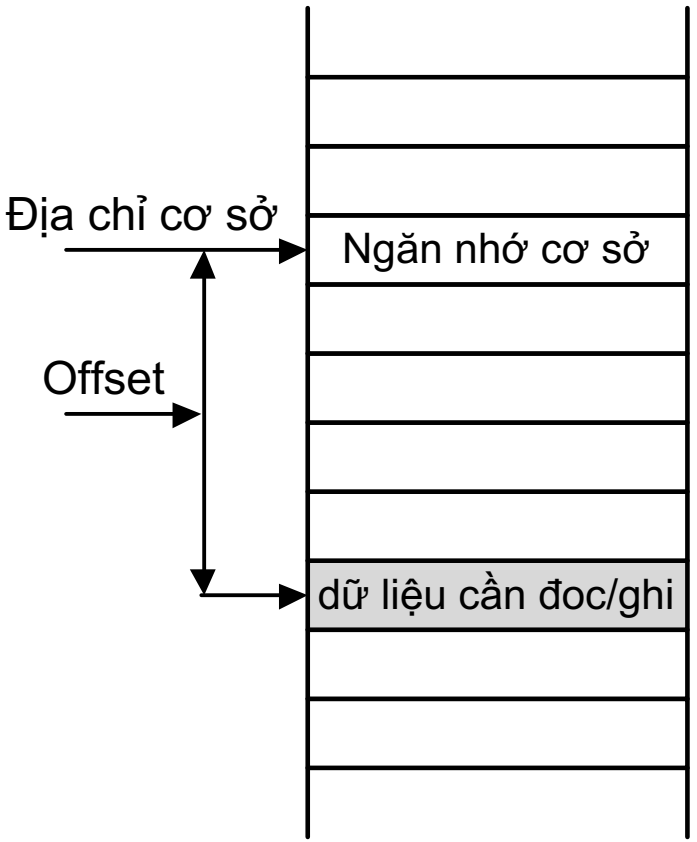
Sử dụng thanh ghi con trỏ

- Trong lệnh cho biết tên thanh ghi con trỏ
- Thanh ghi con trỏ chứa giá trị địa chỉ
- CPU phát địa chỉ này ra để tìm ra ngăn nhớ dữ liệu cần đọc/ghi



Sử dụng địa chỉ cơ sở và dịch chuyển

- Địa chỉ cơ sở (base address): địa chỉ của ngăn nhớ cơ sở
- Giá trị dịch chuyển địa chỉ (offset): gia số địa chỉ giữa ngăn nhớ cần đọc/ghi so với ngăn nhớ cơ sở
- Địa chỉ của ngăn nhớ cần đọc/ghi = (địa chỉ cơ sở) + (offset)
- Có thể sử dụng các thanh ghi để quản lý các tham số này
- Trường hợp riêng:
  - Địa chỉ cơ sở = 0
  - Offset = 0

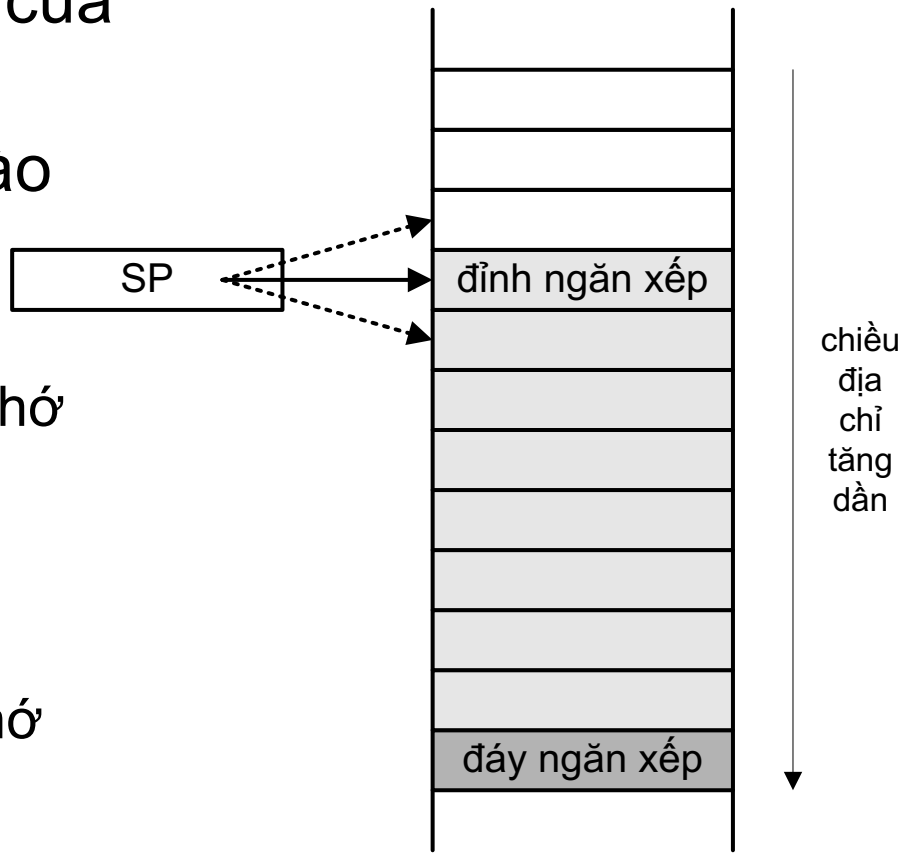


Ngăn xếp (Stack)

- Ngăn xếp là vùng nhớ dữ liệu có cấu trúc LIFO (Last In - First Out vào sau - ra trước)
- Ngăn xếp thường dùng để phục vụ cho chương trình con
- Đáy ngăn xếp là một ngăn nhớ xác định
- Đỉnh ngăn xếp là thông tin nằm ở vị trí trên cùng trong ngăn xếp
- Đỉnh ngăn xếp có thể bị thay đổi

Con trỏ ngăn xếp SP (Stack Pointer)

- SP là thanh ghi chứa địa chỉ của ngăn nhớ đỉnh ngăn xếp
- Khi cất thêm một thông tin vào ngăn xếp:
  - Giảm nội dung của SP
  - Thông tin được cất vào ngăn nhớ được trỏ bởi SP
- Khi lấy một thông tin ra khỏi ngăn xếp:
  - Thông tin được đọc từ ngăn nhớ được trỏ bởi SP
  - Tăng nội dung của SP
- Khi ngăn xếp rỗng, SP trỏ vào đáy



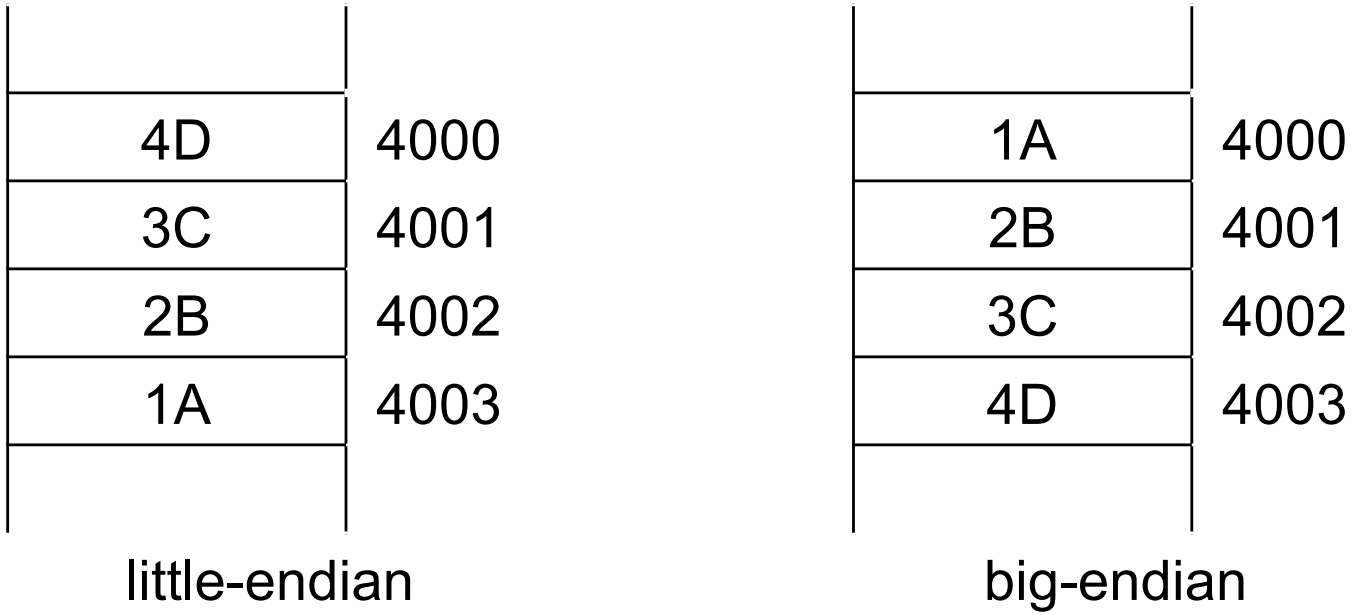
Thứ tự lưu trữ các byte trong bộ nhớ chính

- Bộ nhớ chính được đánh địa chỉ cho từng byte
- Hai cách lưu trữ thông tin nhiều byte:
  - Đầu nhỏ (*Little-endian*): Byte có ý nghĩa thấp được lưu trữ ở ngăn nhớ có địa chỉ nhỏ, byte có ý nghĩa cao được lưu trữ ở ngăn nhớ có địa chỉ lớn.
  - Đầu to (*Big-endian*): Byte có ý nghĩa cao được lưu trữ ở ngăn nhớ có địa chỉ nhỏ, byte có ý nghĩa thấp được lưu trữ ở ngăn nhớ có địa chỉ lớn.
- Các sản phẩm thực tế:
  - Intel x86: little-endian
  - Motorola 680x0, SunSPARC: big-endian
  - MIPS, IA-64: bi-endian (cả hai kiểu)

Ví dụ lưu trữ dữ liệu 32-bit

Số nhị phân	0001	1010	0010	1011	0011	1100	0100	1101

Số Hexa	1A	2B	3C	4D



Tập lệnh

- Mỗi bộ xử lý có một tập lệnh xác định
- Tập lệnh thường có hàng chục đến hàng trăm lệnh
- Mỗi lệnh máy (mã máy) là một chuỗi các bit (0,1) mà bộ xử lý hiểu được để thực hiện một thao tác xác định.
- Các lệnh được mô tả bằng các ký hiệu gợi nhớ dạng text, đó chính là các lệnh của hợp ngữ (assembly language)

Dạng lệnh hợp ngữ

- Mã C:

  a = b + c;

- Ví dụ lệnh hợp ngữ:

  add a, b, c # a = b + c

trong đó:

- add: ký hiệu gợi nhớ chỉ ra thao tác (phép toán) cần thực hiện.
  - Chú ý: mỗi lệnh chỉ thực hiện một thao tác
- b, c: các toán hạng nguồn cho thao tác
- a: toán hạng đích (nơi ghi kết quả)
- phần sau dấu # là lời giải thích (chỉ có tác dụng đến hết dòng)

Các thành phần của lệnh máy

Mã thao tác	Địa chỉ toán hạng

- Mã thao tác (operation code hay opcode): mã hóa cho thao tác mà bộ xử lý phải thực hiện
  - Các thao tác chuyển dữ liệu
  - Các phép toán số học
  - Các phép toán logic
  - Các thao tác chuyển điều khiển (rẽ nhánh, nhảy)
- Địa chỉ toán hạng: chỉ ra nơi chứa các toán hạng mà thao tác sẽ tác động
  - Toán hạng có thể là:
    - Hằng số nằm ngay trong lệnh
    - Nội dung của thanh ghi
    - Nội dung của ngăn nhớ (hoặc cổng vào-ra)

Số lượng địa chỉ toán hạng trong lệnh

- Ba địa chỉ toán hạng:
  - add r1, r2, r3 # r1 = r2 + r3
  - Sử dụng phổ biến trên các kiến trúc hiện nay
- Hai địa chỉ toán hạng:
  - add r1, r2 # r1 = r1 + r2
  - Sử dụng trên Intel x86, Motorola 680x0
- Một địa chỉ toán hạng:
  - add r1 # Acc = Acc + r1
  - Được sử dụng trên kiến trúc thế hệ trước
- 0 địa chỉ toán hạng:
  - Các toán hạng đều được ngầm định ở ngăn xếp
  - Không thông dụng

Các kiến trúc tập lệnh CISC và RISC

- CISC: Complex Instruction Set Computer
  - Máy tính với tập lệnh phức tạp
  - Các bộ xử lý: Intel x86, Motorola 680x0
- RISC: Reduced Instruction Set Computer
  - Máy tính với tập lệnh thu gọn
  - SunSPARC, Power PC, MIPS, ARM ...
  - RISC đối nghịch với CISC
  - Kiến trúc tập lệnh tiên tiến

Các đặc trưng của kiến trúc RISC

- Số lượng lệnh ít
- Hầu hết các lệnh truy nhập toán hạng ở các thanh ghi
- Truy nhập bộ nhớ bằng các lệnh LOAD/STORE (nạp/lưu)
- Thời gian thực hiện các lệnh là như nhau
- Các lệnh có độ dài cố định (thường là 32 bit)
- Số lượng dạng lệnh ít
- Có ít phương pháp định địa chỉ toán hạng
- Có nhiều thanh ghi
- Hỗ trợ các thao tác của ngôn ngữ bậc cao

Kiến trúc tập lệnh MIPS

- MIPS viết tắt cho:

  Microprocessor without **I**nterlocked **P**ipeline **S**tages
- Được phát triển bởi John Hennessy và các đồng nghiệp ở đại học Stanford (1984)
- Được thương mại hóa bởi MIPS Technologies
- Năm 2013 công ty này được bán cho Imagination Technologies (imgtec.com)
- Là kiến trúc RISC điển hình, dễ học
- Được sử dụng trong nhiều sản phẩm thực tế
- Các phần tiếp theo trong chương này sẽ nghiên cứu kiến trúc tập lệnh MIPS 32-bit
  - *Tài liệu: MIPS Reference Data Sheet và Chapter 2 – COD*

5.2. Lệnh hợp ngữ và các toán hạng

- Thực hiện phép cộng: 3 toán hạng
  - Là phép toán phổ biến nhất
  - Hai toán hạng nguồn và một toán hạng đích

```
add a, b, c  # a = b + c
```

- Hầu hết các lệnh số học/logic có dạng trên
- Các lệnh số học sử dụng toán hạng thanh ghi hoặc hằng số

Tập thanh ghi của MIPS

- MIPS có tập 32 thanh ghi 32-bit
  - Được sử dụng thường xuyên
  - Được đánh số từ 0 đến 31 (mã hóa bằng 5-bit)
- Chương trình hợp dịch Assembler đặt tên:
  - Bắt đầu bằng dấu $
  - $t0, $t1, …, $t9 chứa các giá trị tạm thời
  - $s0, $s1, …, $s7 cất các biến
- Qui ước gọi dữ liệu trong MIPS:
  - Dữ liệu 32-bit được gọi là “word”
  - Dữ liệu 16-bit được gọi là “halfword”

Tập thanh ghi của MIPS

Tên thanh ghi	Số hiệu thanh ghi	Công dụng
$zero	0	the constant value 0, chứa hằng số = 0
$at	1	assembler temporary, giá trị tạm thời cho hợp ngữ
$v0-$v1	2-3	procedure return values, các giá trị trả về của thủ tục
$a0-$a3	4-7	procedure arguments, các tham số vào của thủ tục
$t0-$t7	8-15	temporaries, chứa các giá trị tạm thời
$s0-$s7	16-23	saved variables, lưu các biến
$t8-$t9	24-25	more temporarie, chứa các giá trị tạm thời
$k0-$k1	26-27	OS temporaries, các giá trị tạm thời của OS
$gp	28	global pointer, con trỏ toàn cục
$sp	29	stack pointer, con trỏ ngăn xếp
$fp	30	frame pointer, con trỏ khung
$ra	31	procedure return address, địa chỉ trở về của thủ tục

Toán hạng thanh ghi

- Lệnh add, lệnh sub (subtract) chỉ thao tác với toán hạng thanh ghi
  - `add  rd, rs, rt  # (rd) = (rs)+(rt)`
  - `sub  rd, rs, rt  # (rd) = (rs)-(rt)`
- Ví dụ mã C:

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

  - giả thiết: f, g, h, i, j nằm ở $s0, $s1, $s2, $s3, $s4

- Được dịch thành mã hợp ngữ MIPS:

```
add  $t0, $s1, $s2  # $t0 = g + h
add  $t1, $s3, $s4  # $t1 = i + j
sub  $s0, $t0, $t1  # f = (g+h)-(i+j)
```

Toán hạng ở bộ nhớ

- Muốn thực hiện phép toán số học với toán hạng ở bộ nhớ, cần phải:
  - Nạp (load) giá trị từ bộ nhớ vào thanh ghi
  - Thực hiện phép toán trên thanh ghi
  - Lưu (store) kết quả từ thanh ghi ra bộ nhớ
- Bộ nhớ được đánh địa chỉ theo byte
  - MIPS sử dụng 32-bit để đánh địa chỉ cho các byte nhớ và các cổng vào-ra
  - Không gian địa chỉ: **0x00000000 – 0xFFFFFFFF**
  - Mỗi word có độ dài 32-bit chiếm 4-byte trong bộ nhớ, địa chỉ của các word là bội của 4 (địa chỉ của byte đầu tiên)
- MIPS cho phép lưu trữ trong bộ nhớ theo kiểu đầu to (*big-endian*) hoặc kiểu đầu nhỏ (*little-endian*)

Địa chỉ byte nhớ và word nhớ

Dữ liệu hoặc lệnh	Địa chỉ byte (theo Hexa)
byte (8-bit)	**0x0000 0000**
byte	0x0000 0001
byte	0x0000 0002
byte	0x0000 0003
byte	**0x0000 0004**
byte	0x0000 0005
byte	0x0000 0006
byte	0x0000 0007
.	
.	
.	
byte	0xFFFF FFFB
byte	**0xFFFF FFFC**
byte	0xFFFF FFFD
byte	0xFFFF FFFE
byte	0xFFFF FFFF

2^{32} bytes

Dữ liệu hoặc lệnh	Địa chỉ word (theo Hexa)
word (32-bit)	0x0000 0000
word	0x0000 0004
word	0x0000 0008
word	0x0000 000C
word	0x0000 0010
word	0x0000 0014
word	0x0000 0018
.	
.	
.	
word	0xFFFF FFF4
word	0xFFFF FFF8
word	0xFFFF FFFC

2^{30} words

Lệnh load và lệnh store

- Để đọc word dữ liệu 32-bit từ bộ nhớ đưa vào thanh ghi, sử dụng lệnh *load word*

```
lw   rt, imm(rs)    # (rt) = mem[(rs)+imm]
```

  - rs: thanh ghi chứa địa chỉ cơ sở (base address)
  - imm (immediate): hằng số (offset)

  → địa chỉ của word dữ liệu cần đọc = địa chỉ cơ sở + hằng số

  - rt: thanh ghi đích, chứa word dữ liệu được đọc vào

- Để ghi word dữ liệu 32-bit từ thanh ghi đưa ra bộ nhớ, sử dụng lệnh *store word*

```
sw   rt, imm(rs)    # mem[(rs)+imm] = (rt)
```

  - rt: thanh ghi nguồn, chứa word dữ liệu cần ghi ra bộ nhớ
  - rs: thanh ghi chứa địa chỉ cơ sở (base address)
  - imm: hằng số (offset)

  → địa chỉ nơi ghi word dữ liệu = địa chỉ cơ sở + hằng số

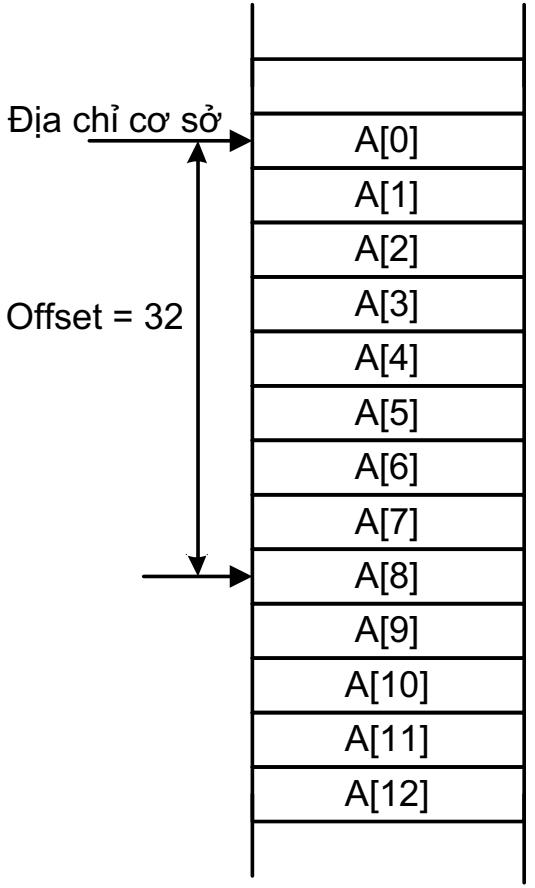
Ví dụ toán hạng bộ nhớ

- Mã C:

// A là mảng các phần tử 32-bit

```
g = h + A[8];
```

  - Cho g ở $s1, h ở $s2
  - $s3 chứa địa chỉ cơ sở của mảng A



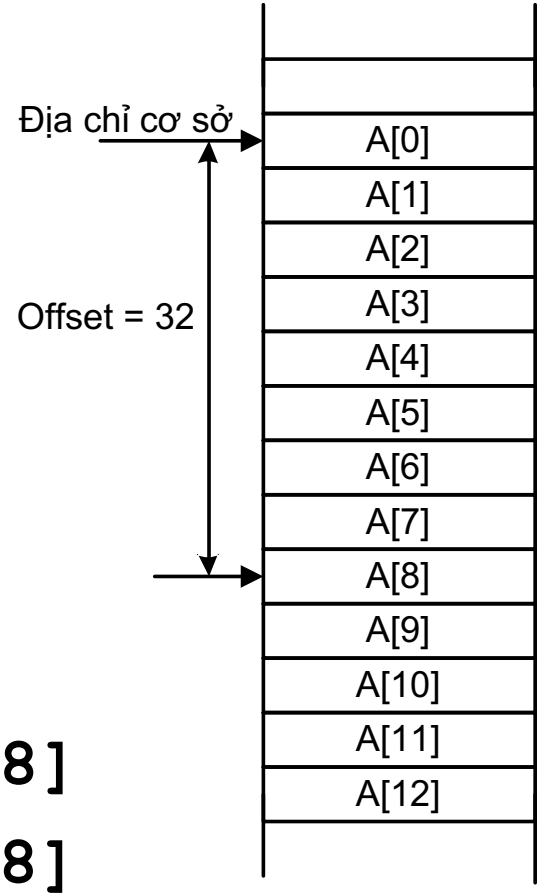
Ví dụ toán hạng bộ nhớ

- Mã C:

// A là mảng các phần tử 32-bit

```
g = h + A[8];
```

  - Cho g ở $s1, h ở $s2
  - $s3 chứa địa chỉ cơ sở của mảng A



- Mã hợp ngữ MIPS:

Chỉ số 8, do đó offset = 32

```
lw   $t0, 32($s3)   # $t0 = A[8]
add  $s1, $s2, $t0  # g = h+A[8]
```

offset

base register

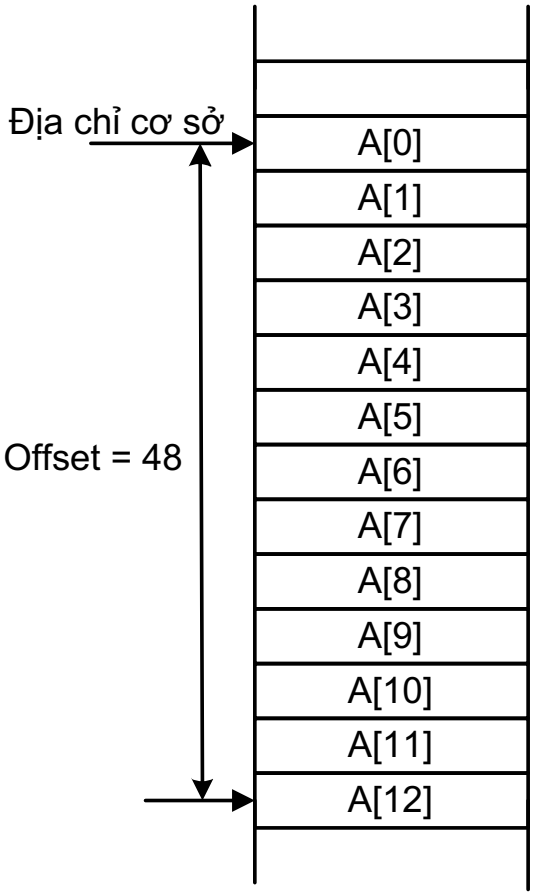
(Chú ý: offset phải là hằng số, có thể dương hoặc âm)

Ví dụ toán hạng bộ nhớ (tiếp)

- Mã C:

```
A[12] = h + A[8];
```

  - h ở $s2
  - $s3 chứa địa chỉ cơ sở của mảng A

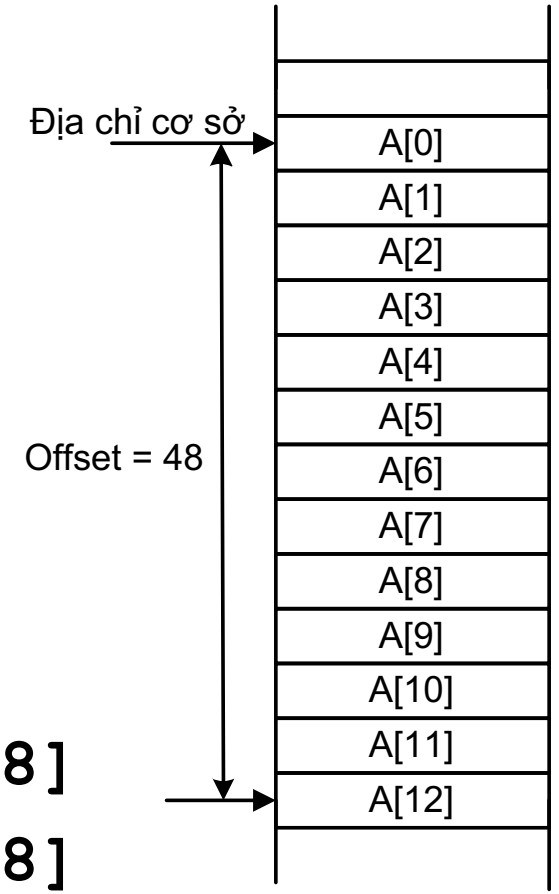


Ví dụ toán hạng bộ nhớ (tiếp)

- Mã C:

```
A[12] = h + A[8];
```

  - h ở $s2
  - $s3 chứa địa chỉ cơ sở của mảng A



- Mã hợp ngữ MIPS:

```
lw   $t0, 32($s3)    # $t0 = A[8]
add  $t0, $s2, $t0   # $t0 = h+A[8]
sw   $t0, 48($s3)    # A[12]=h+A[8]
```

Thanh ghi với Bộ nhớ

- Truy nhập thanh ghi nhanh hơn bộ nhớ
- Thao tác dữ liệu trên bộ nhớ yêu cầu nạp (load) và lưu (store)
  - Cần thực hiện nhiều lệnh hơn
- Chương trình dịch sử dụng các thanh ghi cho các biến nhiều nhất có thể
  - Chỉ sử dụng bộ nhớ cho các biến ít được sử dụng
  - Cần tối ưu hóa sử dụng thanh ghi

Toán hạng tức thì (immediate)

- Dữ liệu hằng số được xác định ngay trong lệnh

```
addi $s3, $s3, 4    # $s3 = $s3+4
```

- Không có lệnh trừ (subi) với giá trị hằng số
  - Sử dụng hằng số âm trong lệnh addi để thực hiện phép trừ

```
addi $s2, $s1, -1   # $s2 = $s1-1
```

Xử lý với số nguyên

- Số nguyên có dấu (biểu diễn bằng bù hai):
  - Với n bit, dải biểu diễn: $[-2^{n-1}, +(2^{n-1}-1)]$
  - Các lệnh add, sub, addi dành cho số nguyên có dấu
- Số nguyên không dấu:
  - Với n bit, dải biểu diễn: $[0, 2^n-1]$
  - Các lệnh addu, subu, addiu dành cho số nguyên không dấu
- Qui ước biểu diễn hằng số nguyên trong hợp ngữ MIPS:
  - số thập phân: 12; 3456; -18
  - số Hexa (bắt đầu bằng 0x): 0x12 ; 0x3456; 0x1AB6

Hằng số Zero

- Thanh ghi 0 của MIPS ($zero hay $0) luôn chứa hằng số 0
  - Không thể thay đổi giá trị
- Hữu ích cho một số thao tác thông dụng
  - Chẳng hạn, chuyển dữ liệu giữa các thanh ghi

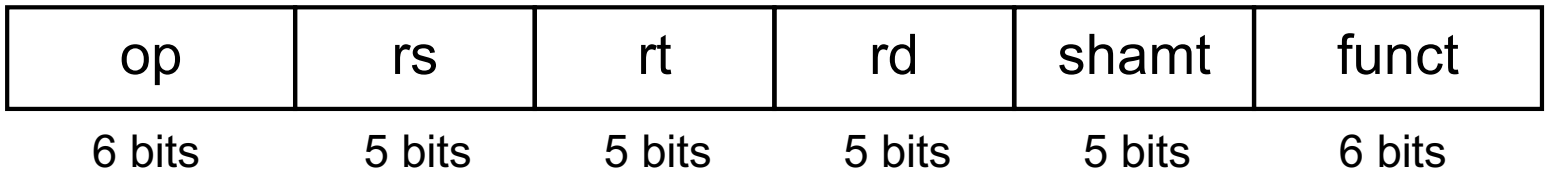
```
add $t2, $s1, $zero   # $t2 = $s1
```

5.3. Mã máy (Machine code)

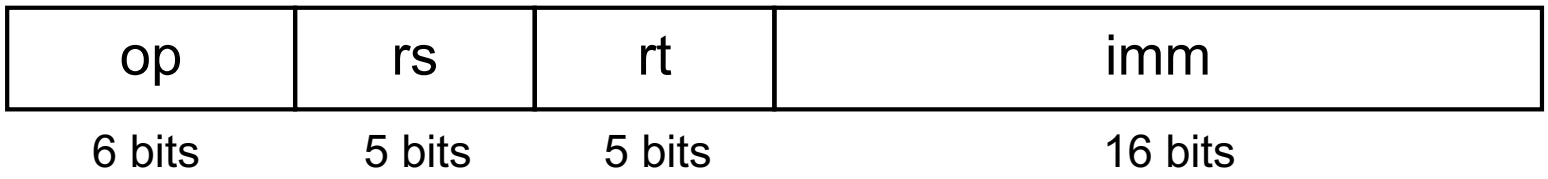
- Các lệnh được mã hóa dưới dạng nhị phân được gọi là mã máy
- Các lệnh của MIPS:
  - Được mã hóa bằng các từ lệnh 32-bit
  - Mỗi lệnh chiếm 4-byte trong bộ nhớ, do vậy địa chỉ của lệnh trong bộ nhớ là bội của 4
  - Có ít dạng lệnh
- Số hiệu thanh ghi được mã hóa bằng 5-bit
  - $t0 – $t7 có số hiệu từ 8 – 15
  - $t8 – $t9 có số hiệu từ 24 – 25
  - $s0 – $s7 có số hiệu từ 16 – 23

Các kiểu lệnh máy của MIPS

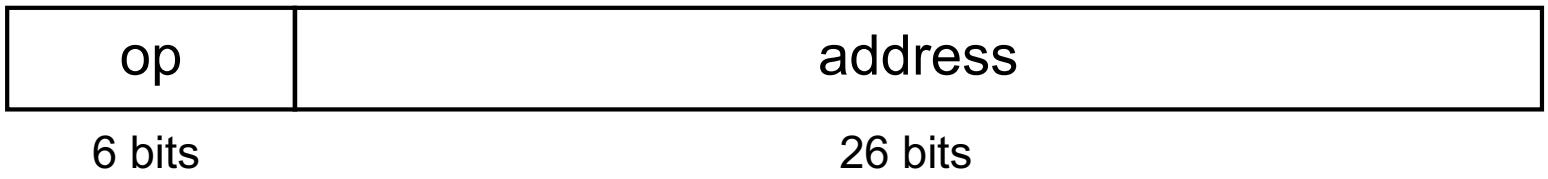
Lệnh kiểu R



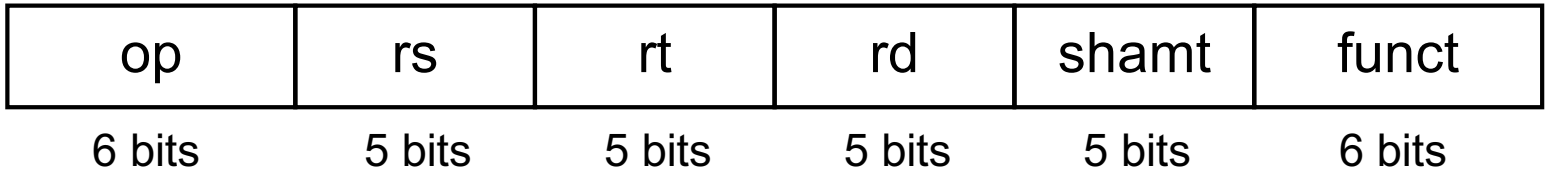
Lệnh kiểu I



Lệnh kiểu J



Lệnh kiểu R (Registers)



- Các trường của lệnh
  - op (operation code - opcode): mã thao tác
    - với các lệnh kiểu R, op = 000000
  - rs: số hiệu thanh ghi nguồn thứ nhất
  - rt: số hiệu thanh ghi nguồn thứ hai
  - rd: số hiệu thanh ghi đích
  - shamt (shift amount): số bit được dịch, chỉ dùng cho lệnh dịch bit, với các lệnh khác shamt = 00000
  - funct (function code): mã hàm

Ví dụ mã máy của lệnh add, sub

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

```
add $t0, $s1, $s2
```

0	$s1	$s2	$t0	0	add
0	17	18	8	0	32
000000	10001	10010	01000	00000	100000

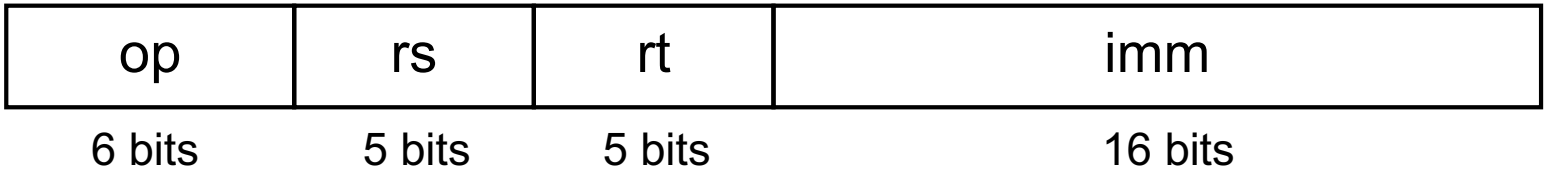
(0x02324020)

```
sub $s0, $t3, $t5
```

0	$t3	$t5	$s0	0	sub
0	11	13	16	0	34
000000	01011	01101	10000	00000	100010

(0x016D8022)

Lệnh kiểu I (Immediate)



- Dùng cho các lệnh số học/logic với toán hạng tức thì và các lệnh load/store (nạp/lưu)
  - rs: số hiệu thanh ghi nguồn (addi) hoặc thanh ghi cơ sở (lw, sw)
  - rt: số hiệu thanh ghi đích (addi, lw) hoặc thanh ghi nguồn (sw)
  - imm (immediate): hằng số nguyên 16-bit

```
addi rt, rs, imm    # (rt) = (rs)+SignExtImm
lw   rt, imm(rs)    # (rt) = mem[(rs)+SignExtImm]
sw   rt, imm(rs)    # mem[(rs)+SignExtImm] = (rt)
```

(SignExtImm: hằng số imm 16-bit được mở rộng theo kiểu số có dấu thành 32-bit)

Mở rộng bit cho hằng số theo số có dấu

- Với các lệnh addi, lw, sw cần cộng nội dung thanh ghi với hằng số:
  - Thanh ghi có độ dài 32-bit
  - Hằng số imm 16-bit, cần mở rộng thành 32-bit theo kiểu số có dấu (Sign-extended)
- Ví dụ mở rộng số 16-bit thành 32-bit theo kiểu số có dấu:

+5 =	0000 0000 0000 0101	16-bit
+5 =	0000 0000 0000 0000 0000 0000 0000 0101	32-bit
-12 =	1111 1111 1111 0100	16-bit
-12 =	1111 1111 1111 1111 1111 1111 1111 0100	32-bit

Ví dụ mã máy của lệnh addi

op	rs	rt	imm
6 bits	5 bits	5 bits	16 bits

```
addi $s0, $s1, 5
```

8	$s1	$s0	5
8	17	16	5
001000	10001	10000	0000 0000 0000 0101

(0x22300005)

```
addi $t1, $s2, -12
```

8	$s2	$t1	-12
8	18	9	-12
001000	10010	01001	1111 1111 1111 0100

(0x2249FFF4)

Ví dụ mã máy của lệnh load và lệnh store

op	rs	rt	imm
6 bits	5 bits	5 bits	16 bits

```
lw $t0, 32($s3)
```

35	$s3	$t0	32
35	19	8	32
100011	10011	01000	0000 0000 0010 0000

(0x8E680020)

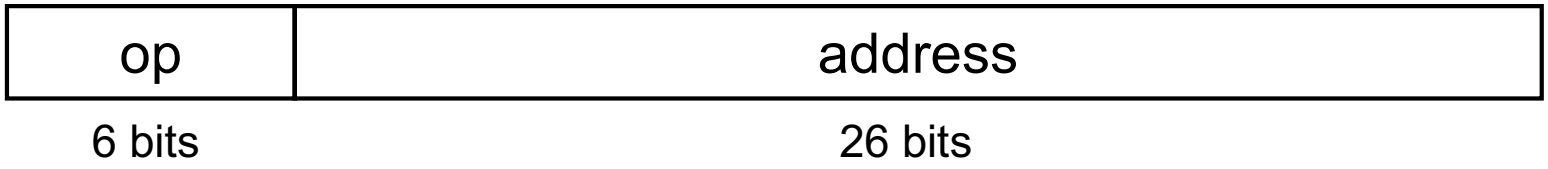
```
sw $s1, 4($t1)
```

43	$t1	$s1	4
43	9	17	4
101011	01001	10001	0000 0000 0000 0100

(0xAD310004)

Lệnh kiểu J (Jump)

- Toán hạng 26-bit địa chỉ
- Được sử dụng cho các lệnh nhảy
  - **`j (jump)`** → op = 000010
  - **`jal (jump and link)`** → op = 000011



5.4. Cơ bản về lập trình hợp ngữ

1. Các lệnh logic
2. Nạp hằng số vào thanh ghi
3. Tạo các cấu trúc điều khiển
4. Lập trình mảng dữ liệu
5. Chương trình con
6. Dữ liệu ký tự
7. Lệnh nhân và lệnh chia
8. Các lệnh với số dấu phẩy động

1. Các lệnh logic

- Các lệnh logic để thao tác trên các bit của dữ liệu

Phép toán logic	Toán tử trong C	Lệnh của MIPS
Shift left	<<	`sll`
Shift right	>>	`srl`
Bitwise AND	&	`and, andi`
Bitwise OR	|	`or, ori`
Bitwise XOR	^	`xor, xori`
Bitwise NOT	~	`nor`

Ví dụ lệnh logic kiểu R

Nội dung các thanh ghi nguồn

$s1	0100	0110	1010	0001	1100	0000	1011	0111
$s2	1111	1111	1111	1111	0000	0000	0000	0000

Mã hợp ngữ	Kết quả thanh ghi đích								
and $s3, $s1, $s2	$s3								
or $s4, $s1, $s2	$s4								
xor $s5, $s1, $s2	$s5								
nor $s6, $s1, $s2	$s6								

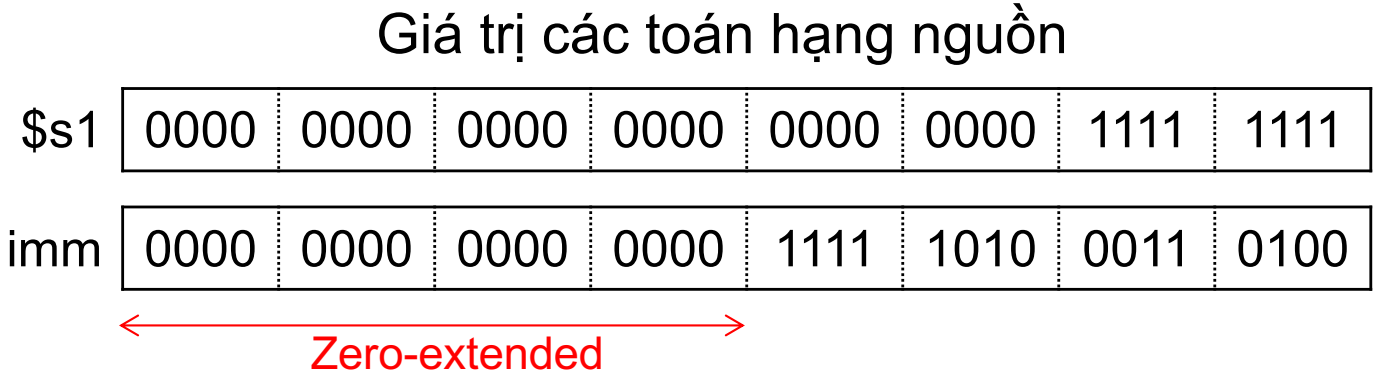
Ví dụ lệnh logic kiểu R

Nội dung các thanh ghi nguồn

$s1	0100	0110	1010	0001	1100	0000	1011	0111
$s2	1111	1111	1111	1111	0000	0000	0000	0000

Mã hợp ngữ		Kết quả thanh ghi đích							
and	$s3, $s1, $s2	$s3	0100	0110	1010	0001	0000	0000	0000	0000
or	$s4, $s1, $s2	$s4	1111	1111	1111	1111	1100	0000	1011	0111
xor	$s5, $s1, $s2	$s5	1011	1001	0101	1110	1100	0000	1011	0111
nor	$s6, $s1, $s2	$s6	0000	0000	0000	0000	0011	1111	0100	1000

Ví dụ lệnh logic kiểu I



Mã hợp ngữ		Kết quả thanh ghi đích
andi	$s2,$s1,0xFA34	$s2
ori	$s3,$s1,0xFA34	$s3
xori	$s4,$s1,0xFA34	$s4

Chú ý: Với các lệnh logic kiểu I, hằng số imm 16-bit được mở rộng thành 32-bit theo số không dấu (zero-extended)

Ví dụ lệnh logic kiểu I

Giá trị các toán hạng nguồn

$s1	0000	0000	0000	0000	0000	0000	1111	1111
imm	0000	0000	0000	0000	1111	1010	0011	0100

Zero-extended

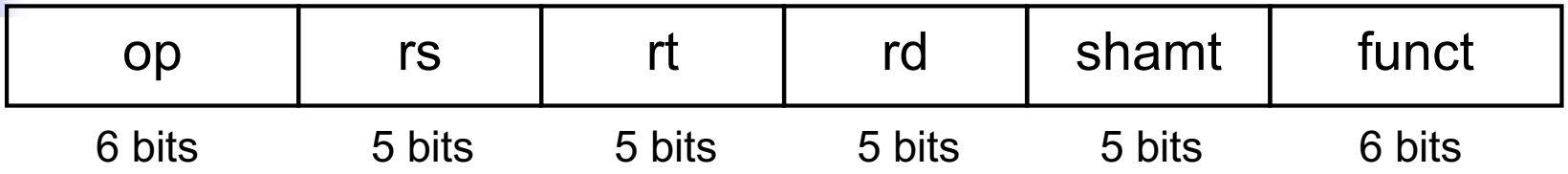
Mã hợp ngữ		Kết quả thanh ghi đích							
andi	$s2,$s1,0xFA34	$s2	0000	0000	0000	0000	0000	0000	0011	0100
ori	$s3,$s1,0xFA34	$s3	0000	0000	0000	0000	1111	1010	1111	1111
xori	$s4,$s1,0xFA34	$s4	0000	0000	0000	0000	1111	1010	1100	1011

Ý nghĩa của các phép toán logic

- Phép AND dùng để giữ nguyên một số bit trong word, xóa các bit còn lại về 0
- Phép OR dùng để giữ nguyên một số bit trong word, thiết lập các bit còn lại lên 1
- Phép XOR dùng để giữ nguyên một số bit trong word, đảo giá trị các bit còn lại
- Phép NOT dùng để đảo các bit trong word
  - Đổi 0 thành 1, và đổi 1 thành 0
  - MIPS không có lệnh NOT, nhưng có lệnh NOR với 3 toán hạng
  - a NOR b == NOT (a OR b)

```
nor  $t0, $t1, $zero   # $t0 = not($t1)
```

Lệnh logic dịch bit



- *shamt*: chỉ ra dịch bao nhiêu vị trí (shift amount)
- *rs*: không sử dụng, thiết lập = 00000
- Thanh ghi đích *rd* nhận giá trị thanh ghi nguồn *rt* đã được dịch trái hoặc dịch phải, *rt* không thay đổi nội dung
- **`sll`** - shift left logical (dịch trái logic)
  - Dịch trái các bit và điền các bit 0 vào bên phải
  - Dịch trái *i* bits là nhân với 2^i (nếu kết quả trong phạm vi biểu diễn 32-bit)
- **`srl`** - shift right logical (dịch phải logic)
  - Dịch phải các bit và điền các bit 0 vào bên trái
  - Dịch phải *i* bits là chia cho 2^i (chỉ với số nguyên không dấu)

Ví dụ lệnh dịch trái sll

Lệnh hợp ngữ:

```
sll $t2, $s0, 4   # $t2 = $s0 << 4
```

Mã máy:

op	rs	rt	rd	shamt	funct
0	0	16	10	4	0
000000	00000	10000	01010	00100	000000

(0x00105100)

Ví dụ kết quả thực hiện lệnh:

$s0	0000	0000	0000	0000	0000	0000	0000	1101	= 13
$t2	0000	0000	0000	0000	0000	0000	1101	0000	= 208 (13x16)

Chú ý: Nội dung thanh ghi $s0 không bị thay đổi

Ví dụ lệnh dịch phải srl

Lệnh hợp ngữ:

```
srl $s2, $s1, 2  # $s2 = $s1 >> 2
```

Mã máy:

op	rs	rt	rd	shamt	funct
0	0	17	18	2	2
000000	00000	10001	10010	00010	000010

(0x00119082)

Ví dụ kết quả thực hiện lệnh:

$s1	0000	0000	0000	0000	0000	0000	0101	0110	= 86
$s2	0000	0000	0000	0000	0000	0000	0001	0101	= 21

[86/4]

2. Nạp hằng số vào thanh ghi

- Trường hợp hằng số 16-bit → sử dụng lệnh addi:
  - Ví dụ: nạp hằng số 0x4F3C vào thanh ghi $s0:

```
addi $s0, $0, 0x4F3C  #$s0 = 0x4F3C
```

- Trong trường hợp hằng số 32-bit → sử dụng lệnh lui và lệnh ori:

```
lui rt, constant_hi16bit
```

  - Copy 16 bit cao của hằng số 32-bit vào 16 bit trái của rt
  - Xóa 16 bits bên phải của rt về 0

```
ori rt,rt,constant_low16bit
```

  - Đưa 16 bit thấp của hằng số 32-bit vào thanh ghi rt

Lệnh lui *(load upper immediate)*

op	rs	rt	imm
6 bits	5 bits	5 bits	16 bits

```
lui $s0, 0x21A0
```

15	0	$s0	0x21A0
15	0	16	0x21A0

Lệnh mã máy

001111	00000	10000	0010 0001 1010 0000

(0x3C1021A0)

Nội dung $s0 sau khi lệnh được thực hiện:

$s0	0010	0001	1010	0000	0000	0000	0000	0000

Ví dụ khởi tạo thanh ghi 32-bit

- Nạp vào thanh ghi $s0 giá trị 32-bit sau:

0010 0001 1010 0000 0100 0000 0011 1011 =0x21A0 403B

```
lui $s0,0x21A0          # nạp 0x21A0 vào nửa cao
                        # của thanh ghi $s0
ori $s0,$s0,0x403B      # nạp 0x403B vào nửa thấp
                        # của thanh ghi $s0
```

Nội dung $s0 sau khi thực hiện lệnh **lui**

$s0	0010	0001	1010	0000	0000	0000	0000	0000

or

	0000	0000	0000	0000	0100	0000	0011	1011

Nội dung $s0 sau khi thực hiện lệnh **ori**

$s0	0010	0001	1010	0000	0100	0000	0011	1011

3. Tạo các cấu trúc điều khiển

- Các cấu trúc rẽ nhánh
  - `if`
  - `if/else`
  - `switch/case`
- Các cấu trúc lặp
  - `while`
  - `do while`
  - `for`

Các lệnh rẽ nhánh và lệnh nhảy

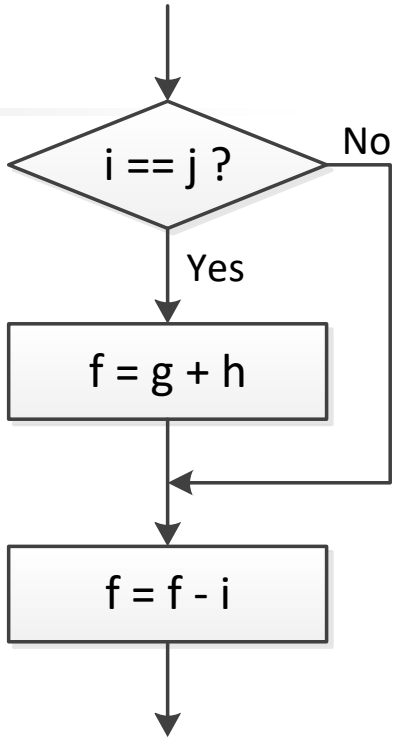
- Các lệnh rẽ nhánh: beq, bne
  - Rẽ nhánh đến lệnh được đánh nhãn nếu điều kiện là đúng, ngược lại, thực hiện tuần tự
  - **`beq rs, rt, L1`**
    - branch on equal
    - nếu (rs == rt) rẽ nhánh đến lệnh ở nhãn L1
  - **`bne rs, rt, L1`**
    - branch on not equal
    - nếu (rs != rt) rẽ nhánh đến lệnh ở nhãn L1
- Lệnh nhảy j
  - **`j L1`**
    - nhảy (jump) không điều kiện đến lệnh ở nhãn L1

Dịch câu lệnh if

- Mã C:

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

  - f, g, h, i, j ở $s0, $s1, $s2, $s3, $s4

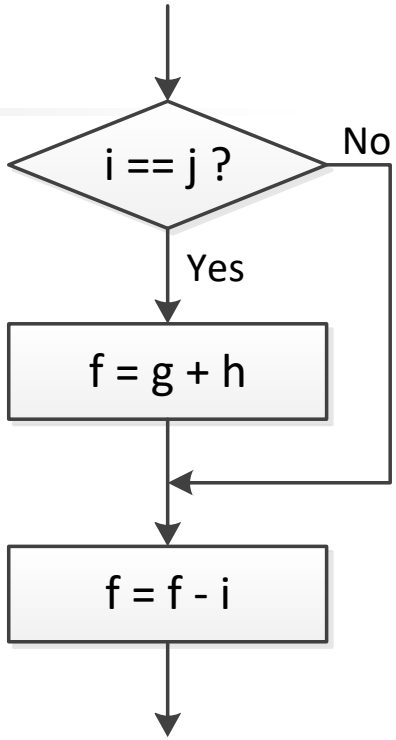


Dịch câu lệnh if

- Mã C:

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

  - f, g, h, i, j ở $s0, $s1, $s2, $s3, $s4



- Mã hợp ngữ MIPS:

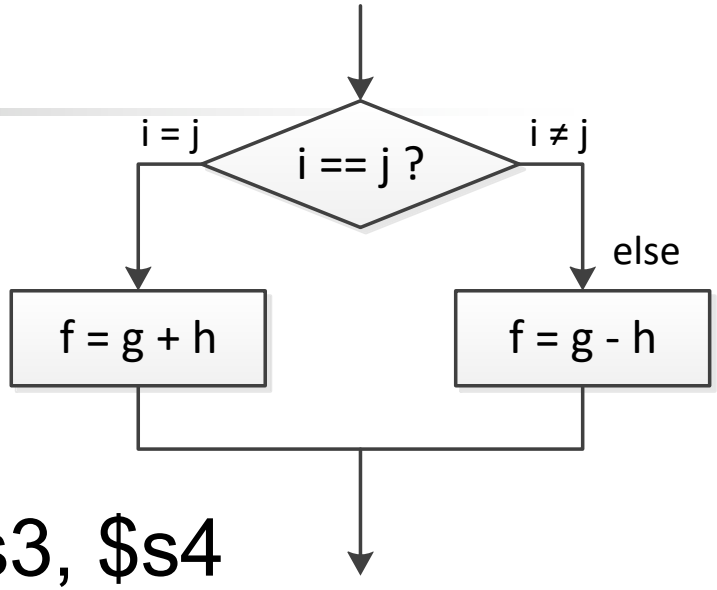
```
# $s0 = f, $s1 = g, $s2 = h
# $s3 = i, $s4 = j
    bne $s3, $s4, L1   # Nếu i=j
    add $s0, $s1, $s2  # thì f=g+h
L1: sub $s0, $s0, $s3  # f=f-i
```

Điều kiện hợp ngữ ngược với điều kiện của ngôn ngữ bậc cao

Dịch câu lệnh if/else

- Mã C:

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

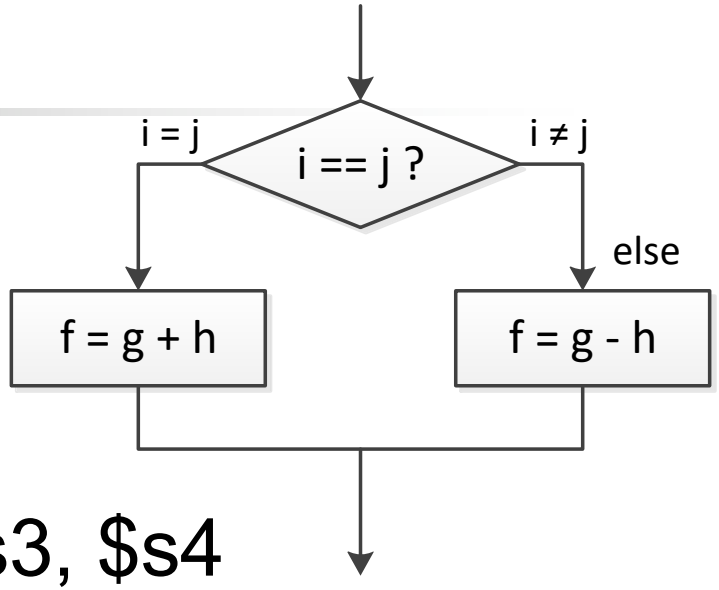


  - f, g, h, i, j ở $s0, $s1, $s2, $s3, $s4

Dịch câu lệnh if/else

- Mã C:

