

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)

Nguyễn Đức Tiến

Version: Apr 2015

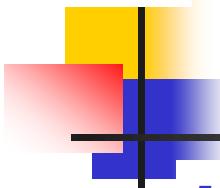
Contact Information

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

- Mobile: 091-313-7399
- e-mail: tiennd@soict.hust.edu.vn
tien.nguyenduc@hust.edu.vn

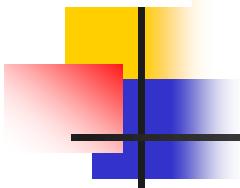
Mục tiêu học phần

- Sinh viên được trang bị các kiến thức cơ sở về kiến trúc tập lệnh và tổ chức của máy tính, cũng như những vấn đề 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 bộ xử lý cụ thể
 - Lập trình hợp ngũ trên một số kiến trúc
 - Đánh giá hiệu năng của các họ máy tính
 - Khai thác và quản trị hiệu quả các hệ thống máy tính
 - Phân tích và thiết kế máy tính



Tài liệu tham khảo chính

- [1] William Stallings - *Computer Organization and Architecture – Designing for Performance* – 2013 (9th edition)
- [2] David A. Patterson & John L. Hennessy - *Computer Organization and Design: The Hardware/Software Interface* – 2012 (revised 4th edition)
- [3] David Money Harris and Sarah L. Harris, *Digital Design and Computer Architecture* – 2013 (2nd edition)
- [4] Andrew S. Tanenbaum - *Structured Computer Organization* – 2012 (6th edition)



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

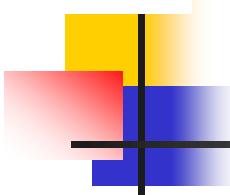
Chú ý: Bài giảng mới nhất Jan 2014

<ftp://dce.hust.edu.vn/khanhnk/CA>

<ftp://dce.hust.edu.vn/tiennd/kientrucmaytinh/>

Chương 1 GIỚI THIỆU CHUNG

Nguyễn Kim Khánh
Trường Đại học Bách khoa Hà Nội



Nội dung

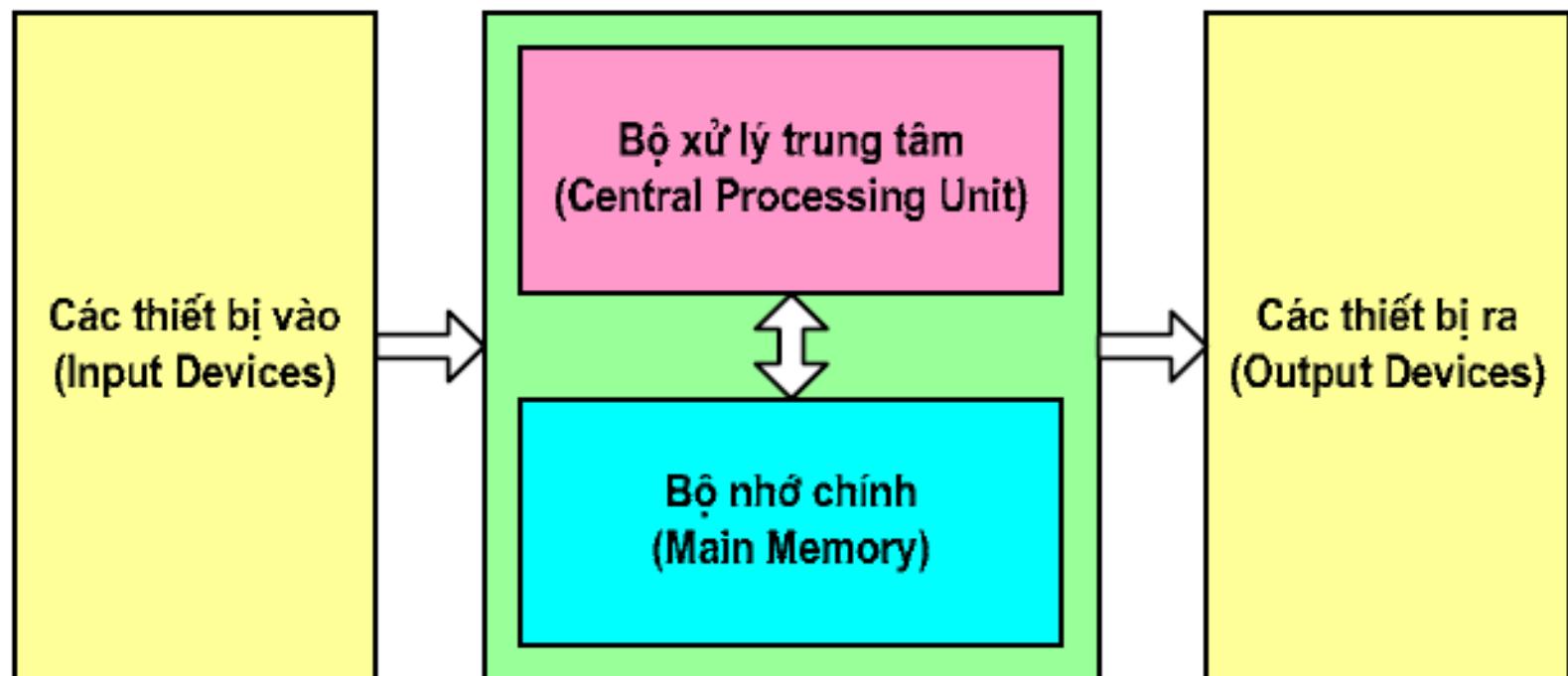
- 1.1. Máy tính và phân loại
- 1.2. Khái niệm kiến trúc máy tính
- 1.3. Sự tiến hóa của máy tính
- 1.4. Hiệu năng máy tính