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



  - f, g, h, i, j ở $s0, $s1, $s2, $s3, $s4

- Mã hợp ngữ MIPS:

```
        bne $s3,$s4,Else   # Nếu i=j
        add $s0,$s1,$s2    # thì f=g+h
        j   Exit           # thoát
Else:   sub $s0,$s1,$s2    # nếu i<>j thì f=g-h
Exit:   …
```

Dịch câu lệnh switch/case

Mã C:

```
switch (amount) {
   case 20:   fee = 2; break;
   case 50:   fee = 3; break;
   case 100: fee = 5; break;
   default: fee = 0;
}
// tương đương với sử dụng các câu lệnh if/else
    if(amount = = 20) fee = 2;
    else if (amount = = 50) fee = 3;
    else if (amount = = 100) fee = 5;
    else fee = 0;
```

Dịch câu lệnh switch/case

Mã hợp ngữ MIPS

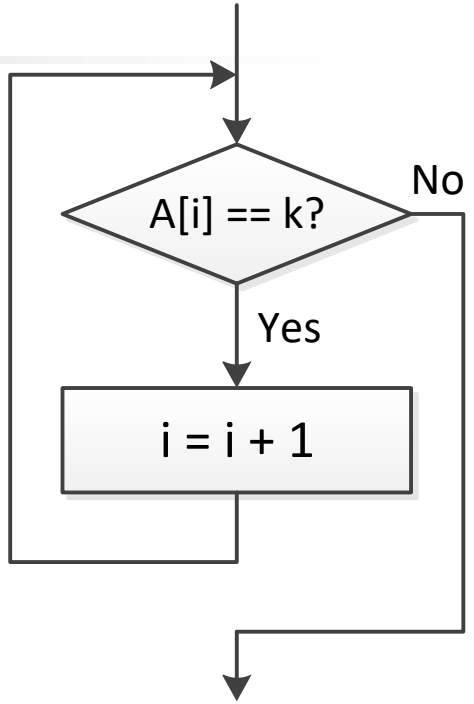
```
# $s0 = amount, $s1 = fee
case20:
    addi    $t0, $0, 20          # $t0 = 20
    bne     $s0, $t0, case50     # amount == 20? if not, skip to case50
    addi    $s1, $0, 2           # if so, fee = 2
    j       done                 # and break out of case
case50:
    addi    $t0, $0, 50          # $t0 = 50
    bne     $s0, $t0, case100    # amount == 50? if not, skip to case100
    addi    $s1, $0, 3           # if so, fee = 3
    j       done                 # and break out of case
case100:
    addi    $t0, $0, 100         # $t0 = 100
    bne     $s0, $t0, default    # amount == 100? if not, skip to default
    addi    $s1, $0, 5           # if so, fee = 5
    j       done                 # and break out of case
default:
   add      $s1 ,$0, $0          # fee = 0
done:
```

Dịch câu lệnh vòng lặp while

- Mã C:

```
while (A[i] == k)  i += 1;
```

  - i ở $s3, k ở $s5
  - địa chỉ cơ sở của mảng A ở $s6

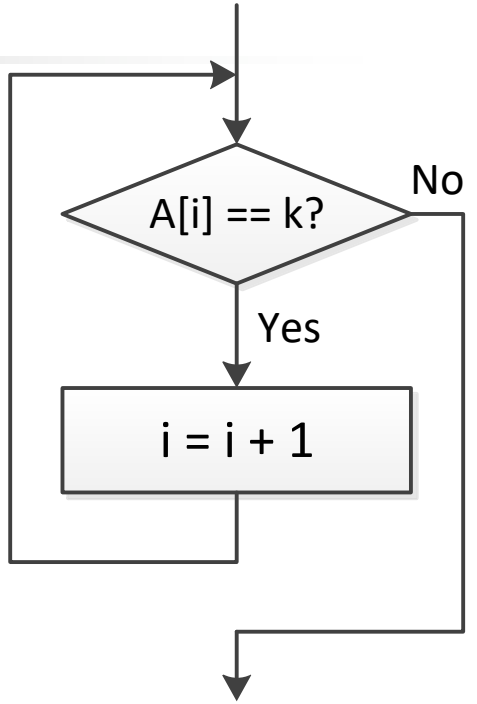


Dịch câu lệnh vòng lặp while

- Mã C:

```
while (A[i] == k)  i += 1;
```

  - i ở $s3, k ở $s5
  - địa chỉ cơ sở của mảng A ở $s6



- Mã hợp ngữ MIPS:

```
Loop: sll  $t1, $s3, 2     # $t1 = 4*i
      add  $t1, $t1, $s6   # $t1 = địa chỉ A[i]
      lw   $t0, 0($t1)     # $t0 = A[i]
      bne  $t0, $s5, Exit  # nếu A[i]<>k thì Exit
      addi $s3, $s3, 1     # nếu A[i]=k thì i=i+1
      j    Loop            # quay lại Loop
Exit: ...
```

Dịch câu lệnh vòng lặp for

- Mã C:

```
  // add the numbers from 0 to 9
int sum = 0;
int i;
for (i=0; i!=10; i++) {
  sum = sum + i;
}
```

Dịch câu lệnh vòng lặp for

- Mã C:

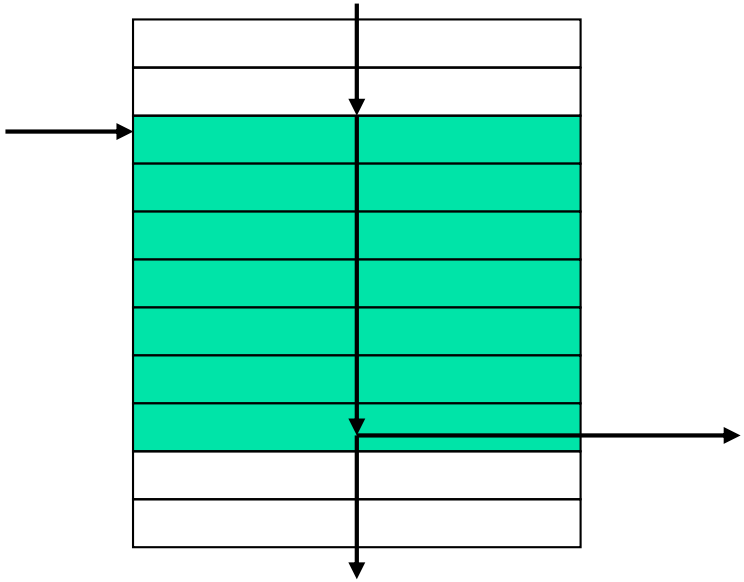
```
  // add the numbers from 0 to 9
int sum = 0;
int i;
for (i=0; i!=10; i++) {
  sum = sum + i;
}
```

- Mã hợp ngữ MIPS:

```
# $s0 = i, $s1 = sum
      addi  $s1, $0, 0       # sum = 0
      add   $s0, $0, $0      # i = 0
      addi  $t0, $0, 10      # $t0 = 10
for:  beq   $s0, $t0, done   # Nếu i=10, thoát
      add   $s1, $s1, $s0    # Nếu i<10 thì sum = sum+i
      addi  $s0, $s0, 1      # tăng i thêm 1
      j     for              # quay lại for
done: ...
```

Khối lệnh cơ sở (basic block)

- Khối lệnh cơ sở là dãy các lệnh với
  - Không có lệnh rẽ nhánh nhúng trong đó (ngoại trừ ở cuối)
  - Không có đích rẽ nhánh tới (ngoại trừ ở vị trí đầu tiên)



- Chương trình dịch xác định khối cơ sở để tối ưu hóa
- Các bộ xử lý tiên tiến có thể tăng tốc độ thực hiện khối cơ sở

Thêm các lệnh thao tác điều kiện

- Lệnh slt (set on less than)

  `slt  rd, rs, rt`

  - Nếu (rs < rt) thì rd = 1; ngược lại rd = 0;

- Lệnh slti

  `slti rt, rs, constant`

  - Nếu (rs < constant) thì rt = 1; ngược lại rt = 0;

- Sử dụng kết hợp với các lệnh beq, bne

```
    slt $t0, $s1, $s2    # nếu ($s1 < $s2)
    bne $t0, $zero, L1   # rẽ nhánh đến L1
    ...
L1:
```

So sánh số có dấu và không dấu

- So sánh số có dấu: **`slt, slti`**
- So sánh số không dấu: **`sltu, sltiu`**
- Ví dụ
  - $s0 = 1111 1111 1111 1111 1111 1111 1111 1111
  - $s1 = 0000 0000 0000 0000 0000 0000 0000 0001
  - **`slt  $t0, $s0, $s1  # signed`**
    - –1 < +1 → $t0 = 1
  - **`sltu $t0, $s0, $s1  # unsigned`**
    - +4,294,967,295 > +1 → $t0 = 0

Ví dụ sử dụng lệnh slt

- Mã C

```
int sum = 0;
int i;

for (i=1; i < 101; i = i*2) {
  sum = sum + i;
}
```

Ví dụ sử dụng lệnh slt

- Mã hợp ngữ MIPS

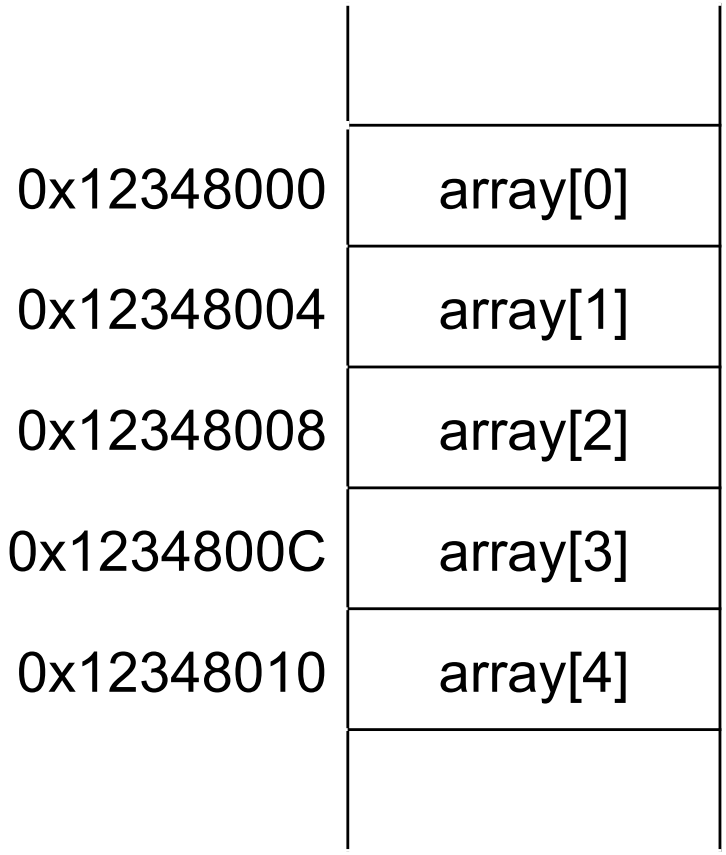
```
# $s0 = i, $s1 = sum
        addi $s1, $0, 0        # sum = 0
        addi $s0, $0, 1        # i = 1
        addi $t0, $0, 101      # t0 = 101
loop:   slt  $t1, $s0, $t0     # Nếu i>= 101
        beq  $t1, $0, done     # thì thoát
        add  $s1, $s1, $s0     # nếu i<101 thì sum=sum+i
        sll  $s0, $s0, 1       # i= 2*i
        j    loop              # lặp lại
done:
```

4. Lập trình với mảng dữ liệu

- Truy cập số lượng lớn các dữ liệu cùng loại
- Chỉ số (Index): truy cập từng phần tử của mảng
- Kích cỡ (Size): số phần tử của mảng

Ví dụ về mảng

- Mảng 5-phần tử, mỗi phần tử có độ dài 32-bit, chiếm 4 byte trong bộ nhớ
- Địa chỉ cơ sở = 0x12348000 (địa chỉ của phần tử đầu tiên của mảng array[0])
- Bước đầu tiên để truy cập mảng: nạp địa chỉ cơ sở vào thanh ghi



Ví dụ truy cập các phần tử

- Mã C

```
int array[5];
array[0] = array[0] * 2;
array[1] = array[1] * 2;
```

- Mã hợp ngữ MIPS

```
# nạp địa chỉ cơ sở của mảng vào $s0
   lui  $s0, 0x1234          # 0x1234 vào nửa cao của $S0
   ori  $s0, $s0, 0x8000     # 0x8000 vào nửa thấp của $s0

   lw   $t1, 0($s0)          # $t1 = array[0]
   sll  $t1, $t1, 1          # $t1 = $t1 * 2
   sw   $t1, 0($s0)          # array[0] = $t1

   lw   $t1, 4($s0)          # $t1 = array[1]
   sll  $t1, $t1, 1          # $t1 = $t1 * 2
   sw   $t1, 4($s0)          # array[1] = $t1
```

Ví dụ vòng lặp truy cập mảng dữ liệu

- Mã C

```
int array[1000];
int i;

for (i=0; i < 1000; i = i + 1)
    array[i] = array[i] * 8;

// giả sử địa chỉ cơ sở của mảng = 0x23b8f000
```

- Mã hợp ngữ MIPS

```
# $s0 = array base address (0x23b8f000), $s1 = i
```

Ví dụ vòng lặp truy cập mảng dữ liệu (tiếp)

Mã hợp ngữ MIPS

```
# $s0 = array base address (0x23b8f000), $s1 = i
# khởi tạo các thanh ghi
      lui  $s0, 0x23b8          # $s0 = 0x23b80000
      ori  $s0, $s0, 0xf000     # $s0 = 0x23b8f000
      addi $s1, $0, 0           # i = 0
      addi $t2, $0, 1000        # $t2 = 1000
# vòng lặp
loop: slt  $t0, $s1, $t2        # i < 1000?
      beq  $t0, $0, done        # if not then done
      sll  $t0, $s1, 2          # $t0 = i*4
      add  $t0, $t0, $s0        # address of array[i]
      lw   $t1, 0($t0)          # $t1 = array[i]
      sll  $t1, $t1, 3          # $t1 = array[i]*8
      sw   $t1, 0($t0)          # array[i] = array[i]*8
      addi $s1, $s1, 1          # i = i + 1
      j    loop                 # repeat
done:
```

5. Chương trình con - thủ tục

- Các bước yêu cầu:
  1. Đặt các tham số vào các thanh ghi
  2. Chuyển điều khiển đến thủ tục
  3. Thực hiện các thao tác của thủ tục
  4. Đặt kết quả vào thanh ghi cho chương trình đã gọi thủ tục
  5. Trở về vị trí đã gọi

Sử dụng các thanh ghi

- $a0 – $a3: các tham số vào (các thanh ghi 4 – 7)
- $v0, $v1: các kết quả ra (các thanh ghi 2 và 3)
- $t0 – $t9: các giá trị tạm thời
  - Có thể được ghi lại bởi thủ tục được gọi
- $s0 – $s7: cất giữ các biến
  - Cần phải cất/khôi phục bởi thủ tục được gọi
- $gp: global pointer - con trỏ toàn cục cho dữ liệu tĩnh (thanh ghi 28)
- $sp: stack pointer - con trỏ ngăn xếp (thanh ghi 29)
- $fp: frame pointer - con trỏ khung (thanh ghi 30)
- $ra: return address - địa chỉ trở về (thanh ghi 31)

Các lệnh gọi thủ tục

- Gọi thủ tục: jump and link

```
jal ProcedureAddress
```

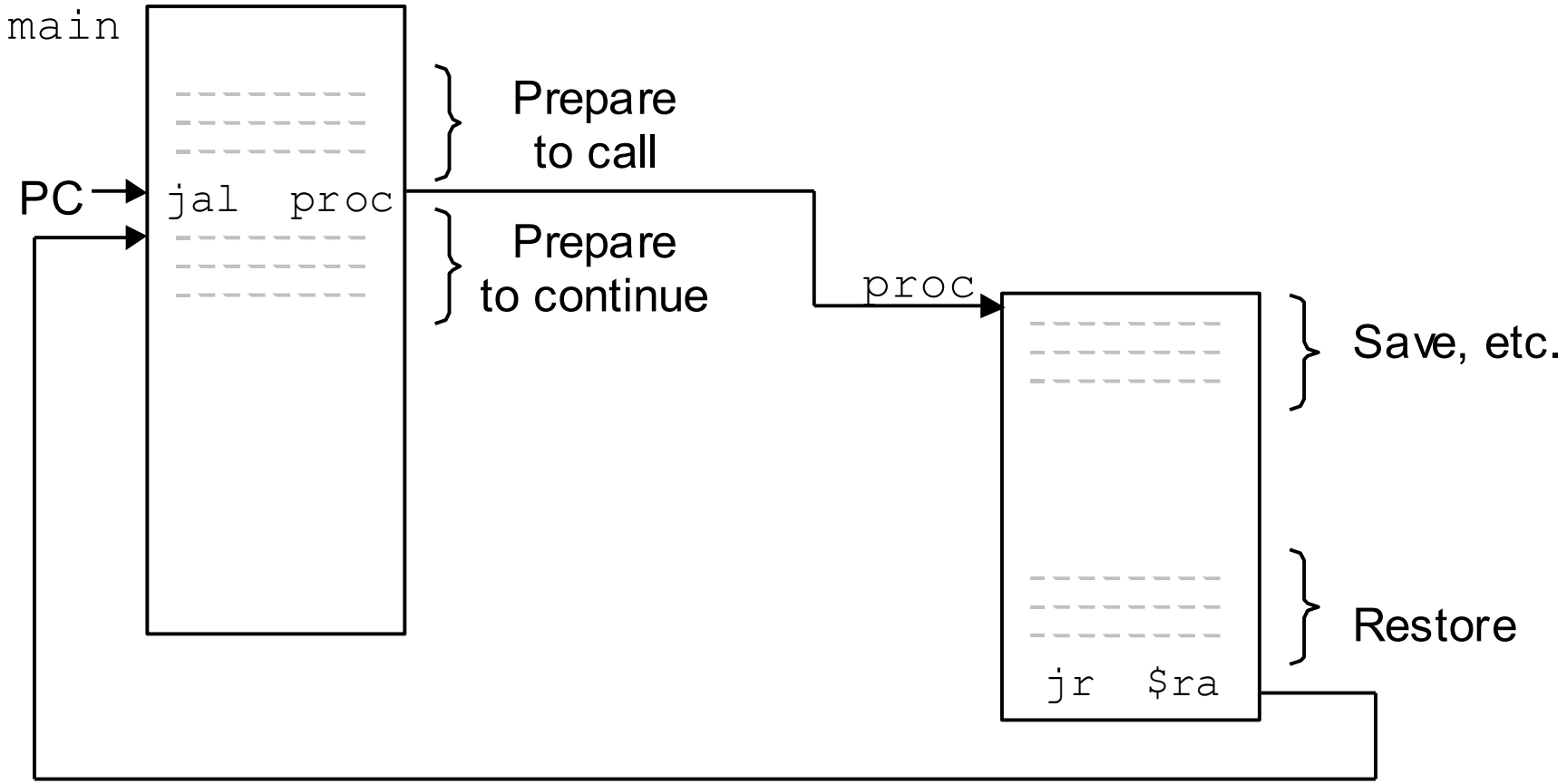
  - Địa chỉ của lệnh kế tiếp (địa chỉ trở về) được cất ở thanh ghi $ra
  - Nhảy đến địa chỉ của thủ tục

- Trở về từ thủ tục: jump register

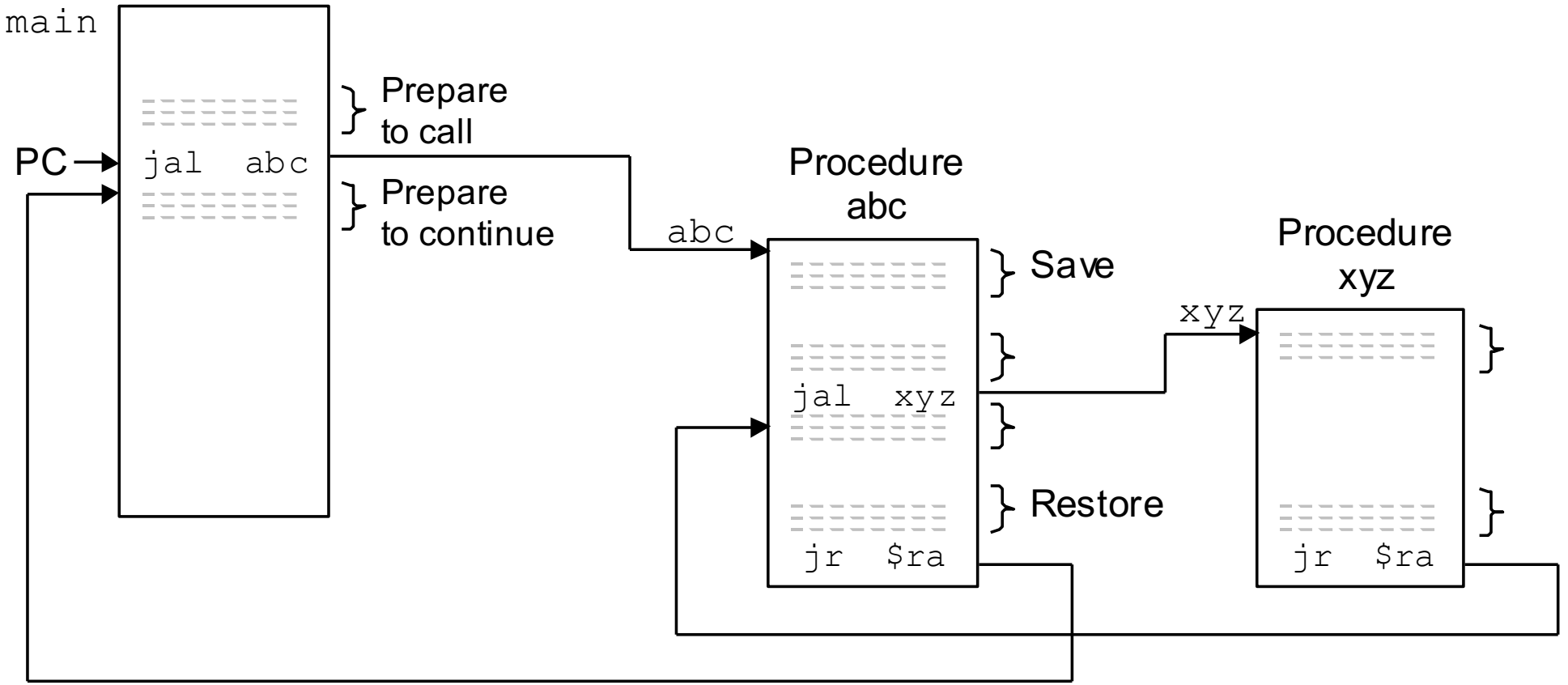
```
jr $ra
```

  - Copy nội dung thanh ghi $ra (đang chứa địa chỉ trở về) trả lại cho bộ đếm chương trình PC

Minh họa gọi Thủ tục



Gọi thủ tục lồng nhau



Ví dụ Thủ tục lá

- Thủ tục lá là thủ tục không có lời gọi thủ tục khác
- Mã C:

```
int leaf_example (int g, h, i, j)
{
    int f;
    f = (g + h) - (i + j);
    return f;
}
```

  - Các tham số g, h, i, j ở $a0, $a1, $a2, $a3
  - f ở $s0 (do đó, cần cất $s0 ra ngăn xếp)
  - $t0 và $t1 được thủ tục dùng để chứa các giá trị tạm thời, cũng cần cất trước khi sử dụng
  - Kết quả ở $v0

Mã hợp ngữ MIPS

```
leaf_example:
        addi   $sp, $sp, -12      # tạo 3 vị trí ở stack
        sw     $t1, 8($sp)        # cất nội dung $t1
        sw     $t0, 4($sp)        # cất nội dung $t0
        sw     $s0, 0($sp)        # cất nội dung $s0
        add    $t0, $a0, $a1      # $t0 = g+h
        add    $t1, $a2, $a3      # $t1 = i+j
        sub    $s0, $t0, $t1      # $s0 = (g+h)-(i+j)
        add    $v0, $s0, $zero    # trả kết quả sang $v0
        lw     $s0, 0($sp)        # khôi phục $s0
        lw     $t0, 4($sp)        # khôi phục $t0
        lw     $t1, 8($sp)        # khôi phục $t1
        addi   $sp, $sp, 12       # xóa 3 mục ở stack
        jr     $ra                # trở về nơi đã gọi
```

Ví dụ Thủ tục cành

- Là thủ tục có gọi thủ tục khác
- Mã C:

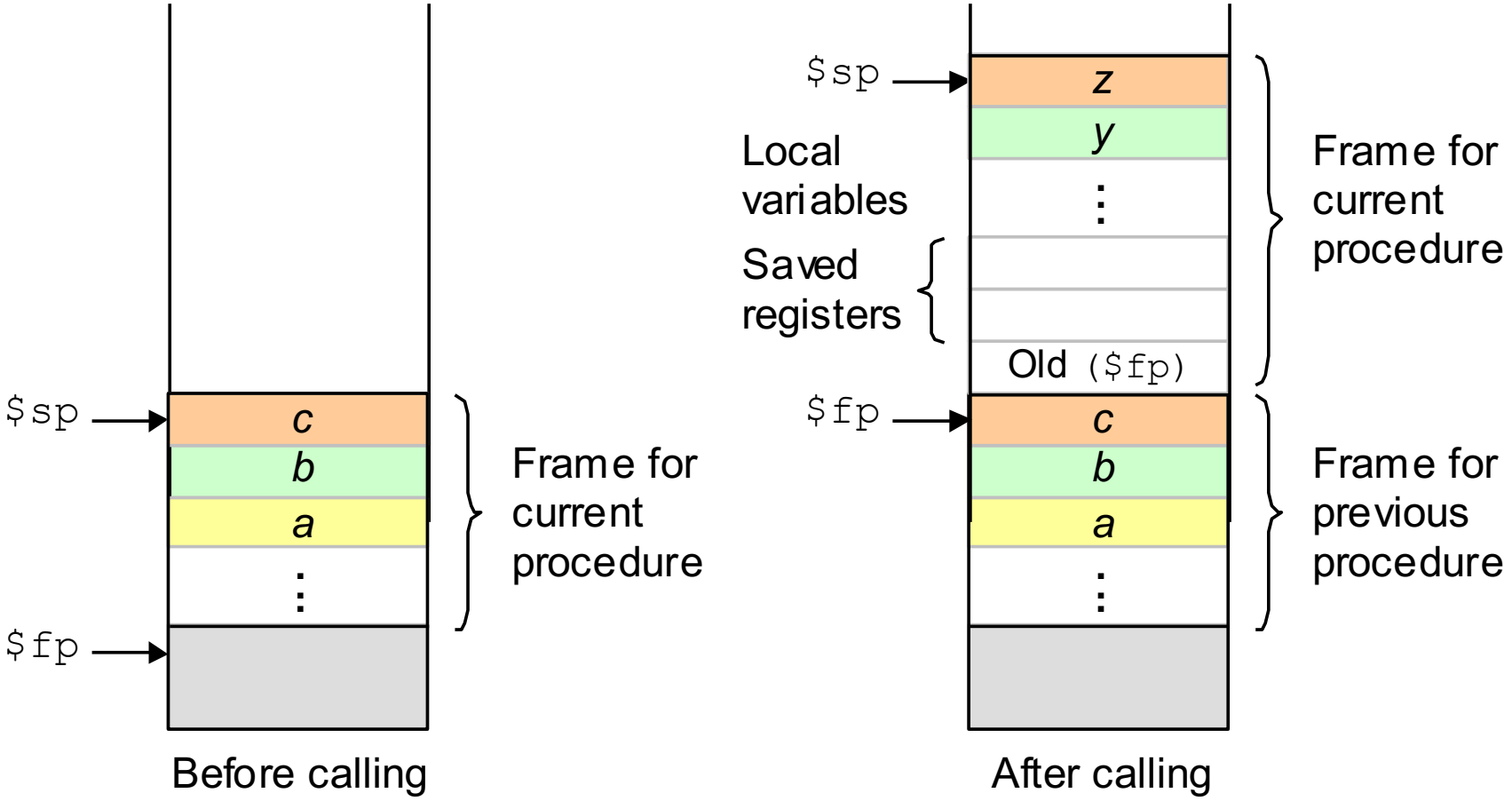
```
int fact (int n)
{
  if (n < 1) return (1);
  else return n * fact(n - 1);
}
```

  - Tham số n ở $a0
  - Kết quả ở $v0

Mã hợp ngữ MIPS

```
fact:
      addi   $sp, $sp, -8       # dành stack cho 2 mục
      sw     $ra, 4($sp)        # cất địa chỉ trở về
      sw     $a0, 0($sp)        # cất tham số n
      slti   $t0, $a0, 1        # kiểm tra n < 1
      beq    $t0, $zero, L1
      addi   $v0, $zero, 1      # nếu đúng, kết quả là 1
      addi   $sp, $sp, 8        # lấy 2 mục từ stack
      jr     $ra                # và trở về
L1:   addi   $a0, $a0, -1       # nếu không, giảm n
      jal    fact               # gọi đệ qui
      lw     $a0, 0($sp)        # khôi phục n ban đầu
      lw     $ra, 4($sp)        # và địa chỉ trở về
      addi   $sp, $sp, 8        # lấy 2 mục từ stack
      mul    $v0, $a0, $v0      # nhân để nhận kết quả
      jr     $ra                # và trở về
```

Sử dụng Stack khi gọi thủ tục



6. Dữ liệu ký tự

- Các tập ký tự được mã hóa theo byte
  - ASCII: 128 ký tự
    - 95 ký thị hiển thị , 33 mã điều khiển
  - Latin-1: 256 ký tự
    - ASCII và các ký tự mở rộng
- Unicode: Tập ký tự 32-bit
  - Được sử dụng trong Java, C++, …
  - Hầu hết các ký tự của các ngôn ngữ trên thế giới và các ký hiệu

Các thao tác với Byte/Halfword

- Có thể sử dụng các phép toán logic
- Nạp/Lưu byte/halfword trong MIPS

- `lb rt, offset(rs)` `lh rt, offset(rs)`
  - Mở rộng theo số có dấu thành 32 bits trong rt

- `lbu rt, offset(rs)` `lhu rt, offset(rs)`
  - Mở rộng theo số không dấu thành 32 bits trong rt

- `sb rt, offset(rs)` `sh rt, offset(rs)`
  - Chỉ lưu byte/halfword bên phải

Ví dụ copy String

- Mã C:

```
void strcpy (char x[], char y[])
{ int i;
  i = 0;
  while ((x[i]=y[i])!='\0')
    i += 1;
}
```

  - Các địa chỉ của x, y ở $a0, $a1
  - i ở $s0

Ví dụ Copy String

- Mã hợp ngữ MIPS

```
strcpy:
    addi $sp, $sp, -4       # adjust stack for 1 item
    sw   $s0, 0($sp)        # save $s0
    add  $s0, $zero, $zero  # i = 0
L1: add  $t1, $s0, $a1      # addr of y[i] in $t1
    lbu  $t2, 0($t1)        # $t2 = y[i]
    add  $t3, $s0, $a0      # addr of x[i] in $t3
    sb   $t2, 0($t3)        # x[i] = y[i]
    beq  $t2, $zero, L2     # exit loop if y[i] == 0
    addi $s0, $s0, 1        # i = i + 1
    j    L1                 # next iteration of loop
L2: lw   $s0, 0($sp)        # restore saved $s0
    addi $sp, $sp, 4        # pop 1 item from stack
    jr   $ra                # and return
```

7. Các lệnh nhân và chia số nguyên

- MIPS có hai thanh ghi 32-bit: HI (high) và LO (low)
- Các lệnh liên quan:
  - `mult rs, rt  # nhân số nguyên có dấu`
  - `multu rs, rt # nhân số nguyên không dấu`
    - Tích 64-bit nằm trong cặp thanh ghi HI/LO
  - `div rs, rt   # chia số nguyên có dấu`
  - `divu rs, rt  # chia số nguyên không dấu`
    - HI: chứa phần dư, LO: chứa thương
  - `mfhi rd      # Move from Hi to rd`
  - `mflo rd      # Move from LO to rd`

8. Các lệnh với số dấu phẩy động (FP)

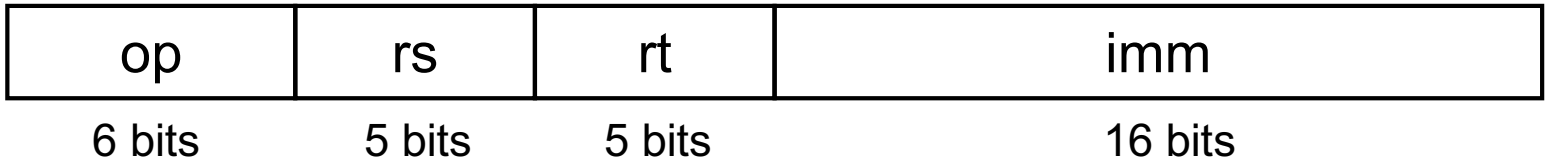
- Các thanh ghi số dấu phẩy động
  - 32 thanh ghi 32-bit (single-precision): $f0, $f1, … $f31
  - Cặp đôi để chứa dữ liệu dạng 64-bit (double-precision): $f0/$f1, $f2/$f3, …
- Các lệnh số dấu phẩy động chỉ thực hiện trên các thanh ghi số dấu phẩy động
- Lệnh load và store với FP
  - **`lwc1, ldc1, swc1, sdc1`**
    - **`Ví dụ:  ldc1 $f8, 32($s2)`**

Các lệnh với số dấu phẩy động

- Các lệnh số học với số FP 32-bit (single-precision)
  - **`add.s, sub.s, mul.s, div.s`**
  - **`VD:  add.s $f0, $f1, $f6`**
- Các lệnh số học với số FP 64-bit (double-precision)
  - **`add.d, sub.d, mul.d, div.d`**
  - **`VD:  mul.d $f4, $f4, $f6`**
- Các lệnh so sánh
  - **`c.xx.s, c.xx.d`** (trong đó *xx* là eq, lt, le, …)
  - Thiết lập hoặc xóa các bit mã điều kiện
  - **`VD:  c.lt.s $f3, $f4`**
- Các lệnh rẽ nhánh dựa trên mã điều kiện
  - **`bc1t, bc1f`**
  - **`VD:  bc1t TargetLabel`**

5.5. Các phương pháp định địa chỉ của MIPS

- Các lệnh Branch chỉ ra:
  - Mã thao tác, hai thanh ghi, hằng số
- Hầu hết các đích rẽ nhánh là rẽ nhánh gần
  - Rẽ xuôi hoặc rẽ ngược



- Định địa chỉ tương đối với PC
  - PC-relative addressing
  - Địa chỉ đích = PC + hằng số imm × 4
  - Chú ý: trước đó PC đã được tăng lên
  - Hằng số imm 16-bit có giá trị trong dải $[-2^{15} , +2^{15} - 1]$

Lệnh beq, bne

op	rs	rt	imm
6 bits	5 bits	5 bits	16 bits

```
beq $s0, $s1, Exit
bne $s0, $s1, Exit
```

4 or 5	16	17	Exit

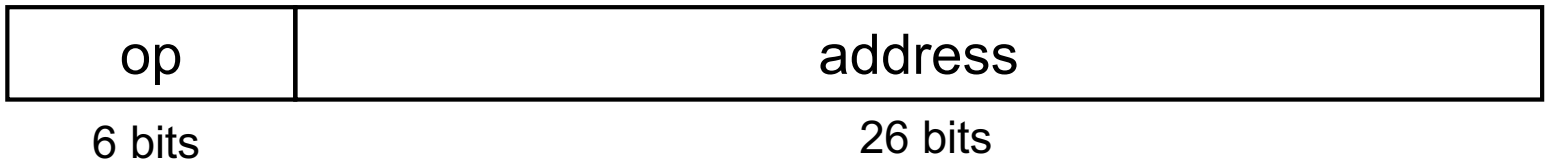
khoảng cách tương đối tính theo word

Lệnh mã máy

beq	000100	10000	10001	0000 0000 0000 0110
bne	000101	10000	10001	0000 0000 0000 0110

Địa chỉ hóa cho lệnh Jump

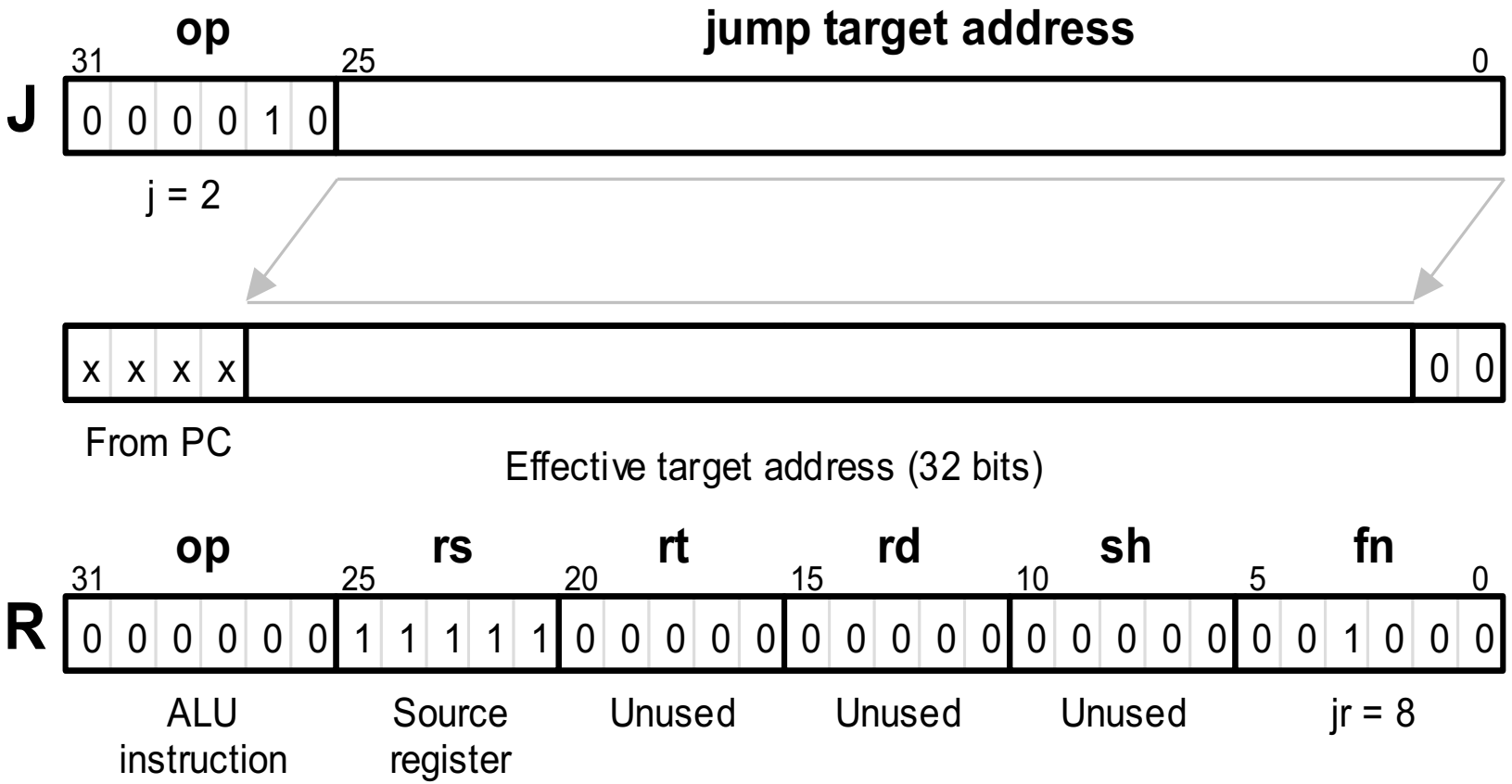
- Đích của lệnh Jump (j và jal) có thể là bất kỳ chỗ nào trong chương trình
  - Cần mã hóa đầy đủ địa chỉ trong lệnh



- Định địa chỉ nhảy (giả) trực tiếp (Pseudo)Direct jump addressing
  - Địa chỉ đích = $PC_{31...28}$: (address × 4)

Ví dụ mã lệnh j và jr

```
j    L1        # nhảy đến vị trí có nhãn L1
jr   $ra       # nhảy đến vị trí có địa chỉ ở $ra;
               # $ra chứa địa chỉ trở về
```



Ví dụ mã hóa lệnh

```
Loop: sll  $t1, $s3, 2
      add  $t1, $t1, $s6
      lw   $t0, 0($t1)
      bne  $t0, $s5, Exit
      addi $s3, $s3, 1
      j    Loop
Exit: …
```

0x80000	0	0	19	9	2	0
0x80004	0	9	22	9	0	32
0x80008	35	9	8	0		
0x8000C	5	8	21	2		
0x80010	8	19	19	1		
0x80014	2	0x20000				
0x80018						

Rẽ nhánh xa

- Nếu đích rẽ nhánh là quá xa để mã hóa với offset 16-bit, assembler sẽ viết lại code
- Ví dụ

```
        beq $s0, $s1, L1
        (lệnh kế tiếp)
        ...
  L1:
```

sẽ được thay bằng đoạn lệnh sau:

```
        bne $s0, $s1, L2
        j L1
  L2:   (lệnh kế tiếp)
        ...
  L1:
```

Tóm tắt về các phương pháp định địa chỉ

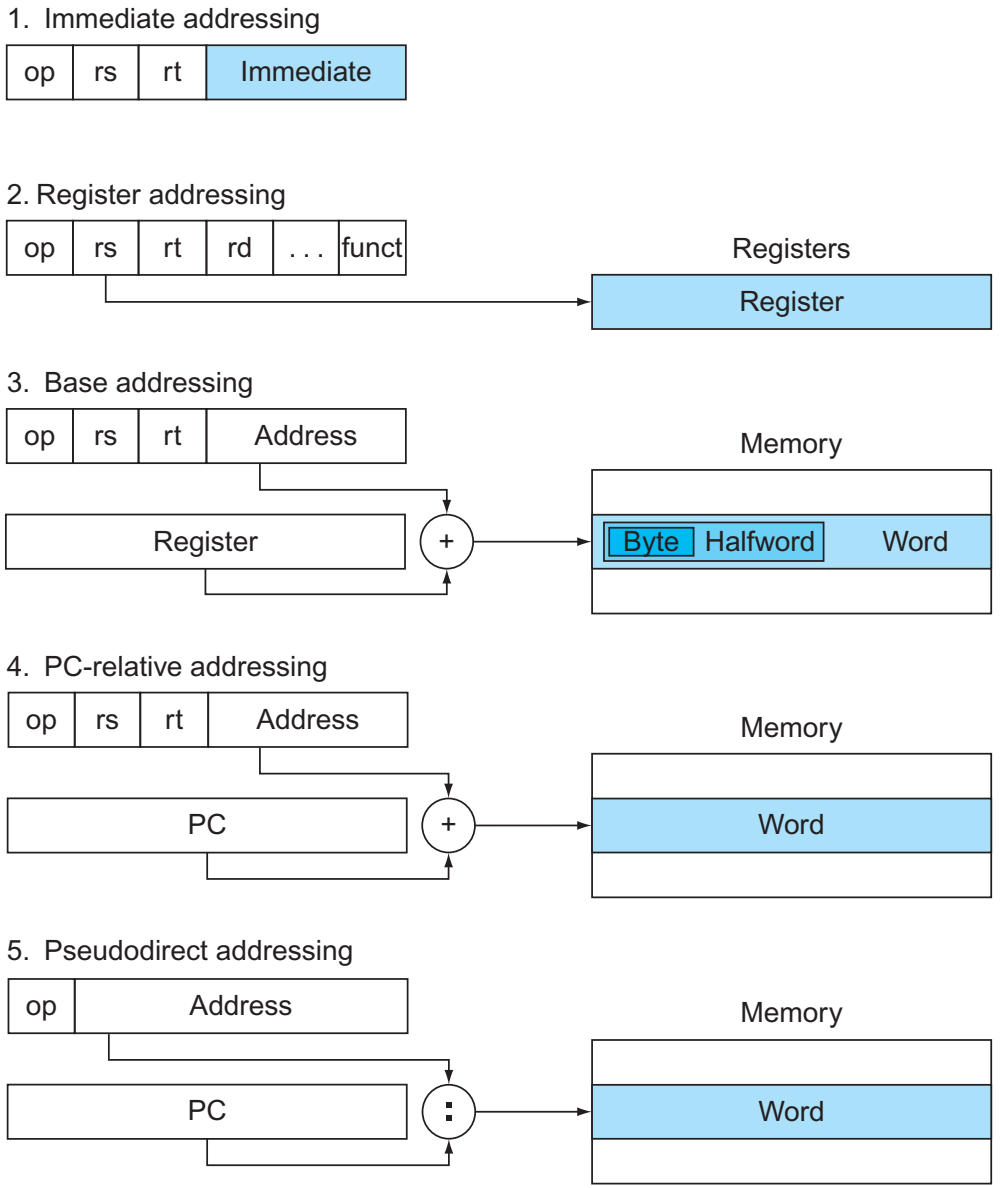
1. Định địa chỉ tức thì

2. Định địa chỉ thanh ghi

3. Định địa chỉ cơ sở

4. Định địa chỉ tương đối với PC

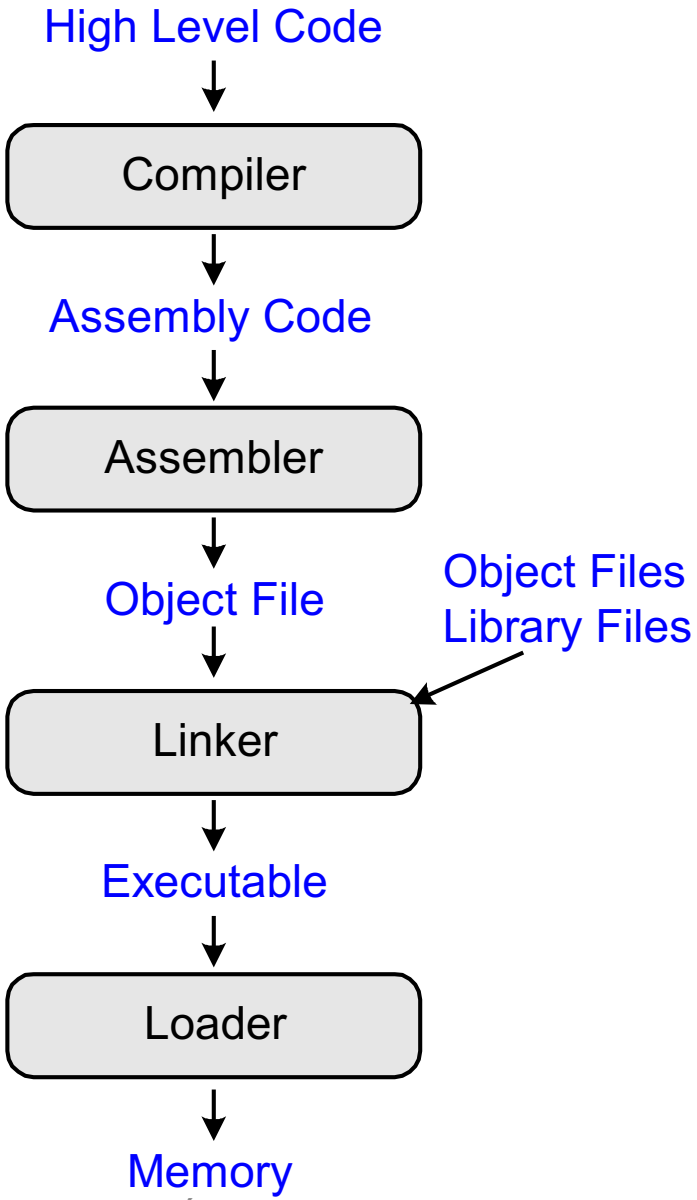
5. Định địa chỉ giả trực tiếp



5.6. Dịch và chạy chương trình hợp ngữ

- Các phần mềm lập trình hợp ngữ MIPS:
  - MARS
  - MipsIt
  - QtSpim
- MIPS Reference Data

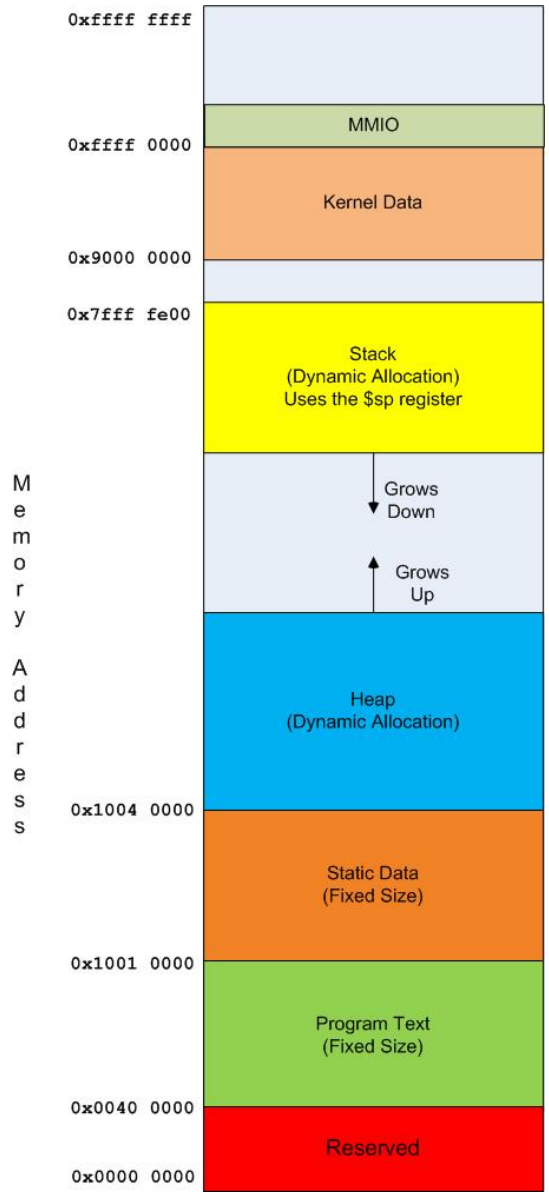
Dịch và chạy ứng dụng



Chương trình trong bộ nhớ

- Các lệnh (instructions)
- Dữ liệu
  - Toàn cục/tĩnh: được cấp phát trước khi chương trình bắt đầu thực hiện
  - Động: được cấp phát trong khi chương trình thực hiện
- Bộ nhớ:
  - 2^{32} bytes = 4 GiB
  - Địa chỉ từ 0x00000000 đến 0xFFFFFFFF

Bản đồ bộ nhớ của MIPS



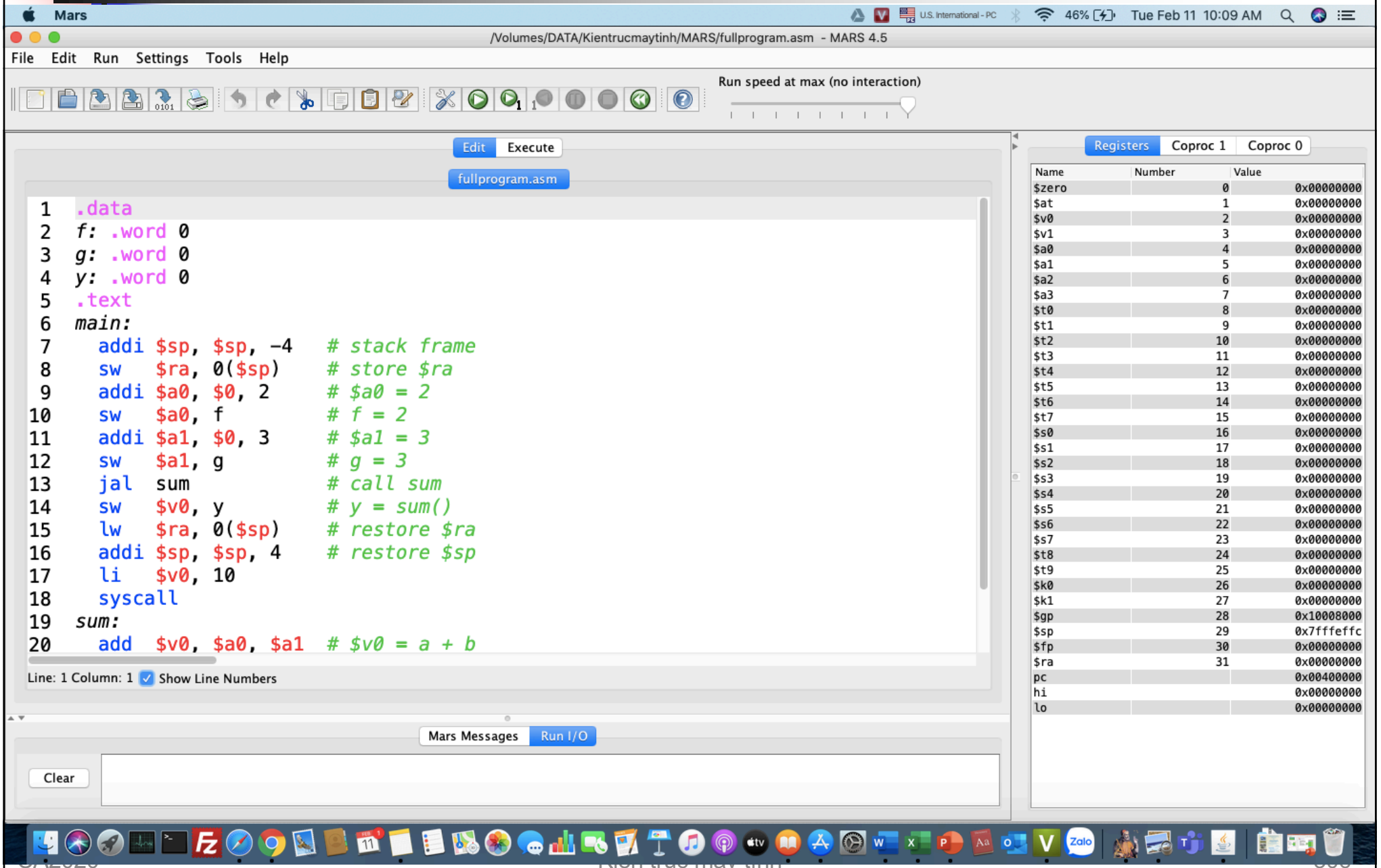
Ví dụ: Mã C

```
int f, g, y;  // global
  variables

int main(void)
{
  f = 2;
  g = 3;
  y = sum(f, g);
  return y;
}

int sum(int a, int b) {
  return (a + b);
}
```

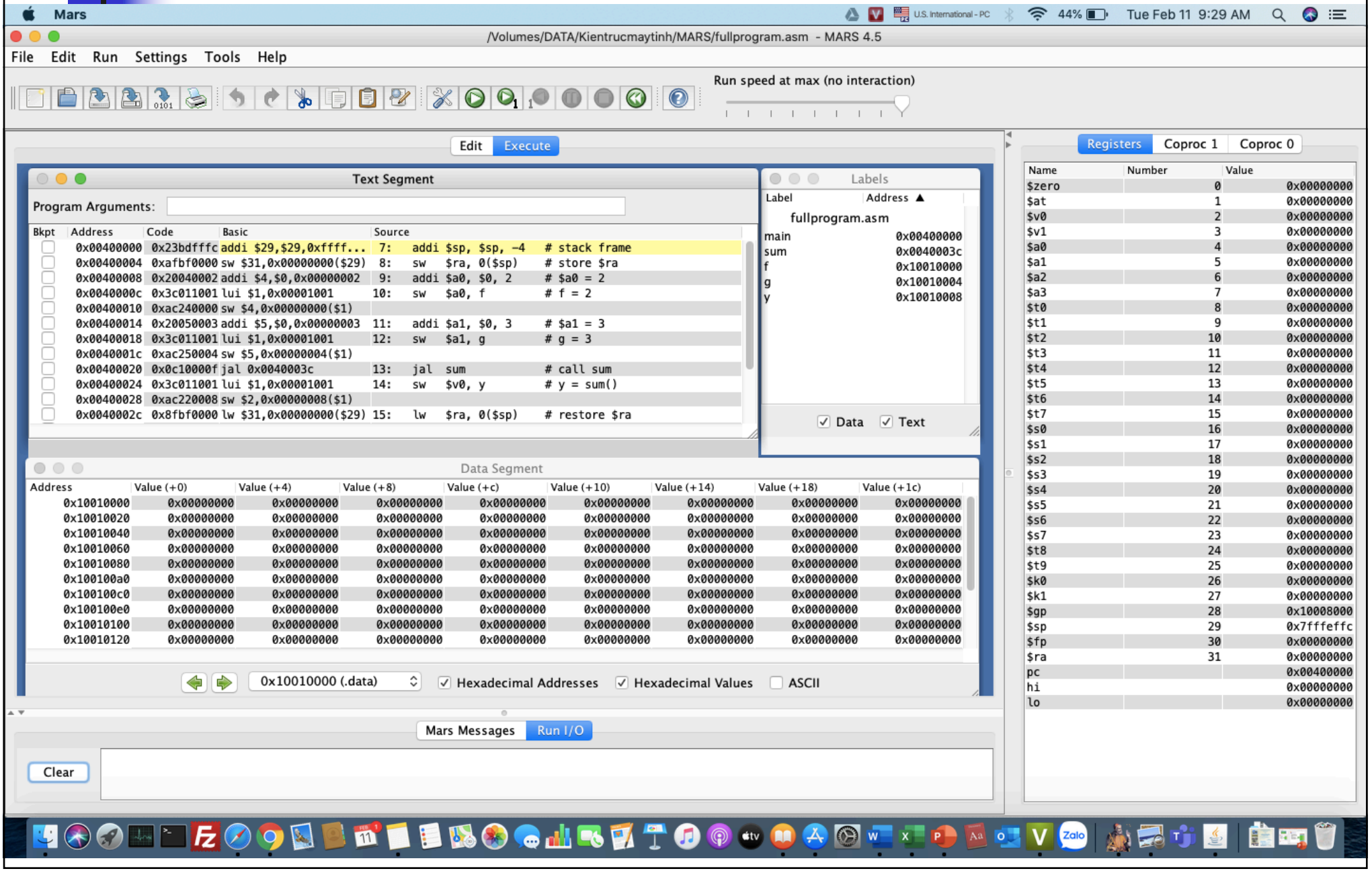
Ví dụ chương trình hợp ngữ MIPS

```
.data
f: .word 0
g: .word 0
y: .word 0
.text
main:
  addi $sp, $sp, -4    # stack frame
  sw   $ra, 0($sp)     # store $ra
  addi $a0, $0, 2      # $a0 = 2
  sw   $a0, f          # f = 2
  addi $a1, $0, 3      # $a1 = 3
  sw   $a1, g          # g = 3
  jal  sum             # call sum
  sw   $v0, y          # y = sum()
  lw   $ra, 0($sp)     # restore $ra
  addi $sp, $sp, 4     # restore $sp
  li   $v0, 10
  syscall
sum:
  add  $v0, $a0, $a1   # $v0 = a + b
  jr   $ra             # return
```

Viết chương trình trên phần mềm MARS



Thực thi chương trình trên MARS



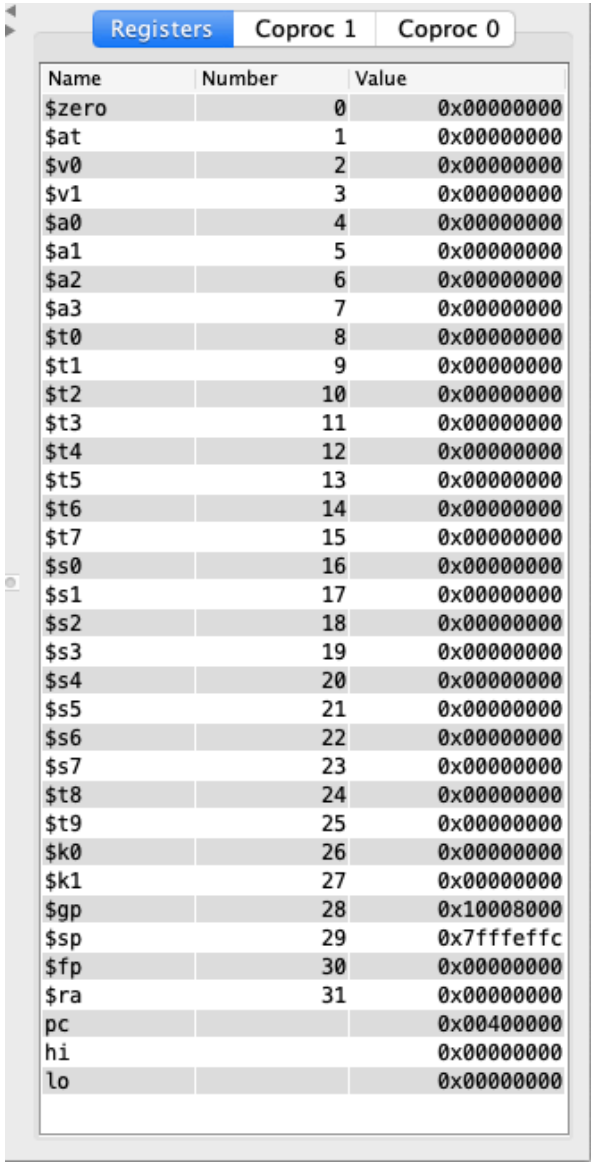
Vùng nhớ lệnh

Text Segment

Program Arguments:

Bkpt	Address	Code	Basic	Source
	0x00400000	0x23bdfffc	addi $29,$29,0xfffffffc	7: addi $sp, $sp, -4 # stack frame
	0x00400004	0xafbf0000	sw $31,0x00000000($29)	8: sw $ra, 0($sp) # store $ra
	0x00400008	0x20040002	addi $4,$0,0x00000002	9: addi $a0, $0, 2 # $a0 = 2
	0x0040000c	0x3c011001	lui $1,0x00001001	10: sw $a0, f # f = 2
	0x00400010	0xac240000	sw $4,0x00000000($1)	
	0x00400014	0x20050003	addi $5,$0,0x00000003	11: addi $a1, $0, 3 # $a1 = 3
	0x00400018	0x3c011001	lui $1,0x00001001	12: sw $a1, g # g = 3
	0x0040001c	0xac250004	sw $5,0x00000004($1)	
	0x00400020	0x0c10000f	jal 0x0040003c	13: jal sum # call sum
	0x00400024	0x3c011001	lui $1,0x00001001	14: sw $v0, y # y = sum()
	0x00400028	0xac220008	sw $2,0x00000008($1)	
	0x0040002c	0x8fbf0000	lw $31,0x00000000($29)	15: lw $ra, 0($sp) # restore $ra
	0x00400030	0x23bd0004	addi $29,$29,0x00000004	16: addi $sp, $sp, 4 # restore $sp
	0x00400034	0x2402000a	addiu $2,$0,0x0000000a	17: li $v0, 10
	0x00400038	0x0000000c	syscall	18: syscall
	0x0040003c	0x00851020	add $2,$4,$5	20: add $v0, $a0, $a1 # $v0 = a + b
	0x00400040	0x03e00008	jr $31	21: jr $ra # return

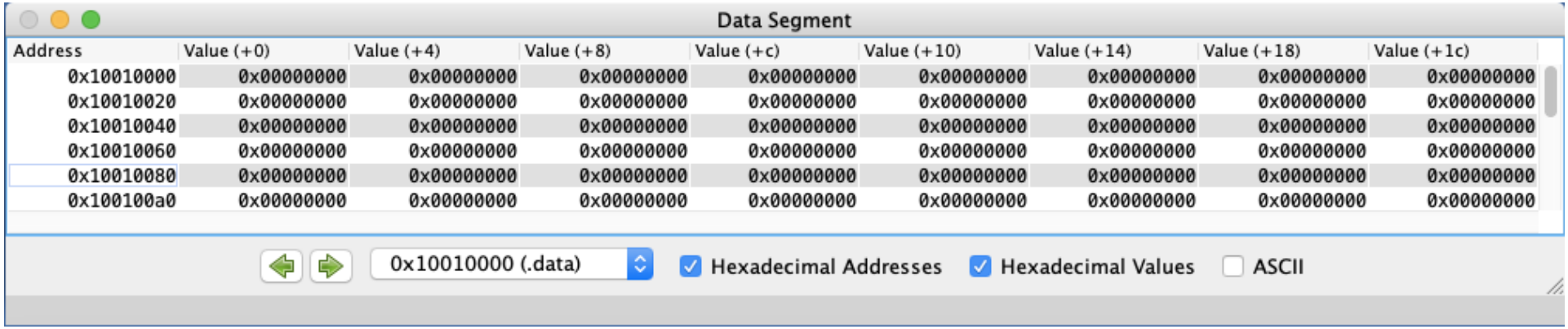
Tập thanh ghi



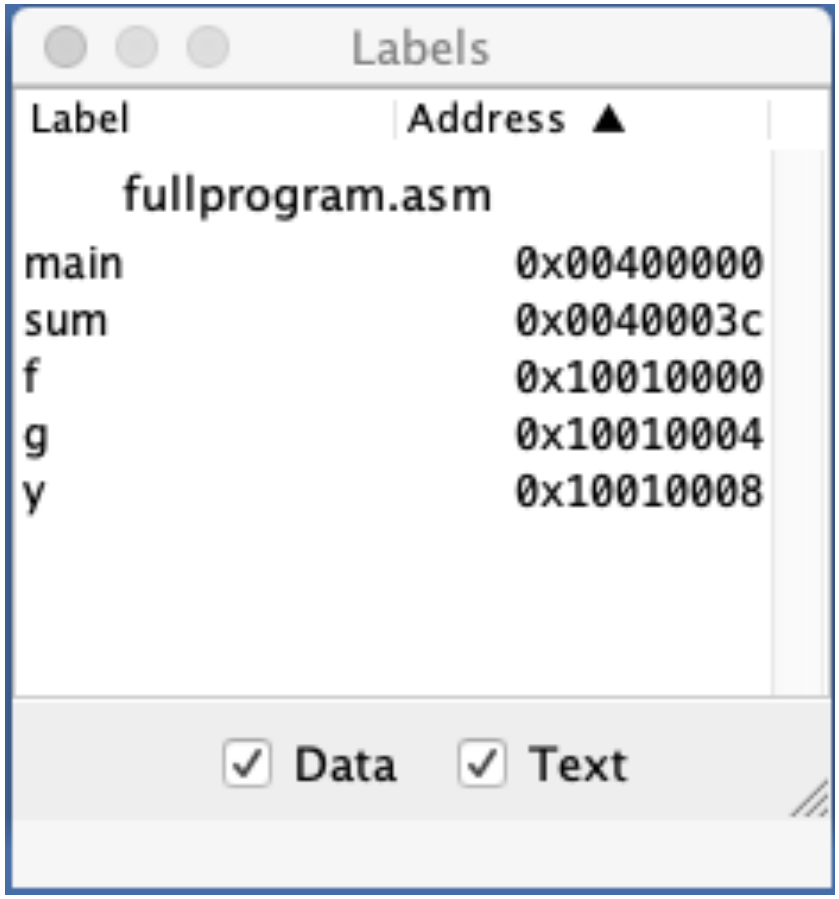
Registers | Coproc 1 | Coproc 0

Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000000
$v0	2	0x00000000
$v1	3	0x00000000
$a0	4	0x00000000
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x00000000
$t1	9	0x00000000
$t2	10	0x00000000
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x00000000
$s1	17	0x00000000
$s2	18	0x00000000
$s3	19	0x00000000
$s4	20	0x00000000
$s5	21	0x00000000
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x10008000
$sp	29	0x7fffeffc
$fp	30	0x00000000
$ra	31	0x00000000
pc		0x00400000
hi		0x00000000
lo		0x00000000

Vùng nhớ dữ liệu



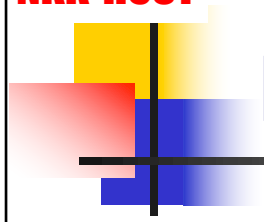
Nhãn và biến


Labels
Label
Address ▲
fullprogram.asm
main 0x00400000
sum 0x0040003c
f 0x10010000
g 0x10010004
y 0x10010008
Data
Text

Ví dụ lệnh giả (Pseudoinstruction)

Pseudoinstruction	MIPS Instructions
`li $s0,0x1234AA77`	`lui $at,0x1234` `ori $s0,$at,0xAA77`
`mul $s0,$s1,$s2`	`mult $s1,$s2` `mflo $s0`
`not $t1,$t2`	`nor $t1,$t2,$0`
`move $s1,$s2`	`addu $s1, $0, $s2`
`nop`	`sll $0, $0, 0`

Hết chương 5



Chương 6
BỘ XỬ LÝ

Nguyễn Kim Khánh
Trường Đại học Bách khoa Hà Nội

Nội dung học phần

Chương 1. Giới thiệu chung

Chương 2. Cơ bản về logic số

Chương 3. Hệ thống máy tính

Chương 4. Số học máy tính

Chương 5. Kiến trúc tập lệnh

Chương 6. Bộ xử lý

Chương 7. Bộ nhớ máy tính

Chương 8. Hệ thống vào-ra

Chương 9. Các kiến trúc song song

Nội dung của chương 6

6.1. Tổ chức của CPU

6.2. Thiết kế đơn vị điều khiển

6.3. Kỹ thuật đường ống lệnh

6.4. Ví dụ thiết kế bộ xử lý theo kiến trúc MIPS (*)

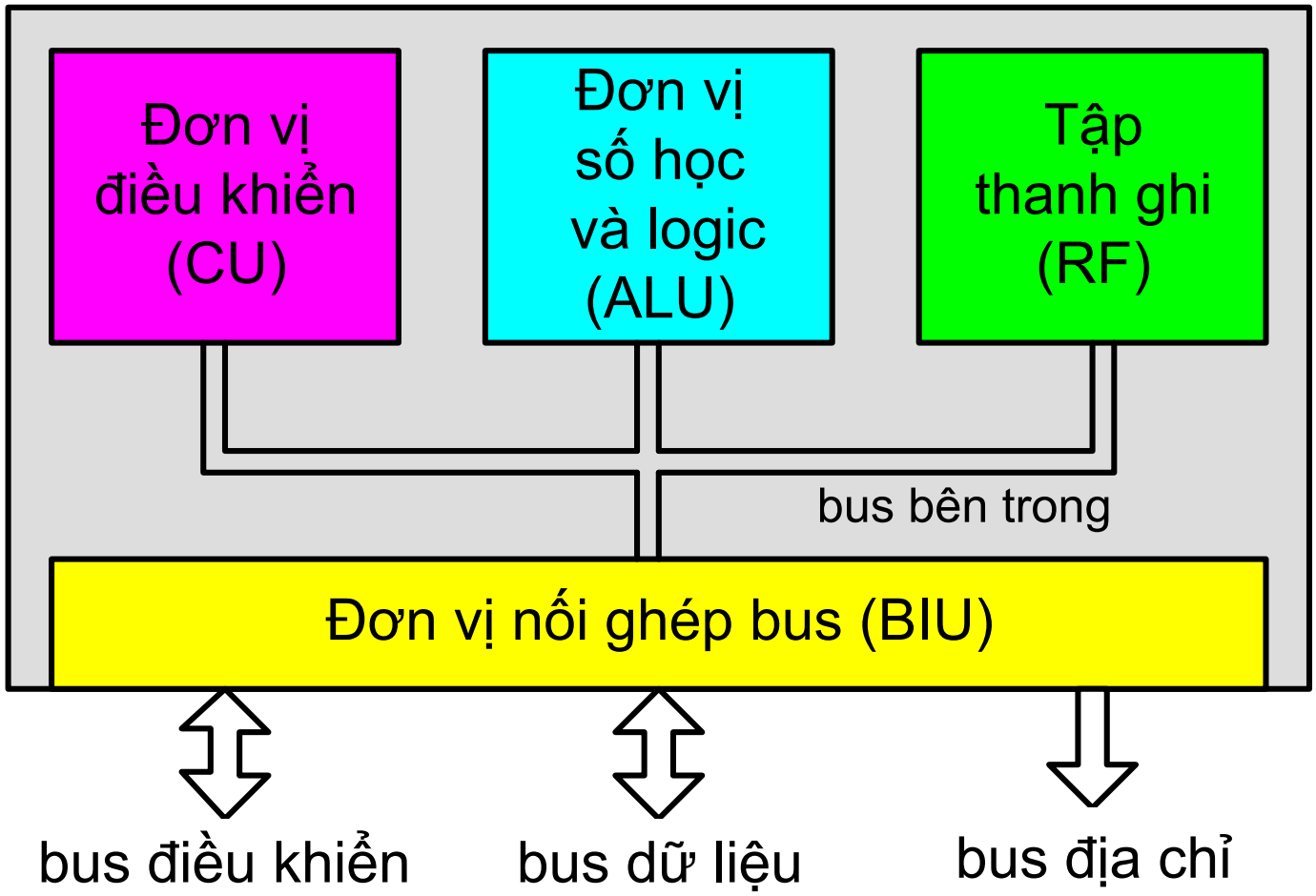
(*) *dành cho Chương trình Tài năng và Chất lượng cao*

6.1. Tổ chức của CPU

1. Cấu trúc cơ bản của CPU

- Nhiệm vụ của CPU:
  - Nhận lệnh (Fetch Instruction): CPU đọc lệnh từ bộ nhớ
  - Giải mã lệnh (Decode Instruction): xác định thao tác mà lệnh yêu cầu
  - Nhận dữ liệu (Fetch Data): nhận dữ liệu từ bộ nhớ hoặc các cổng vào-ra
  - Xử lý dữ liệu (Process Data): thực hiện phép toán số học hay phép toán logic với các dữ liệu
  - Ghi dữ liệu (Write Data): ghi dữ liệu ra bộ nhớ hay cổng vào-ra

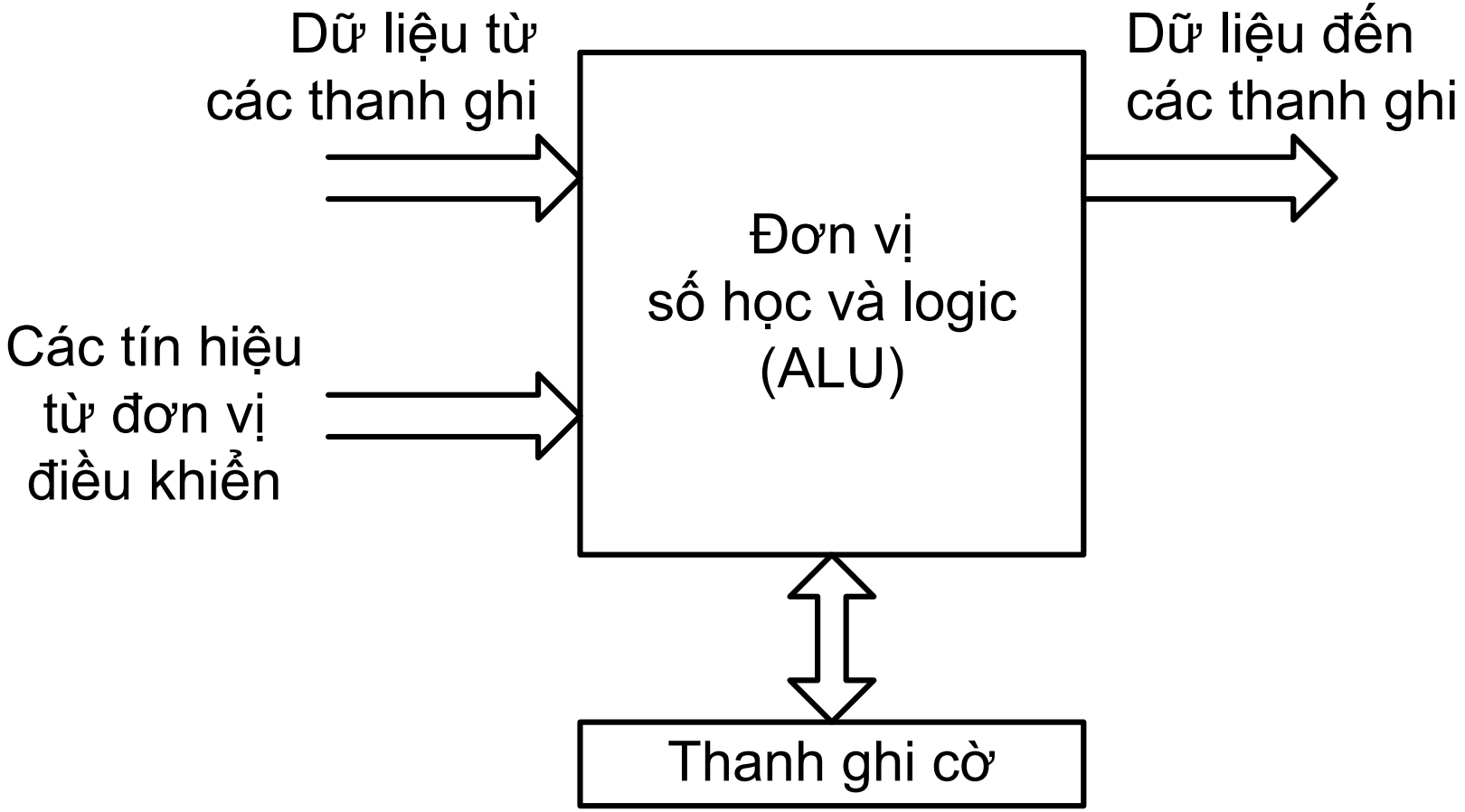
Sơ đồ cấu trúc cơ bản của CPU



2. Đơn vị số học và logic

- Chức năng: Thực hiện các phép toán số học và phép toán logic:
  - Số học: cộng, trừ, nhân, chia, đảo dấu
  - Logic: AND, OR, XOR, NOT, phép dịch bit

Mô hình kết nối ALU

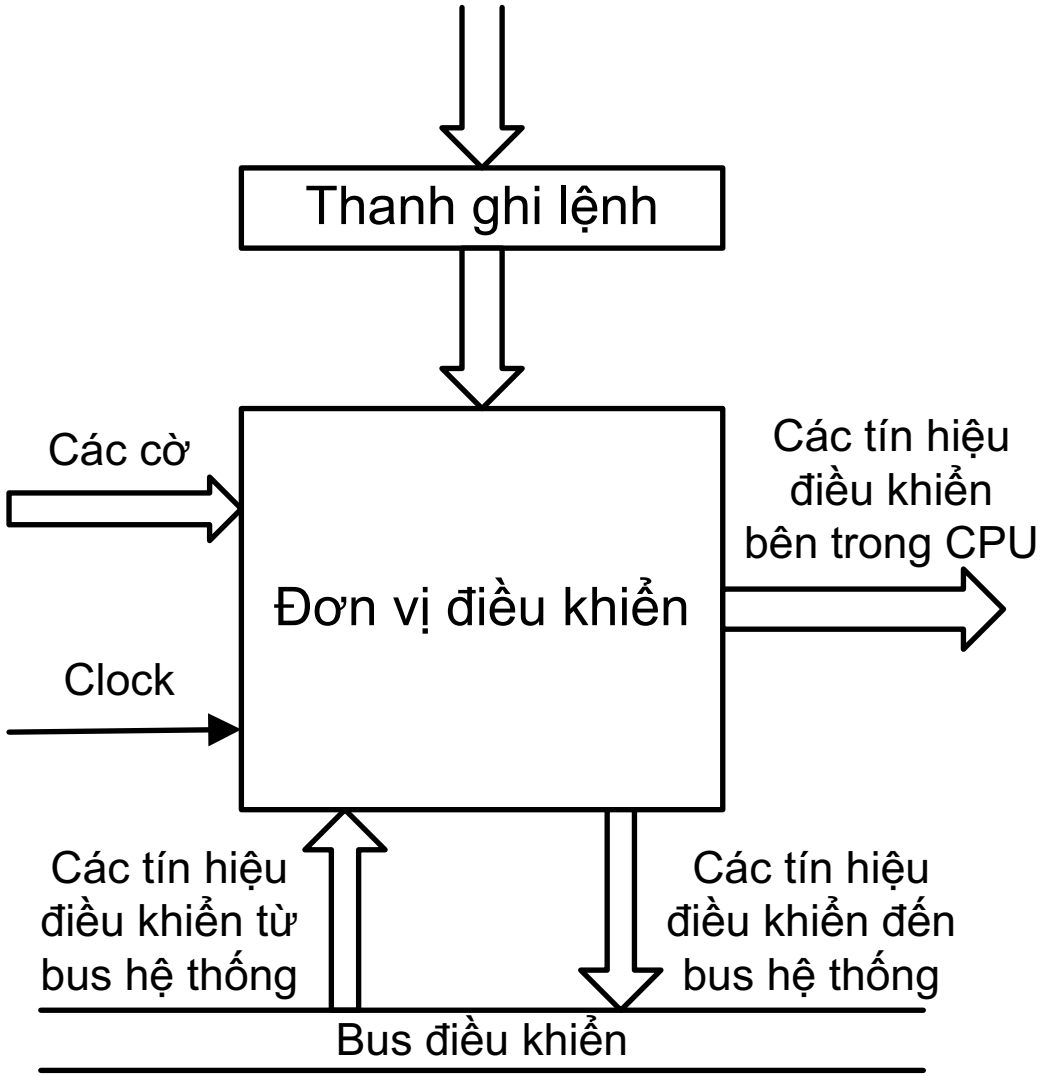


Thanh ghi cờ: hiển thị trạng thái của kết quả phép toán

3. Đơn vị điều khiển

- Chức năng
  - Điều khiển nhận lệnh từ bộ nhớ đưa vào CPU
  - Tăng nội dung của PC để trỏ sang lệnh kế tiếp
  - Giải mã lệnh đã được nhận để xác định thao tác mà lệnh yêu cầu
  - Phát ra các tín hiệu điều khiển thực hiện lệnh
  - Nhận các tín hiệu yêu cầu từ bus hệ thống và đáp ứng với các yêu cầu đó.

Mô hình kết nối đơn vị điều khiển



Các tín hiệu đưa đến đơn vị điều khiển

- Clock: tín hiệu nhịp từ mạch tạo dao động bên ngoài
- Lệnh từ thanh ghi lệnh đưa đến để giải mã
- Các cờ từ thanh ghi cờ cho biết trạng thái của CPU
- Các tín hiệu yêu cầu từ bus điều khiển

Các tín hiệu phát ra từ đơn vị điều khiển

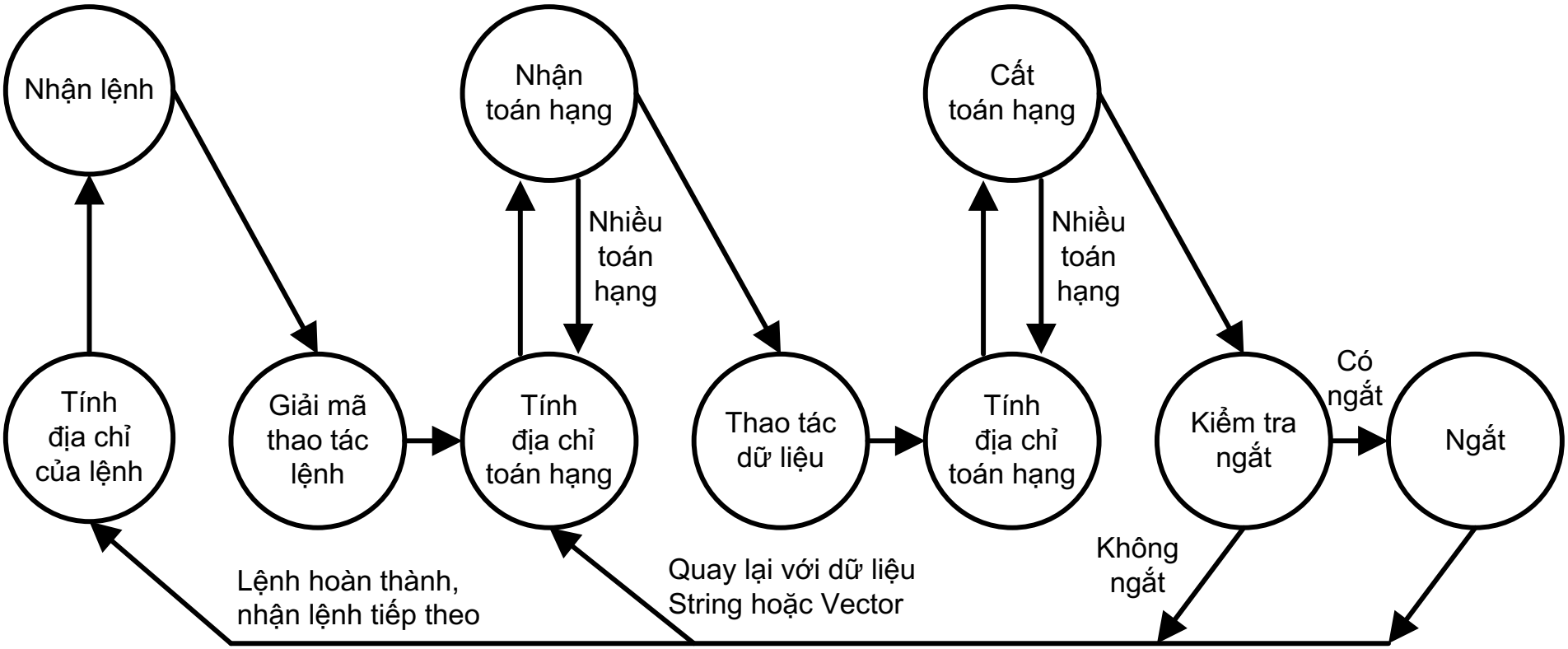
- Các tín hiệu điều khiển bên trong CPU:
  - Điều khiển các thanh ghi
  - Điều khiển ALU
- Các tín hiệu điều khiển bên ngoài CPU:
  - Điều khiển bộ nhớ
  - Điều khiển các mô-đun vào-ra

4. Hoạt động của chu trình lệnh

Chu trình lệnh

- Nhận lệnh
- Giải mã lệnh
- Nhận toán hạng
- Thực hiện lệnh
- Cất toán hạng
- Ngắt

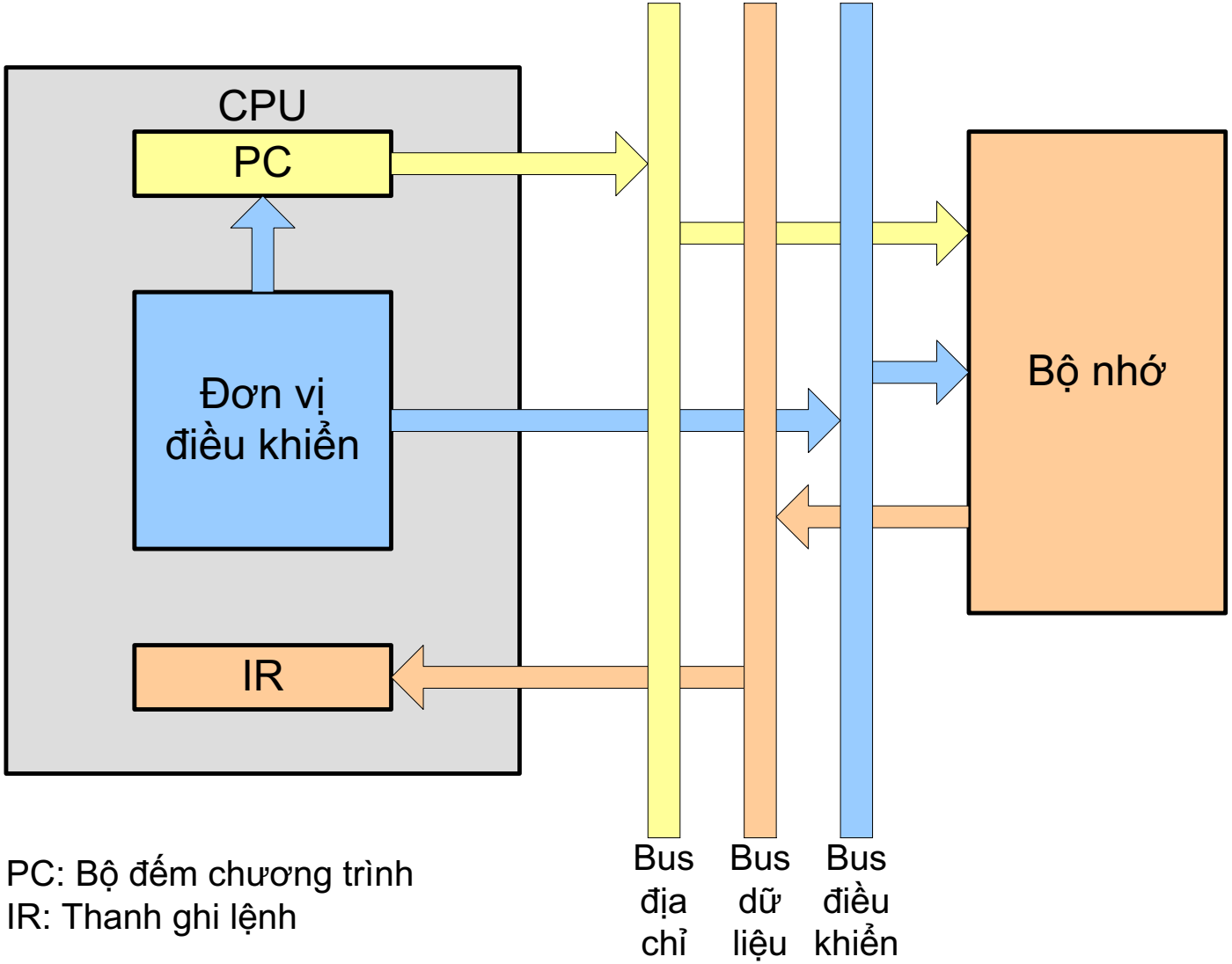
Giản đồ trạng thái chu trình lệnh



Nhận lệnh

- CPU đưa địa chỉ của lệnh cần nhận từ bộ đếm chương trình PC ra bus địa chỉ
- CPU phát tín hiệu điều khiển đọc bộ nhớ
- Lệnh từ bộ nhớ được đặt lên bus dữ liệu và được CPU copy vào thanh ghi lệnh IR
- CPU tăng nội dung PC để trỏ sang lệnh kế tiếp

Sơ đồ mô tả quá trình nhận lệnh



PC: Bộ đếm chương trình
IR: Thanh ghi lệnh

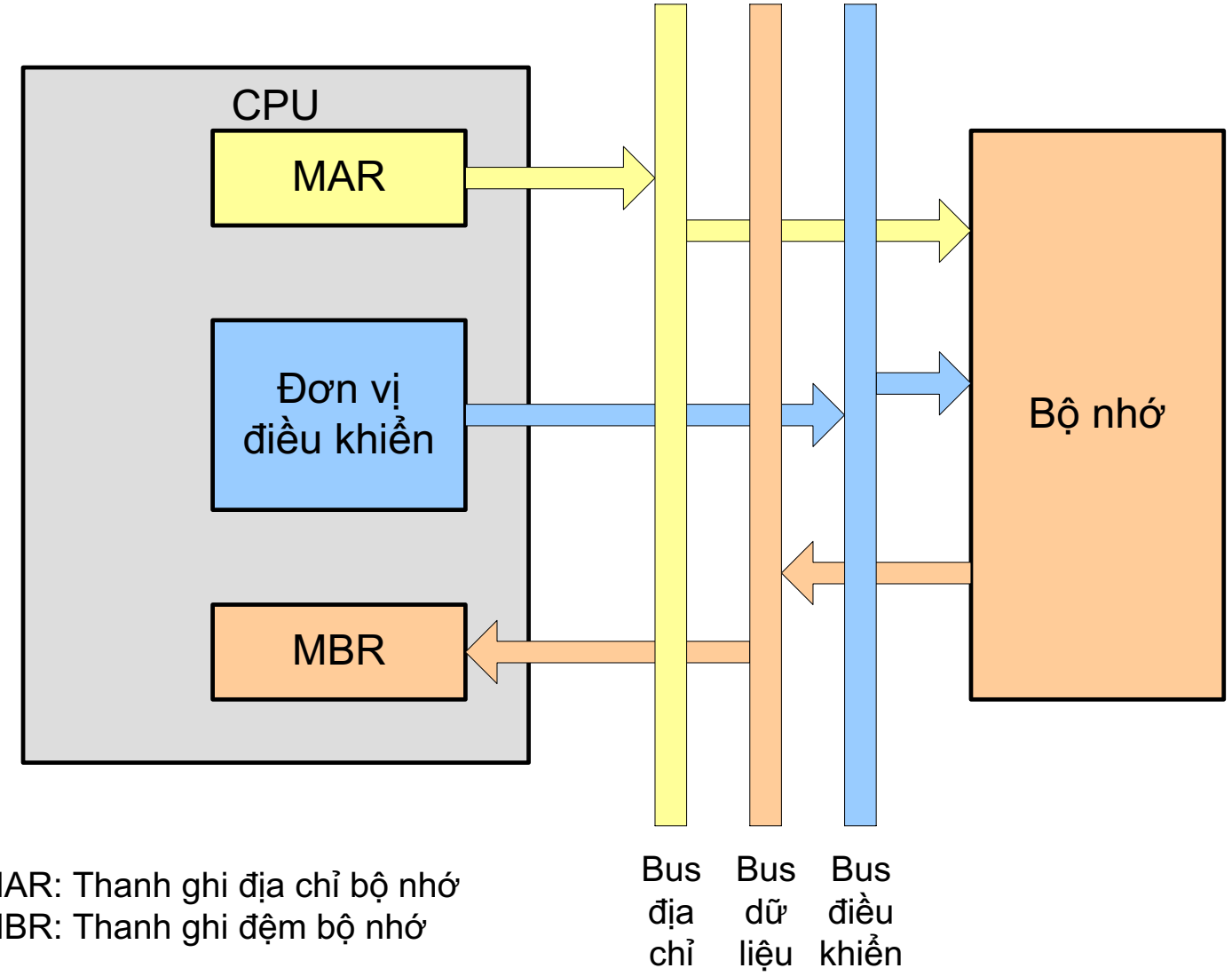
Giải mã lệnh

- Lệnh từ thanh ghi lệnh IR được đưa đến đơn vị điều khiển
- Đơn vị điều khiển tiến hành giải mã lệnh để xác định thao tác phải thực hiện
- Giải mã lệnh xảy ra bên trong CPU

Nhận dữ liệu từ bộ nhớ

- CPU đưa địa chỉ của toán hạng ra bus địa chỉ
- CPU phát tín hiệu điều khiển đọc
- Toán hạng được đọc vào CPU
- Tương tự như nhận lệnh

Sơ đồ mô tả nhận dữ liệu từ bộ nhớ



MAR: Thanh ghi địa chỉ bộ nhớ
MBR: Thanh ghi đệm bộ nhớ

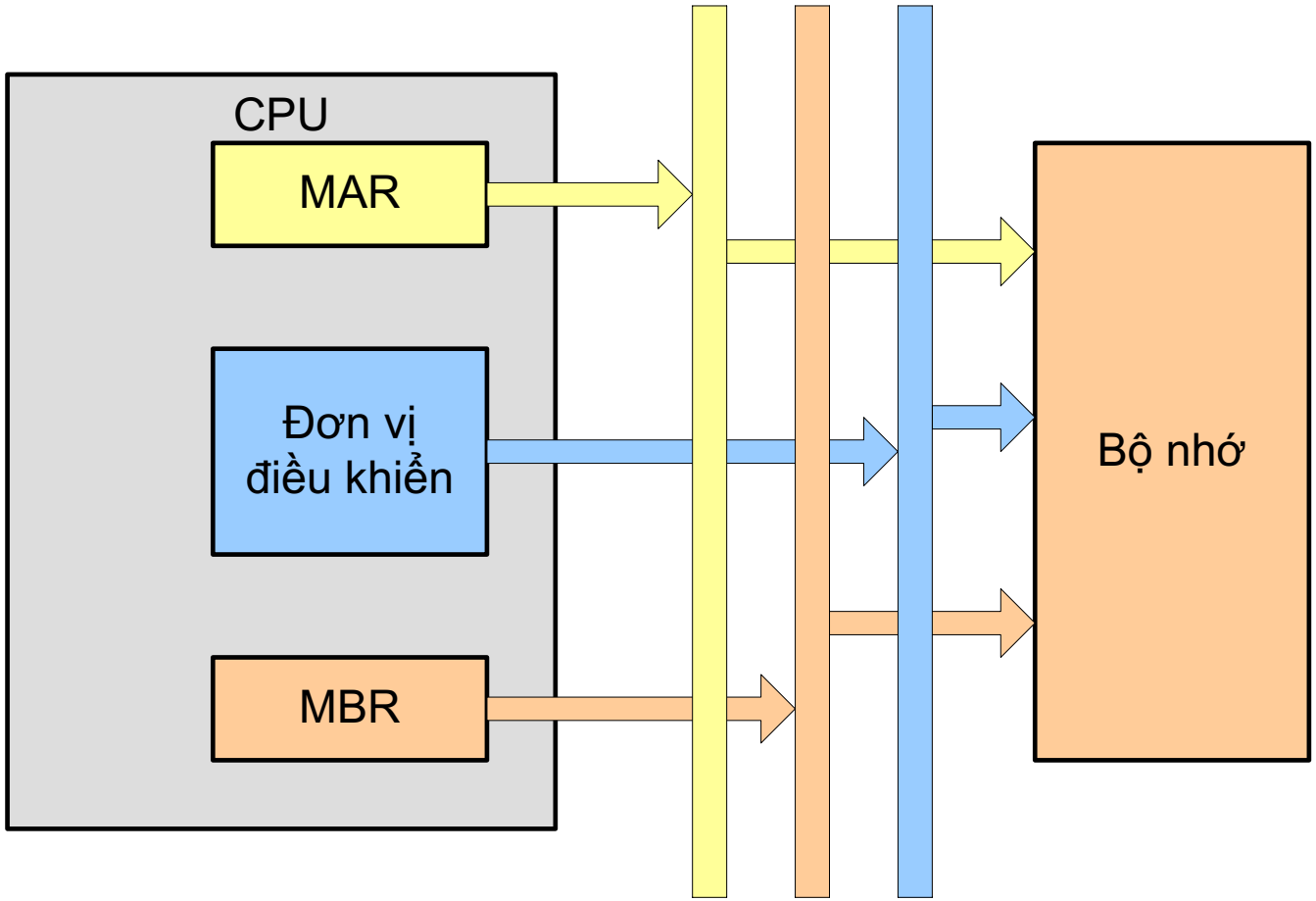
Thực hiện lệnh

- Có nhiều dạng tuỳ thuộc vào lệnh
- Có thể là:
  - Đọc/Ghi bộ nhớ
  - Vào/Ra
  - Chuyển giữa các thanh ghi
  - Phép toán số học/logic
  - Chuyển điều khiển (rẽ nhánh)
  - ...

Ghi toán hạng

- CPU đưa địa chỉ ra bus địa chỉ
- CPU đưa dữ liệu cần ghi ra bus dữ liệu
- CPU phát tín hiệu điều khiển ghi
- Dữ liệu trên bus dữ liệu được copy đến vị trí xác định

Sơ đồ mô tả quá trình ghi toán hạng

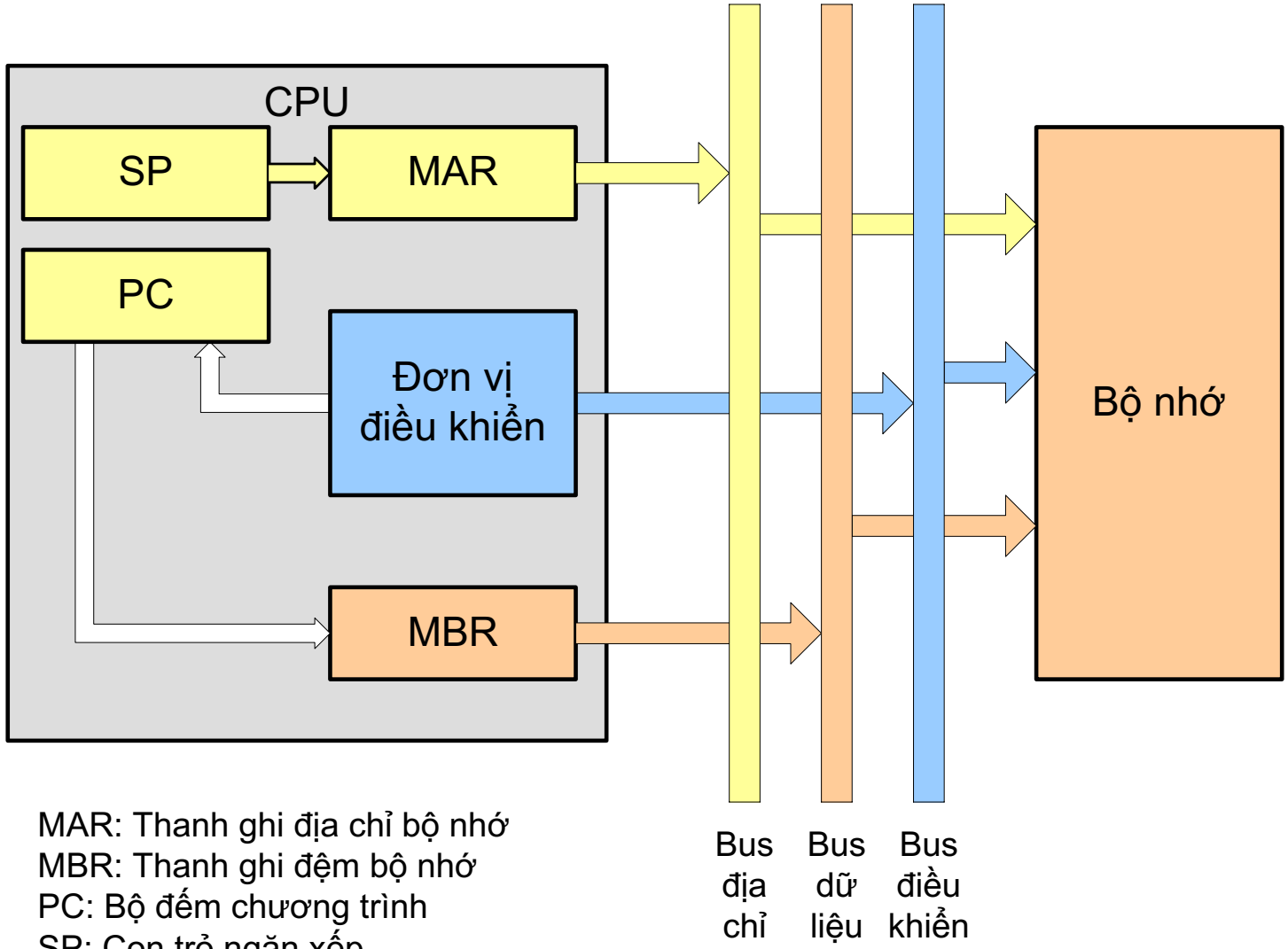


MAR: Thanh ghi địa chỉ bộ nhớ
MBR: Thanh ghi đệm bộ nhớ

Ngắt

- Nội dung của bộ đếm chương trình PC (địa chỉ trở về sau khi ngắt) được đưa ra bus dữ liệu
- CPU đưa địa chỉ (thường được lấy từ con trỏ ngăn xếp SP) ra bus địa chỉ
- CPU phát tín hiệu điều khiển ghi bộ nhớ
- Địa chỉ trở về trên bus dữ liệu được ghi ra vị trí xác định (ở ngăn xếp)
- Địa chỉ lệnh đầu tiên của chương trình con điều khiển ngắt được nạp vào PC

Sơ đồ mô tả chu trình ngắt



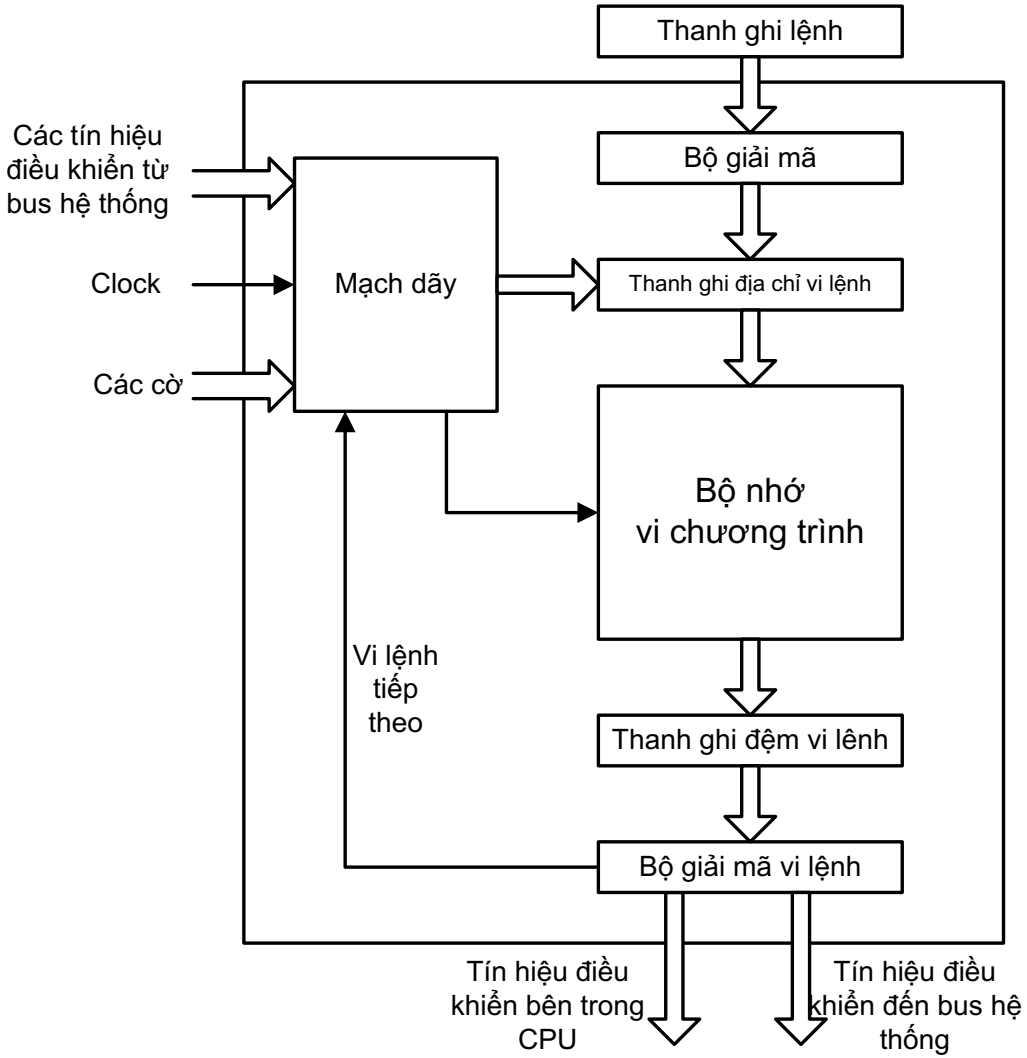
MAR: Thanh ghi địa chỉ bộ nhớ
MBR: Thanh ghi đệm bộ nhớ
PC: Bộ đếm chương trình
SP: Con trỏ ngăn xếp

6.2. Các phương pháp thiết kế đơn vị điều khiển

- Đơn vị điều khiển vi chương trình (Microprogrammed Control Unit)
- Đơn vị điều khiển nối kết cứng (Hardwired Control Unit)

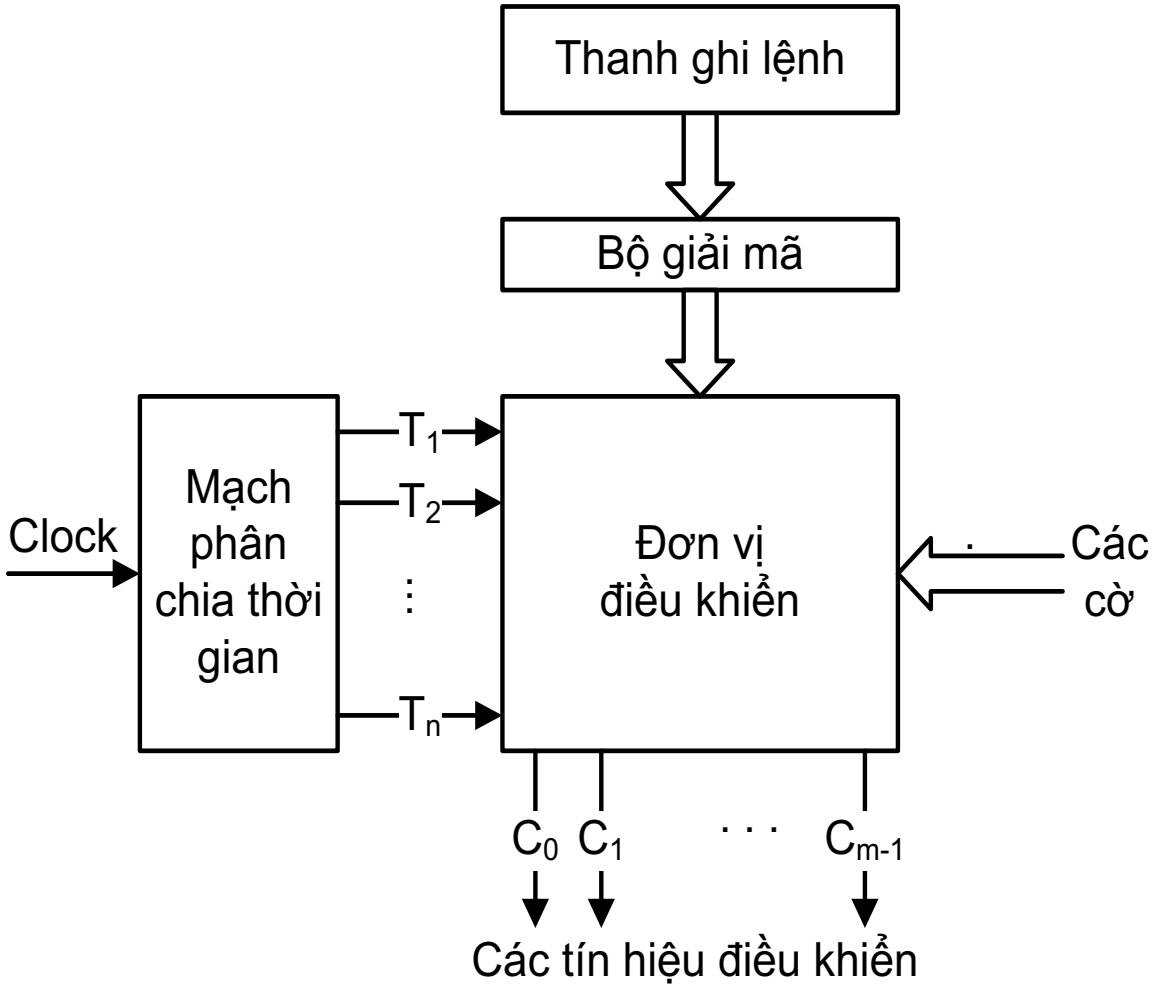
1. Đơn vị điều khiển vi chương trình

- Bộ nhớ vi chương trình (ROM) lưu trữ các vi chương trình (microprogram)
- Một vi chương trình bao gồm các vi lệnh (microinstruction)
- Mỗi vi lệnh mã hoá cho một vi thao tác (microoperation)
- Để hoàn thành một lệnh cần thực hiện một hoặc một vài vi chương trình
- Tốc độ chậm



2. Đơn vị điều khiển nối kết cứng

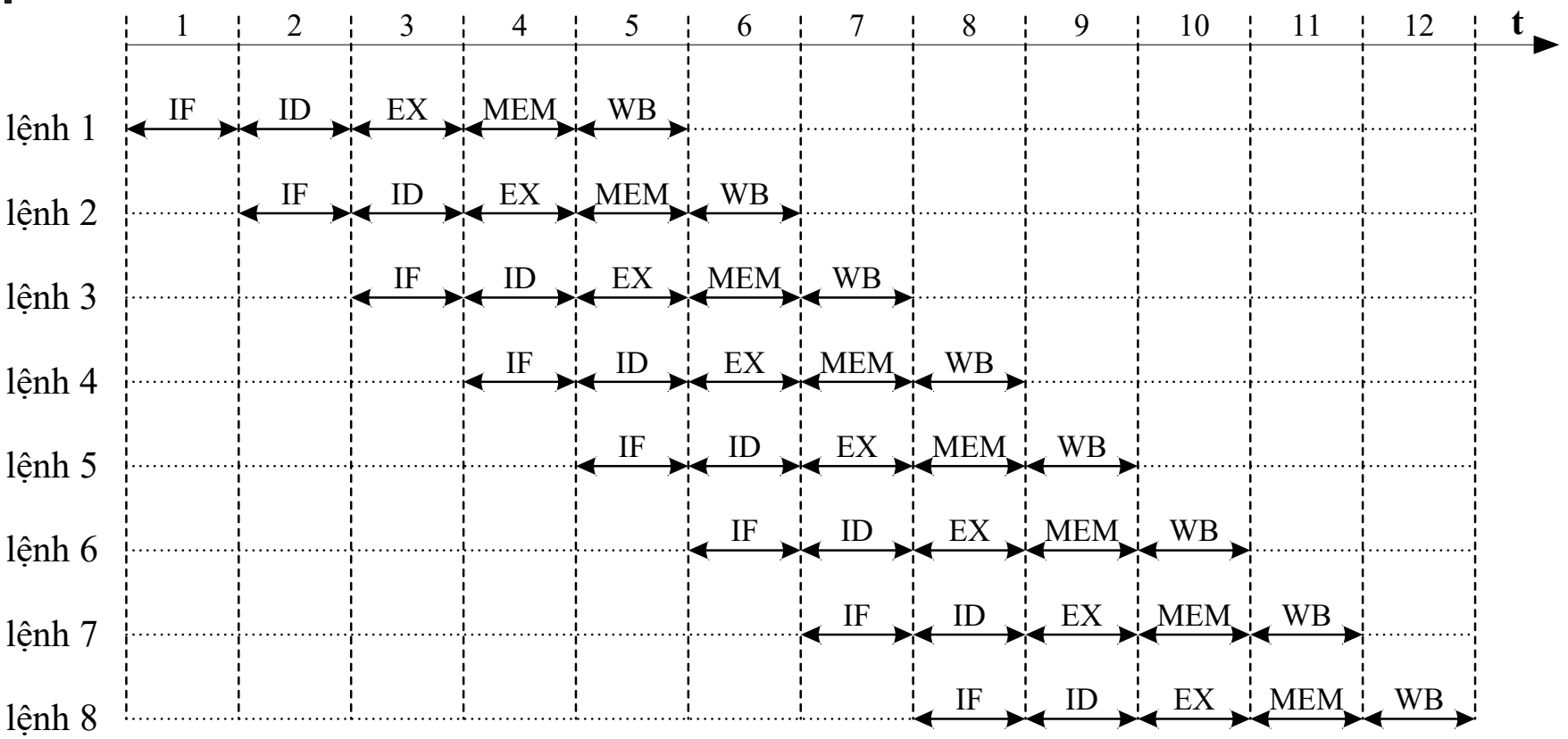
- Sử dụng mạch cứng để giải mã và tạo các tín hiệu điều khiển thực hiện lệnh
- Tốc độ nhanh
- Đơn vị điều khiển phức tạp



6.3. Kỹ thuật đường ống lệnh

- Kỹ thuật đường ống lệnh (Instruction Pipelining): Chia chu trình lệnh thành các công đoạn và cho phép thực hiện gối lên nhau (như dây chuyền lắp ráp)
- Chẳng hạn bộ xử lý MIPS có 5 công đoạn:
  1. IF: Instruction fetch from memory – Nhận lệnh từ bộ nhớ
  2. ID: Instruction decode & register read – Giải mã lệnh và đọc thanh ghi
  3. EX: Execute operation or calculate address – Thực hiện thao tác hoặc tính toán địa chỉ
  4. MEM: Access memory operand – Truy nhập toán hạng bộ nhớ
  5. WB: Write result back to register – Ghi kết quả trả về thanh ghi

Biểu đồ thời gian của đường ống lệnh



Thời gian thực hiện 1 công đoạn = T

Thời gian thực hiện tuần tự 8 lệnh: 8 x 5T = 40T

Thời gian thực hiện đường ống 8 lệnh: (1 x 5T) + [(8-1) x T] = 12T

Các mối trở ngại (Hazard) của đường ống lệnh

- Hazard: Tình huống ngăn cản bắt đầu của lệnh tiếp theo ở chu kỳ tiếp theo
  - Hazard cấu trúc: do tài nguyên được yêu cầu đang bận
  - Hazard dữ liệu: cần phải đợi để lệnh trước hoàn thành việc đọc/ghi dữ liệu
  - Hazard điều khiển: do rẽ nhánh gây ra

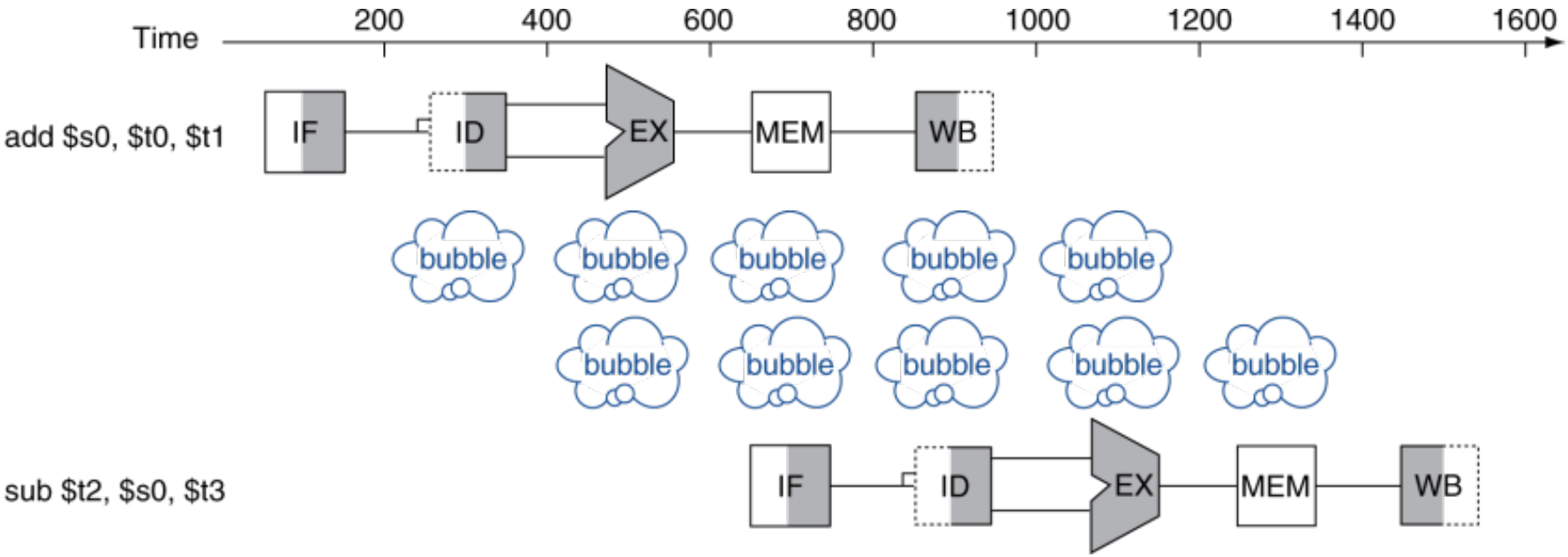
Hazard cấu trúc

- Xung đột khi sử dụng tài nguyên
- Trong đường ống của MIPS với một bộ nhớ dùng chung
  - Lệnh Load/store yêu cầu truy cập dữ liệu
  - Nhận lệnh cần trì hoãn cho chu kỳ đó
- Bởi vậy, datapath kiểu đường ống yêu cầu bộ nhớ lệnh và bộ nhớ dữ liệu tách rời (hoặc cache lệnh/cache dữ liệu tách rời)

Hazard dữ liệu

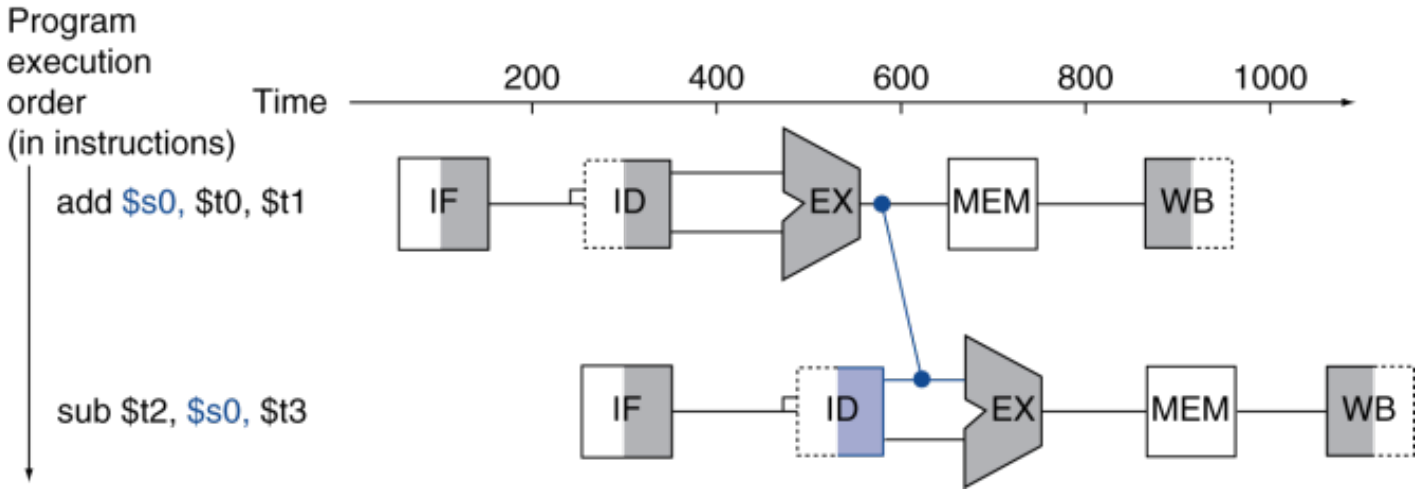
- Lệnh phụ thuộc vào việc hoàn thành truy cập dữ liệu của lệnh trước đó

```
add $s0, $t0, $t1
sub $t2, $s0, $t3
```



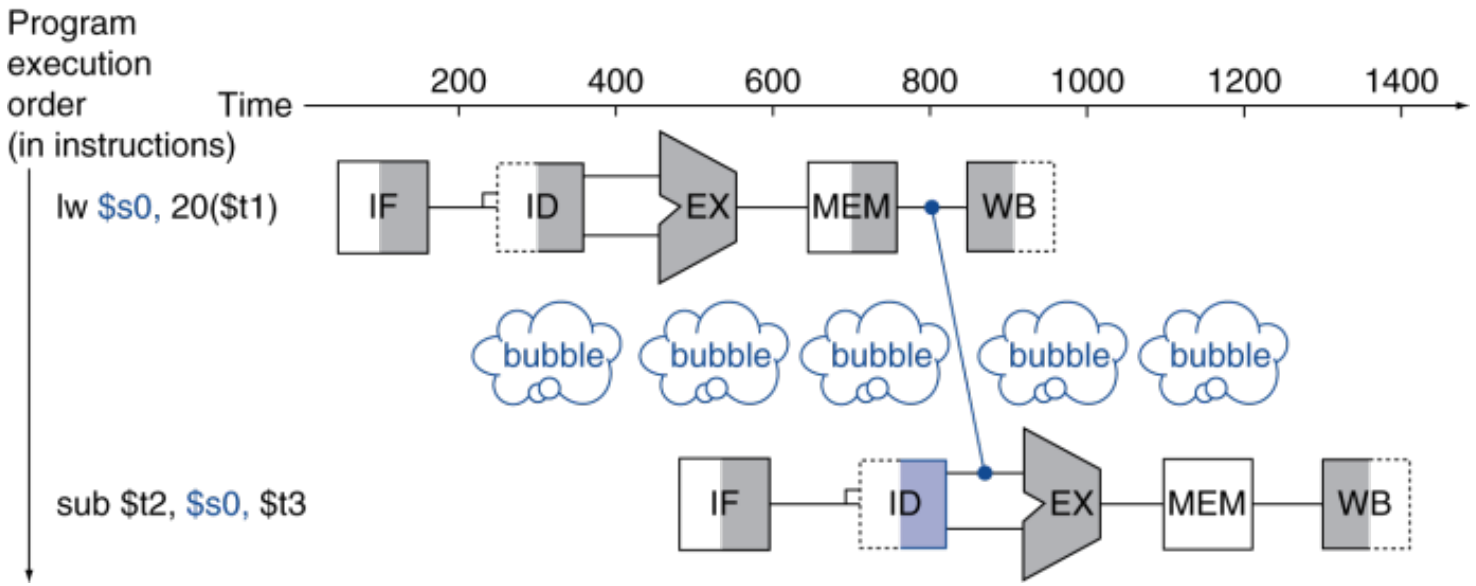
Forwarding (gửi vượt trước)

- Sử dụng kết quả ngay sau khi nó được tính
  - Không đợi đến khi kết quả được lưu đến thanh ghi
  - Yêu cầu có đường kết nối thêm trong datapath



Hazard dữ liệu với lệnh load

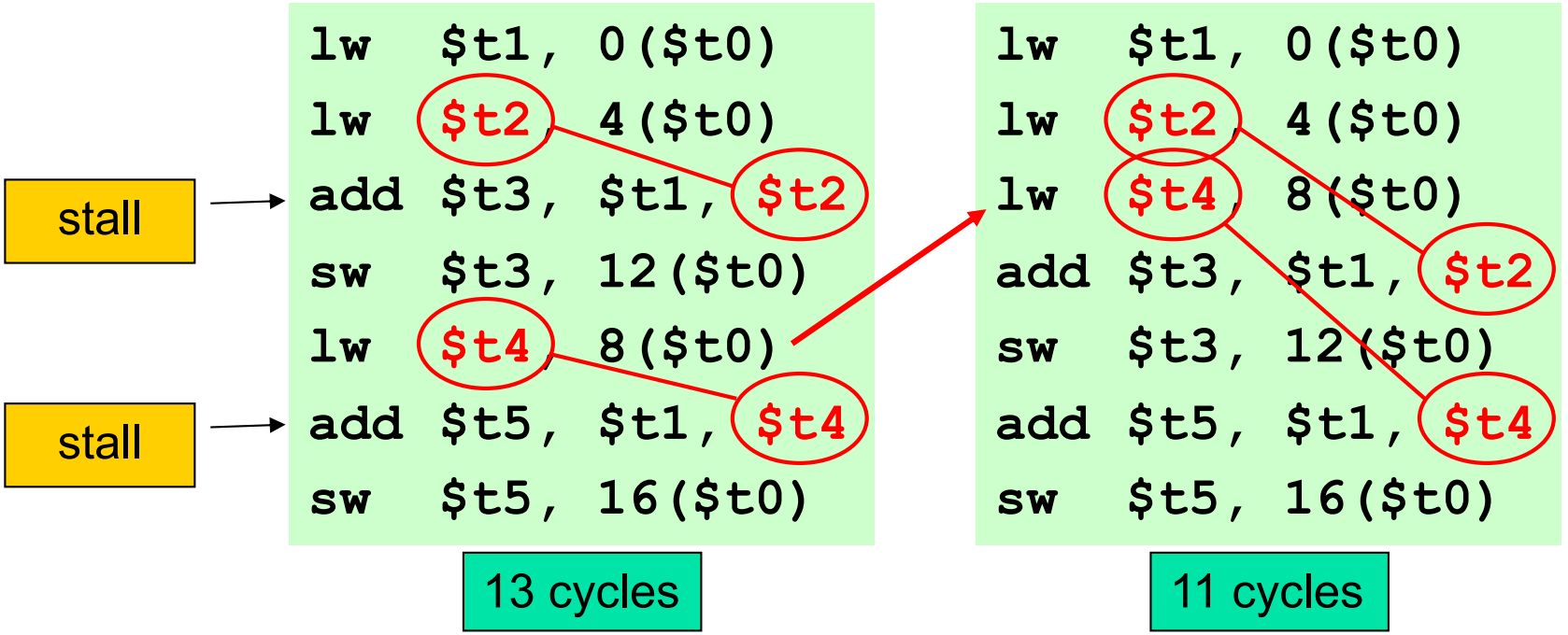
- Không phải luôn luôn có thể tránh trì hoãn bằng cách forwarding
  - Nếu giá trị chưa được tính khi cần thiết
  - Không thể chuyển ngược thời gian
  - Cần chèn bước trì hoãn (stall hay bubble)



Lập lịch mã để tránh trì hoãn

- Thay đổi trình tự mã để tránh sử dụng kết quả load ở lệnh tiếp theo
- Mã C:

```
a = b + e; c = b + f;
```

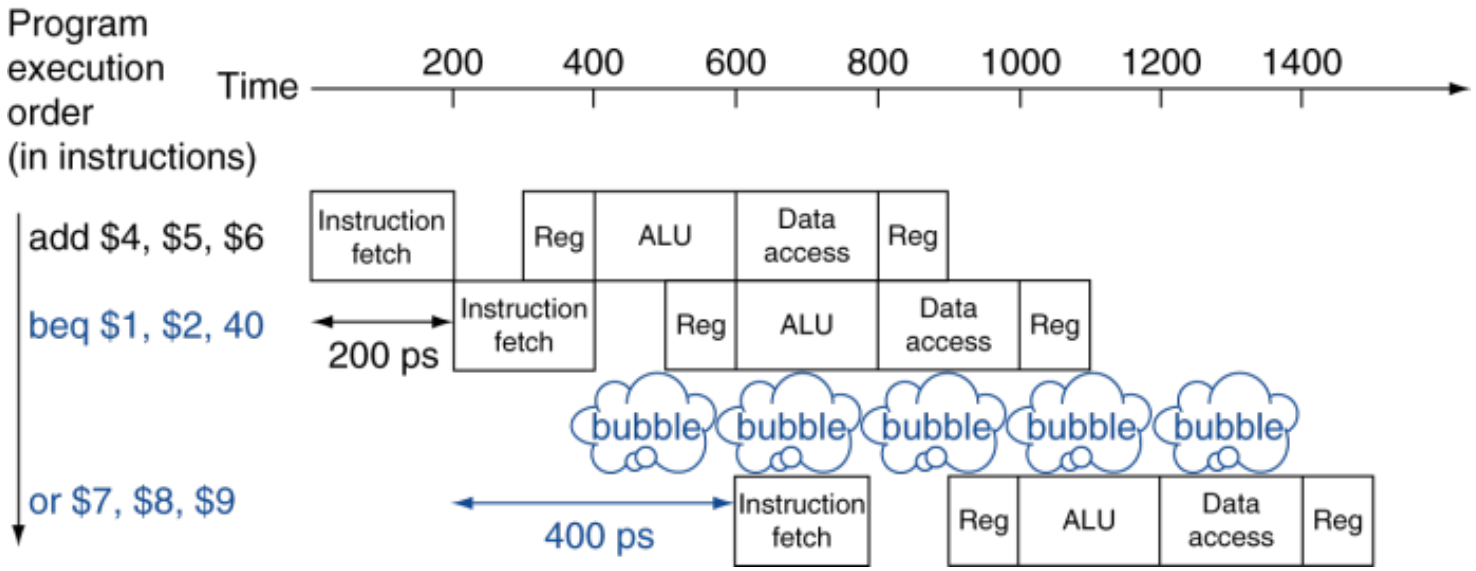


Hazard điều khiển

- Rẽ nhánh xác định luồng điều khiển
  - Nhận lệnh tiếp theo phụ thuộc vào kết quả rẽ nhánh
  - Đường ống không thể luôn nhận đúng lệnh
    - Vẫn đang làm ở công đoạn giải mã lệnh (ID) của lệnh rẽ nhánh
- Với đường ống của MIPS
  - Cần so sánh thanh ghi và tính địa chỉ đích sớm trong đường ống
  - Thêm phần cứng để thực hiện việc đó trong công đoạn ID

Trì hoãn khi rẽ nhánh

- Đợi cho đến khi kết quả rẽ nhánh đã được xác định trước khi nhận lệnh tiếp theo

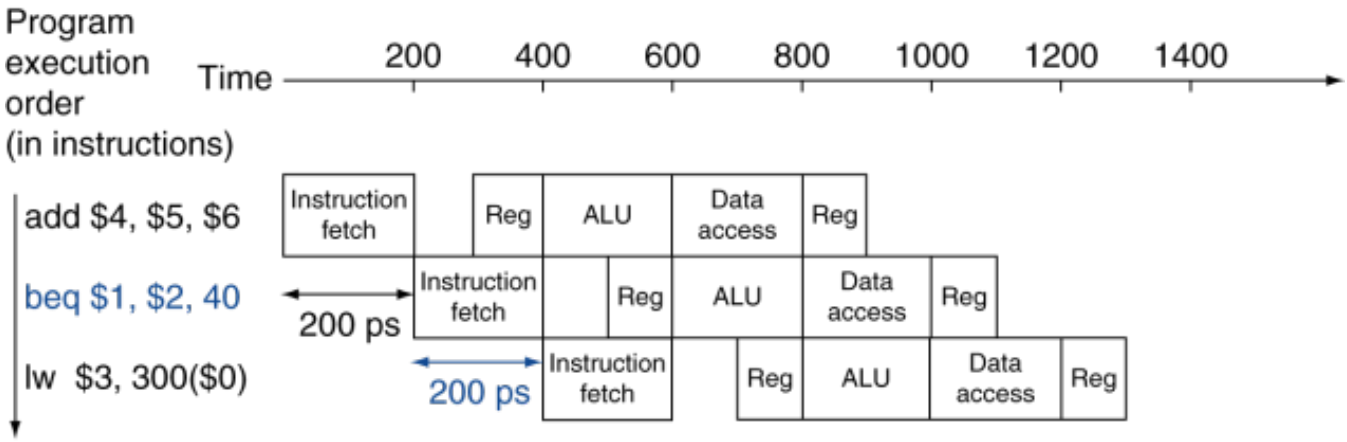


Dự đoán rẽ nhánh

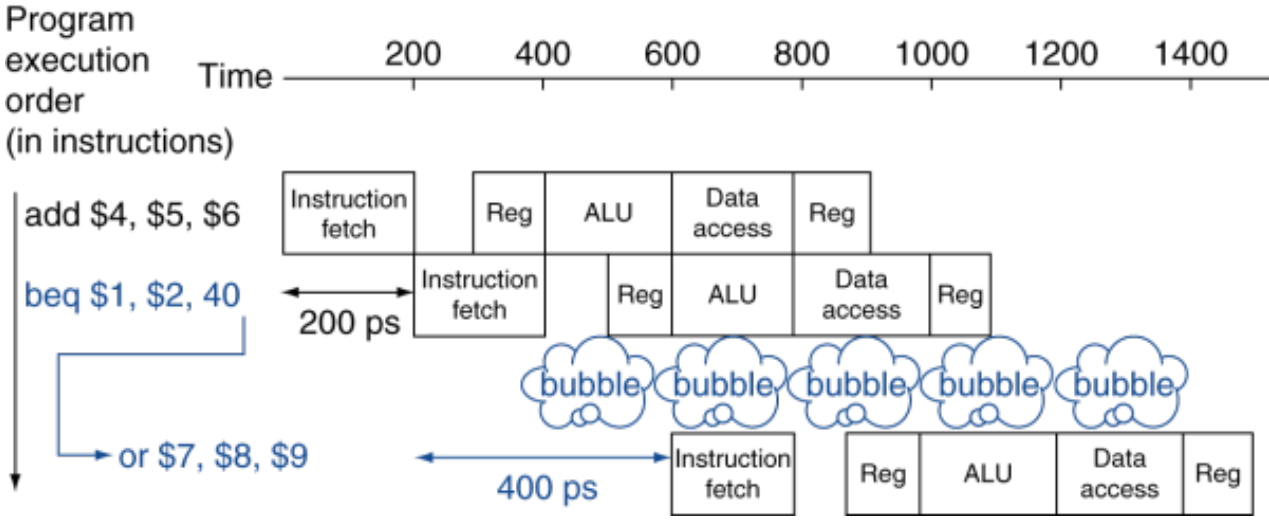
- Những đường ống dài hơn không thể sớm xác định dễ dàng kết quả rẽ nhánh
  - Cách trì hoãn không đáp ứng được
- Dự đoán kết quả rẽ nhánh
  - Chỉ trì hoãn khi dự đoán là sai
- Với MIPS
  - Có thể dự đoán rẽ nhánh không xẩy ra
  - Nhận lệnh ngay sau lệnh rẽ nhánh (không làm trễ)

MIPS với dự đoán rẽ nhánh không xẩy ra

Prediction correct



Prediction incorrect

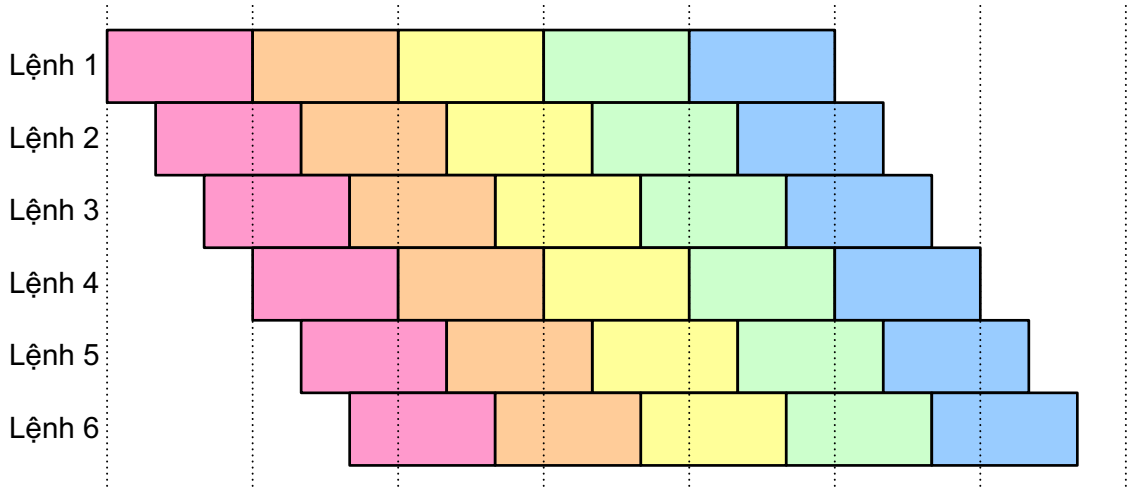


Đặc điểm của đường ống

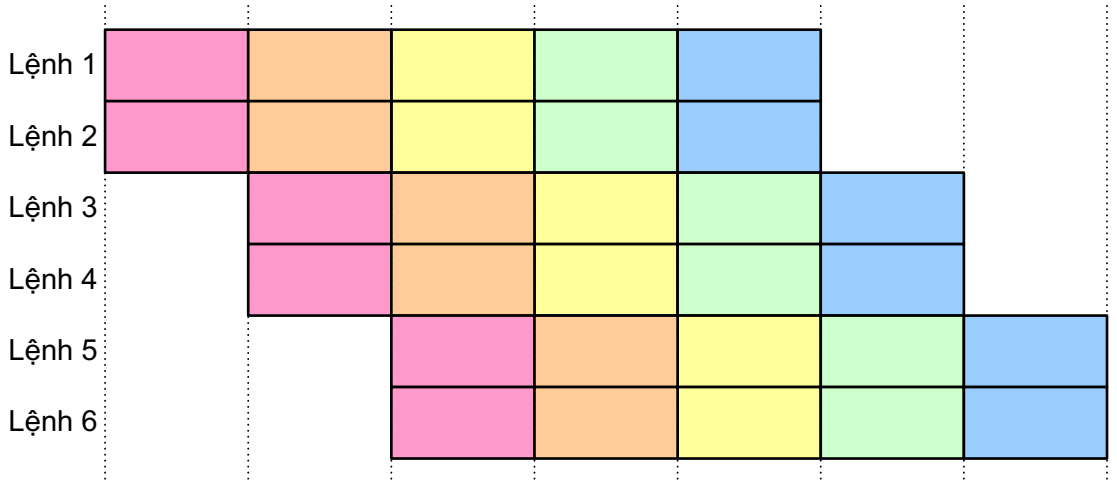
- Kỹ thuật đường ống cải thiện hiệu năng bằng cách tăng số lệnh thực hiện
  - Thực hiện nhiều lệnh đồng thời
  - Mỗi lệnh có cùng thời gian thực hiện
- Các dạng hazard:
  - Cấu trúc, dữ liệu, điều khiển
- Thiết kế tập lệnh ảnh hưởng đến độ phức tạp của việc thực hiện đường ống

Tăng cường khả năng song song mức lệnh

- Tăng số công đoạn của đường ống



- Siêu vô hướng (Superscalar)



6.4. Thiết kế bộ xử lý theo kiến trúc MIPS(*)

Dành riêng cho Chương trình Tài năng và Chất lượng cao

MIPS.pptx

Hết chương 6

Chương 7
BỘ NHỚ MÁY TÍNH

Nguyễn Kim Khánh
Trường Đại học Bách khoa Hà Nội

Nội dung học phần

Chương 1. Giới thiệu chung

Chương 2. Cơ bản về logic số

Chương 3. Hệ thống máy tính

Chương 4. Số học máy tính

Chương 5. Kiến trúc tập lệnh

Chương 6. Bộ xử lý

Chương 7. Bộ nhớ máy tính

Chương 8. Hệ thống vào-ra

Chương 9. Các kiến trúc song song

Nội dung của chương 7

7.1. Tổng quan hệ thống nhớ

7.2. Bộ nhớ chính

7.3. Bộ nhớ đệm (cache)

7.4. Bộ nhớ ngoài

7.5. Bộ nhớ ảo

7.1. Tổng quan hệ thống nhớ

1. Các đặc trưng của bộ nhớ

- Vị trí
  - Bên trong CPU:
    - tập thanh ghi
  - Bộ nhớ trong:
    - bộ nhớ chính
    - bộ nhớ đệm (cache)
  - Bộ nhớ ngoài:
    - các thiết bị lưu trữ
- Dung lượng
  - Độ dài từ nhớ (tính bằng bit)
  - Số lượng từ nhớ

Các đặc trưng của bộ nhớ (tiếp)

- Đơn vị truyền
  - Từ nhớ
  - Khối nhớ
- Phương pháp truy nhập
  - Truy nhập tuần tự (băng từ)
  - Truy nhập trực tiếp (các loại đĩa)
  - Truy nhập ngẫu nhiên (bộ nhớ bán dẫn)
  - Truy nhập liên kết (cache)

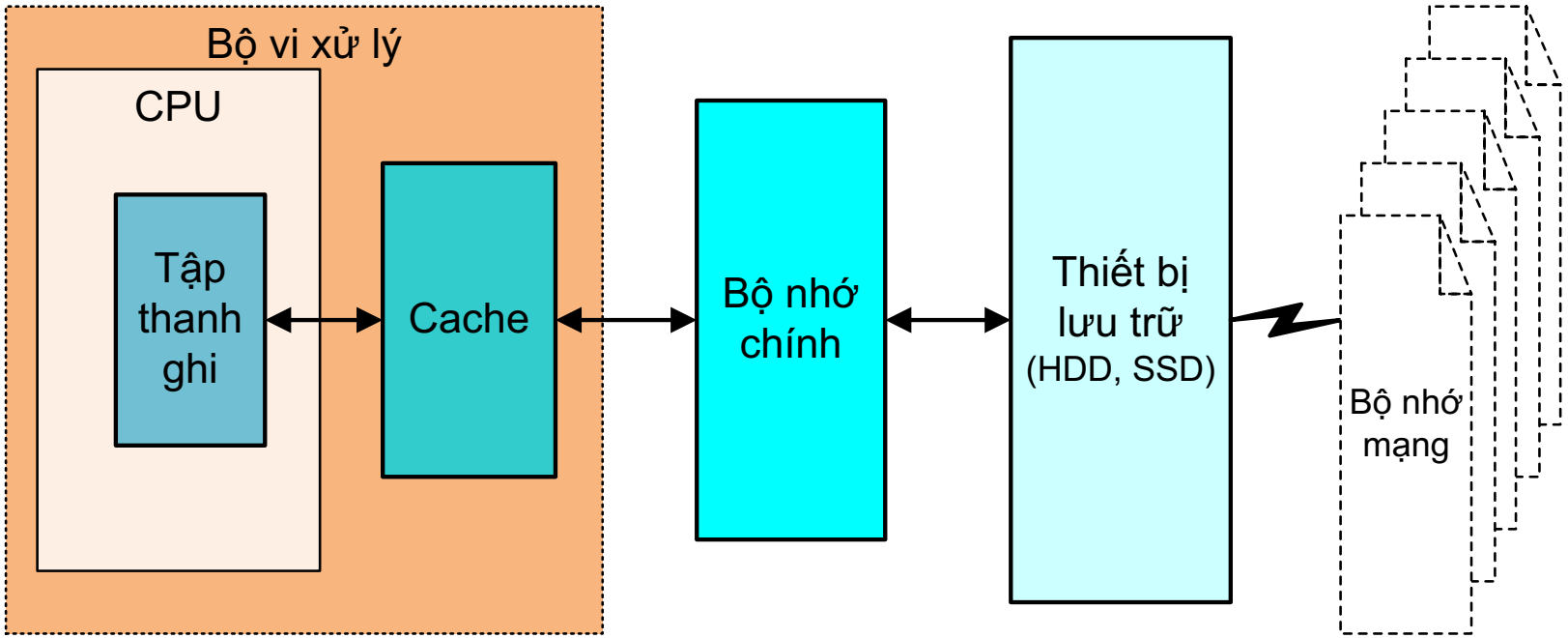
Các đặc trưng của bộ nhớ (tiếp)

- Hiệu năng (performance)
  - Thời gian truy nhập
  - Chu kỳ nhớ
  - Tốc độ truyền
- Kiểu vật lý
  - Bộ nhớ bán dẫn
  - Bộ nhớ từ
  - Bộ nhớ quang

Các đặc trưng của bộ nhớ (tiếp)

- Các đặc tính vật lý
  - Khả biến / Không khả biến (volatile / nonvolatile)
  - Xoá được / không xoá được
- Tổ chức

2. Phân cấp bộ nhớ



Từ trái sang phải:

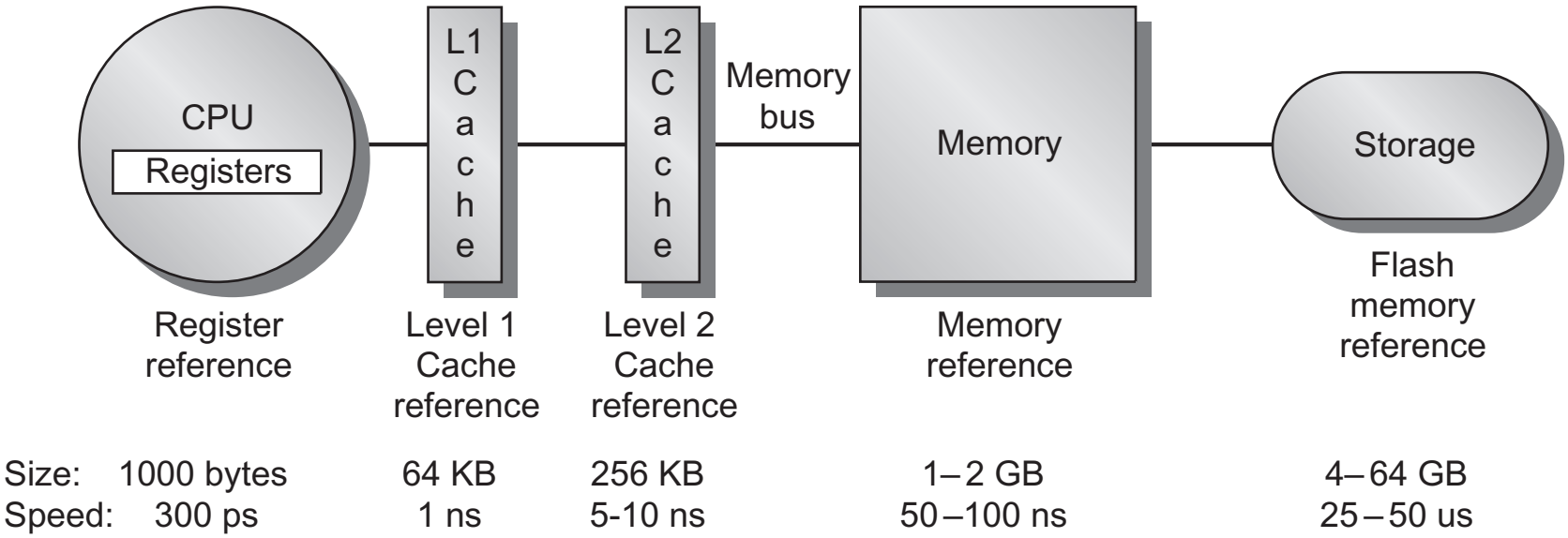
- dung lượng tăng dần
- tốc độ giảm dần
- giá thành cùng dung lượng giảm dần

Công nghệ bộ nhớ

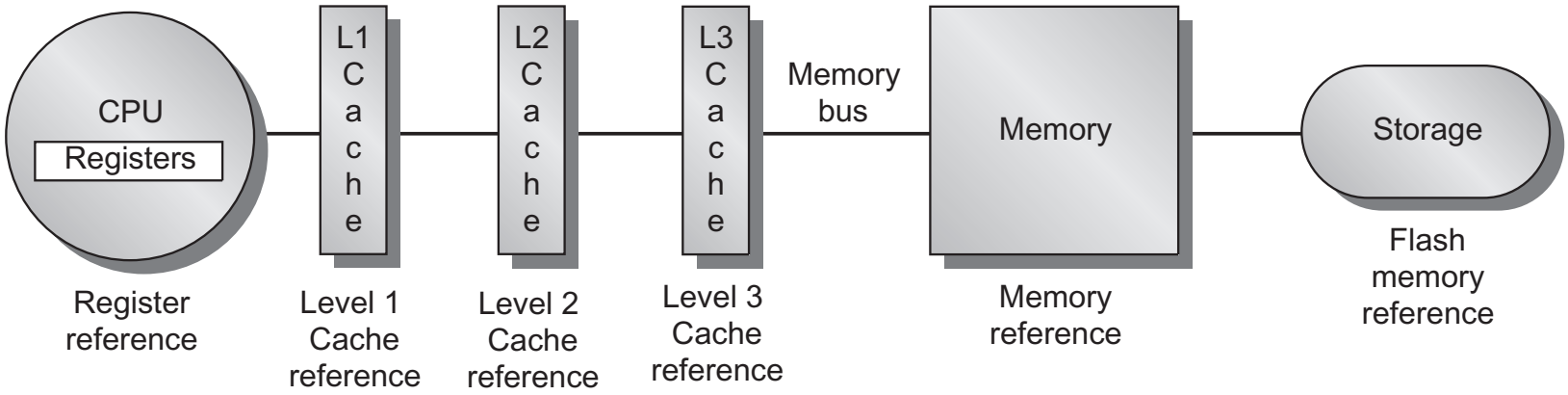
Công nghệ bộ nhớ	Thời gian truy nhập	Giá thành/GiB (2012)
SRAM	0,5 – 2,5 ns	$500 – $1000
DRAM	50 – 70 ns	$10 – $20
Flash memory	5000 – 50 000 ns	$0,75 – $1
HDD	5 – 20 ms	$0,05 – $0,1

- Bộ nhớ lý tưởng
  - Thời gian truy nhập như SRAM
  - Dung lượng và giá thành như ổ đĩa cứng

Phân cấp bộ nhớ cho thiết bị di động

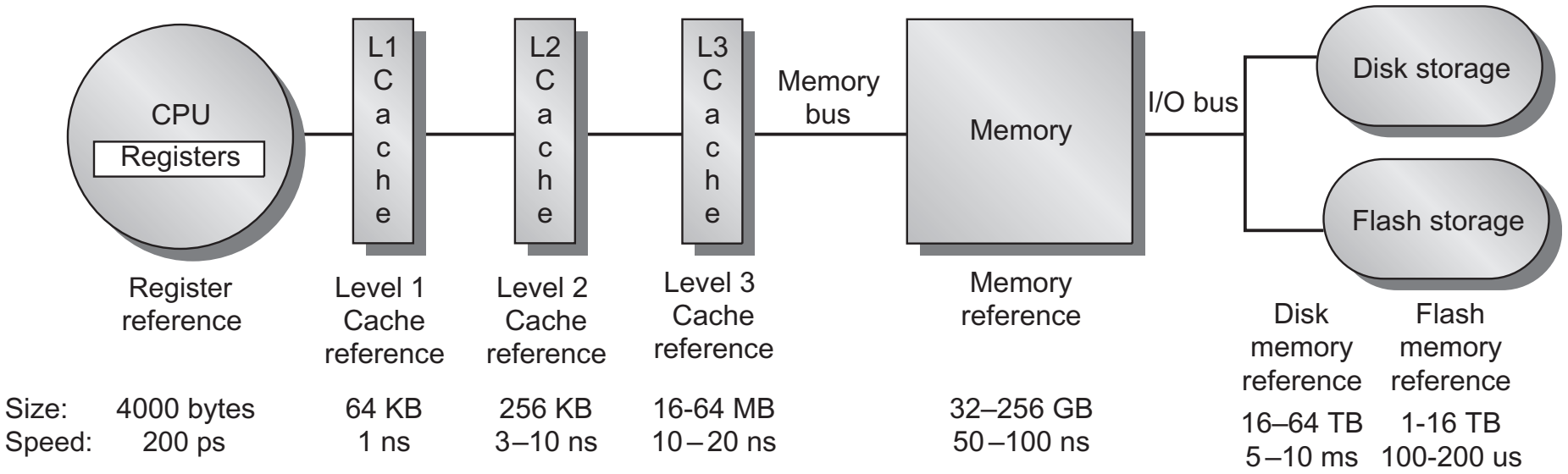


Phân cấp bộ nhớ cho máy tính PC



Laptop	Size:	1000 bytes	64 KB	256 KB	4-8 MB	4 –16 GB	256 GB-1 TB
	Speed:	300 ps	1 ns	3–10 ns	10 – 20 ns	50 –100 ns	50-100 uS
Desktop	Size:	2000 bytes	64 KB	256 KB	8-32 MB	8–64 GB	256 GB-2 TB
	Speed:	300 ps	1 ns	3–10 ns	10 – 20 ns	50 –100 ns	50-100 uS

Phân cấp bộ nhớ cho máy chủ



Nguyên lý cục bộ hoá tham chiếu bộ nhớ

- Trong một khoảng thời gian đủ nhỏ CPU thường chỉ tham chiếu các thông tin trong một khối nhớ cục bộ
- Ví dụ:
  - Cấu trúc chương trình tuần tự
  - Vòng lặp có thân nhỏ
  - Cấu trúc dữ liệu mảng

7.2. Bộ nhớ chính

1. Bộ nhớ bán dẫn

Kiểu bộ nhớ	Tiêu chuẩn	Khả năng xoá	Cơ chế ghi	Tính khả biến
Read Only Memory (ROM)	Bộ nhớ chỉ đọc	Không xoá được	Mặt nạ	Không khả biến
Programmable ROM (PROM)			Bằng điện	
Erasable PROM (EPROM)	Bộ nhớ hầu như chỉ đọc	bằng tia cực tím, cả chip		
Electrically Erasable PROM (EEPROM)		bằng điện, mức từng byte		
Flash memory	Bộ nhớ đọc-ghi	bằng điện, từng khối		
Random Access Memory (RAM)		bằng điện, mức từng byte	Bằng điện	Khả biến

ROM (Read Only Memory)

- Bộ nhớ không khả biến
- Lưu trữ các thông tin sau:
  - Thư viện các chương trình con
  - Các chương trình điều khiển hệ thống (BIOS)
  - Các bảng chức năng
  - Vi chương trình

Các kiểu ROM

- ROM mặt nạ:
  - thông tin được ghi khi sản xuất
- PROM (Programmable ROM)
  - Cần thiết bị chuyên dụng để ghi
  - Chỉ ghi được một lần
- EPROM (Erasable PROM)
  - Cần thiết bị chuyên dụng để ghi
  - Xóa được bằng tia tử ngoại
  - Ghi lại được nhiều lần
- EEPROM (Electrically Erasable PROM)
  - Có thể ghi theo từng byte
  - Xóa bằng điện

Bộ nhớ Flash

- Ghi theo khối
- Xóa bằng điện
- Dung lượng lớn

RAM (Random Access Memory)

- Bộ nhớ đọc-ghi (Read/Write Memory)
- Khả biến
- Lưu trữ thông tin tạm thời
- Có hai loại: SRAM và DRAM
  (Static and Dynamic)

SRAM (Static) – RAM tĩnh

- Các bit được lưu trữ bằng các Flip-Flop

→ thông tin ổn định

- Cấu trúc phức tạp
- Dung lượng chip nhỏ
- Tốc độ nhanh
- Đắt tiền
- Dùng làm bộ nhớ cache

DRAM (Dynamic) – RAM động

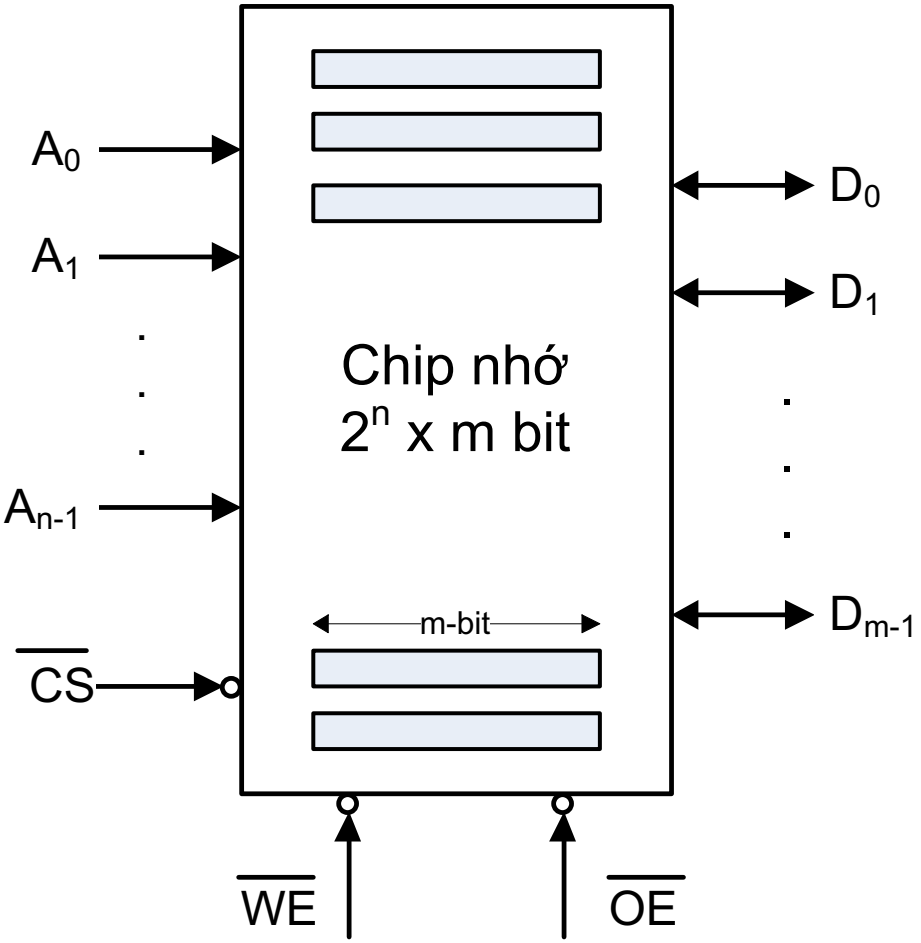
- Các bit được lưu trữ trên tụ điện
  → cần phải có mạch làm tươi
- Cấu trúc đơn giản
- Dung lượng lớn
- Tốc độ chậm hơn
- Rẻ tiền hơn
- Dùng làm bộ nhớ chính

Một số DRAM tiên tiến thông dụng

- Cải tiến để tăng tốc độ
- Synchronous DRAM (SDRAM): làm việc được đồng bộ bởi xung clock
- DDR-SDRAM (Double Data Rate SDRAM)
- DDR3, DDR4

Tổ chức của chip nhớ

Sơ đồ cơ bản của chip nhớ



Hình ảnh module nhớ

Các tín hiệu của chip nhớ

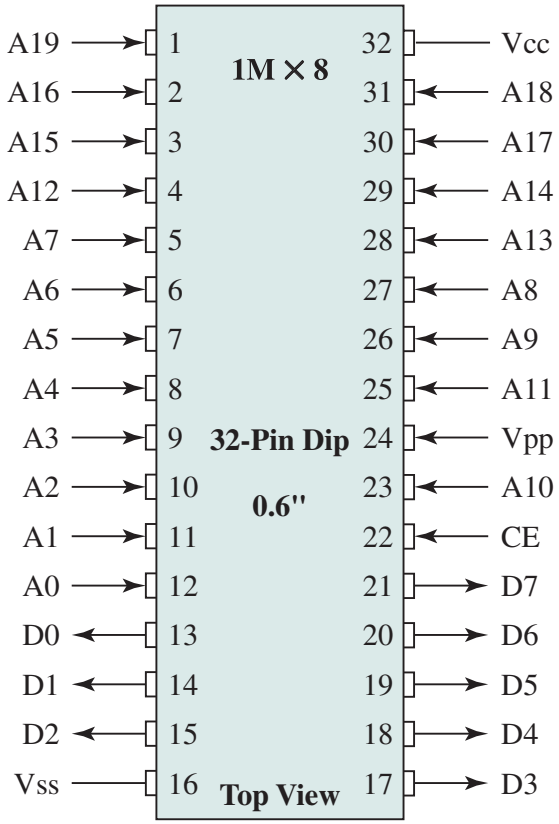
- Các đường địa chỉ: $A_{n-1} \div A_0$ → có 2^n từ nhớ
- Các đường dữ liệu: $D_{m-1} \div D_0$ → độ dài từ nhớ = m bit
- Dung lượng chip nhớ = 2^n x m bit
- Các đường điều khiển:
  - Tín hiệu chọn chip CS (Chip Select)
  - Tín hiệu điều khiển đọc OE (Output Enable)
  - Tín hiệu điều khiển ghi WE (Write Enable)

(Các tín hiệu điều khiển thường tích cực với mức 0)

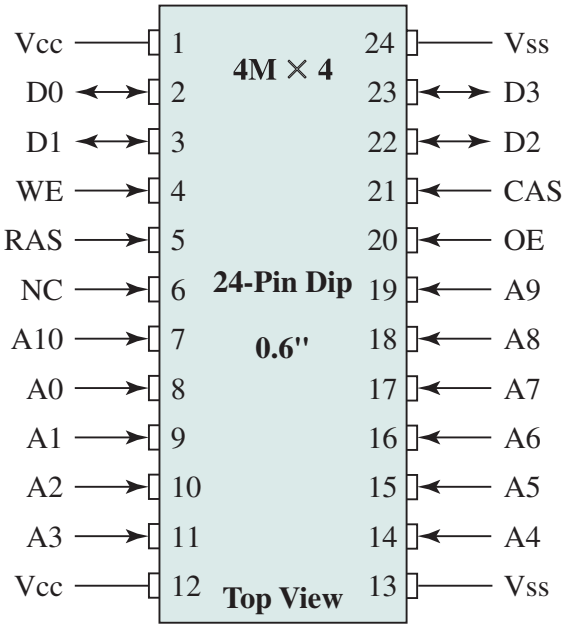
Tổ chức của DRAM

- Dùng n đường địa chỉ dồn kênh → cho phép truyền 2n bit địa chỉ
- Tín hiệu chọn địa chỉ hàng RAS (Row Address Select)
- Tín hiệu chọn địa chỉ cột CAS (Column Address Select)
- Dung lượng của DRAM= 2^{2n} x m bit

Ví dụ chip nhớ



(a) 8-Mbit EPROM



(b) 16-Mbit DRAM

Thiết kế mô-đun nhớ bán dẫn

- Dung lượng chip nhớ 2^n x m bit
- Cần thiết kế để tăng dung lượng:
  - Thiết kế tăng độ dài từ nhớ
  - Thiết kế tăng số lượng từ nhớ
  - Thiết kế kết hợp

Tăng độ dài từ nhớ

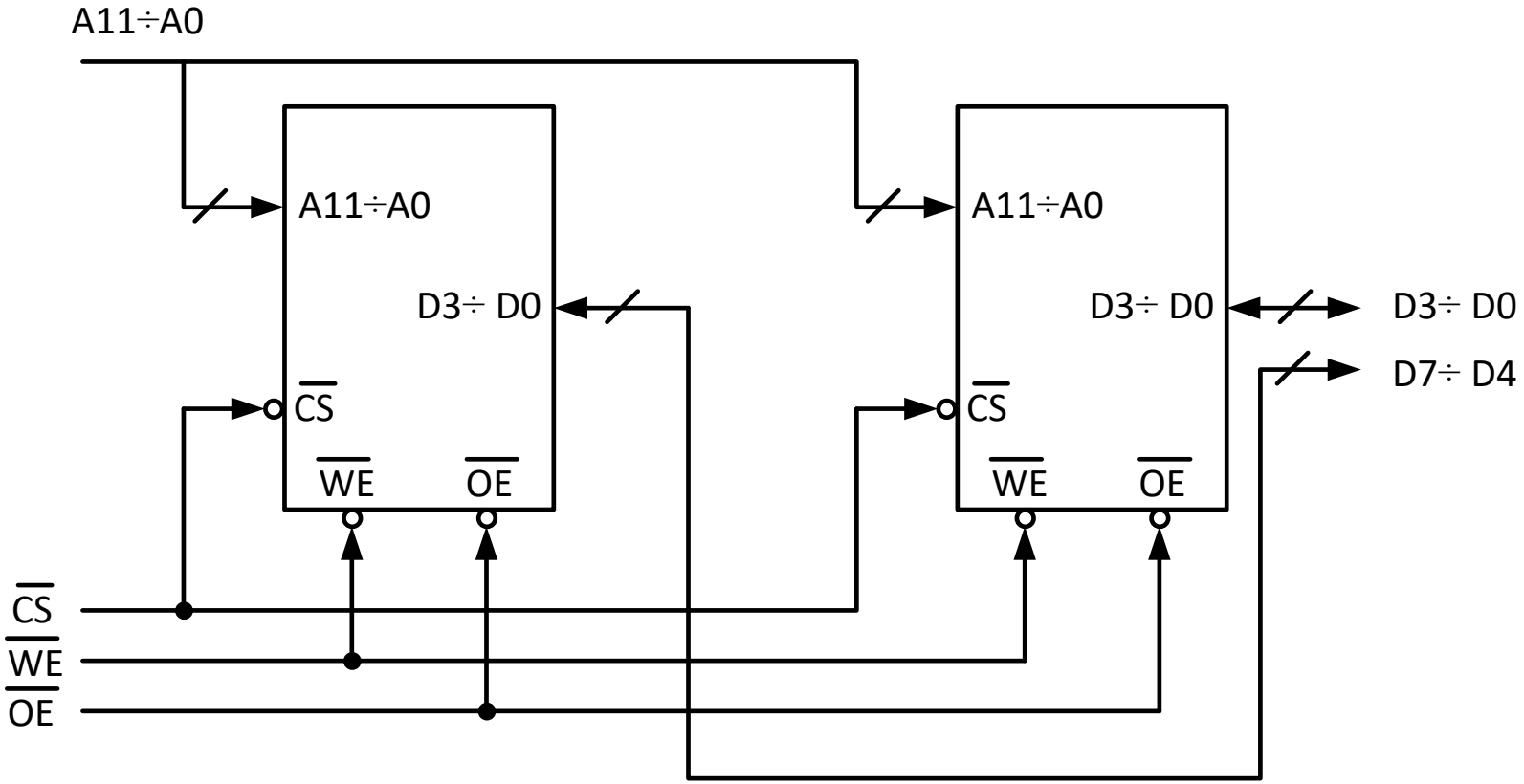
VD1:

- Cho chip nhớ SRAM 4K x 4 bit
- Thiết kế mô-đun nhớ 4K x 8 bit

Giải:

- Dung lượng chip nhớ = 2^{12} x 4 bit
- chip nhớ có:
  - 12 chân địa chỉ
  - 4 chân dữ liệu
- mô-đun nhớ cần có:
  - 12 chân địa chỉ
  - 8 chân dữ liệu

Sơ đồ ví dụ tăng độ dài từ nhớ



Bài toán tăng độ dài từ nhớ tổng quát

- Cho chip nhớ 2^n x mbit
- Thiết kế mô-đun nhớ 2^n x (k.m) bit
- Dùng k chip nhớ

Tăng số lượng từ nhớ

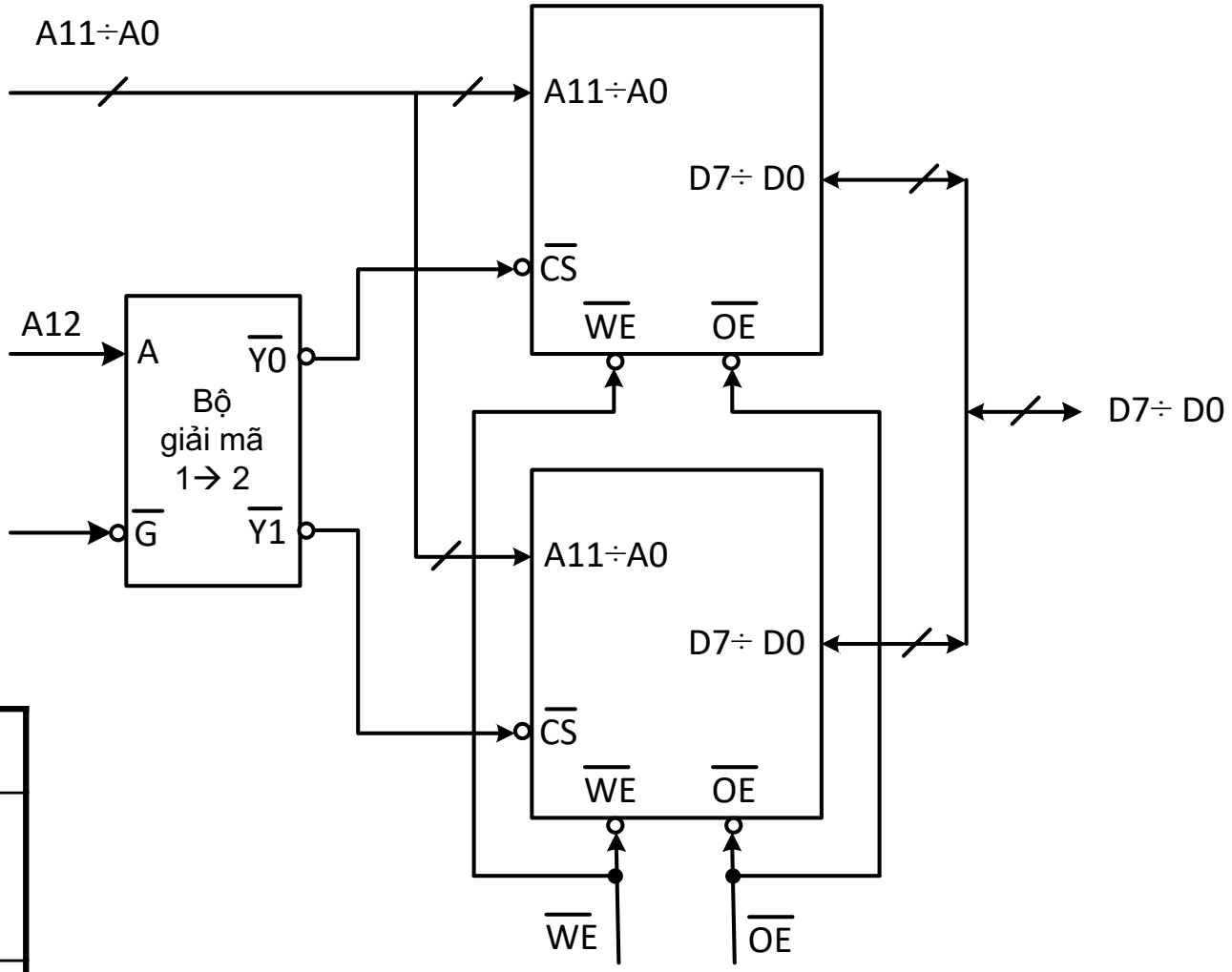
VD2:

- Cho chip nhớ SRAM 4K x 8 bit
- Thiết kế mô-đun nhớ 8K x 8 bit

Giải:

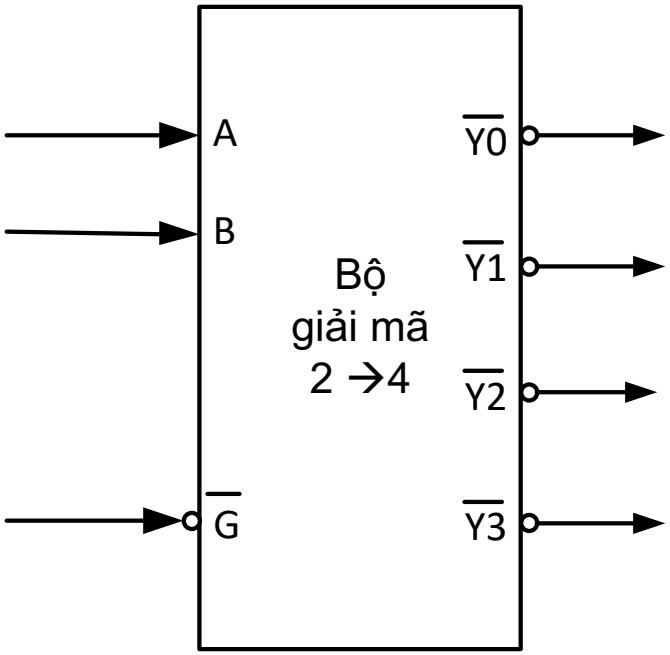
- Dung lượng chip nhớ = 2^{12} x 8 bit
- chip nhớ có:
  - 12 chân địa chỉ
  - 8 chân dữ liệu
- Dung lượng mô-đun nhớ = 2^{13} x 8 bit
  - 13 chân địa chỉ
  - 8 chân dữ liệu

Sơ đồ ví dụ tăng số lượng từ nhớ



$\overline{G}$	A	$\overline{Y0}$	$\overline{Y1}$
0	0	0	1
0	1	1	0
1	x	1	1

Bộ giải mã 2→4



$\overline{G}$	B	A	$\overline{Y0}$	$\overline{Y1}$	$\overline{Y2}$	$\overline{Y3}$
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0
1	x	x	1	1	1	1

Thiết kế kết hợp

Ví dụ 3:

- Cho chip nhớ SRAM 4K x 4 bit
- Thiết kế mô-đun nhớ 8K x 8 bit

Phương pháp thực hiện:

- Kết hợp ví dụ 1 và ví dụ 2
- Tự vẽ sơ đồ thiết kế

2. Các đặc trưng cơ bản của bộ nhớ chính

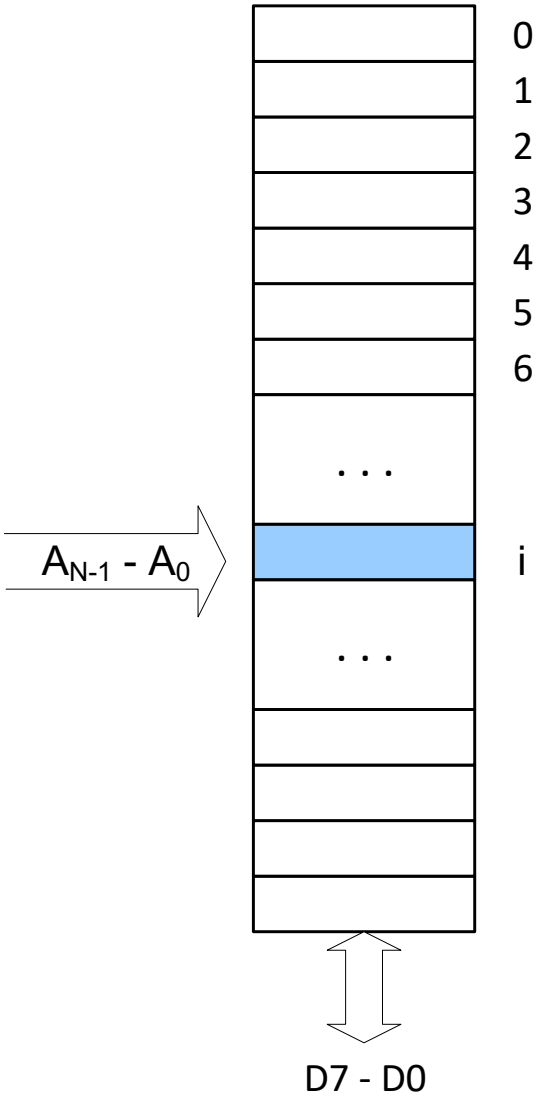
- Chứa các chương trình đang thực hiện và các dữ liệu đang được sử dụng
- Tồn tại trên mọi hệ thống máy tính
- Bao gồm các ngăn nhớ được đánh địa chỉ trực tiếp bởi CPU
- Dung lượng của bộ nhớ chính nhỏ hơn không gian địa chỉ bộ nhớ mà CPU quản lý.
- Việc quản lý logic bộ nhớ chính tuỳ thuộc vào hệ điều hành

Tổ chức bộ nhớ đan xen (interleaved memory)

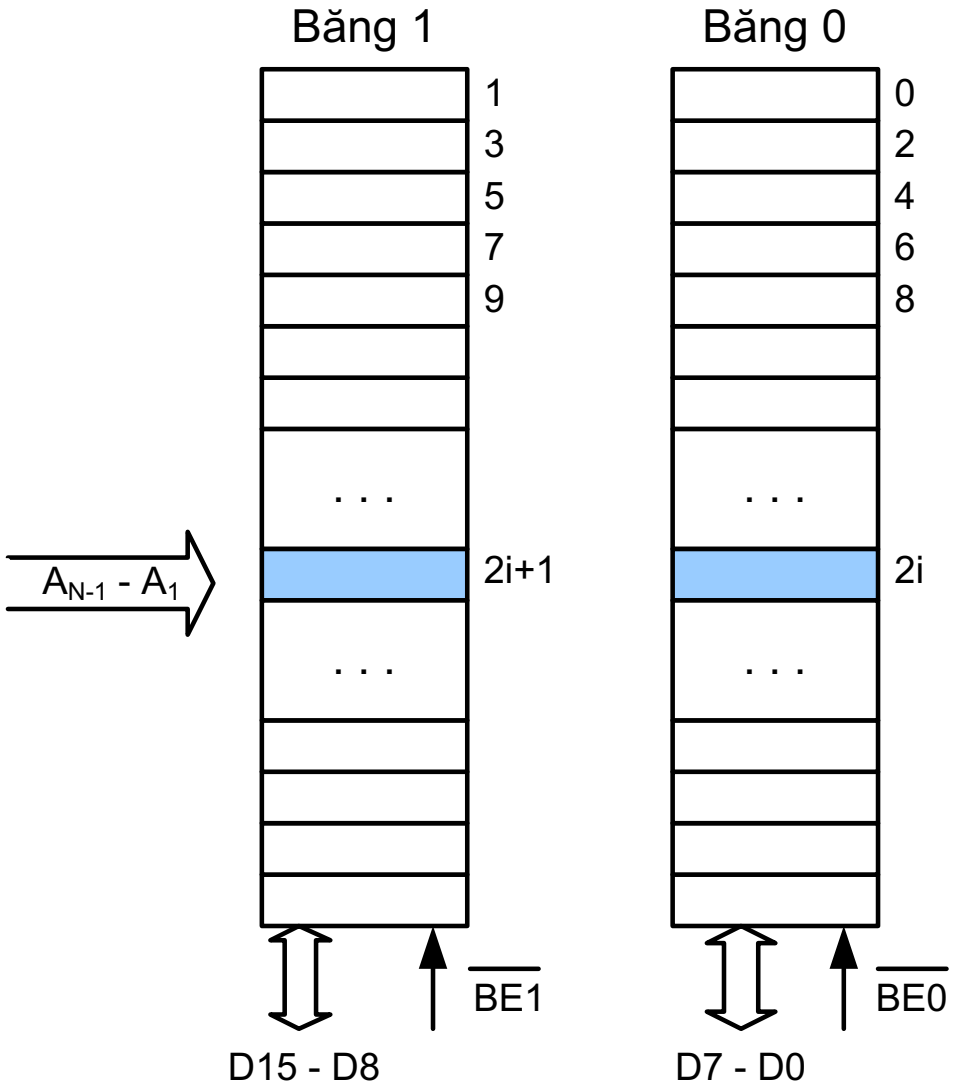
- Độ rộng của bus dữ liệu để trao đổi với bộ nhớ: m = 8, 16, 32, 64,128 ... bit
- Các ngăn nhớ được tổ chức theo byte

→ tổ chức bộ nhớ vật lý khác nhau

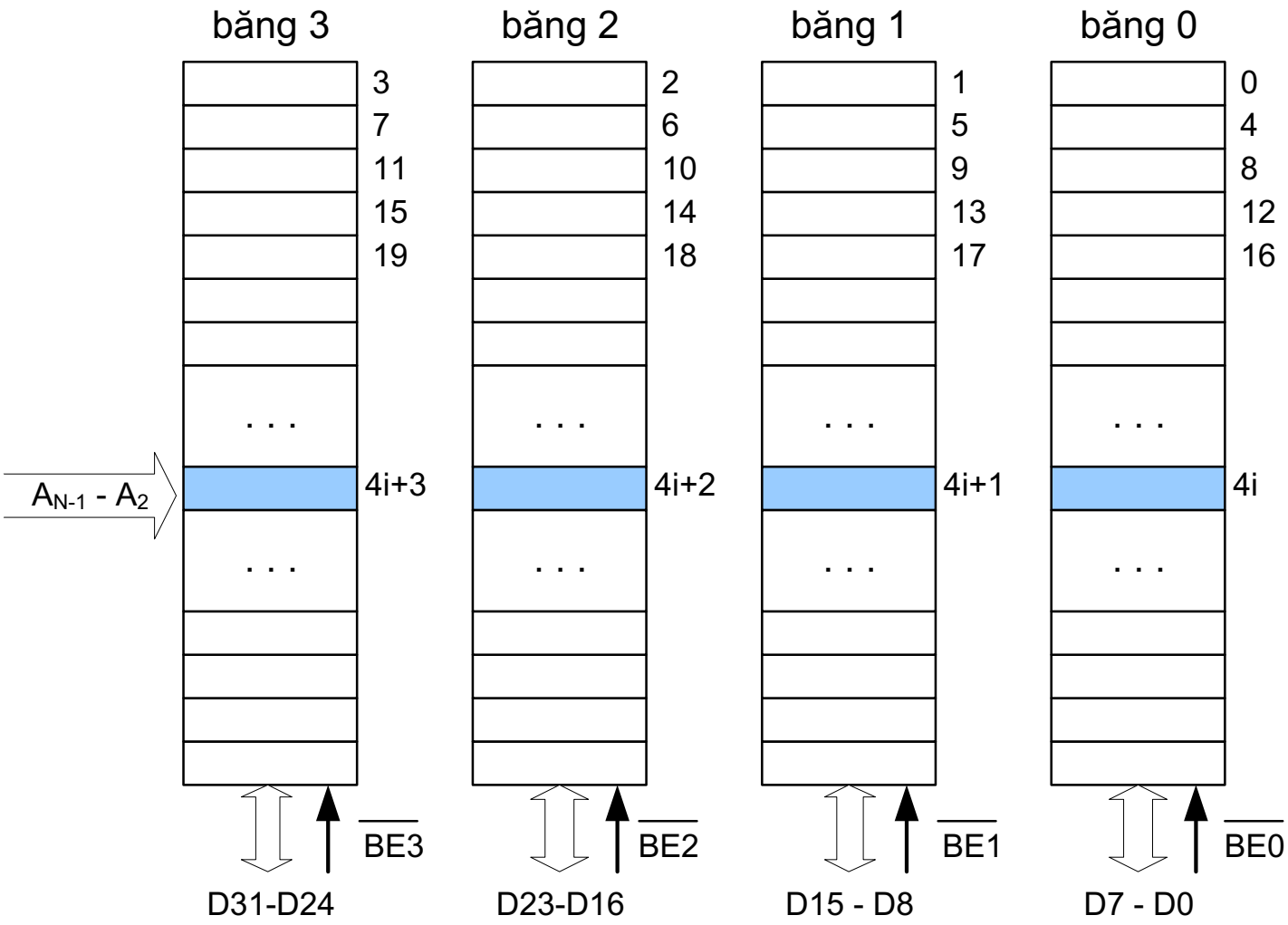
m=8bit → một băng nhớ tuyến tính



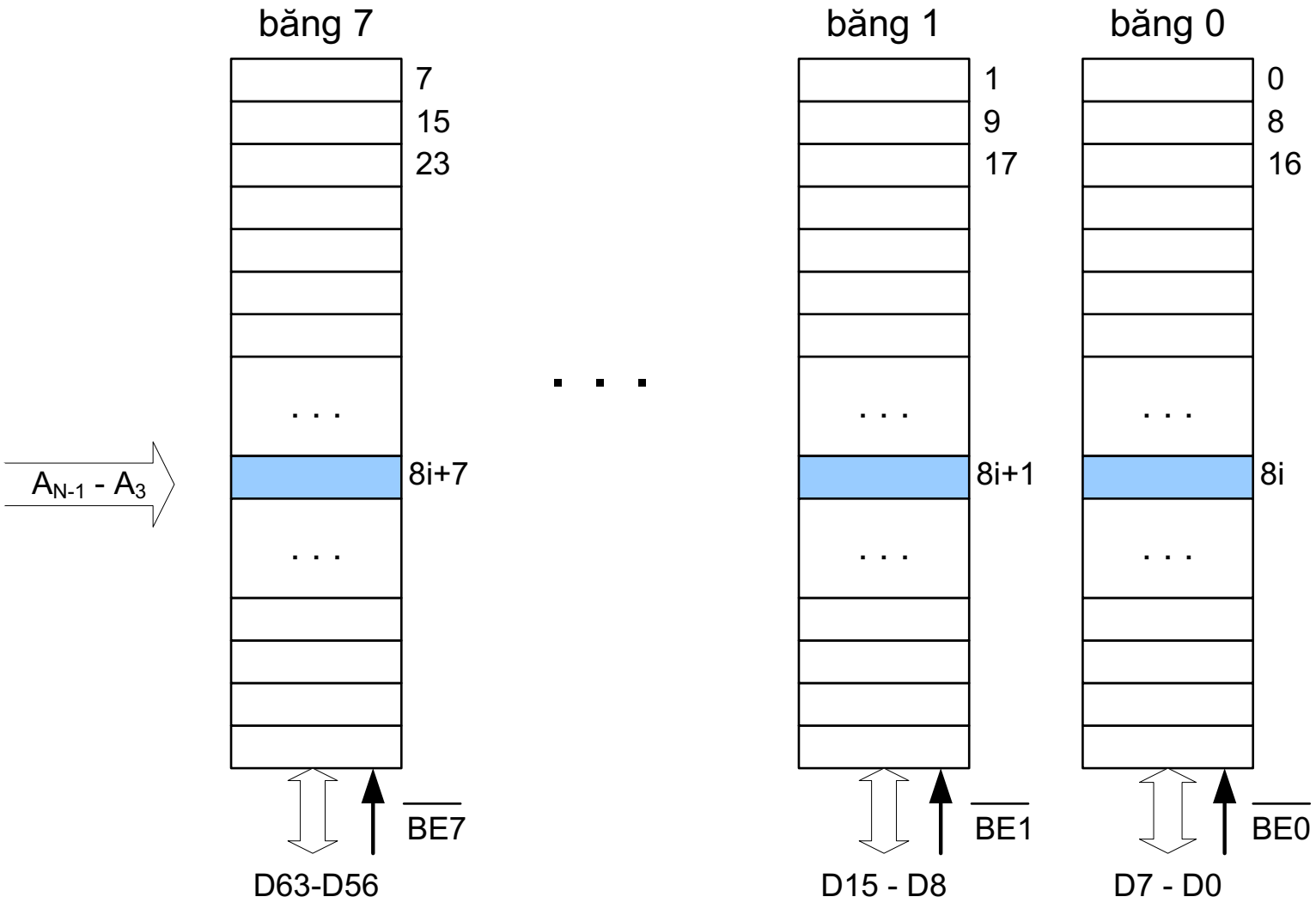
m = 16bit → hai băng nhớ đan xen



m = 32bit → bốn băng nhớ đan xen



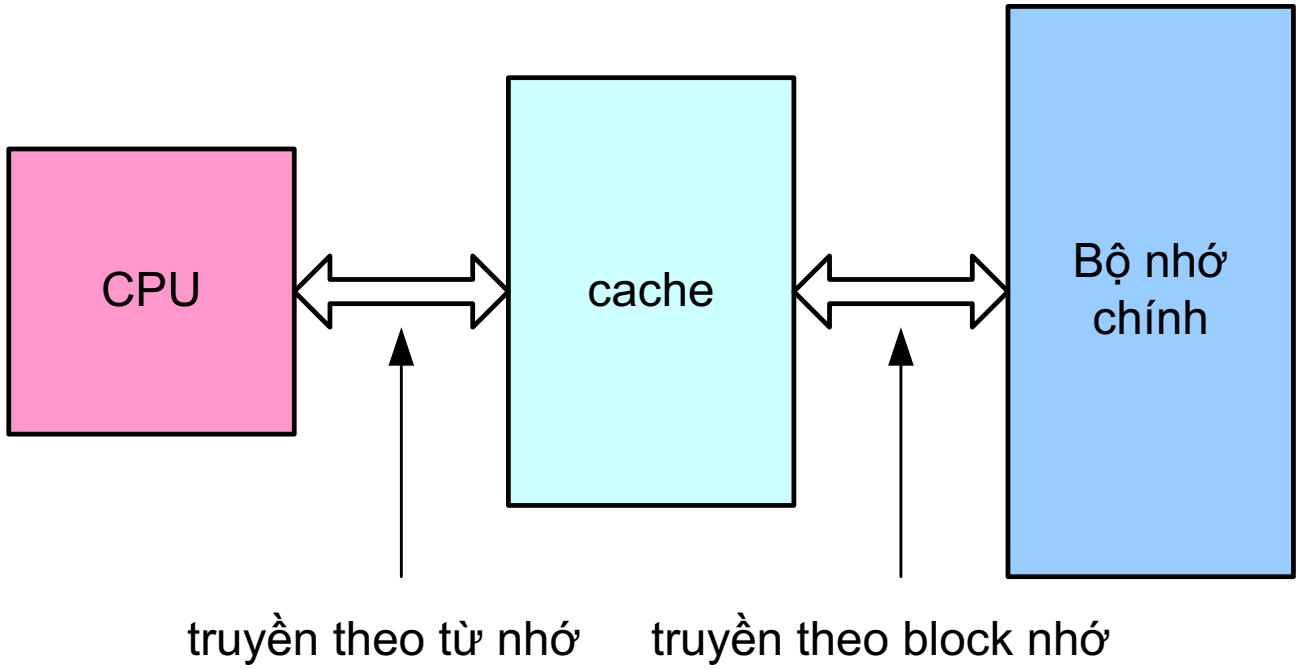
m = 64bit → tám băng nhớ đan xen



7.3. Bộ nhớ đệm (cache memory)

1. Nguyên tắc chung của cache

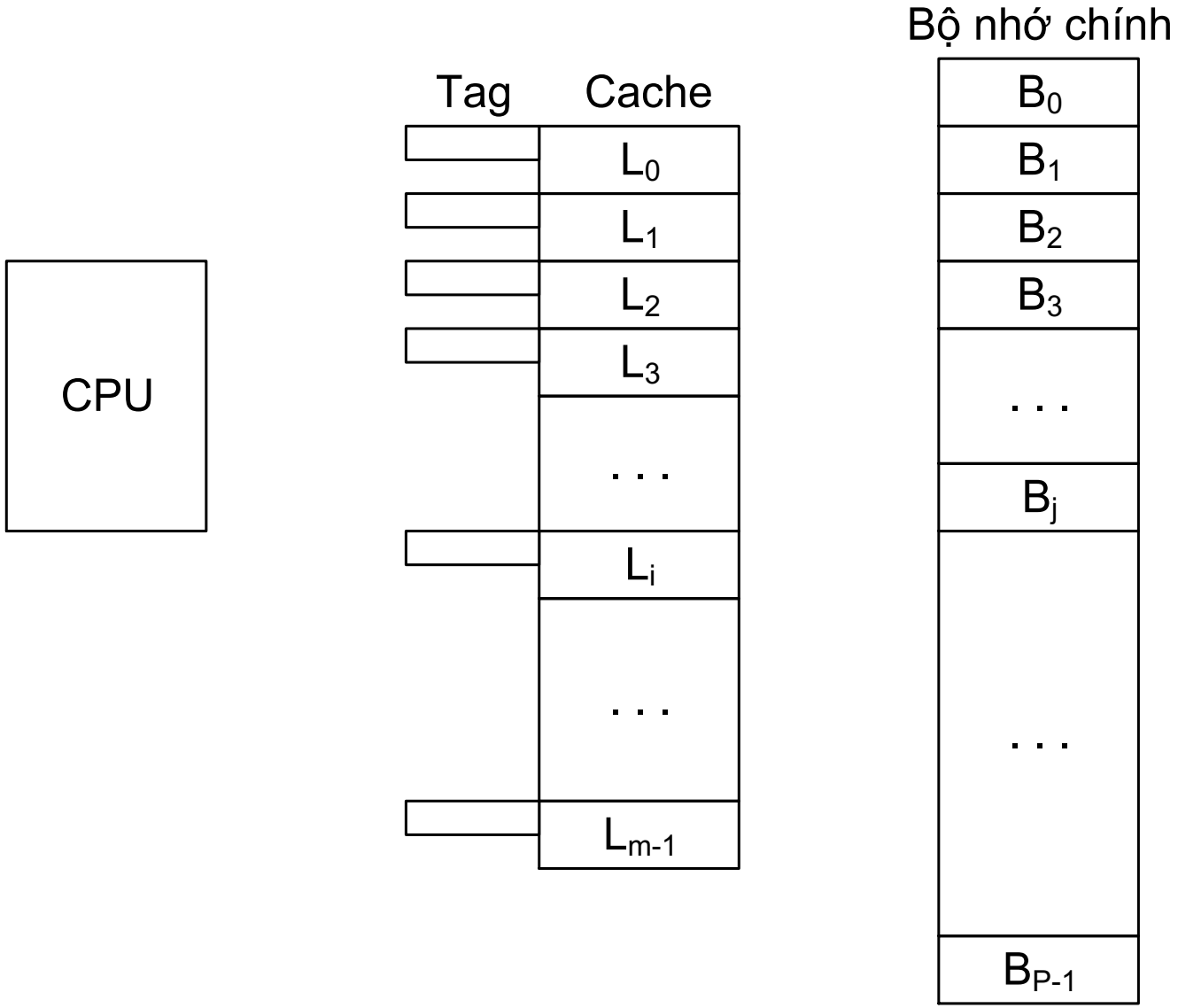
- Cache có tốc độ nhanh hơn bộ nhớ chính
- Cache được đặt giữa CPU và bộ nhớ chính nhằm tăng tốc độ CPU truy cập bộ nhớ
- Cache có thể được đặt trên chip CPU



Ví dụ về thao tác của cache

- CPU yêu cầu nội dung của ngăn nhớ
- CPU kiểm tra trên cache với dữ liệu này
- Nếu có, CPU nhận dữ liệu từ cache (nhanh)
- Nếu không có, đọc Block nhớ chứa dữ liệu từ bộ nhớ chính vào cache
- Tiếp đó chuyển dữ liệu từ cache vào CPU

Cấu trúc chung của cache / bộ nhớ chính



Cấu trúc chung của cache / bộ nhớ chính (tiếp)

- Bộ nhớ chính có 2^N byte nhớ
- Bộ nhớ chính và cache được chia thành các khối có kích thước bằng nhau
  - Bộ nhớ chính: $B_0, B_1, B_2, \ldots, B_{p-1}$ (p Blocks)
  - Bộ nhớ cache: $L_0, L_1, L_2, \ldots, L_{m-1}$ (m Lines)
  - Kích thước của Block (Line) = 8,16,32,64,128 byte
- Mỗi Line trong cache có một thẻ nhớ (Tag) được gắn vào

Cấu trúc chung của cache / bộ nhớ chính (tiếp)

- Một số Block của bộ nhớ chính được nạp vào các Line của cache
- Nội dung Tag (thẻ nhớ) cho biết Block nào của bộ nhớ chính hiện đang được chứa ở Line đó
- Nội dung Tag được cập nhật mỗi khi Block từ bộ nhớ chính nạp vào Line đó
- Khi CPU truy nhập (đọc/ghi) một từ nhớ, có hai khả năng xảy ra:
  - Từ nhớ đó có trong cache (cache hit)
  - Từ nhớ đó không có trong cache (cache miss).

2. Các phương pháp ánh xạ

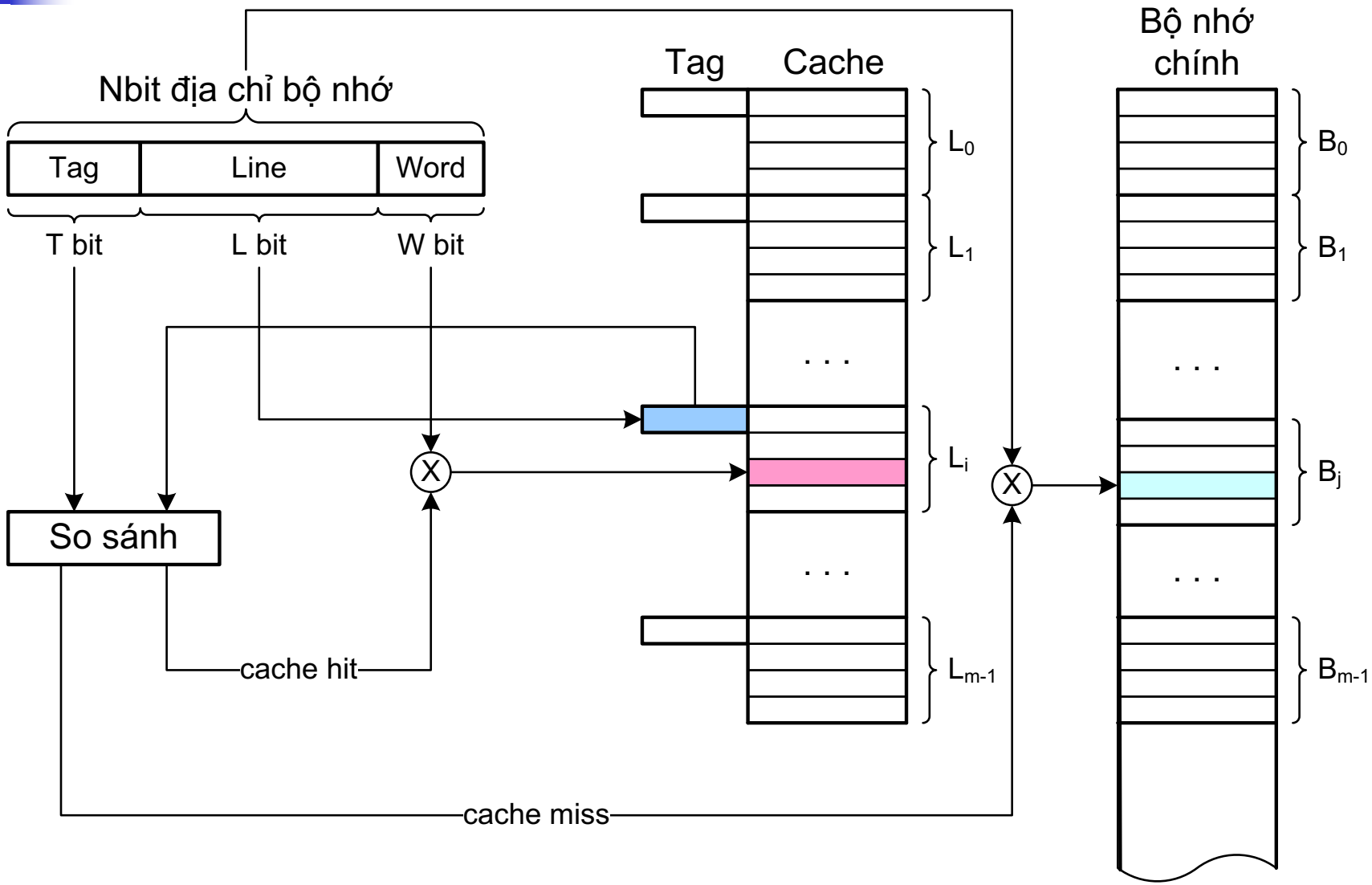
(Chính là các phương pháp tổ chức bộ nhớ cache)

- Ánh xạ trực tiếp
  (Direct mapping)
- Ánh xạ liên kết toàn phần
  (Fully associative mapping)
- Ánh xạ liên kết tập hợp
  (Set associative mapping)

Ánh xạ trực tiếp

- Mỗi Block của bộ nhớ chính chỉ có thể được nạp vào một Line của cache:
  - $B_0 \rightarrow L_0$
  - $B_1 \rightarrow L_1$
  - ....
  - $B_{m-1} \rightarrow L_{m-1}$
  - $B_m \rightarrow L_0$
  - $B_{m+1} \rightarrow L_1$
  - ....
- Tổng quát
  - B_j chỉ có thể nạp vào $L_{j \bmod m}$
  - m là số *Line* của *cache*.

Ánh xạ trực tiếp (tiếp)



Ánh xạ trực tiếp (tiếp)

- Địa chỉ N bit của bộ nhớ chính chia thành ba trường:
  - Trường Word gồm W bit xác định một từ nhớ trong *Block* hay *Line:*

    2^W = kích thước của *Block* hay *Line*
  - Trường *Line* gồm L bit xác định một trong số các *Line* trong *cache:*

    2^L = số *Line* trong *cache* = m
  - Trường *Tag* gồm T bit:

    T = N - (W+L)

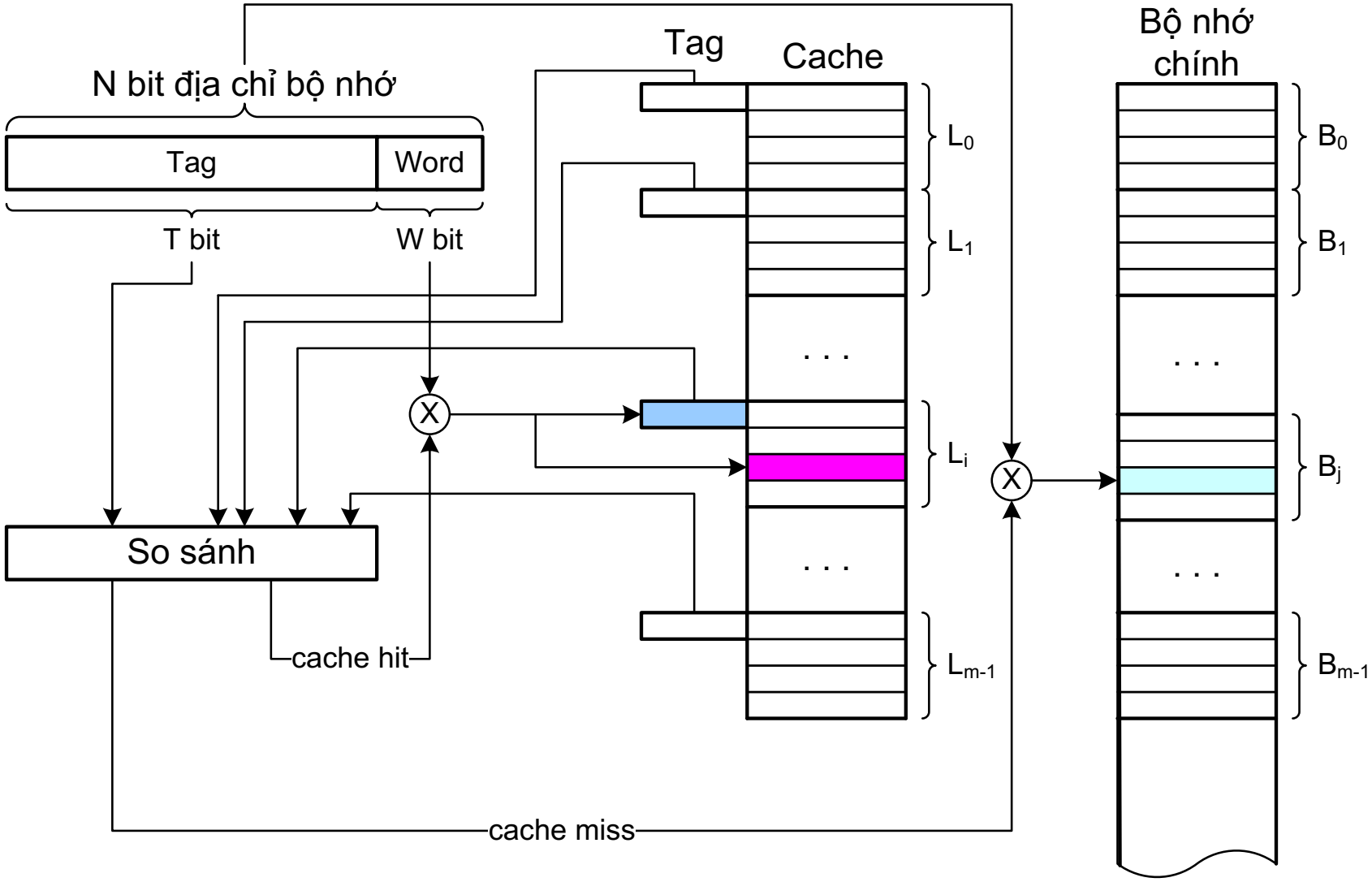
Ánh xạ trực tiếp (tiếp)

- Mỗi thẻ nhớ (Tag) của một Line chứa được T bit
- Khi Block từ bộ nhớ chính được nạp vào Line của cache thì Tag ở đó được cập nhật giá trị là T bit địa chỉ bên trái của Block đó
- Khi CPU muốn truy nhập một từ nhớ thì nó phát ra một địa chỉ N bit cụ thể
  - Nhờ vào giá trị L bit của trường Line sẽ tìm ra Line tương ứng
  - Đọc nội dung Tag ở Line đó (T bit), rồi so sánh với T bit bên trái của địa chỉ vừa phát ra
    - Giống nhau: cache hit
    - Khác nhau: cache miss
- Ưu điểm: Bộ so sánh đơn giản
- Nhược điểm: Xác suất *cache hit* thấp

Ánh xạ liên kết toàn phần

- Mỗi *Block* có thể nạp vào bất kỳ *Line* nào của *cache*
- Địa chỉ của bộ nhớ chính chia thành hai trường:
  - Trường Word
  - Trường *Tag* dùng để xác định *Block* của bộ nhớ chính
- Tag xác định Block đang nằm ở Line đó

Ánh xạ liên kết toàn phần (tiếp)



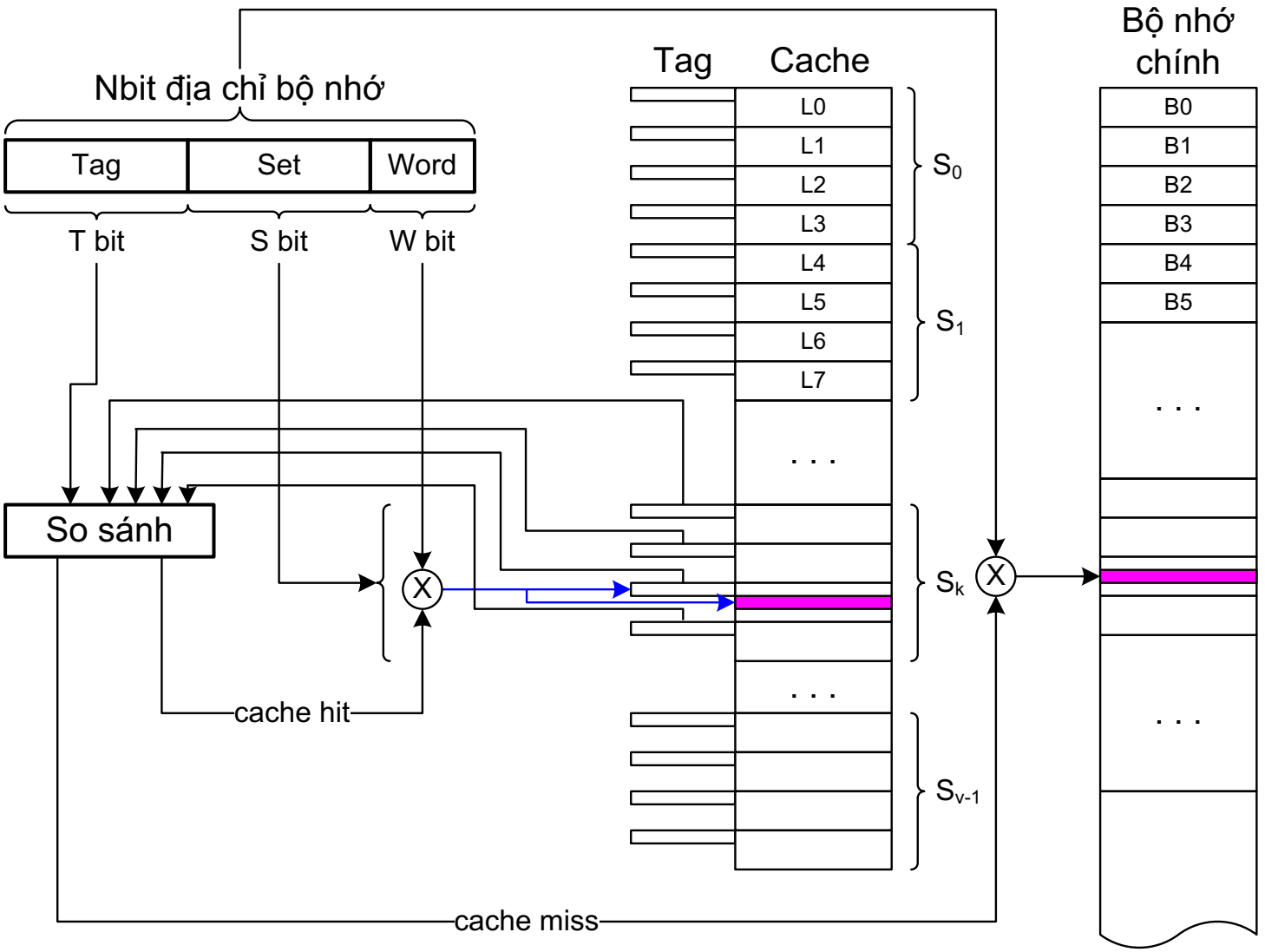
Ánh xạ liên kết toàn phần (tiếp)

- Mỗi thẻ nhớ (Tag) của một Line chứa được T bit
- Khi Block từ bộ nhớ chính được nạp vào Line của cache thì Tag ở đó được cập nhật giá trị là T bit địa chỉ bên trái của Block đó
- Khi CPU muốn truy nhập một từ nhớ thì nó phát ra một địa chỉ N bit cụ thể
  - So sánh T bit bên trái của địa chỉ vừa phát ra với lần lượt nội dung của các Tag trong cache
    - Nếu gặp giá trị bằng nhau: cache hit xảy ra ở Line đó
    - Nếu không có giá trị nào bằng: cache miss
- Ưu điểm: Xác suất *cache hit* cao
- Nhược điểm:
  - So sánh đồng thời với tất cả các Tag → mất nhiều thời gian
  - Bộ so sánh phức tạp
- Ít sử dụng

Ánh xạ liên kết tập hợp

- Dung hòa cho hai phương pháp trên
- Cache được chia thành các Tập (Set)
- Mỗi một Set chứa một số Line
- Ví dụ:
  - 4 Line/Set → 4-way associative mapping
- Ánh xạ theo nguyên tắc sau:
  - $B_0 \rightarrow S_0$
  - $B_1 \rightarrow S_1$
  - $B_2 \rightarrow S_2$
  - .....

Ánh xạ liên kết tập hợp (tiếp)



Ánh xạ liên kết tập hợp (tiếp)

- Kích thước *Block* = 2^W Word
- Trường *Set* có S bit dùng để xác định một trong số các Set trong cache. 2^S = Số *Set trong cache*
- Trường *Tag* có T bit: T = N - (W+S)
- Khi CPU muốn truy nhập một từ nhớ thì nó phát ra một địa chỉ N bit cụ thể
  - Nhờ vào giá trị S bit của trường Set sẽ tìm ra Set tương ứng
  - So sánh T bit bên trái của địa chỉ vừa phát ra với lần lượt nội dung của các Tag trong Set đó
    - Nếu gặp giá trị bằng nhau: cache hit xảy ra ở Line tương ứng
    - Nếu không có giá trị nào bằng: cache miss
- Tổng quát cho cả hai phương pháp trên
- Thông dụng với: 2,4,8,16Lines/Set

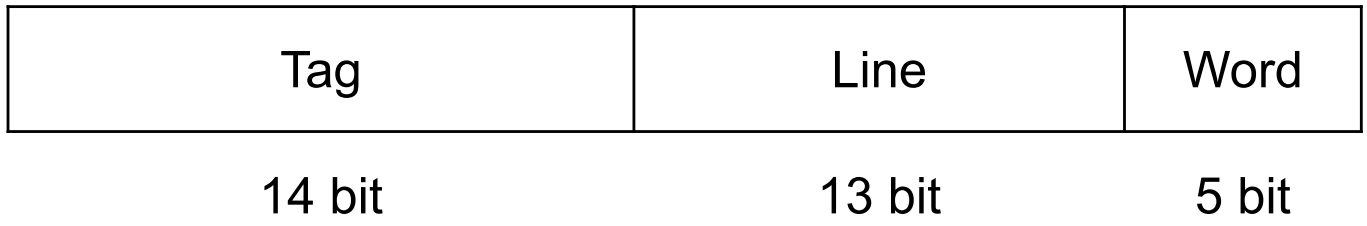
Ví dụ về ánh xạ địa chỉ

- Giả sử máy tính đánh địa chỉ cho từng byte
- Không gian địa chỉ bộ nhớ chính = 4GiB
- Dung lượng bộ nhớ *cache* là 256KiB
- Kích thước *Line* (*Block*) = 32byte.
- Xác định số bit của các trường địa chỉ cho ba trường hợp tổ chức:
  - Ánh xạ trực tiếp
  - Ánh xạ liên kết toàn phần
  - Ánh xạ liên kết tập hợp 4 đường

Với ánh xạ trực tiếp

- Bộ nhớ chính = 4GiB = 2^{32} byte → Số bit địa chỉ của bộ nhớ chính là: N = 32 bit
- *Cache* = 256 KiB = 2^{18} byte
- Kích thước *Line* = 32 byte = 2^5 byte → số bit địa chỉ của trường Word là: W = 5 bit
- Số *Line* trong *cache* = $2^{18}/ 2^5 = 2^{13}$ *Line* → số bit địa chỉ trường Line là: L = 13 bit
- Số bit địa chỉ của trường Tag là:

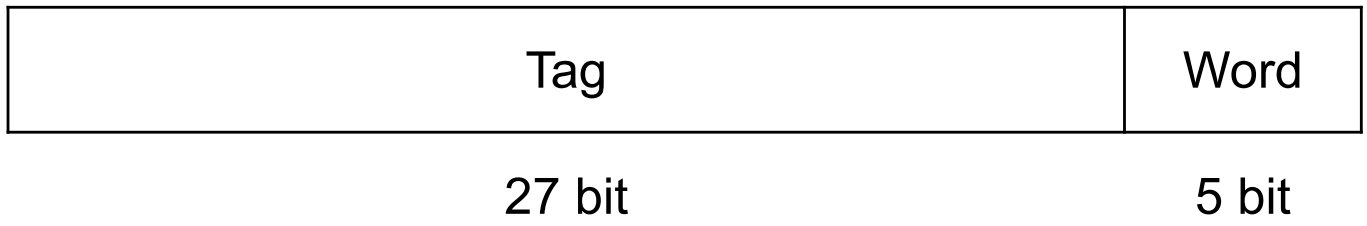
  T = 32 - (13 + 5) = 14 bit



Với ánh xạ liên kết toàn phần

- Bộ nhớ chính = 4GiB = 2^{32} byte → số bit địa chỉ của bộ nhớ chính là: N = 32 bit
- Kích thước *Line* = 32 byte = 2^5 byte → số bit địa chỉ của trường Word là: W = 5 bit
- Số bit địa chỉ của trường *Tag* là:

T = 32 - 5 = 27 bit



Với ánh xạ liên kết tập hợp 4 đường

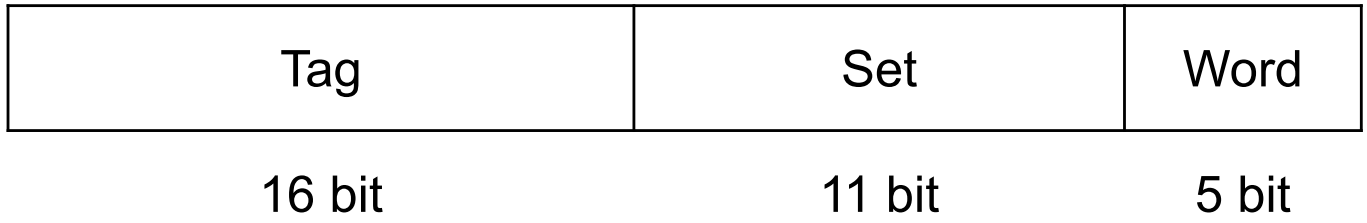
- Bộ nhớ chính = 4GiB = 2^{32} byte → số bit địa chỉ của bộ nhớ chính là: N = 32 bit
- Kích thước *Line* = 32 byte = 2^5 byte → số bit địa chỉ của trường Word là: W = 5 bit
- Số *Line* trong *cache* = $2^{18}/ 2^5 = 2^{13}$ *Line*
- Một *Set* có 4 *Line* = 2^2 *Line*

→ số *Set* trong *cache* = $2^{13}/ 2^2 = 2^{11}$ *Set*

→ số bit địa chỉ của trường Set là: S = 11 bit

- Số bit địa chỉ của trường *Tag* là:

T = 32 - (11 + 5) = 16 bit



3. Thay thế block trong cache

Với ánh xạ trực tiếp:

- Không phải lựa chọn
- Mỗi Block chỉ ánh xạ vào một Line xác định
- Thay thế Block ở Line đó

Thay thế block trong cache (tiếp)

Với ánh xạ liên kết: cần có thuật giải thay thế:

- Random: Thay thế ngẫu nhiên
- FIFO (First In First Out): Thay thế *Block* nào nằm lâu nhất ở trong *Set* đó
- LFU (Least Frequently Used): Thay thế *Block* nào trong *Set* có số lần truy nhập ít nhất trong cùng một khoảng thời gian
- LRU (Least Recently Used): Thay thế Block ở trong *Set* tương ứng có thời gian lâu nhất không được tham chiếu tới
- Tối ưu nhất: LRU

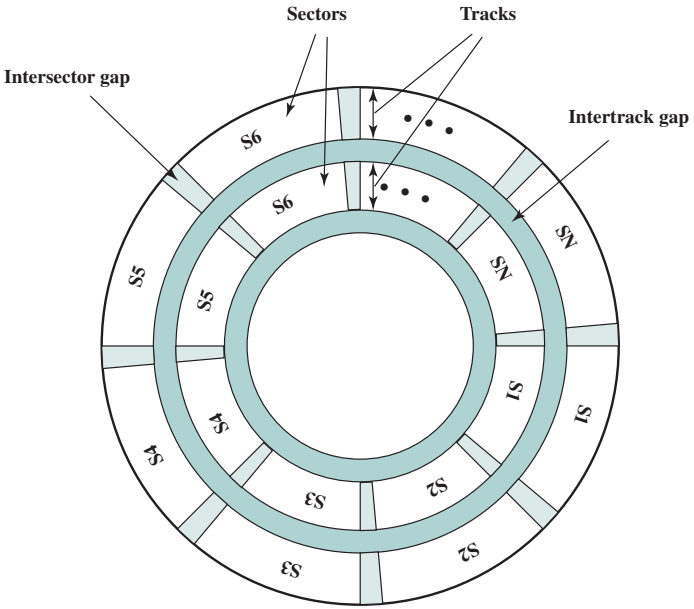
4. Phương pháp ghi dữ liệu khi cache hit

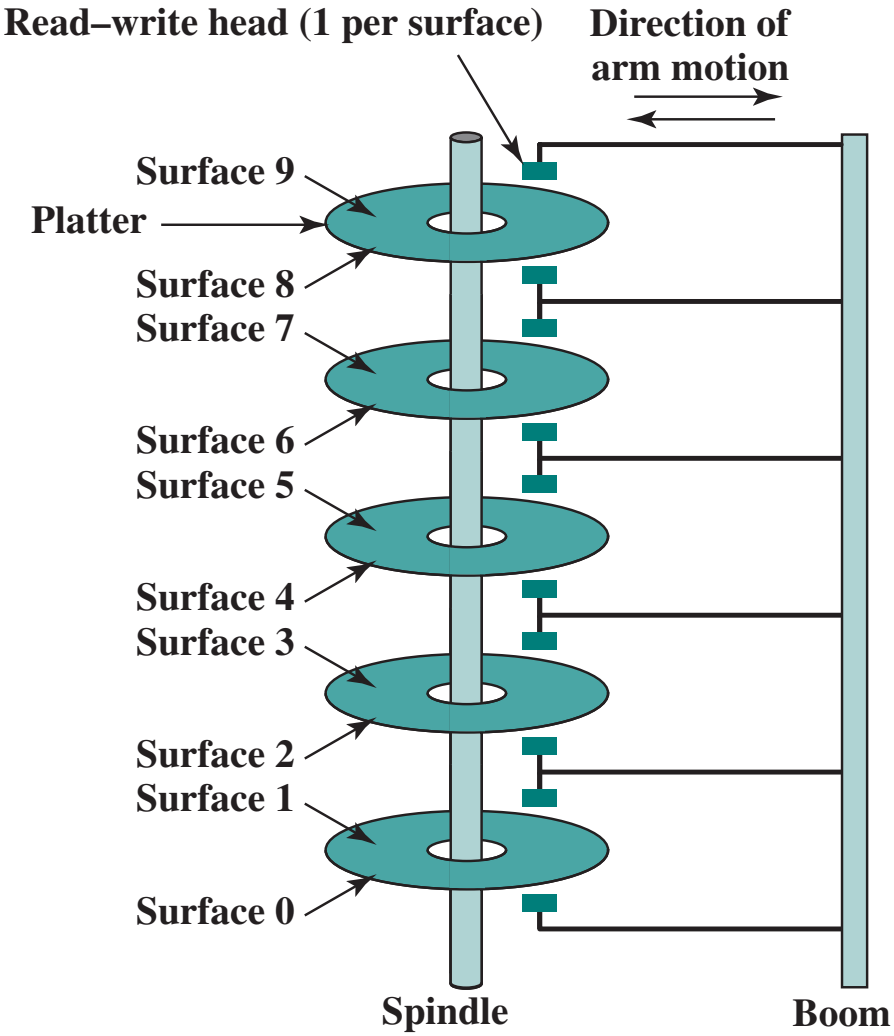
- Ghi xuyên qua (Write-through):
  - ghi cả cache và cả bộ nhớ chính
  - tốc độ chậm
- Ghi trả sau (Write-back):
  - chỉ ghi ra cache
  - tốc độ nhanh
  - khi Block trong cache bị thay thế cần phải ghi trả cả Block về bộ nhớ chính

7.4. Bộ nhớ ngoài

- Tồn tại dưới dạng các thiết bị lưu trữ
- Các kiểu bộ nhớ ngoài
  - Băng từ: ít sử dụng
  - Đĩa từ: Ổ đĩa cứng HDD (Hard Disk Drive)
  - Đĩa quang: CD, DVD
  - Bộ nhớ Flash:
    - Ổ nhớ thể rắn SSD (Solid State Drive)
    - USB flash
    - Thẻ nhớ

Ổ đĩa cứng (HDD –Hard Disk Drive)





HDD

- Dung lượng lớn
- Tốc độ đọc/ghi chậm
- Tốn năng lượng
- Dễ bị lỗi cơ học
- Rẻ tiền

Ổ SSD (Solid State Drive)

- Bộ nhớ bán dẫn flash
- Không khả biến
- Tốc độ nhanh
- Tiêu thụ năng lượng ít
- Gồm nhiều chip nhớ flash và cho phép truy cập song song
- Ít bị lỗi
- Đắt tiền

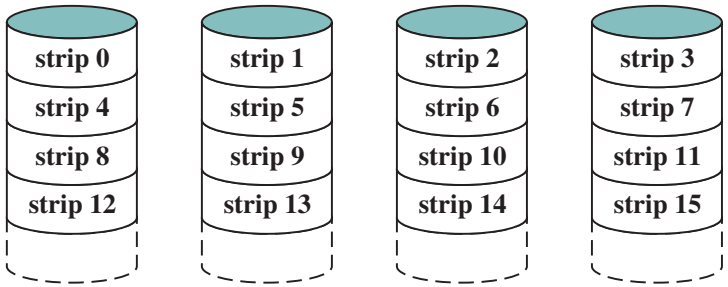
Đĩa quang

- CD (Compact Disc)
  - Dung lượng thông dụng 650MB
- DVD
  - Digital Video Disc hoặc Digital Versatile Disk
  - Ghi một hoặc hai mặt
  - Một hoặc hai lớp trên một mặt
  - Thông dụng: 4,7GB/lớp

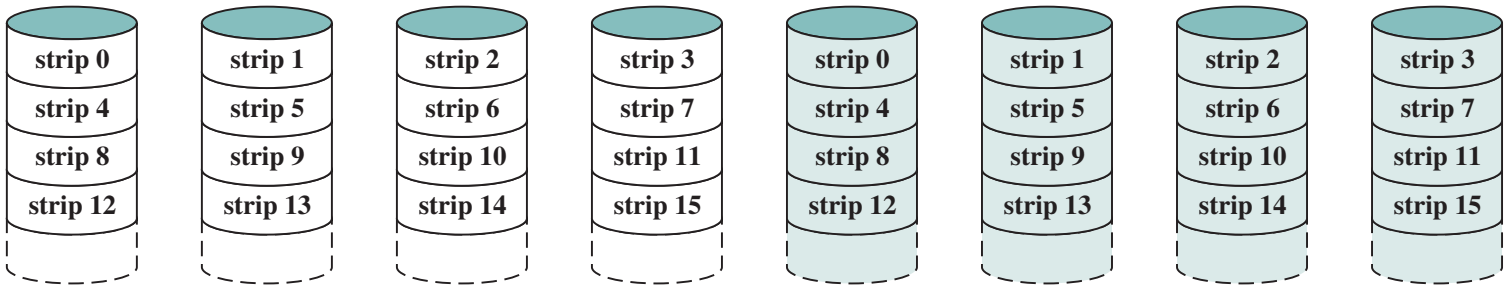
Hệ thống lưu trữ dung lượng lớn: RAID

- Redundant Array of Inexpensive Disks
- (Redundant Array of Independent Disks)
- Tập các ổ đĩa cứng vật lý được OS coi như một ổ logic duy nhất → dung lượng lớn
- Dữ liệu được lưu trữ phân tán trên các ổ đĩa vật lý → truy cập song song (nhanh)
- Lưu trữ thêm thông tin dư thừa, cho phép khôi phục lại thông tin trong trường hợp đĩa bị hỏng → an toàn thông tin
- 7 loại phổ biến (RAID 0 – 6)

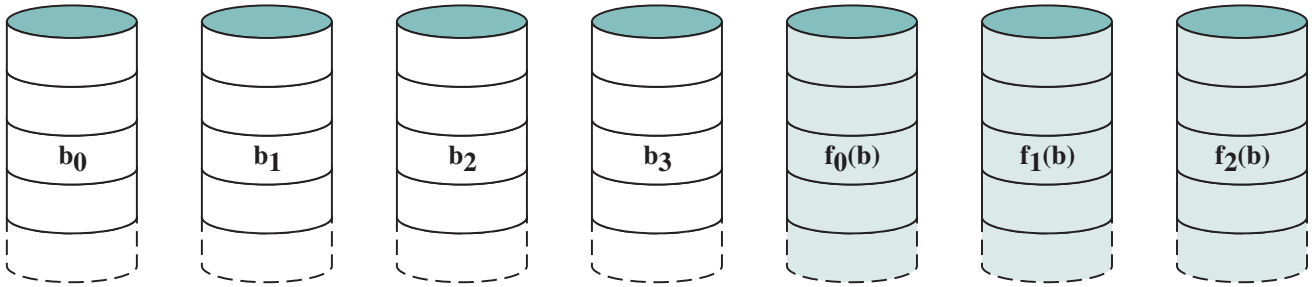
RAID 0, 1, 2



(a) RAID 0 (Nonredundant)

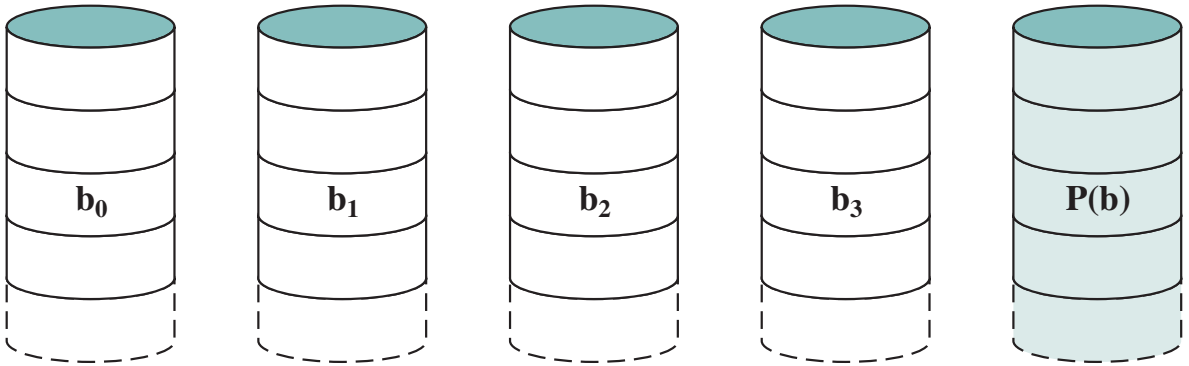


(b) RAID 1 (Mirrored)

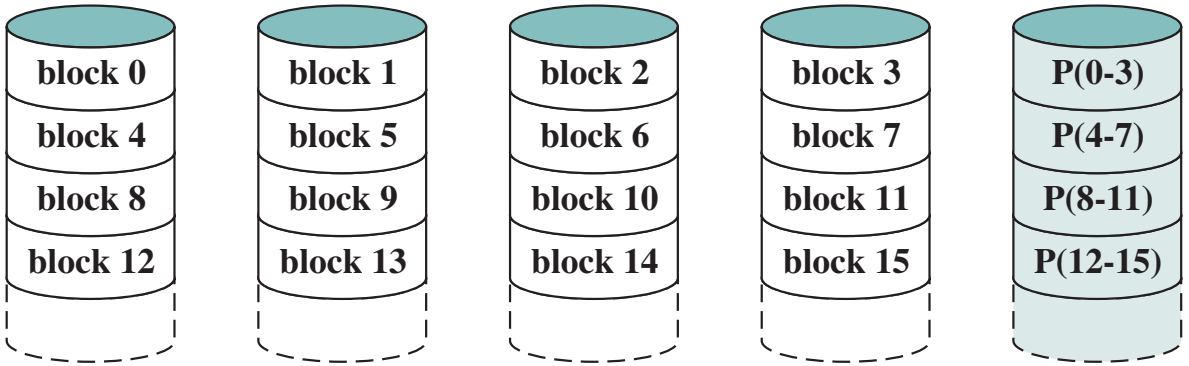


(c) RAID 2 (Redundancy through Hamming code)

RAID 3 & 4

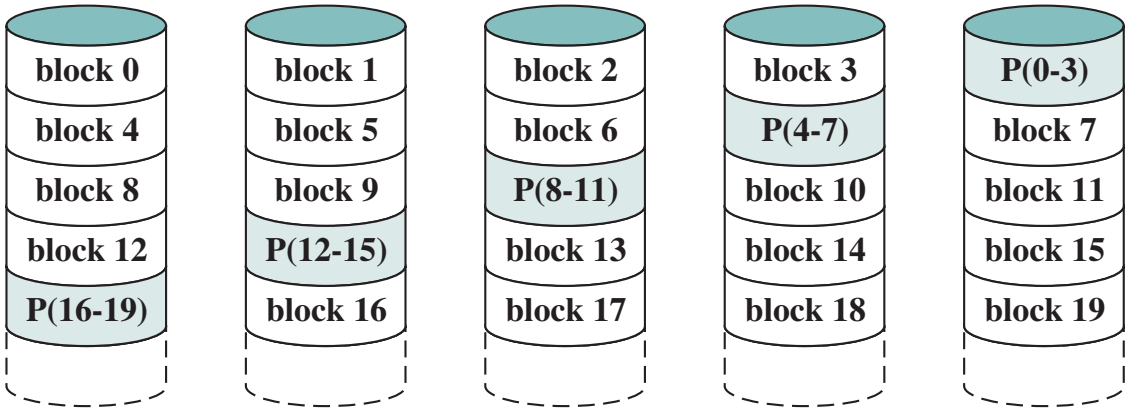


(d) RAID 3 (Bit-interleaved parity)

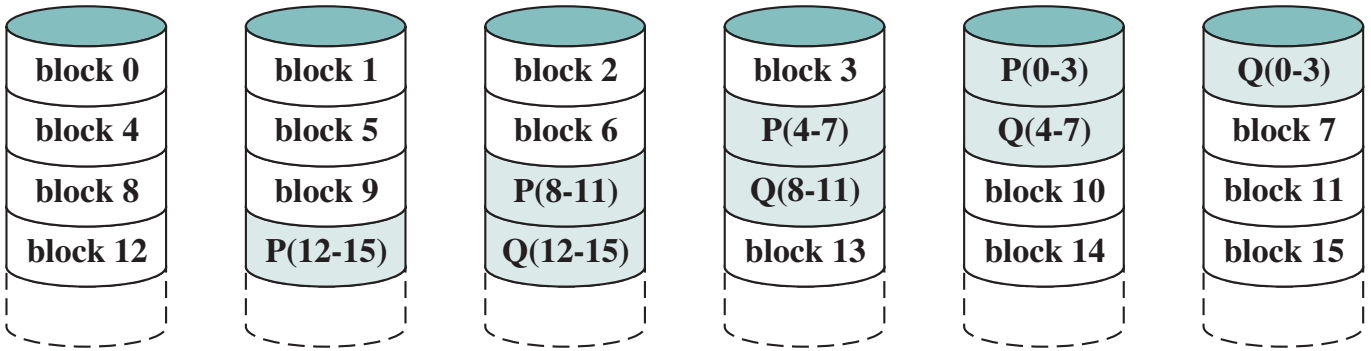


(e) RAID 4 (Block-level parity)

RAID 5 & 6

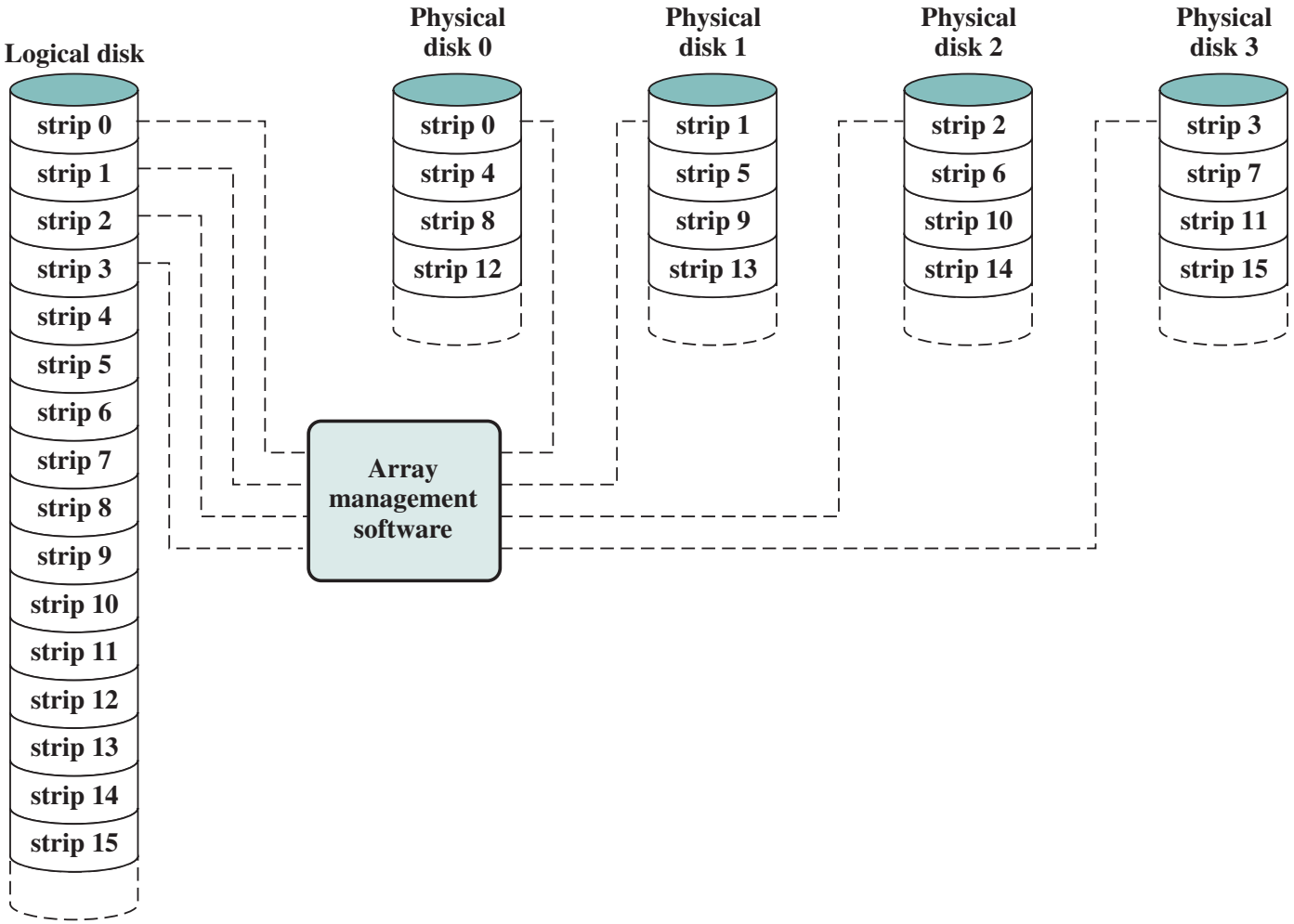


(f) RAID 5 (Block-level distributed parity)



(g) RAID 6 (Dual redundancy)

Ánh xạ dữ liệu của RAID 0



7.5. Bộ nhớ ảo (Virtual Memory)

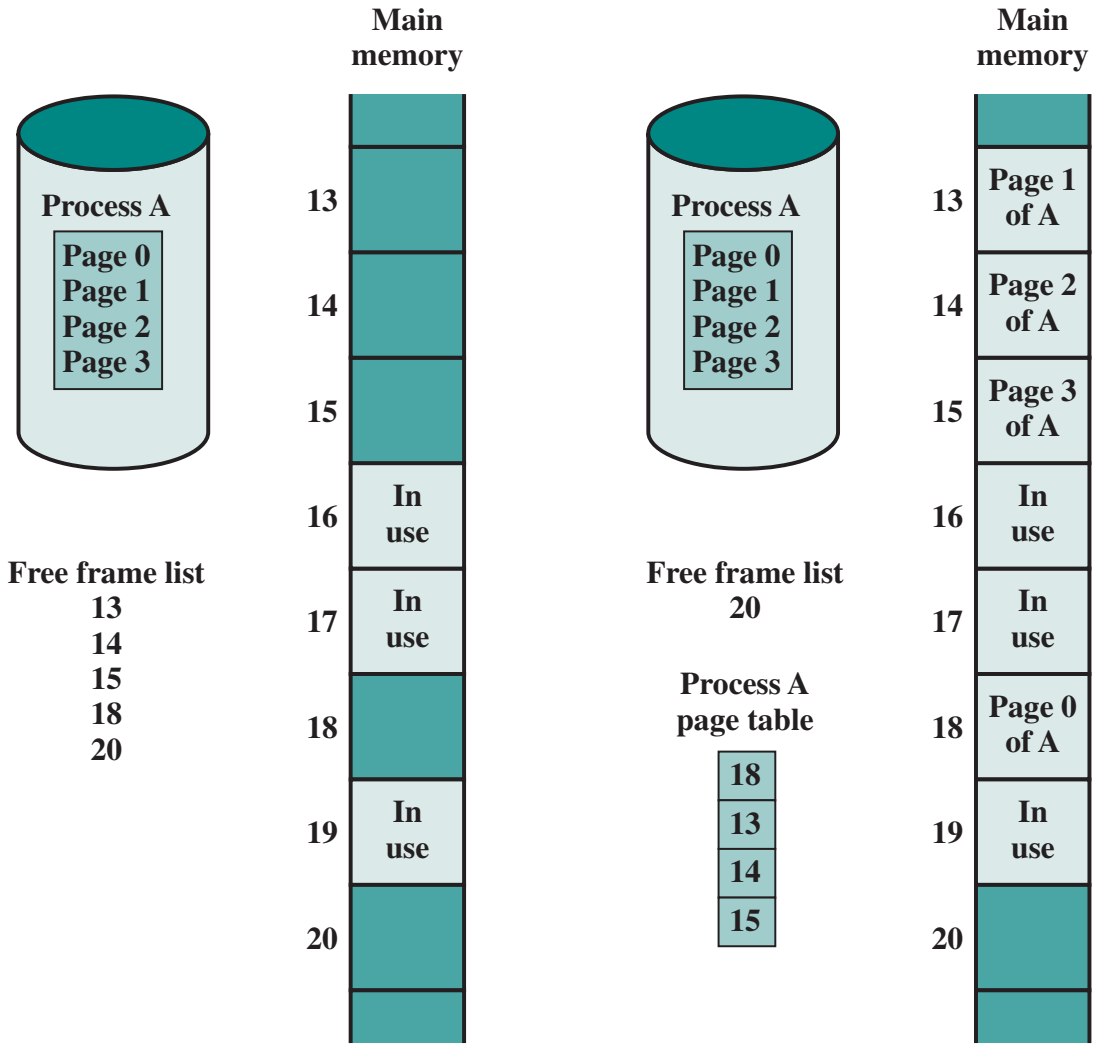
- Khái niệm bộ nhớ ảo: gồm bộ nhớ chính và bộ nhớ ngoài mà được CPU coi như là một bộ nhớ duy nhất (bộ nhớ chính).
- Các kỹ thuật thực hiện bộ nhớ ảo:
  - Kỹ thuật phân trang: Chia không gian địa chỉ bộ nhớ thành các trang nhớ có kích thước bằng nhau và nằm liền kề nhau

    Thông dụng: kích thước trang = 4KiB
  - Kỹ thuật phân đoạn: Chia không gian nhớ thành các đoạn nhớ có kích thước thay đổi, các đoạn nhớ có thể gối lên nhau.

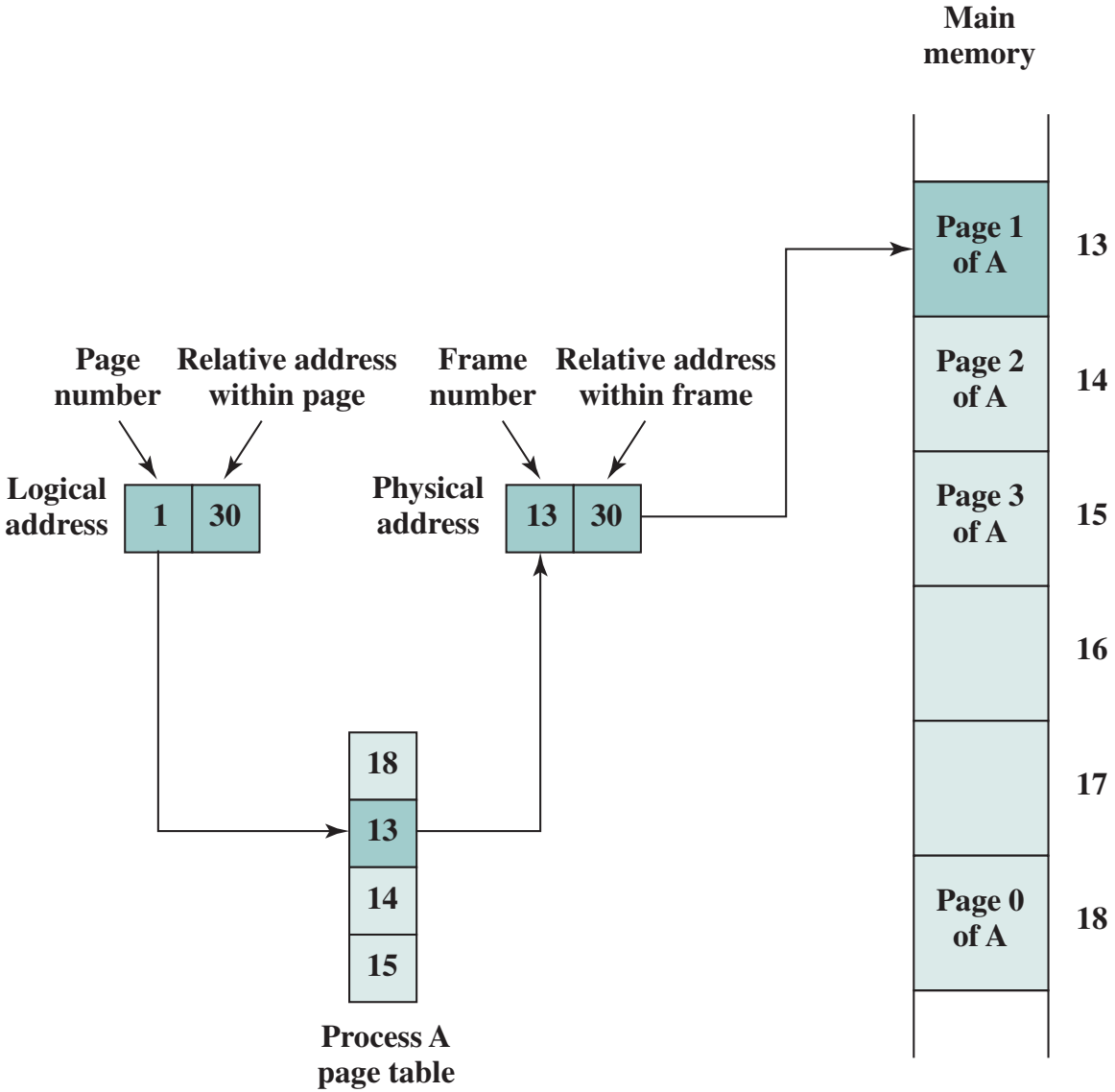
Phân trang

- Phân chia bộ nhớ thành các phần có kích thước bằng nhau gọi là các khung trang
- Chia chương trình (tiến trình) thành các trang
- Cấp phát số hiệu khung trang yêu cầu cho tiến trình
- OS duy trì danh sách các khung trang nhớ trống
- Tiến trình không yêu cầu các khung trang liên tiếp
- Sử dụng bảng trang để quản lý

Cấp phát các khung trang



Địa chỉ logic và địa chỉ vật lý của phân trang



Nguyên tắc làm việc của bộ nhớ ảo phân trang

- Phân trang theo yêu cầu
  - Không yêu cầu tất cả các trang của tiến trình nằm trong bộ nhớ
  - Chỉ nạp vào bộ nhớ những trang được yêu cầu
- Lỗi trang
  - Trang được yêu cầu không có trong bộ nhớ
  - HĐH cần hoán đổi trang yêu cầu vào
  - Có thể cần hoán đổi một trang nào đó ra để lấy chỗ
  - Cần chọn trang để đưa ra

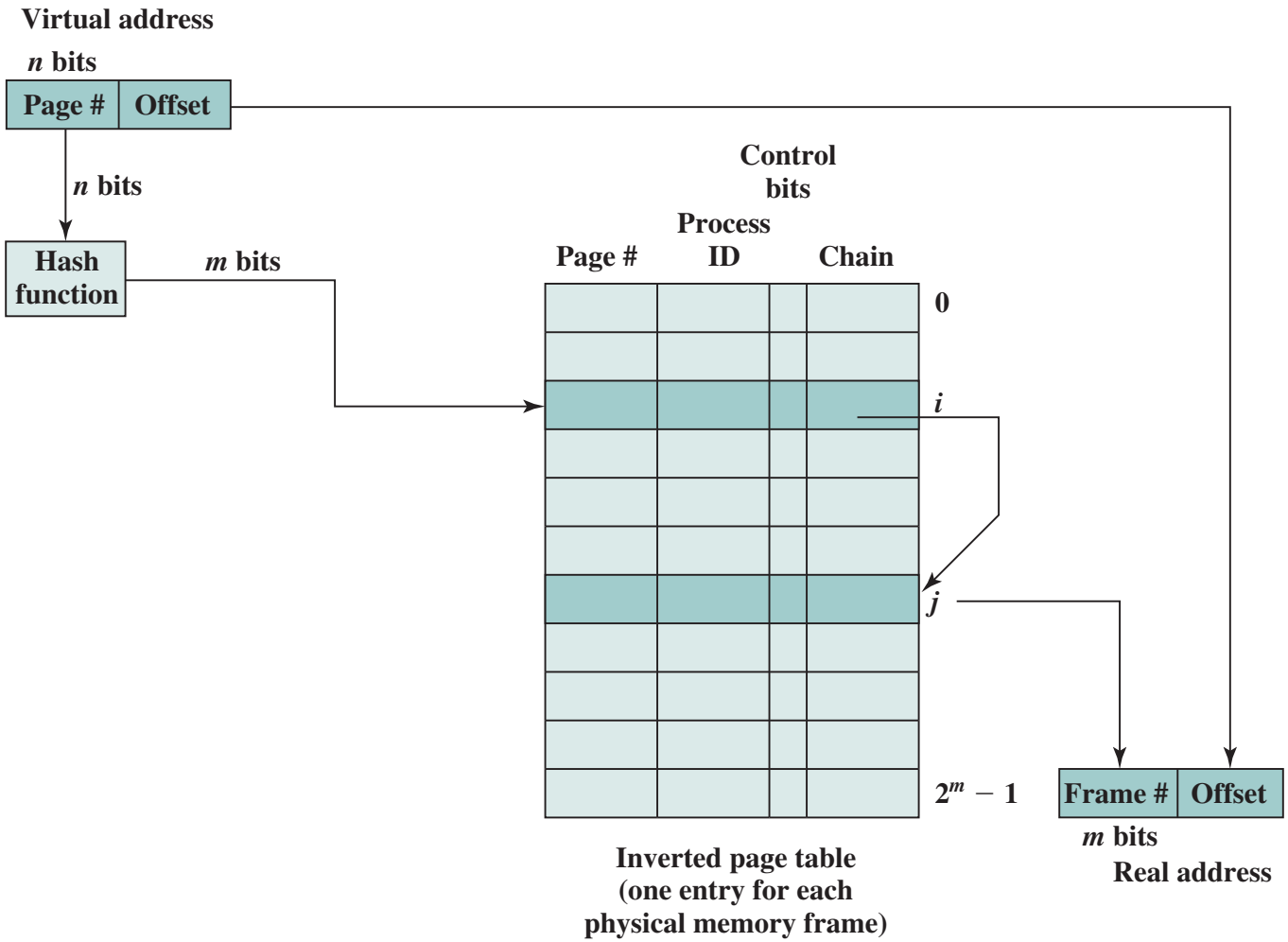
Thất bại

- Quá nhiều tiến trình trong bộ nhớ quá nhỏ
- OS tiêu tốn toàn bộ thời gian cho việc hoán đổi
- Có ít hoặc không có công việc nào được thực hiện
- Đĩa luôn luôn sáng
- Giải pháp:
  - Thuật toán thay trang
  - Giảm bớt số tiến trình đang chạy
  - Thêm bộ nhớ

Lợi ích

- Không cần toàn bộ tiến trình nằm trong bộ nhớ để chạy
- Có thể hoán đổi trang được yêu cầu
- Như vậy có thể chạy những tiến trình lớn hơn tổng bộ nhớ sẵn dùng
- Bộ nhớ chính được gọi là bộ nhớ thực
- Người dùng cảm giác bộ nhớ lớn hơn bộ nhớ thực

Cấu trúc bảng trang



Bộ nhớ trên máy tính PC

- Bộ nhớ cache: tích hợp trên chip vi xử lý:
  - L1: cache lệnh và cache dữ liệu
  - L2, L3
- Bộ nhớ chính: Tồn tại dưới dạng các mô-đun nhớ RAM

Bộ nhớ trên PC (tiếp)

- ROM BIOS chứa các chương trình sau:
  - Chương trình POST (Power On Self Test)
  - Chương trình CMOS Setup
  - Chương trình Bootstrap loader
  - Các trình điều khiển vào-ra cơ bản (BIOS)
- CMOS RAM:
  - Chứa thông tin cấu hình hệ thống
  - Đồng hồ hệ thống
  - Có pin nuôi riêng
- Video RAM: quản lý thông tin của màn hình
- Các loại bộ nhớ ngoài

Hết chương 7

Chương 8
HỆ THỐNG VÀO-RA

Nguyễn Kim Khánh
Trường Đại học Bách khoa Hà Nội

Nội dung học phần

Chương 1. Giới thiệu chung
Chương 2. Cơ bản về logic số
Chương 3. Hệ thống máy tính
Chương 4. Số học máy tính
Chương 5. Kiến trúc tập lệnh
Chương 6. Bộ xử lý
Chương 7. Bộ nhớ máy tính
Chương 8. Hệ thống vào-ra
Chương 9. Các kiến trúc song song

Nội dung của chương 8

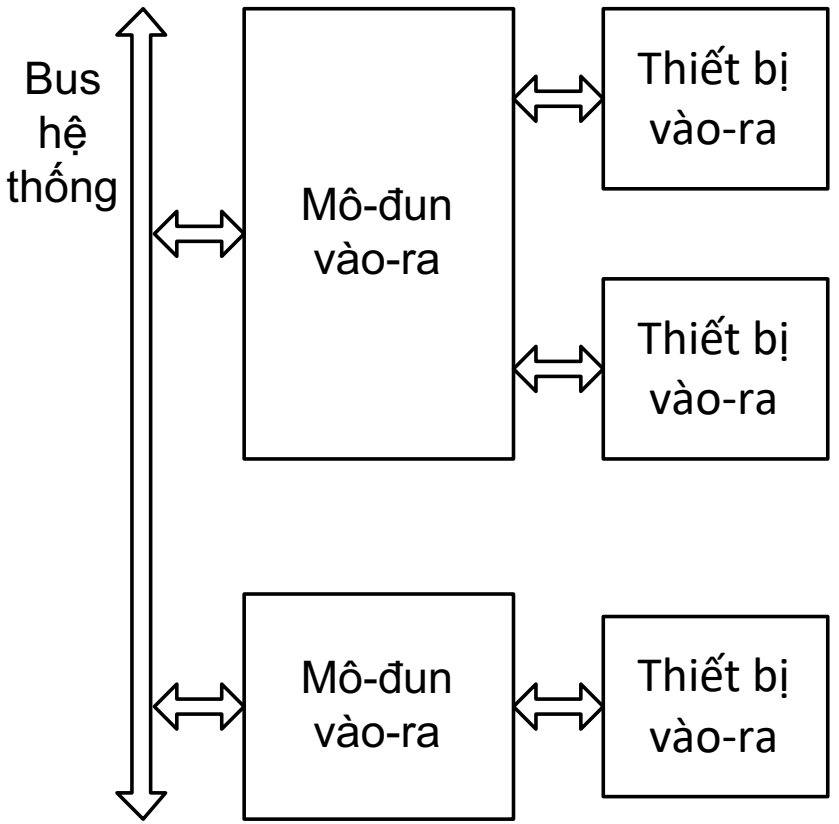
8.1. Tổng quan về hệ thống vào-ra

8.2. Các phương pháp điều khiển vào-ra

8.3. Nối ghép thiết bị vào-ra

8.1. Tổng quan về hệ thống vào-ra

- Chức năng: Trao đổi thông tin giữa máy tính với bên ngoài
- Các thao tác cơ bản:
  - Vào dữ liệu (Input)
  - Ra dữ liệu (Output)
- Các thành phần chính:
  - Các thiết bị vào-ra
  - Các mô-đun vào-ra



Đặc điểm của hệ thống vào-ra

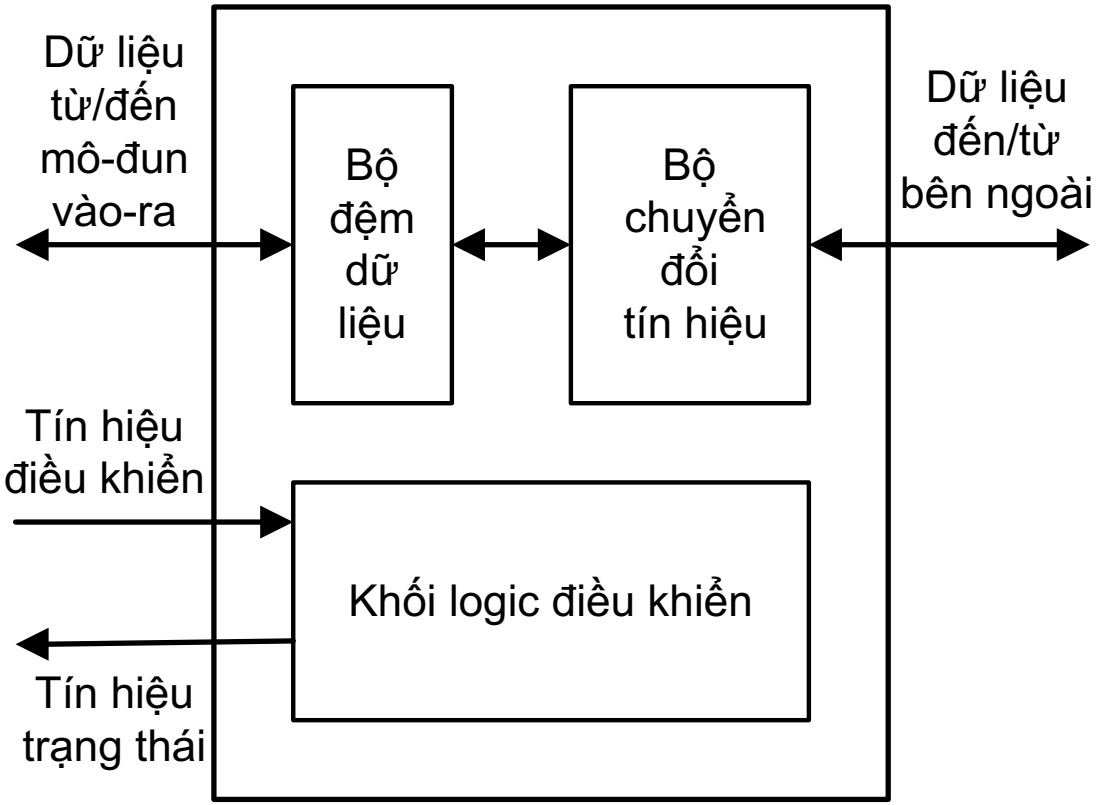
- Tồn tại đa dạng các thiết bị vào-ra khác nhau về:
  - Nguyên tắc hoạt động
  - Tốc độ
  - Khuôn dạng dữ liệu
- Tất cả các thiết bị vào-ra đều chậm hơn CPU và RAM

→ Cần có các mô-đun vào-ra để nối ghép các thiết bị với CPU và bộ nhớ chính

Thiết bị vào-ra

- Còn gọi là thiết bị ngoại vi (Peripherals)
- Chức năng: chuyển đổi dữ liệu giữa bên trong và bên ngoài máy tính
- Phân loại:
  - Thiết bị vào (Input Devices)
  - Thiết bị ra (Output Devices)
  - Thiết bị lưu trữ (Storage Devices)
  - Thiết bị truyền thông (Communication Devices)
- Giao tiếp:
  - Người - máy
  - Máy - máy

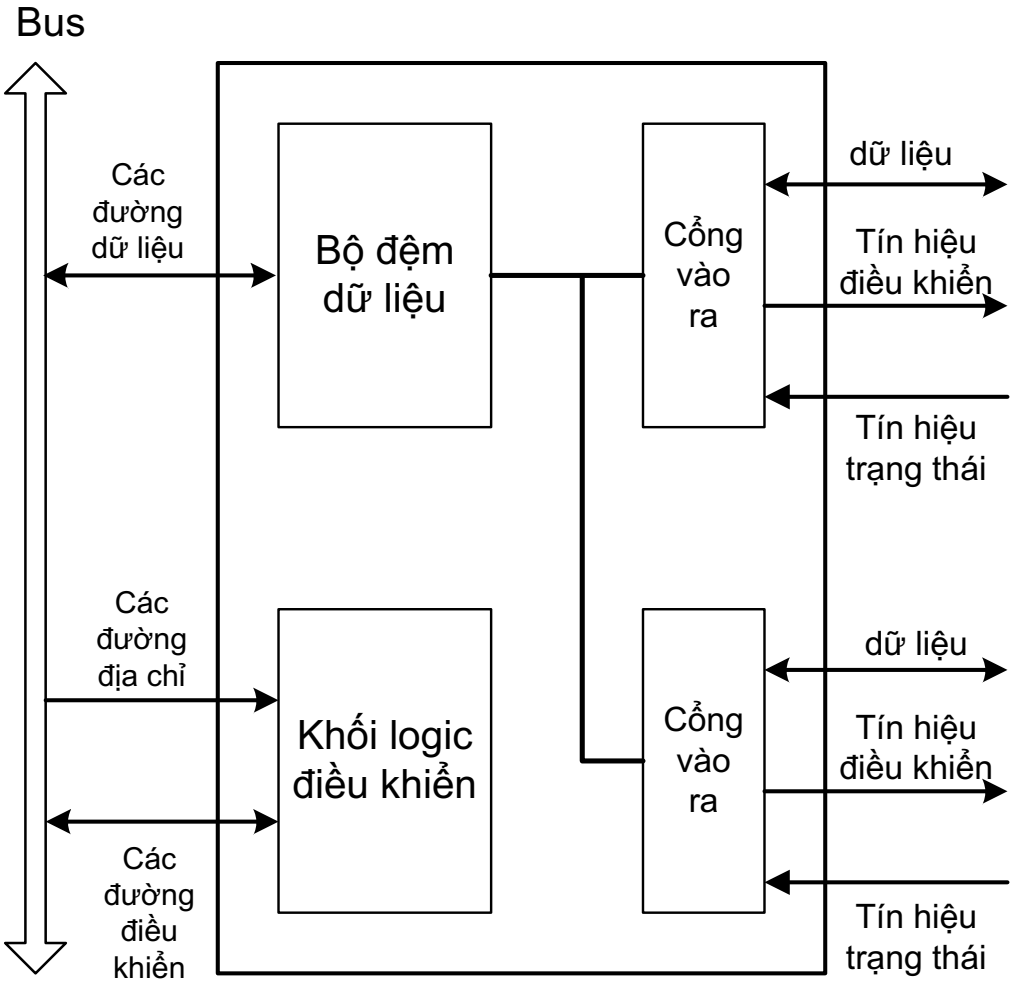
Cấu trúc chung của thiết bị vào-ra



Mô-đun vào-ra

- Chức năng:
  - Điều khiển và định thời
  - Trao đổi thông tin với CPU hoặc bộ nhớ chính
  - Trao đổi thông tin với thiết bị vào-ra
  - Đệm giữa bên trong máy tính với thiết bị vào-ra
  - Phát hiện lỗi của thiết bị vào-ra

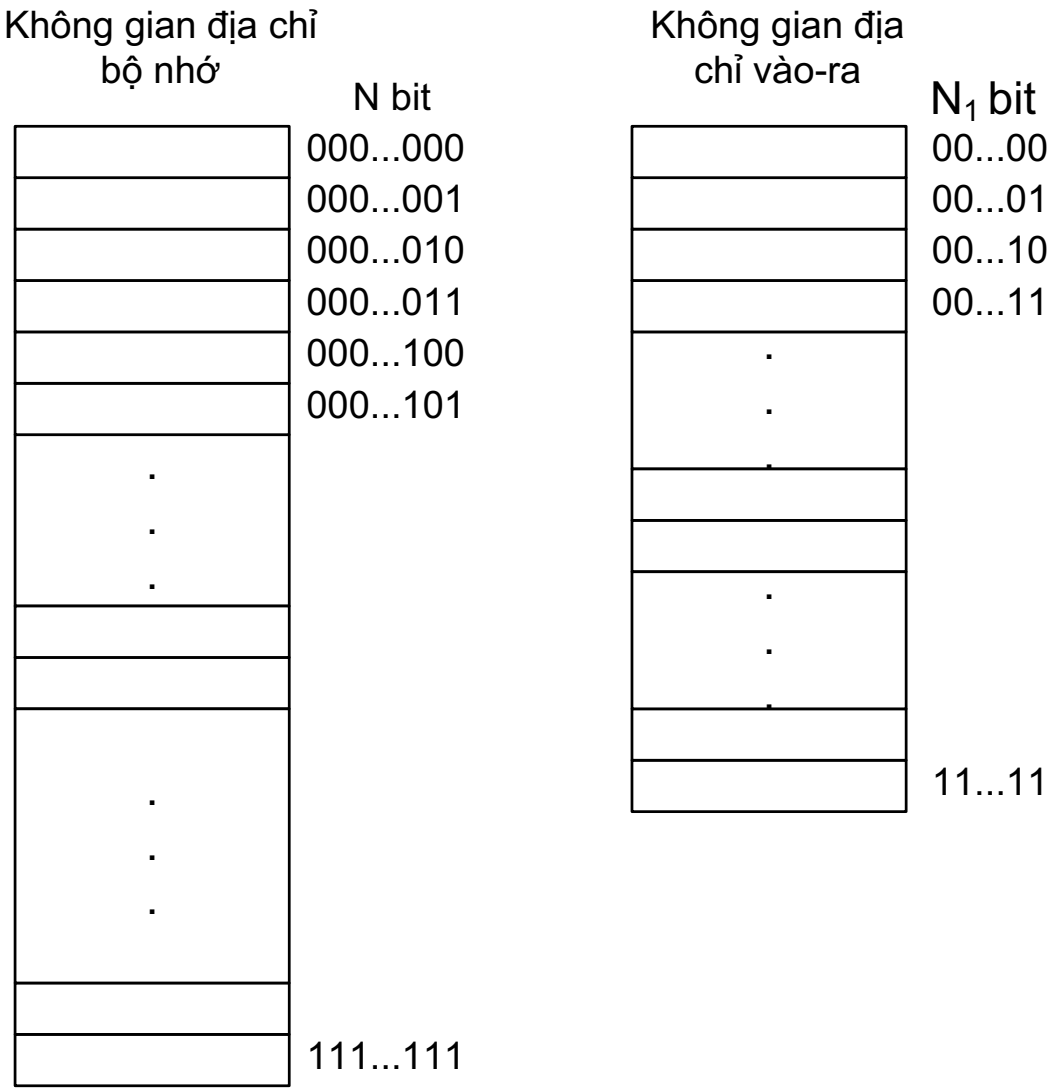
Cấu trúc của mô-đun vào-ra



4. Địa chỉ hóa cổng vào-ra (IO addressing)

- Hầu hết các bộ xử lý chỉ có một không gian địa chỉ chung cho cả các ngăn nhớ và các cổng vào-ra
  - Các bộ xử lý 680x0 của Motorola
  - Các bộ xử lý theo kiến trúc RISC: MIPS, ARM, ...
- Một số bộ xử lý có hai không gian địa chỉ tách biệt:
  - Không gian địa chỉ bộ nhớ
  - Không gian địa chỉ vào-ra
  - Ví dụ: Intel x86

Không gian địa chỉ tách biệt



Các phương pháp địa chỉ hoá cổng vào-ra

- Vào-ra theo bản đồ bộ nhớ
  (Memory mapped IO)
- Vào-ra riêng biệt
  (Isolated IO hay IO mapped IO)

Vào-ra theo bản đồ bộ nhớ

- Cổng vào-ra được đánh địa chỉ theo không gian địa chỉ bộ nhớ
- CPU coi cổng vào-ra như ngăn nhớ
- Lập trình trao đổi dữ liệu với cổng vào-ra bằng các lệnh truy nhập dữ liệu bộ nhớ
- Có thể thực hiện trên mọi hệ thống
- Ví dụ: Bộ xử lý MIPS
  - 32-bit địa chỉ cho một không gian địa chỉ chung cho cả các ngăn nhớ và các cổng vào-ra
  - Các cổng vào-ra được gắn các địa chỉ thuộc vùng địa chỉ dự trữ
  - Vào/ra dữ liệu: sử dụng lệnh load/store

Ví dụ lập trình vào-ra cho MIPS

- Ví dụ: Có hai cổng vào-ra được gán địa chỉ:
  - Cổng 1: 0xFFFFFFF4
  - Cổng 2: 0xFFFFFFF8

- Ghi giá trị 0x41 ra cổng 1

```
addi    $t0, $0, 0x41       # đưa giá trị 0x41
sw      $t0, 0xFFF4($0)     # ra cổng 1
```

Chú ý: giá trị 16-bit 0xFFF4 được sign-extended thành 32-bit 0xFFFFFFF4

- Đọc dữ liệu từ cổng 2 đưa vào $t3

```
lw      $t3, 0xFFF8($0)     # đọc dữ liệu cổng 2 đưa vào $t3
```

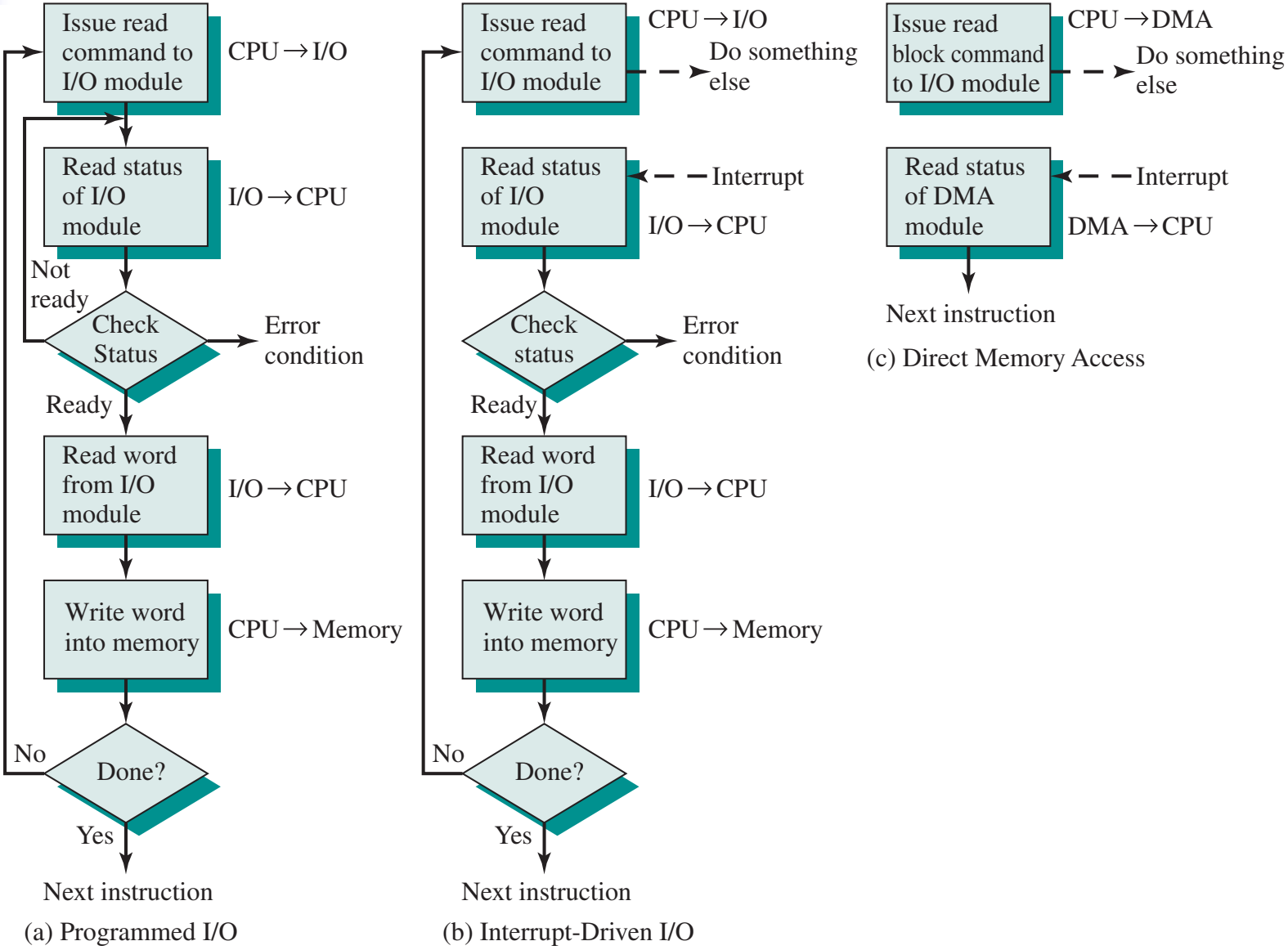
Vào-ra riêng biệt (Isolated IO)

- Cổng vào-ra được đánh địa chỉ theo không gian địa chỉ vào-ra riêng
- Lập trình trao đổi dữ liệu với cổng vào-ra bằng các lệnh vào-ra chuyên dụng
- Ví dụ: Intel x86
  - Dùng 8-bit hoặc 16-bit địa chỉ cho không gian địa chỉ vào-ra riêng
  - Có hai lệnh vào-ra chuyên dụng
    - Lệnh IN: nhận dữ liệu từ cổng vào
    - Lệnh OUT: đưa dữ liệu đến cổng ra

8.2. Các phương pháp điều khiển vào-ra

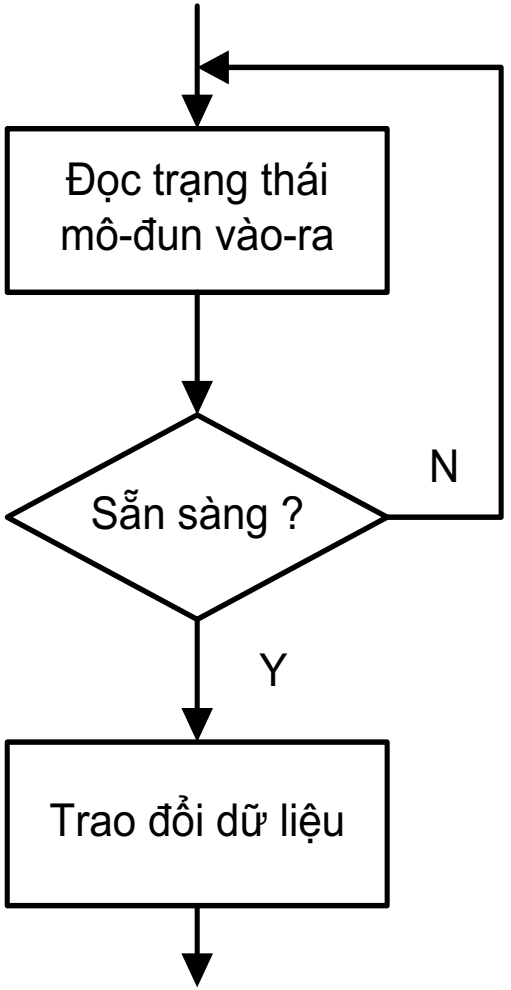
- Vào-ra bằng chương trình
  (Programmed IO)
- Vào-ra điều khiển bằng ngắt
  (Interrupt Driven IO)
- Truy nhập bộ nhớ trực tiếp - DMA
  (Direct Memory Access)

Ba kỹ thuật thực hiện vào một khối dữ liệu



1. Vào-ra bằng chương trình

- Nguyên tắc chung:
  - CPU điều khiển trực tiếp vào-ra bằng chương trình → cần phải lập trình vào-ra để trao đổi dữ liệu giữa CPU với mô-đun vào-ra
  - CPU nhanh hơn thiết bị vào-ra rất nhiều lần, vì vậy trước khi thực hiện lệnh vào-ra, chương trình cần đọc và kiểm tra trạng thái sẵn sàng của mô-đun vào-ra



Các tín hiệu điều khiển vào-ra

- Tín hiệu ***điều khiển*** *(Control):* kích hoạt thiết bị vào-ra
- Tín hiệu ***kiểm tra*** *(Test):* kiểm tra trạng thái của mô-đun vào-ra và thiết bị vào-ra
- Tín hiệu điều khiển ***đọc*** *(Read):* yêu cầu mô-đun vào-ra nhận dữ liệu từ thiết bị vào-ra và đưa vào bộ đệm dữ liệu, rồi CPU nhận dữ liệu đó
- Tín hiệu điều khiển ***ghi*** *(Write):* yêu cầu mô-đun vào-ra lấy dữ liệu trên bus dữ liệu đưa đến bộ đệm dữ liệu rồi chuyển ra thiết bị vào-ra

Các lệnh vào-ra

- Với vào-ra theo bản đồ bộ nhớ: sử dụng các lệnh trao đổi dữ liệu với bộ nhớ để trao đổi dữ liệu với cổng vào-ra
- Với vào-ra riêng biệt: sử dụng các lệnh vào-ra chuyên dụng (IN, OUT)

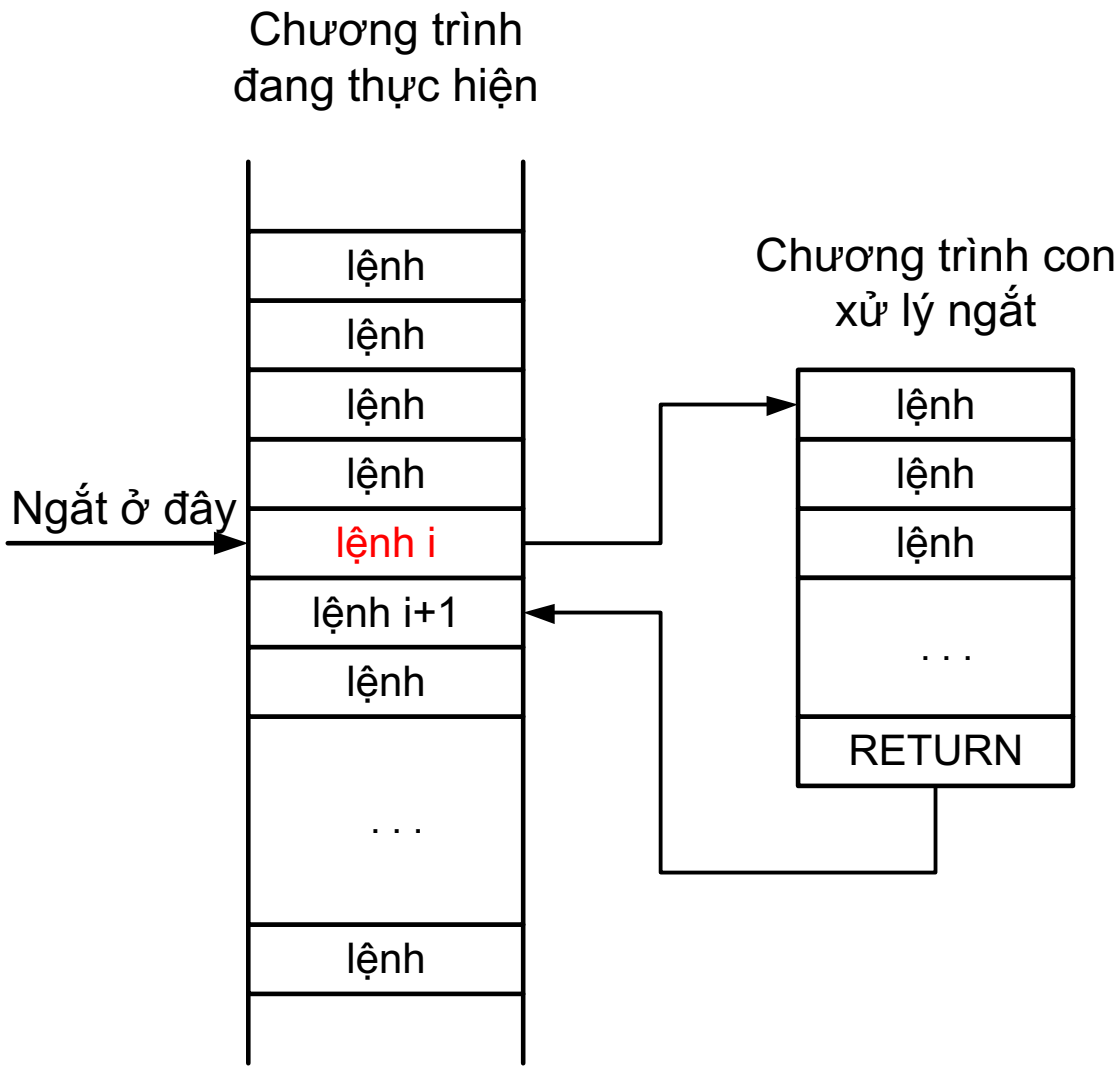
Đặc điểm

- Vào-ra do ý muốn của người lập trình
- CPU trực tiếp điều khiển trao đổi dữ liệu giữa CPU với mô-đun vào-ra
- CPU đợi trạng thái sẵn sàng của mô-đun vào-ra (thông qua vòng lặp)→ tiêu tốn nhiều thời gian của CPU

2. Vào-ra điều khiển bằng ngắt

- Nguyên tắc chung:
  - CPU không phải đợi trạng thái sẵn sàng của mô-đun vào-ra, CPU thực hiện một chương trình nào đó
  - Khi mô-đun vào-ra sẵn sàng thì nó phát tín hiệu ngắt CPU
  - CPU thực hiện chương trình con xử lý ngắt vào-ra tương ứng để trao đổi dữ liệu
  - CPU trở lại tiếp tục thực hiện chương trình đang bị ngắt

Chuyển điều khiển đến chương trình con ngắt



Hoạt động vào dữ liệu: nhìn từ mô-đun vào-ra

- Mô-đun vào-ra nhận tín hiệu điều khiển *đọc* từ CPU
- Mô-đun vào-ra nhận dữ liệu từ thiết bị vào-ra, trong khi đó CPU làm việc khác
- Khi đã có dữ liệu → mô-đun vào-ra phát tín hiệu ngắt CPU
- CPU yêu cầu dữ liệu
- Mô-đun vào-ra chuyển dữ liệu đến CPU

Hoạt động vào dữ liệu: nhìn từ CPU

- Phát tín hiệu điều khiển ***đọc***
- Làm việc khác
- Cuối mỗi chu trình lệnh, kiểm tra tín hiệu yêu cầu ngắt
- Nếu bị ngắt:
  - Cất ngữ cảnh (nội dung các thanh ghi liên quan)
  - Thực hiện chương trình con xử lý ngắt để vào dữ liệu
  - Khôi phục ngữ cảnh của chương trình đang thực hiện

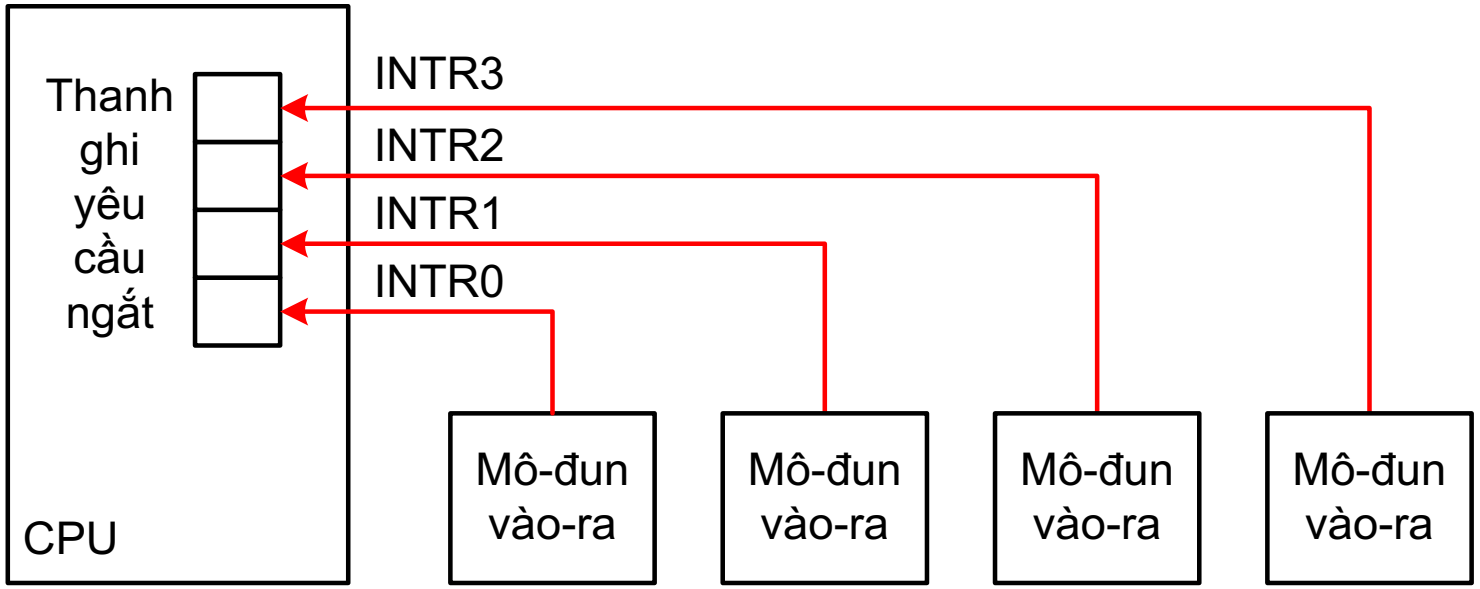
Các vấn đề nảy sinh khi thiết kế

- Làm thế nào để xác định được mô-đun vào-ra nào phát tín hiệu ngắt ?
- CPU làm như thế nào khi có nhiều yêu cầu ngắt cùng xẩy ra ?

Các phương pháp nối ghép ngắt

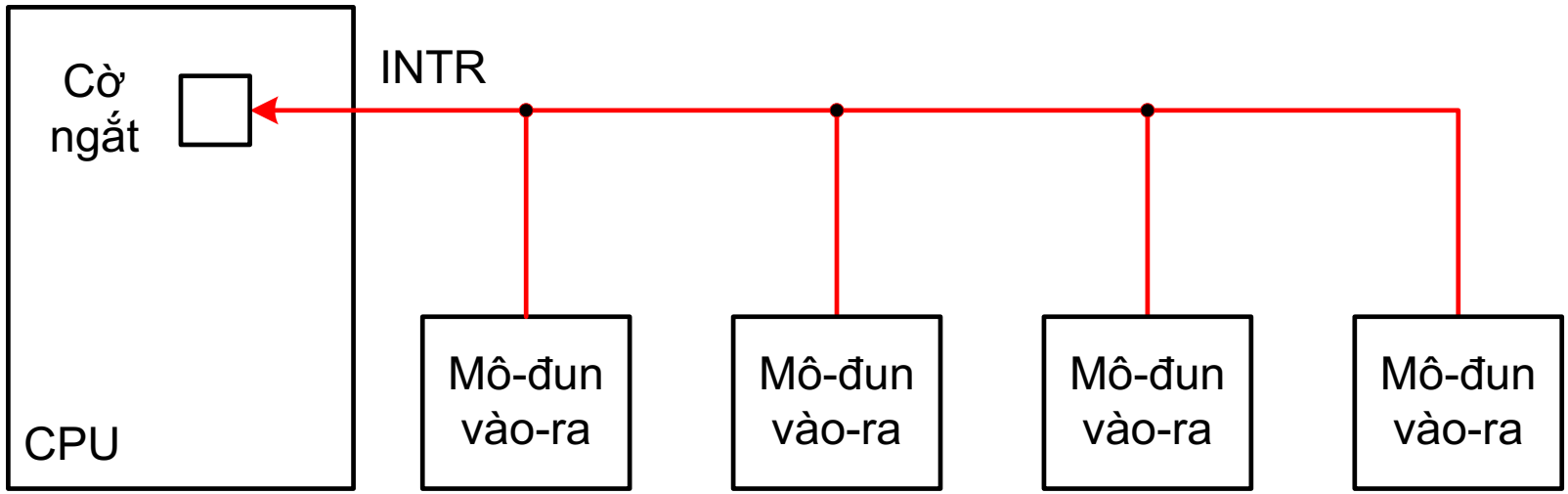
- Sử dụng nhiều đường yêu cầu ngắt
- Hỏi vòng bằng phần mềm (Software Poll)
- Hỏi vòng bằng phần cứng (Daisy Chain or Hardware Poll)
- Sử dụng bộ điều khiển ngắt (PIC)

Nhiều đường yêu cầu ngắt



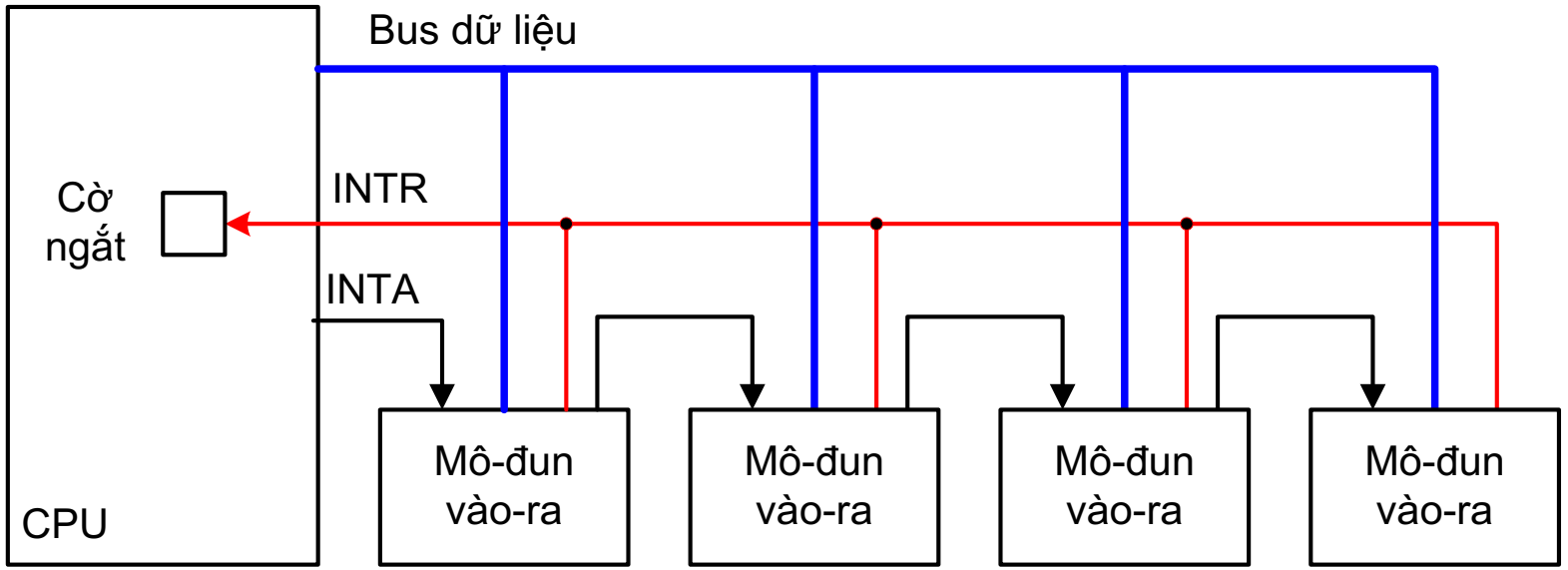
- Mỗi mô-đun vào-ra được nối với một đường yêu cầu ngắt
- CPU phải có nhiều đường tín hiệu yêu cầu ngắt
- Hạn chế số lượng mô-đun vào-ra
- Các đường ngắt được qui định mức ưu tiên

Hỏi vòng bằng phần mềm



- CPU thực hiện phần mềm hỏi lần lượt từng mô-đun vào-ra
- Chậm
- Thứ tự các mô-đun được hỏi vòng chính là thứ tự ưu tiên

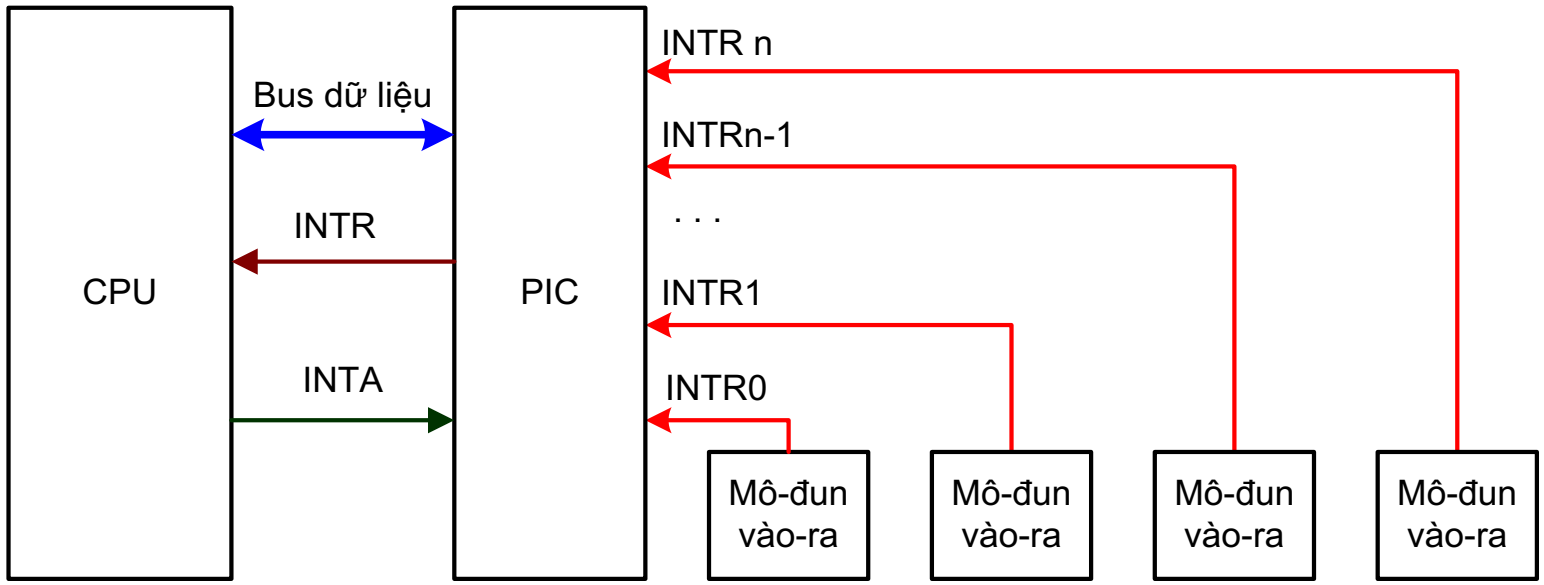
Hỏi vòng bằng phần cứng



Hỏi vòng bằng phần cứng (tiếp)

- CPU phát tín hiệu chấp nhận ngắt (INTA) đến mô-đun vào-ra đầu tiên
- Nếu mô-đun vào-ra đó không gây ra ngắt thì nó gửi tín hiệu đến mô-đun kế tiếp cho đến khi xác định được mô-đun gây ngắt
- Thứ tự các mô-đun vào-ra kết nối trong chuỗi xác định thứ tự ưu tiên

Bộ điều khiển ngắt lập trình được



- PIC – Programmable Interrupt Controller
- PIC có nhiều đường vào yêu cầu ngắt có qui định mức ưu tiên
- PIC chọn một yêu cầu ngắt không bị cấm có mức ưu tiên cao nhất gửi tới CPU

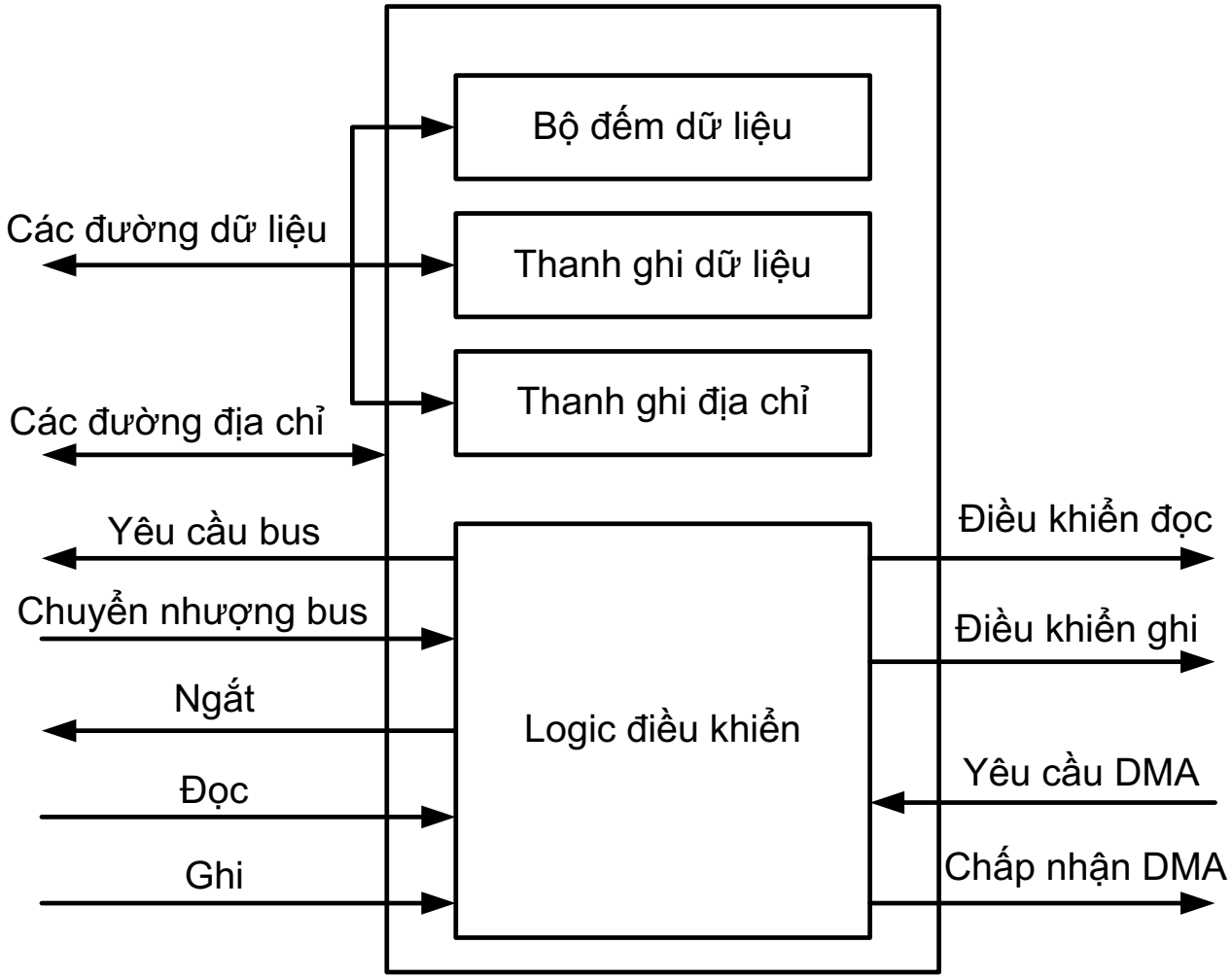
Đặc điểm của vào-ra điều khiển bằng ngắt

- Có sự kết hợp giữa phần cứng và phần mềm
  - Phần cứng: gây ngắt CPU
  - Phần mềm: trao đổi dữ liệu giữa CPU với mô-đun vào-ra
- CPU trực tiếp điều khiển vào-ra
- CPU không phải đợi mô-đun vào-ra, do đó hiệu quả sử dụng CPU tốt hơn

3. DMA (Direct Memory Access)

- Vào-ra bằng chương trình và bằng ngắt do CPU trực tiếp điều khiển:
  - Chiếm thời gian của CPU
- Để khắc phục dùng kỹ thuật DMA
  - Sử dụng mô-đun điều khiển vào-ra chuyên dụng, gọi là DMAC (Controller), điều khiển trao đổi dữ liệu giữa mô-đun vào-ra với bộ nhớ chính

Sơ đồ cấu trúc của DMAC



Các thành phần của DMAC

- Thanh ghi dữ liệu: chứa dữ liệu trao đổi
- Thanh ghi địa chỉ: chứa địa chỉ ngăn nhớ dữ liệu
- Bộ đếm dữ liệu: chứa số từ dữ liệu cần trao đổi
- Logic điều khiển: điều khiển hoạt động của DMAC

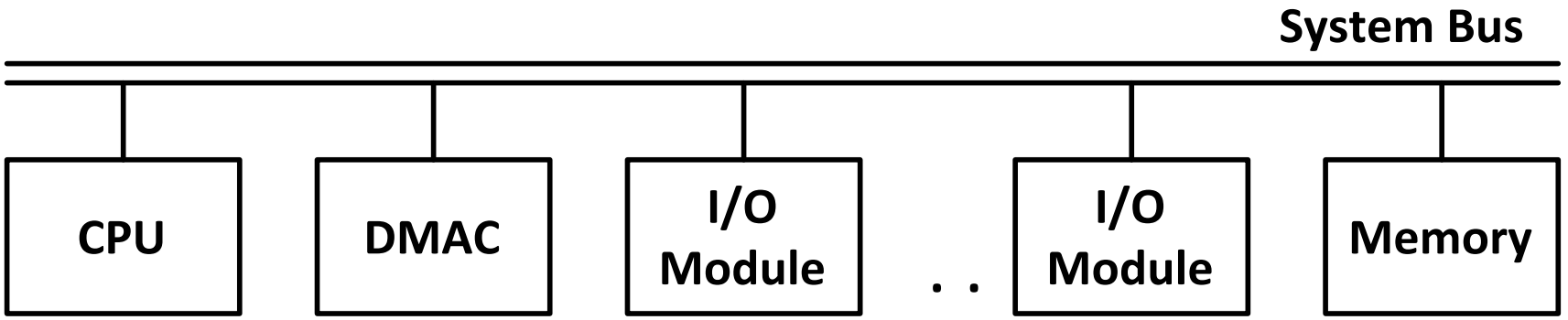
Hoạt động DMA

- CPU "nói" cho DMAC
  - Vào hay Ra dữ liệu
  - Địa chỉ thiết bị vào-ra (cổng vào-ra tương ứng)
  - Địa chỉ đầu của mảng nhớ chứa dữ liệu → nạp vào thanh ghi địa chỉ
  - Số từ dữ liệu cần truyền → nạp vào bộ đếm dữ liệu
- CPU làm việc khác
- DMAC điều khiển trao đổi dữ liệu
- Sau khi truyền được một từ dữ liệu thì:
  - nội dung thanh ghi địa chỉ tăng
  - nội dung bộ đếm dữ liệu giảm
- Khi bộ đếm dữ liệu = 0, DMAC gửi tín hiệu ngắt CPU để báo kết thúc DMA

Các kiểu thực hiện DMA

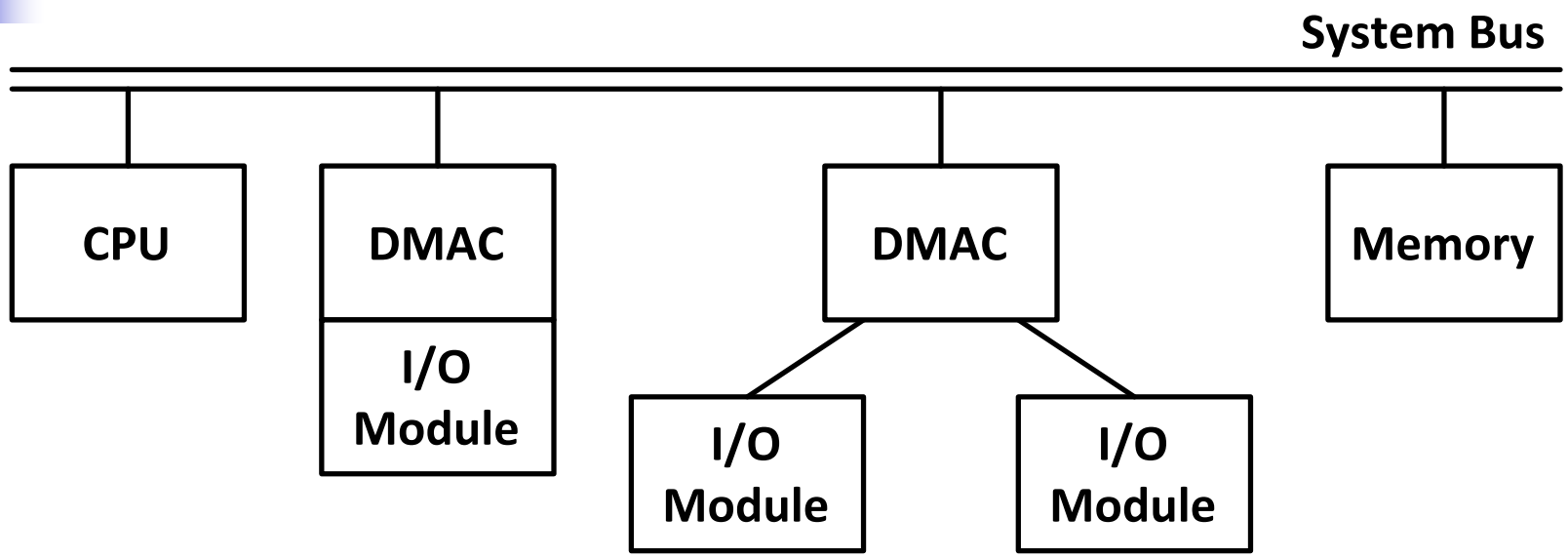
- DMA truyền theo khối (Block-transfer DMA): DMAC sử dụng bus để truyền xong cả khối dữ liệu
- DMA lấy chu kỳ (Cycle Stealing DMA): DMAC cưỡng bức CPU treo tạm thời từng chu kỳ bus, DMAC chiếm bus thực hiện truyền một từ dữ liệu.
- DMA trong suốt (Transparent DMA): DMAC nhận biết những chu kỳ nào CPU không sử dụng bus thì chiếm bus để trao đổi một từ dữ liệu.

Cấu hình DMA (1)



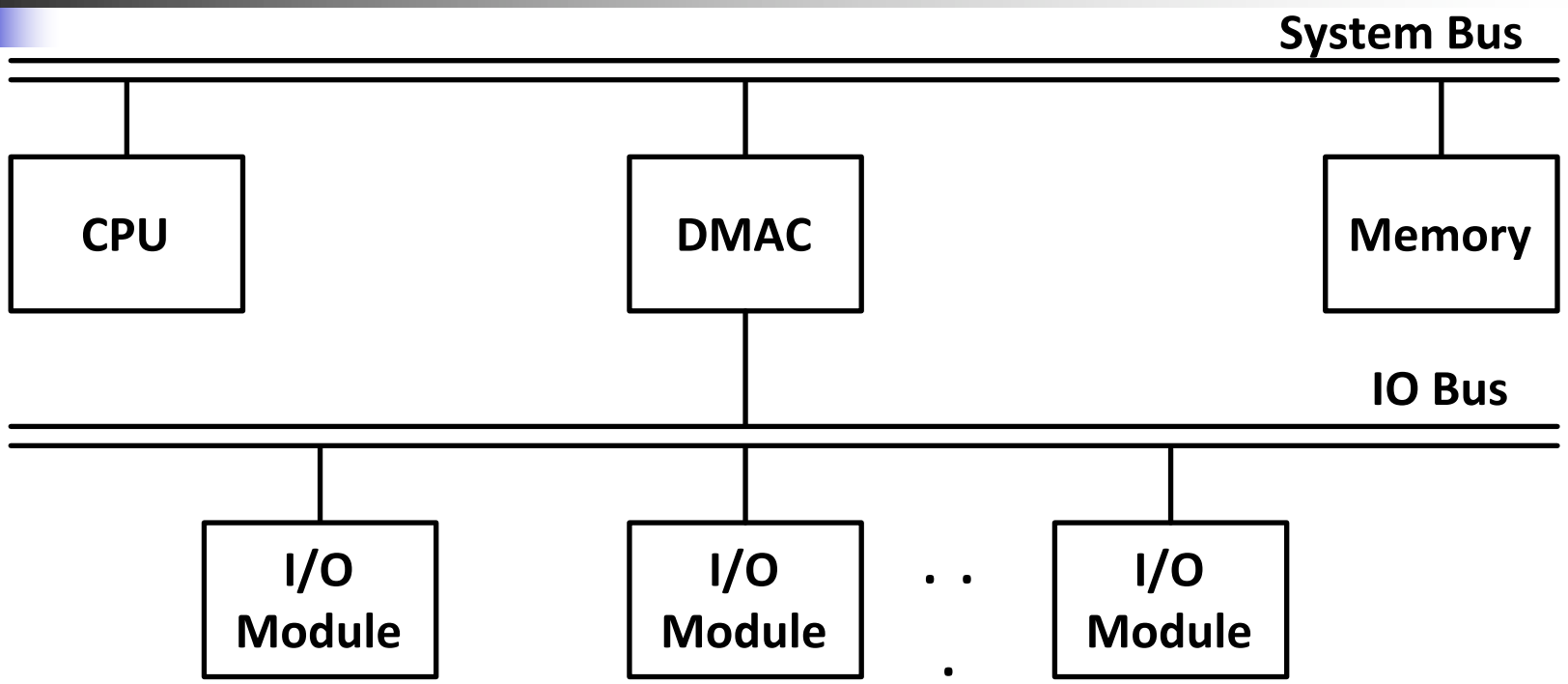
- Mỗi lần trao đổi một dữ liệu, DMAC sử dụng bus hai lần
  - Giữa mô-đun vào-ra với DMAC
  - Giữa DMAC với bộ nhớ

Cấu hình DMA (2)



- DMAC điều khiển một hoặc vài mô-đun vào-ra
- Mỗi lần trao đổi một dữ liệu, DMAC sử dụng bus một lần
  - Giữa DMAC với bộ nhớ

Cấu hình DMA (3)



- Bus vào-ra tách rời hỗ trợ tất cả các thiết bị cho phép DMA
- Mỗi lần trao đổi một dữ liệu, DMAC sử dụng bus một lần
  - Giữa DMAC với bộ nhớ

Đặc điểm của DMA

- CPU không tham gia trong quá trình trao đổi dữ liệu
- DMAC điều khiển trao đổi dữ liệu giữa bộ nhớ chính với mô-đun vào-ra (hoàn toàn bằng phần cứng)→ tốc độ nhanh
- Phù hợp với các yêu cầu trao đổi mảng dữ liệu có kích thước lớn

4. Bộ xử lý vào-ra

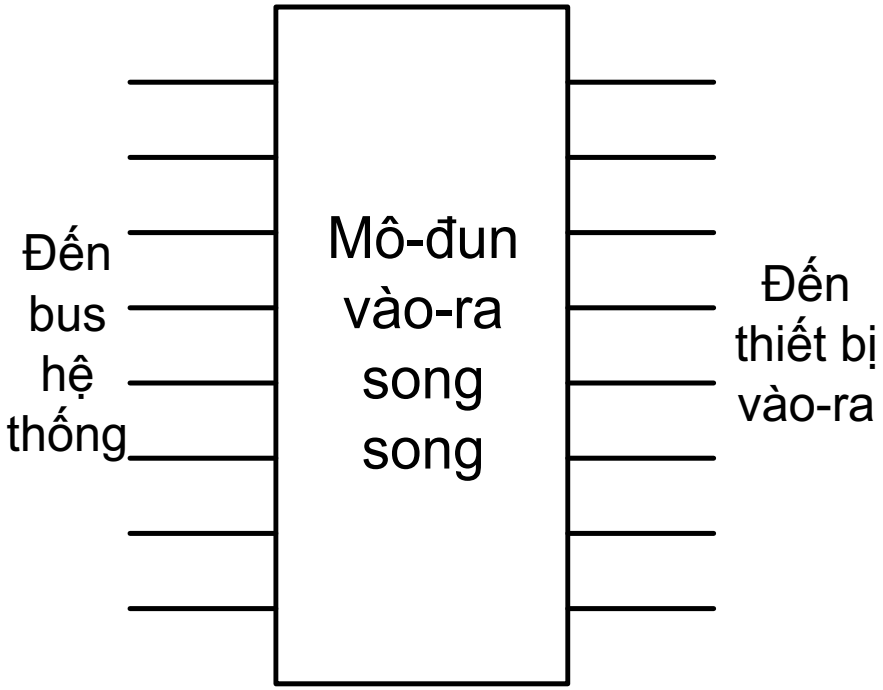
- Việc điều khiển vào-ra được thực hiện bởi một bộ xử lý vào-ra chuyên dụng
- Bộ xử lý vào-ra hoạt động theo chương trình của riêng nó
- Chương trình của bộ xử lý vào-ra có thể nằm trong bộ nhớ chính hoặc nằm trong một bộ nhớ riêng

8.3. Nối ghép thiết bị vào-ra

1. Các kiểu nối ghép vào-ra

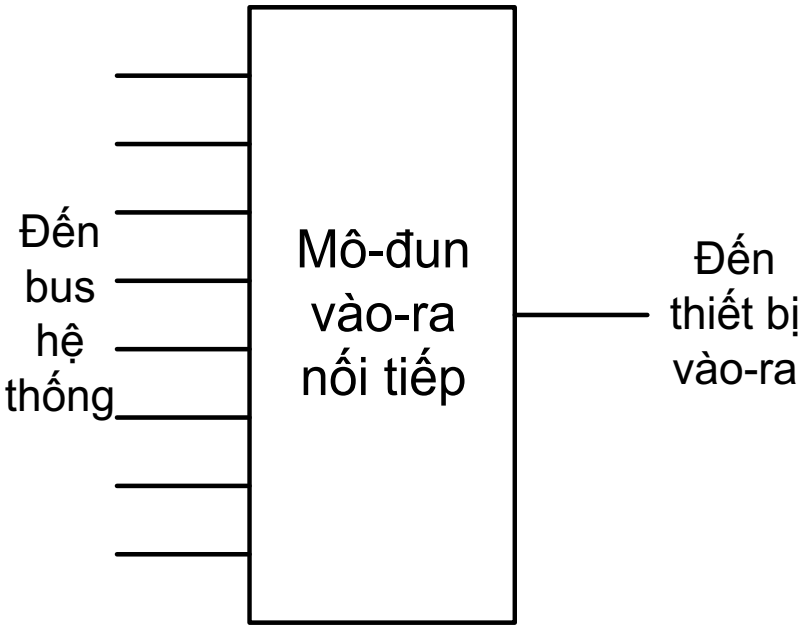
- Nối ghép song song
- Nối ghép nối tiếp

Nối ghép song song



- Truyền nhiều bit song song
- Tốc độ nhanh
- Cần nhiều đường truyền dữ liệu

Nối ghép nối tiếp

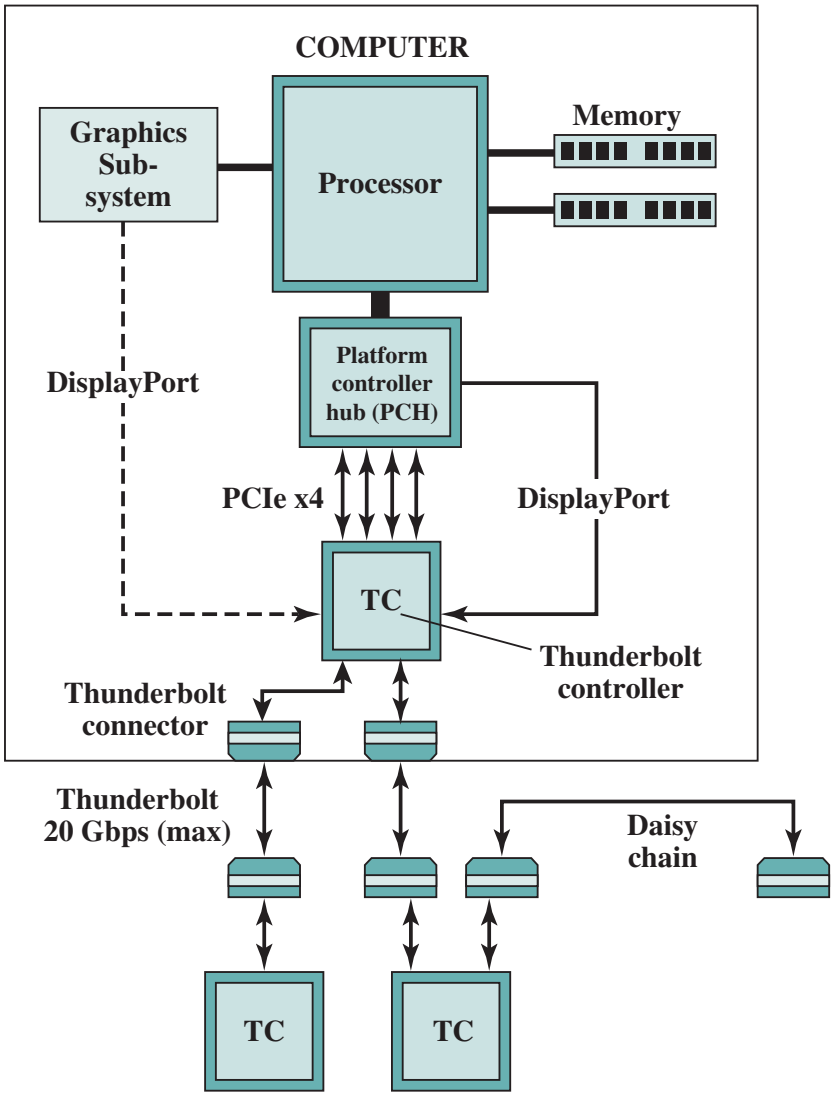


- Truyền lần lượt từng bit
- Cần có bộ chuyển đổi từ dữ liệu song song sang nối tiếp hoặc/và ngược lại
- Tốc độ chậm hơn
- Cần ít đường truyền dữ liệu

2. Các cấu hình nối ghép

- Điểm tới điểm (Point to Point)
  - Thông qua một cổng vào-ra nối ghép với một thiết bị
- Điểm tới đa điểm (Point to Multipoint)
  - Thông qua một cổng vào-ra cho phép nối ghép được với nhiều thiết bị
  - Ví dụ:
    - USB (Universal Serial Bus): 127 thiết bị
    - IEEE 1394 (FireWire): 63 thiết bị
    - Thunderbolt

Thunderbolt



Hết chương 8

Chương 9
CÁC KIẾN TRÚC SONG SONG

Nguyễn Kim Khánh
Trường Đại học Bách khoa Hà Nội

Nội dung học phần

Chương 1. Giới thiệu chung
Chương 2. Cơ bản về logic số
Chương 3. Hệ thống máy tính
Chương 4. Số học máy tính
Chương 5. Kiến trúc tập lệnh
Chương 6. Bộ xử lý
Chương 7. Bộ nhớ máy tính
Chương 8. Hệ thống vào-ra
Chương 9. Các kiến trúc song song

Nội dung của chương 9

9.1. Phân loại kiến trúc máy tính

9.2. Đa xử lý bộ nhớ dùng chung

9.3. Đa xử lý bộ nhớ phân tán

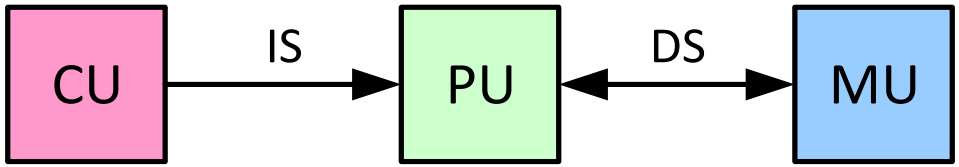
9.4. Bộ xử lý đồ họa đa dụng

9.1. Phân loại kiến trúc máy tính

Phân loại kiến trúc máy tính (Michael Flynn -1966)

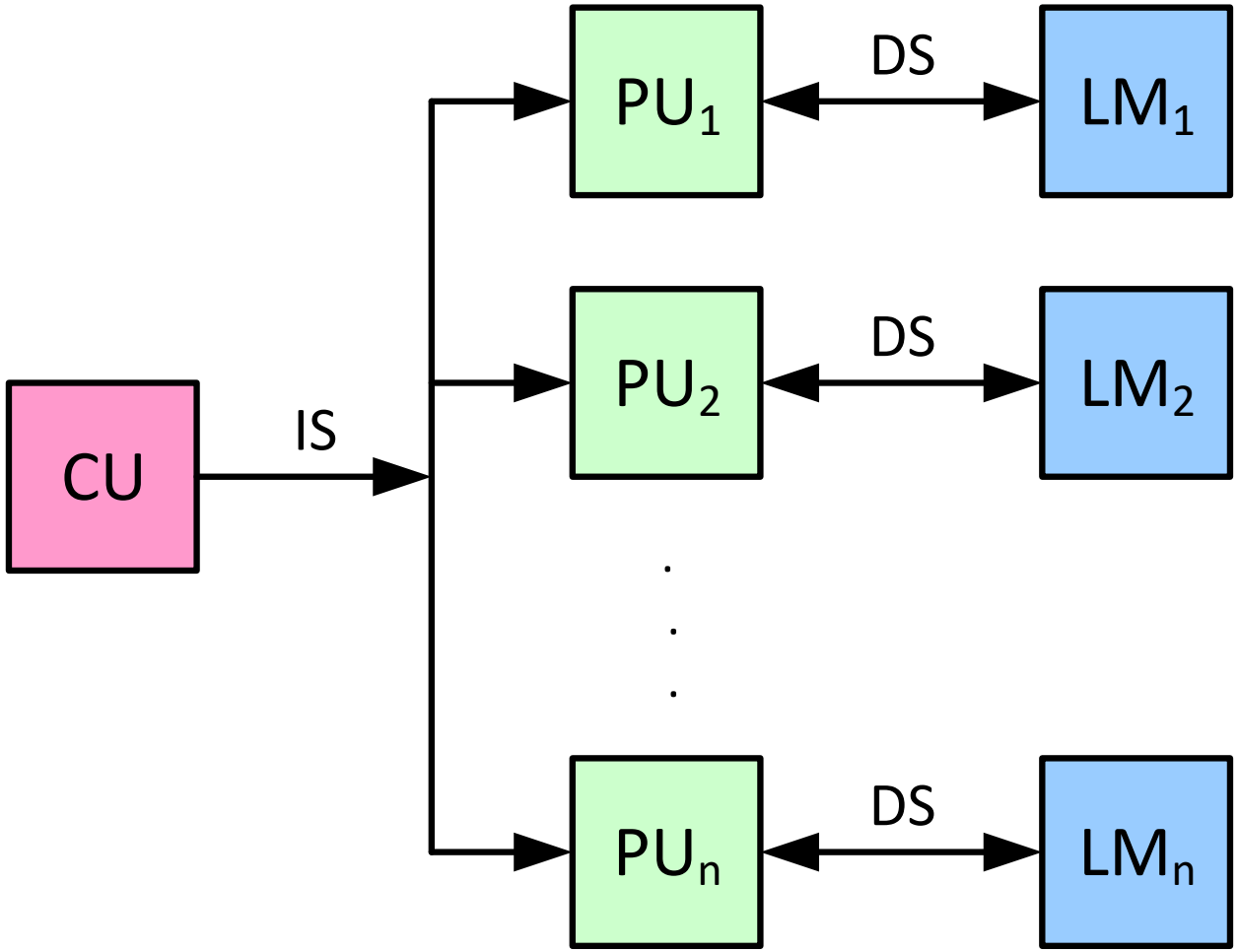
- SISD - Single Instruction Stream, Single Data Stream
- SIMD - Single Instruction Stream, Multiple Data Stream
- MISD - Multiple Instruction Stream, Single Data Stream
- MIMD - Multiple Instruction Stream, Multiple Data Stream

SISD



- CU: Control Unit
- PU: Processing Unit
- MU: Memory Unit
- Một bộ xử lý
- Đơn dòng lệnh
- Dữ liệu được lưu trữ trong một bộ nhớ
- Chính là Kiến trúc von Neumann (tuần tự)

SIMD



SIMD (tiếp)

- Đơn dòng lệnh điều khiển đồng thời các đơn vị xử lý PUs
- Mỗi đơn vị xử lý có một bộ nhớ dữ liệu riêng LM (local memory)
- Mỗi lệnh được thực hiện trên một tập các dữ liệu khác nhau
- Các mô hình SIMD
  - Vector Computer
  - Array processor

MISD

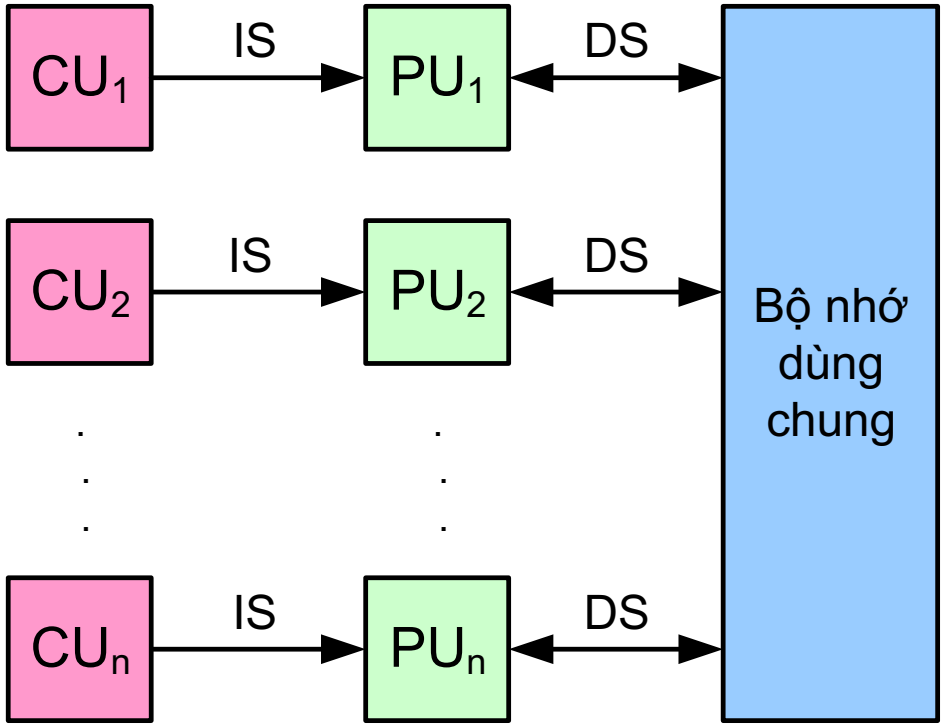
- Một luồng dữ liệu cùng được truyền đến một tập các bộ xử lý
- Mỗi bộ xử lý thực hiện một dãy lệnh khác nhau.
- Chưa tồn tại máy tính thực tế
- Có thể có trong tương lai

MIMD

- Tập các bộ xử lý
- Các bộ xử lý đồng thời thực hiện các dãy lệnh khác nhau trên các dữ liệu khác nhau
- Các mô hình MIMD
  - Multiprocessors (Shared Memory)
  - Multicomputers (Distributed Memory)

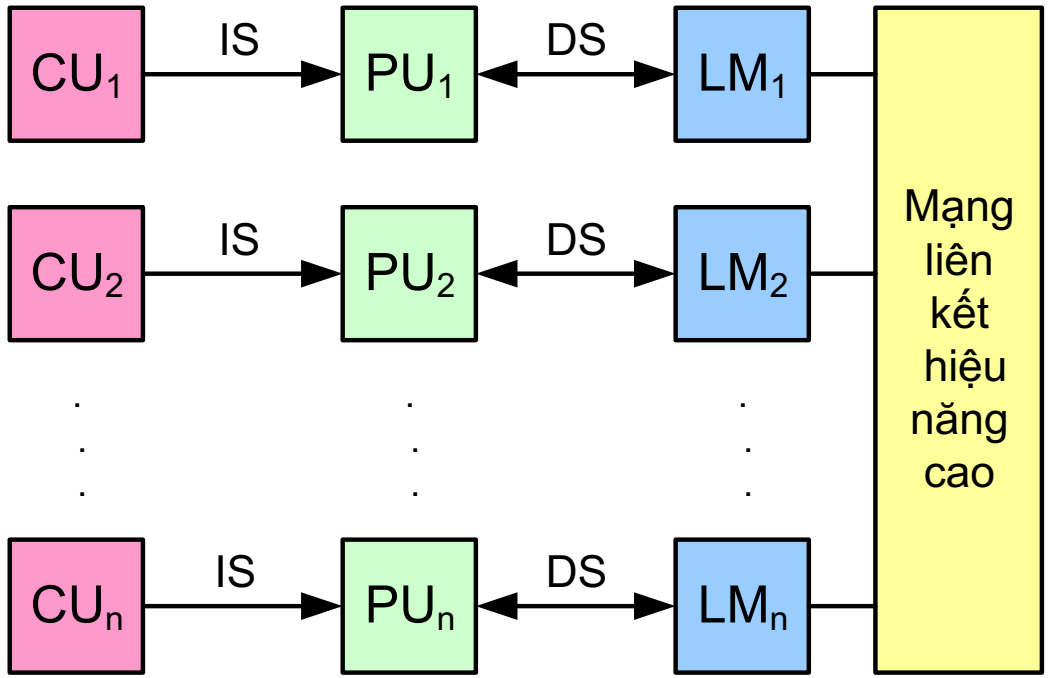
MIMD - Shared Memory

Đa xử lý bộ nhớ dùng chung
(shared memory mutiprocessors)



MIMD - Distributed Memory

Đa xử lý bộ nhớ phân tán
(distributed memory mutiprocessors or multicomputers)



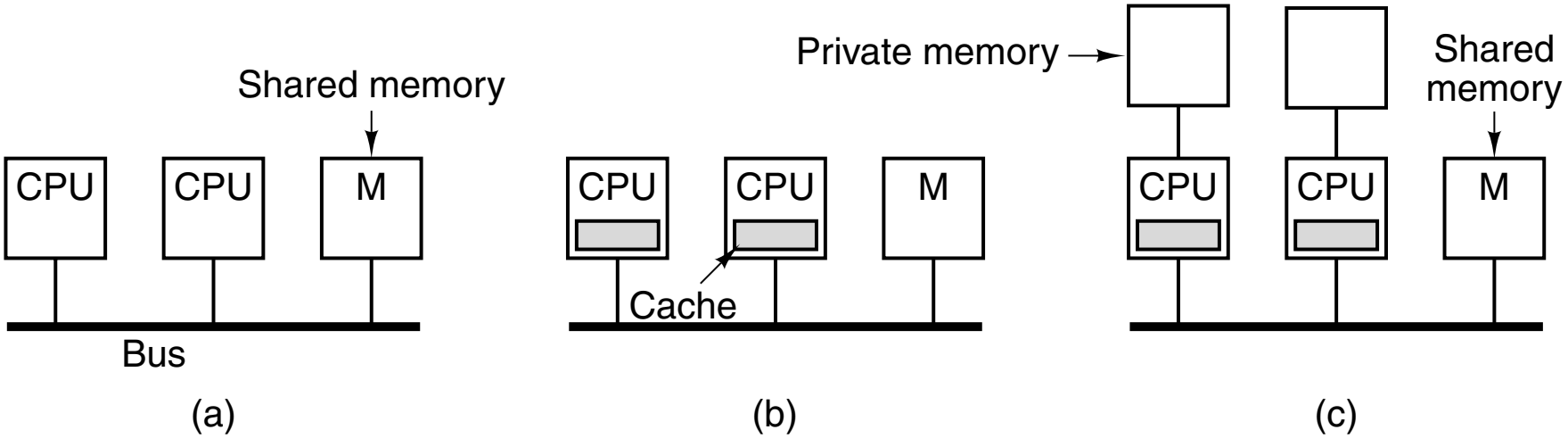
Phân loại các kỹ thuật song song

- Song song mức lệnh
  - pipeline
  - superscalar
- Song song mức dữ liệu
  - SIMD
- Song song mức luồng
  - MIMD
- Song song mức yêu cầu
  - Cloud computing

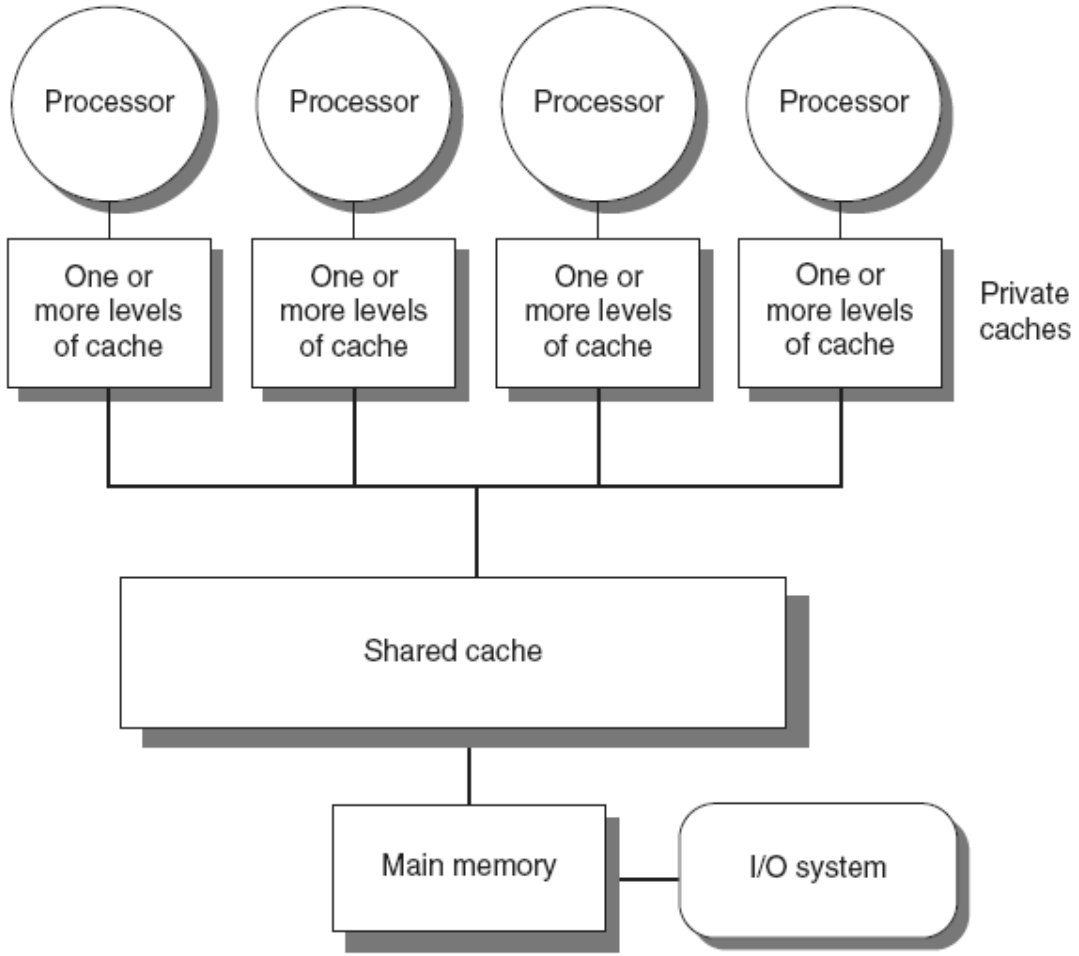
9.2. Đa xử lý bộ nhớ dùng chung

- Hệ thống đa xử lý đối xứng (SMP-Symmetric Multiprocessors)
- Hệ thống đa xử lý không đối xứng (NUMA – Non-Uniform Memory Access)
- Bộ xử lý đa lõi (Multicore Processors)

SMP hay UMA (Uniform Memory Access)



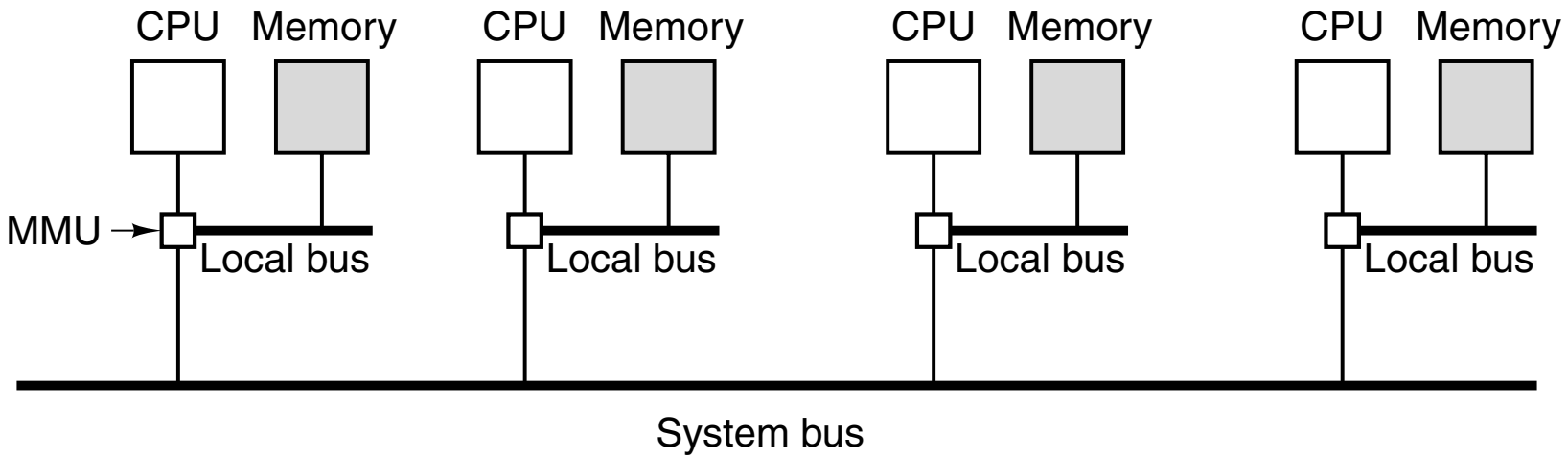
SMP hay UMA (Uniform Memory Access)



SMP (tiếp)

- Một máy tính có n >= 2 bộ xử lý giống nhau
- Các bộ xử lý dùng chung bộ nhớ và hệ thống vào-ra
- Thời gian truy cập bộ nhớ là bằng nhau với các bộ xử lý
- Các bộ xử lý có thể thực hiện chức năng giống nhau
- Hệ thống được điều khiển bởi một hệ điều hành phân tán
- Hiệu năng: Các công việc có thể thực hiện song song
- Khả năng chịu lỗi

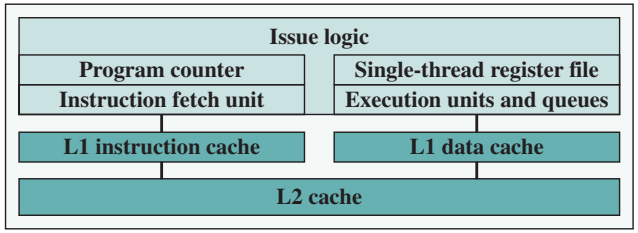
NUMA (Non-Uniform Memory Access)



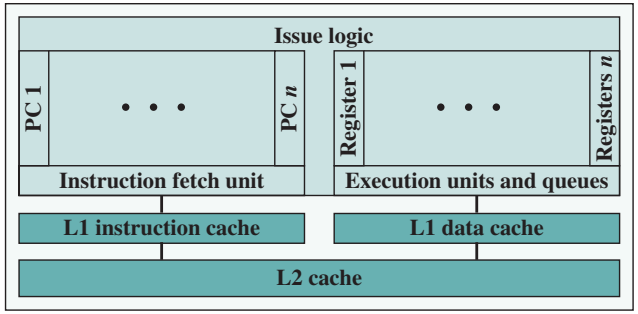
- Có một không gian địa chỉ chung cho tất cả CPU
- Mỗi CPU có thể truy cập từ xa sang bộ nhớ của CPU khác
- Truy nhập bộ nhớ từ xa chậm hơn truy nhập bộ nhớ cục bộ

Bộ xử lý đa lõi (multicores)

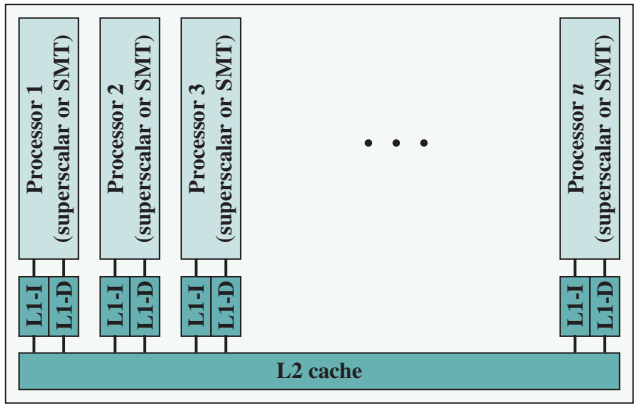
- Thay đổi của bộ xử lý:
  - Tuần tự
  - Pipeline
  - Siêu vô hướng
  - Đa luồng
  - Đa lõi: nhiều CPU trên một chip



(a) Superscalar

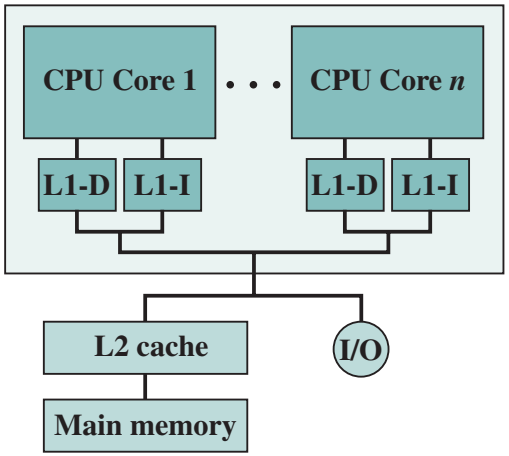


(b) Simultaneous multithreading

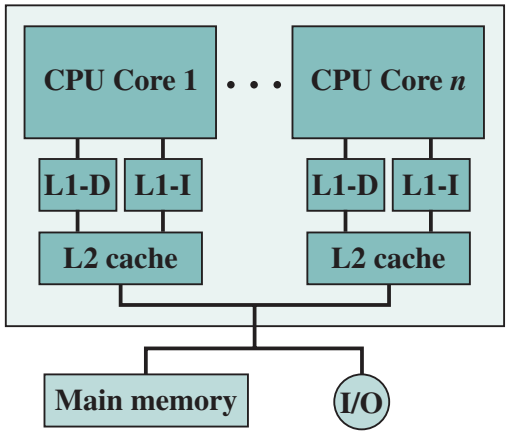


(c) Multicore

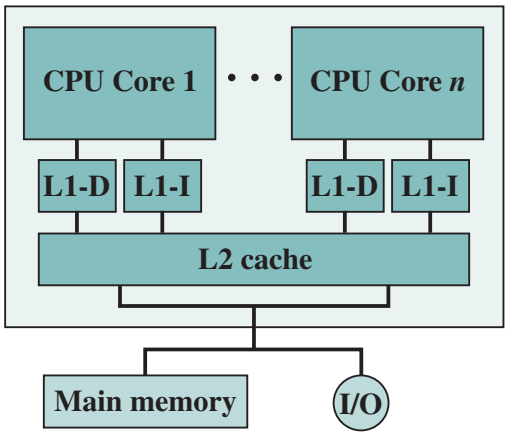
Các dạng tổ chức bộ xử lý đa lõi



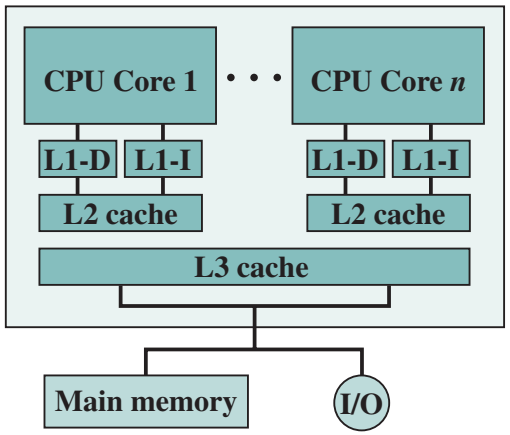
(a) Dedicated L1 cache



(b) Dedicated L2 cache



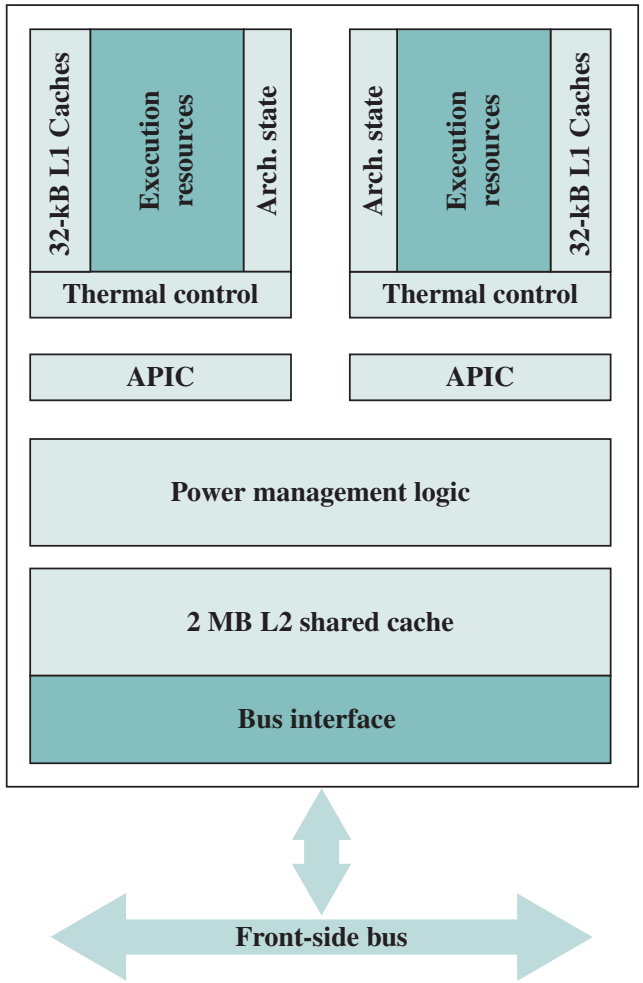
(c) Shared L2 cache



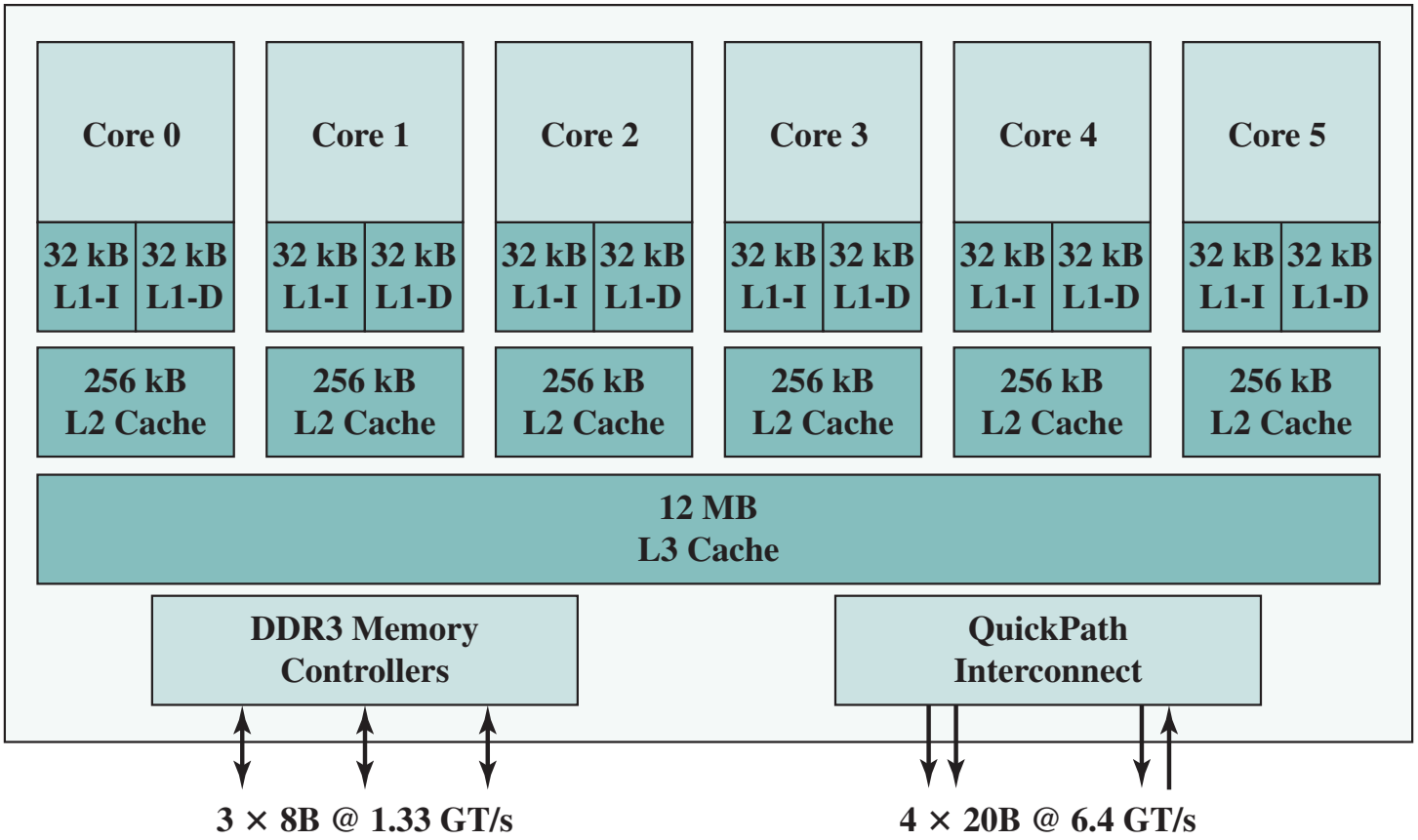
(d) Shared L3 cache

Intel - Core Duo

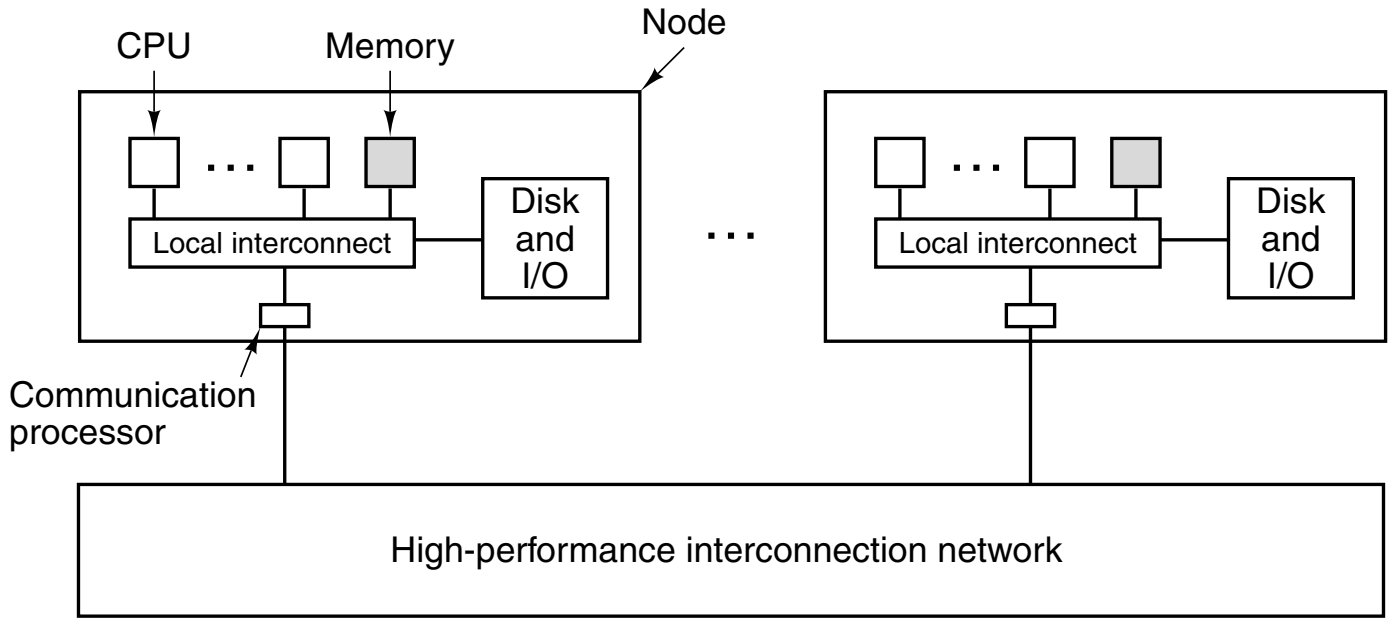
- 2006
- Two x86 superscalar, shared L2 cache
- Dedicated L1 cache per core
  - 32KiB instruction and 32KiB data
- 2MiB shared L2 cache



Intel Core i7-990X

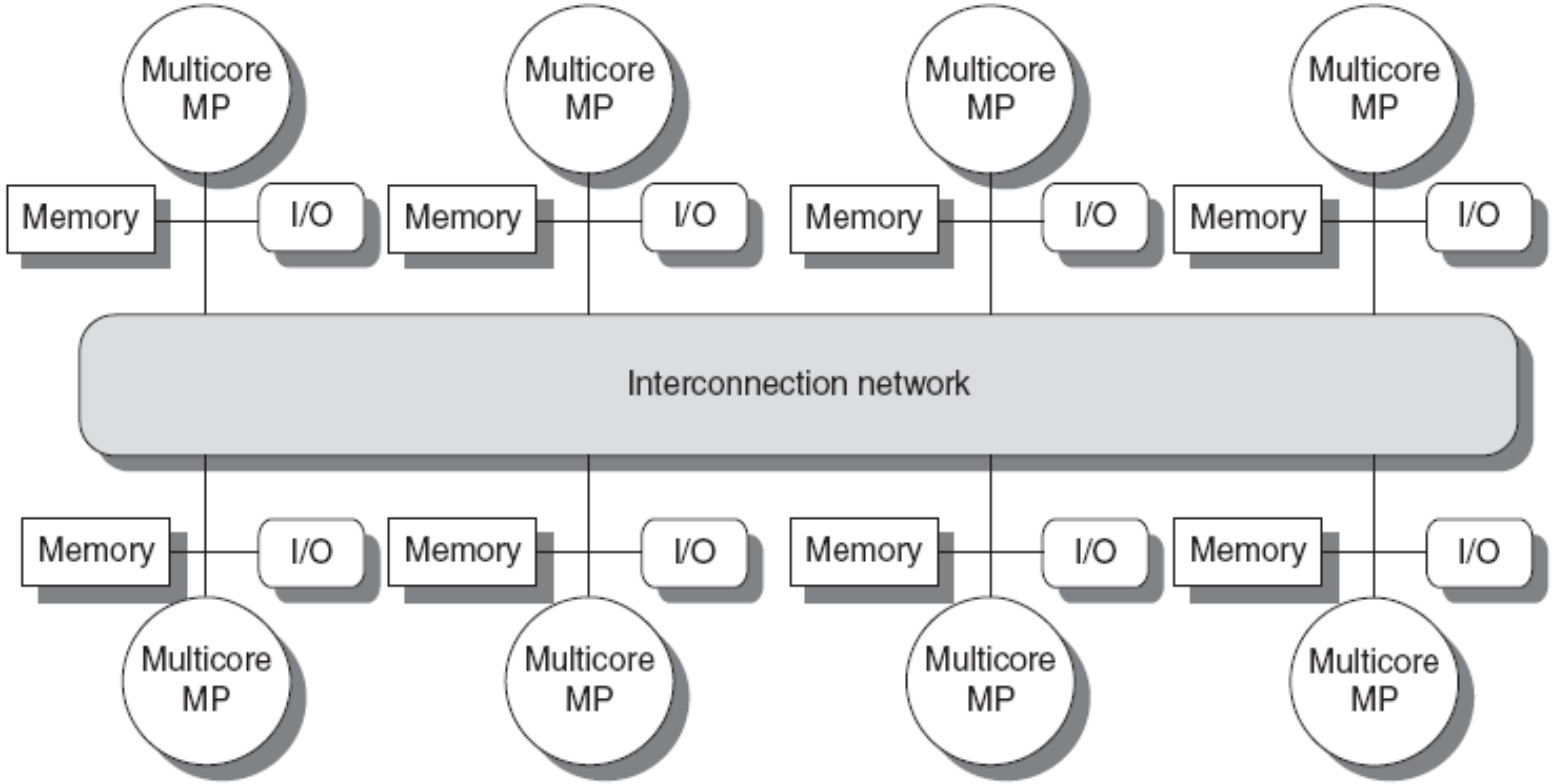


9.3. Đa xử lý bộ nhớ phân tán

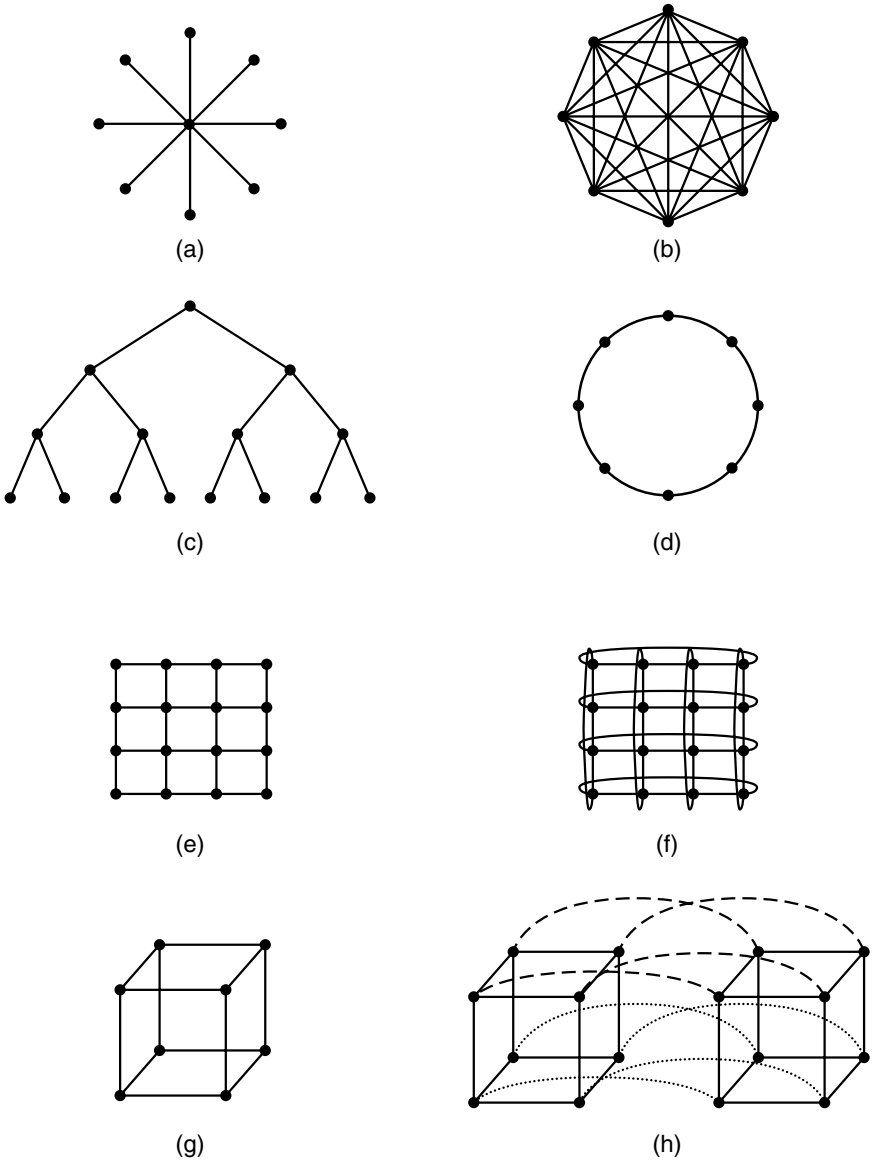


- Máy tính qui mô lớn (Warehouse Scale Computers or Massively Parallel Processors – MPP)
- Máy tính cụm (clusters)

Đa xử lý bộ nhớ phân tán



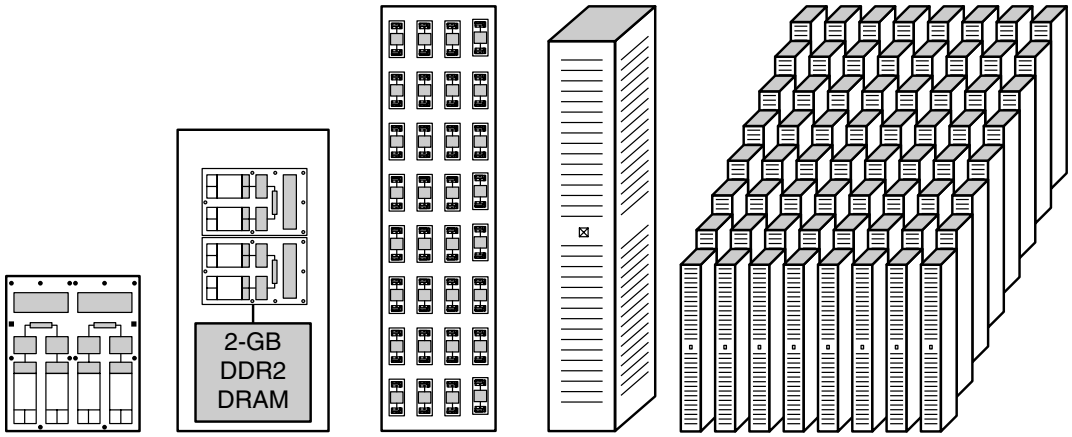
Mạng liên kết



Massively Parallel Processors

- Hệ thống qui mô lớn
- Đắt tiền: nhiều triệu USD
- Dùng cho tính toán khoa học và các bài toán có số phép toán và dữ liệu rất lớn
- Siêu máy tính

IBM Blue Gene/P

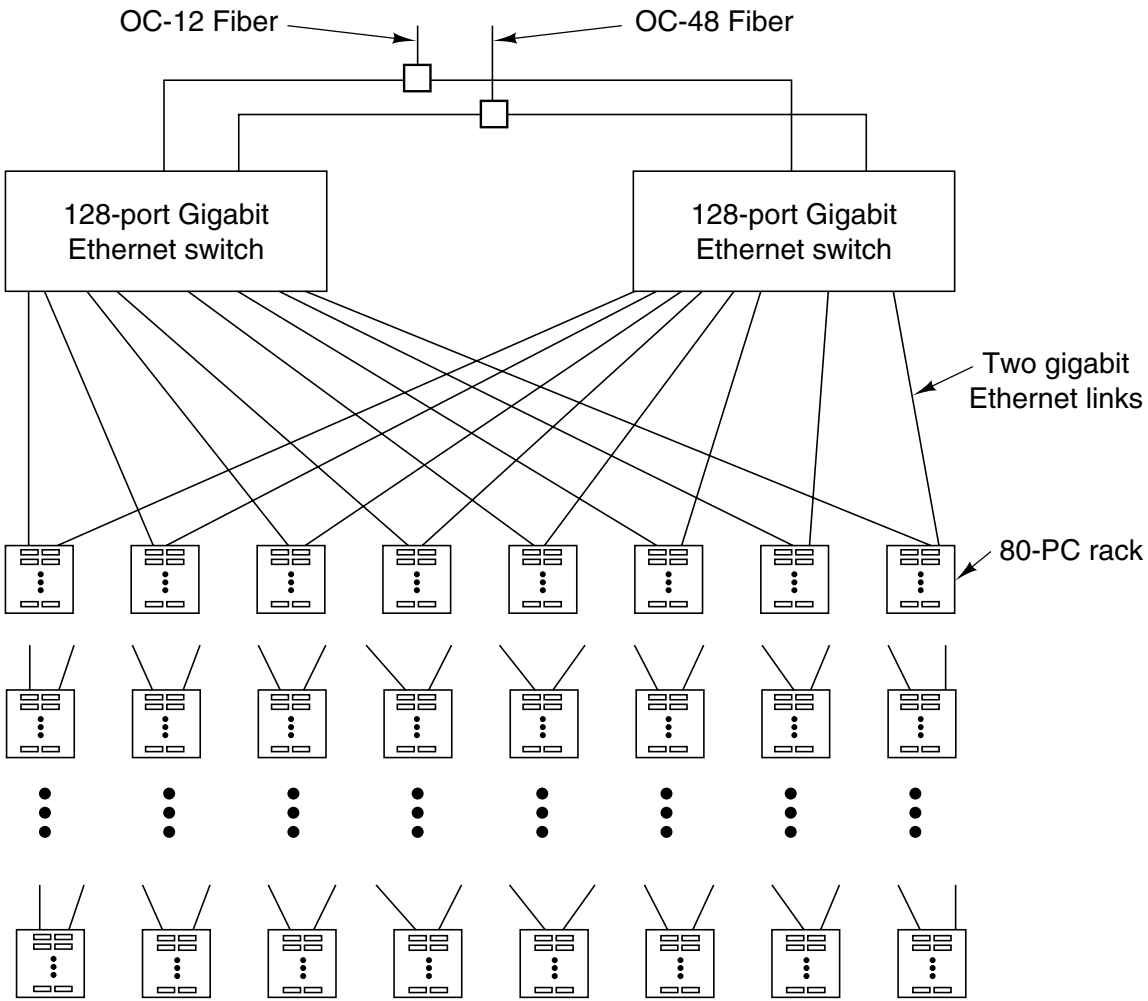


Chip:	Card	Board	Cabinet	System
4 processors 8-MB L3 cache	1 Chip 4 CPUs 2 GB	32 Cards 32 Chips 128 CPUs 64 GB	32 Boards 1024 Cards 1024 Chips 4096 CPUs 2 TB	72 Cabinets 73728 Cards 73728 Chips 294912 CPUs 144 TB
(a)	(b)	(c)	(d)	(e)

Cluster

- Nhiều máy tính được kết nối với nhau bằng mạng liên kết tốc độ cao (~ Gbps)
- Mỗi máy tính có thể làm việc độc lập (PC hoặc SMP)
- Mỗi máy tính được gọi là một node
- Các máy tính có thể được quản lý làm việc song song theo nhóm (cluster)
- Toàn bộ hệ thống có thể coi như là một máy tính song song
- Tính sẵn sàng cao
- Khả năng chịu lỗi lớn

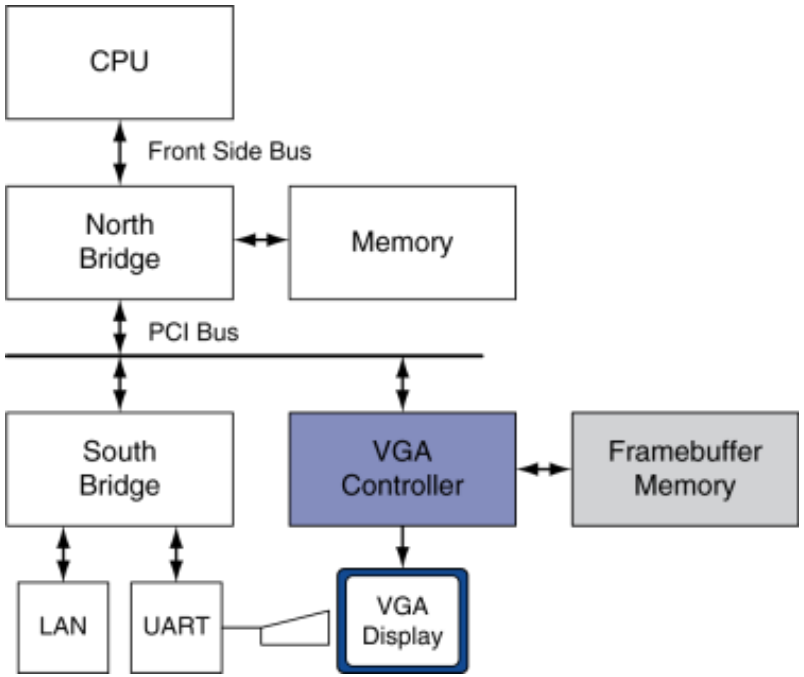
PC Cluster của Google

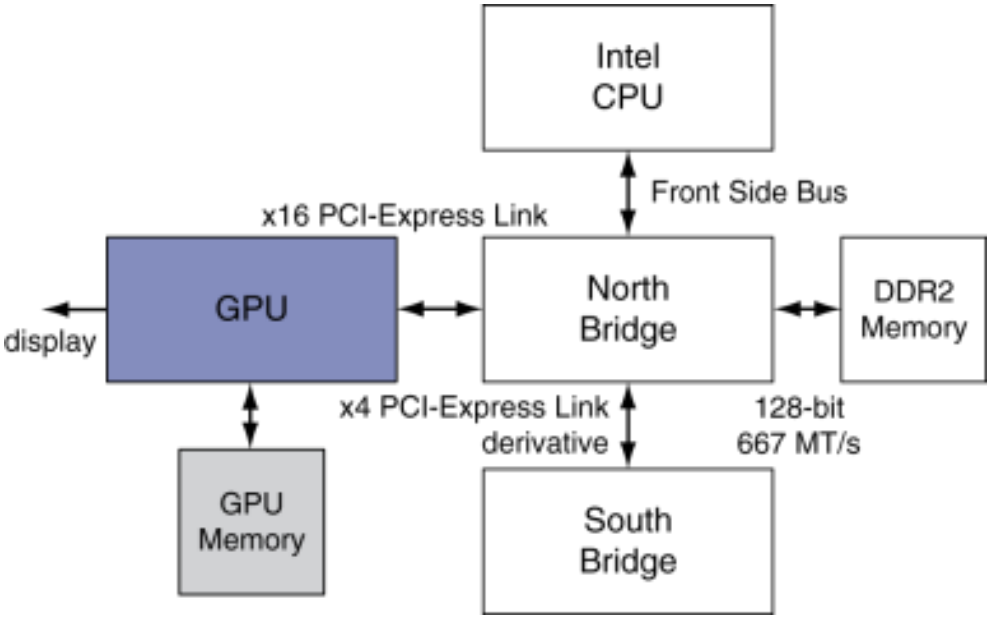


9.4. Bộ xử lý đồ họa đa dụng

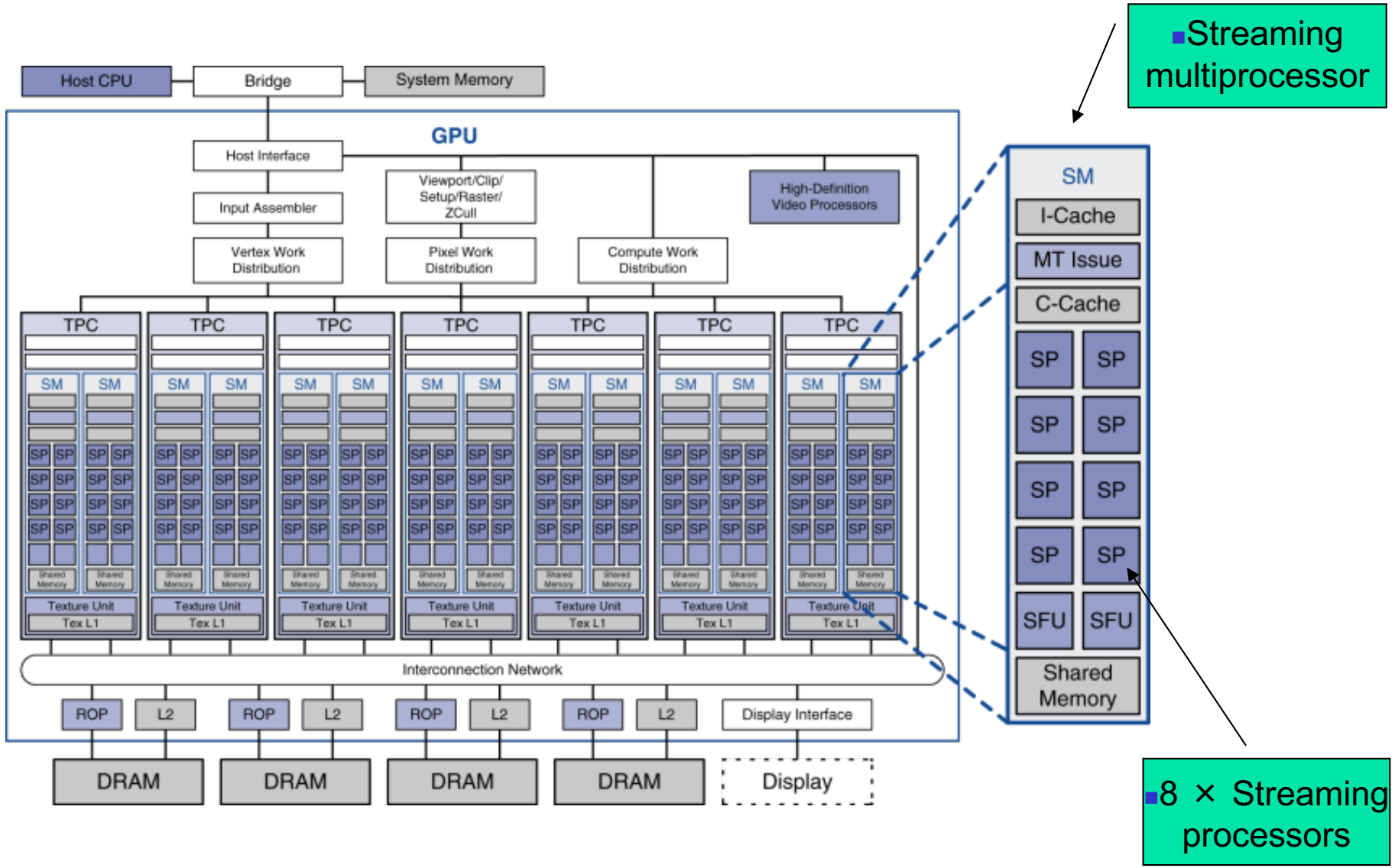
- Kiến trúc SIMD
- Xuất phát từ bộ xử lý đồ họa GPU (Graphic Processing Unit) hỗ trợ xử lý đồ họa 2D và 3D: xử lý dữ liệu song song
- GPGPU – General purpose Graphic Processing Unit
- Hệ thống lai CPU/GPGPU
  - CPU là host: thực hiện theo tuần tự
  - GPGPU: tính toán song song

Bộ xử lý đồ họa trong máy tính

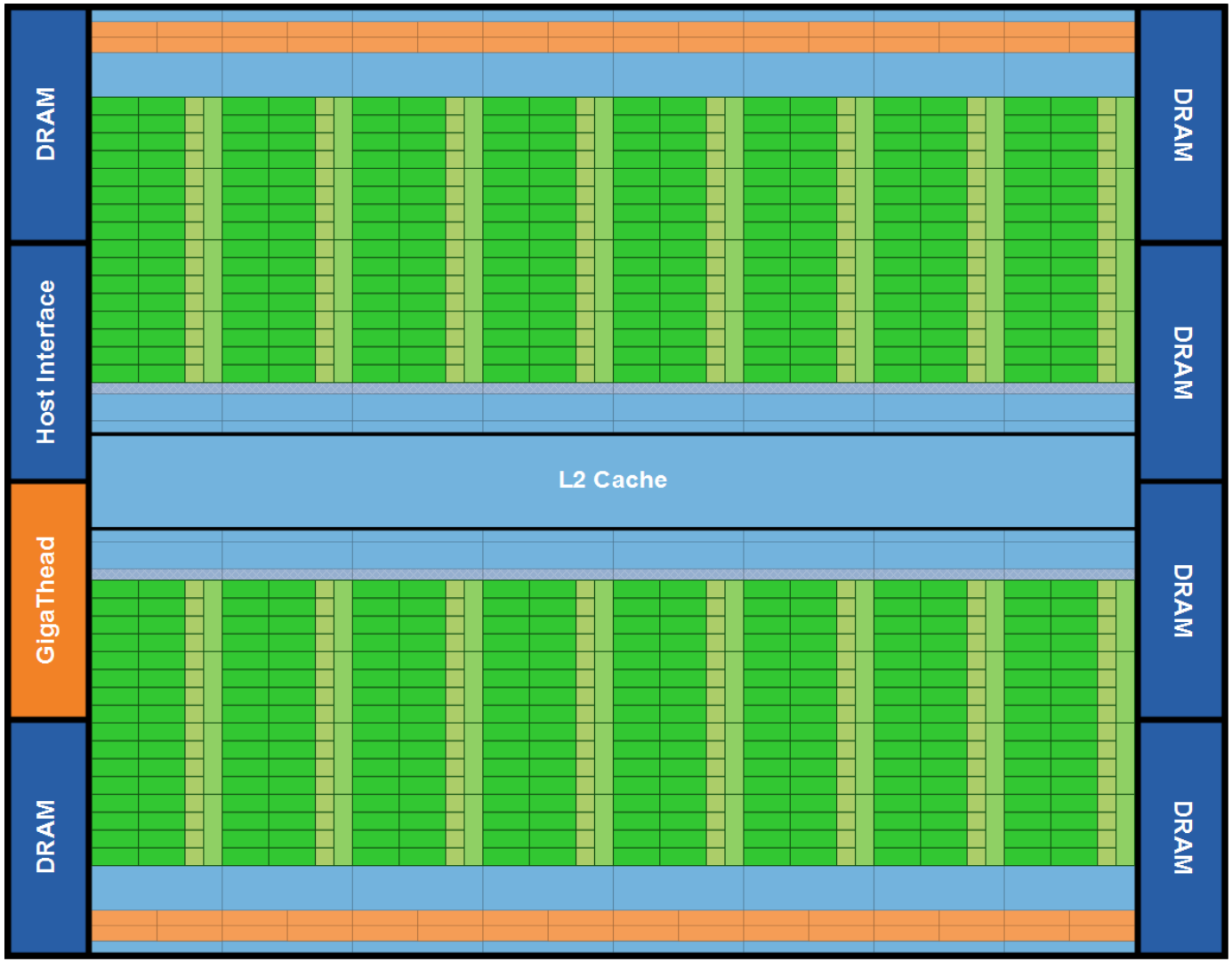




GPGPU: NVIDIA Tesla

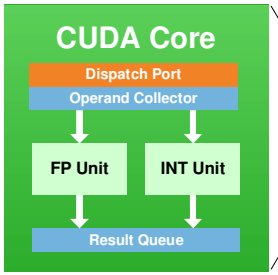


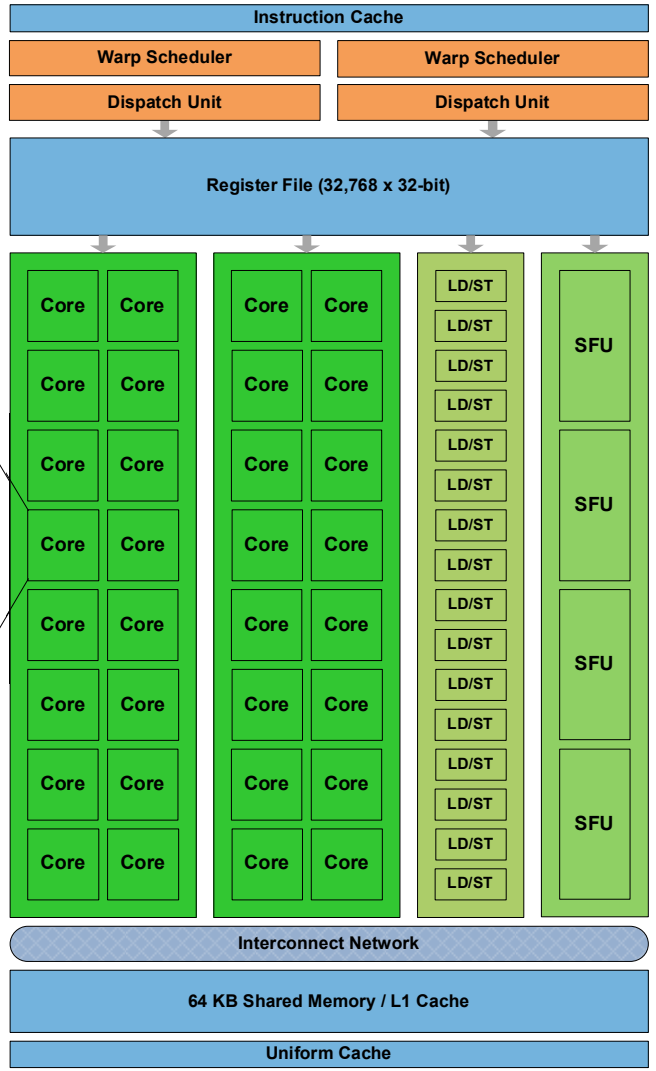
GPGPU: NVIDIA Fermi



NVIDIA Fermi

- Có 16 Streaming Multiprocessors (SM)
- Mỗi SM có 32 CUDA cores.
- Mỗi CUDA core (Cumpute Unified Device Architecture) có 01 FPU và 01 IU





Hết