1.1. Máy tính và phân loại máy tính

1. Máy tính

- **Máy tính (Computer)** là thiết bị điện tử thực hiện các công việc sau:
 - Nhận thông tin vào,
 - Xử lý thông tin theo dãy các lệnh được nhớ sẵn bên trong,
 - Đưa 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.**

Máy tính



2. Phân loại máy tính

- Phân loại truyền thống:
 - Máy vi tính (Microcomputers)
 - Máy tính nhỏ (Minicomputers)
 - Máy tính lớn (Mainframe Computers)
 - Siêu máy tính (Supercomputers)

Phân loại máy tính hiện đại [P&H]

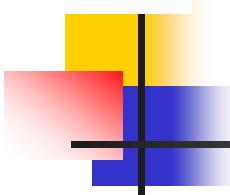
- Thiết bị di động cá nhân (Personal Mobile Devices):
 - Smartphones, Tablet
- Máy tính cá nhân (Personal Computers)
 - Desktop computers, Laptop computers
- Máy chủ (Servers)
 - Thực chất là Máy phục vụ
 - Dùng trong mạng theo mô hình Client/Server
- Máy tính cụm/máy tính qui mô lớn (Clusters/Warehouse Scale Computers):
 - Sử dụng tại các trung tâm tính toán, trung tâm dữ liệu
 - Supercomputers
- Máy tính nhúng (Embedded Computers)
 - Đặt ẩn trong thiết bị khác
 - Được thiết kế chuyên dụng

1.2. Khái niệm kiến trúc máy tính

- Định nghĩa trước đây về kiến trúc máy tính:
 - Là thiết kế kiến trúc tập lệnh (Instruction Set Architecture – ISA)
 - Các thuộc tính của máy tính theo cách nhìn người lập trình (hardware/software interface)
 - Là định nghĩa hẹp

Định nghĩa của Hennessy/ Patterson

- **Kiến trúc máy tính bao gồm:**
 - **Kiến trúc tập lệnh** (Instruction Set Architecture): nghiên cứu máy tính theo cách nhìn của người lập trình (hardware/software interface).
 - **Tổ chức máy tính** (Computer Organization) hay **Vi kiến trúc** (Microarchitecture): nghiên cứu thiết kế máy tính ở mức cao, chẳng hạn như hệ thống nhớ, cấu trúc bus, thiết kế bên trong CPU.
 - **Phần cứng** (Hardware): nghiên cứu thiết kế logic chi tiết và công nghệ đóng gói của máy tính.
- **Kiến trúc tập lệnh thay đổi chậm, tổ chức và phần cứng máy tính thay đổi rất nhanh.**

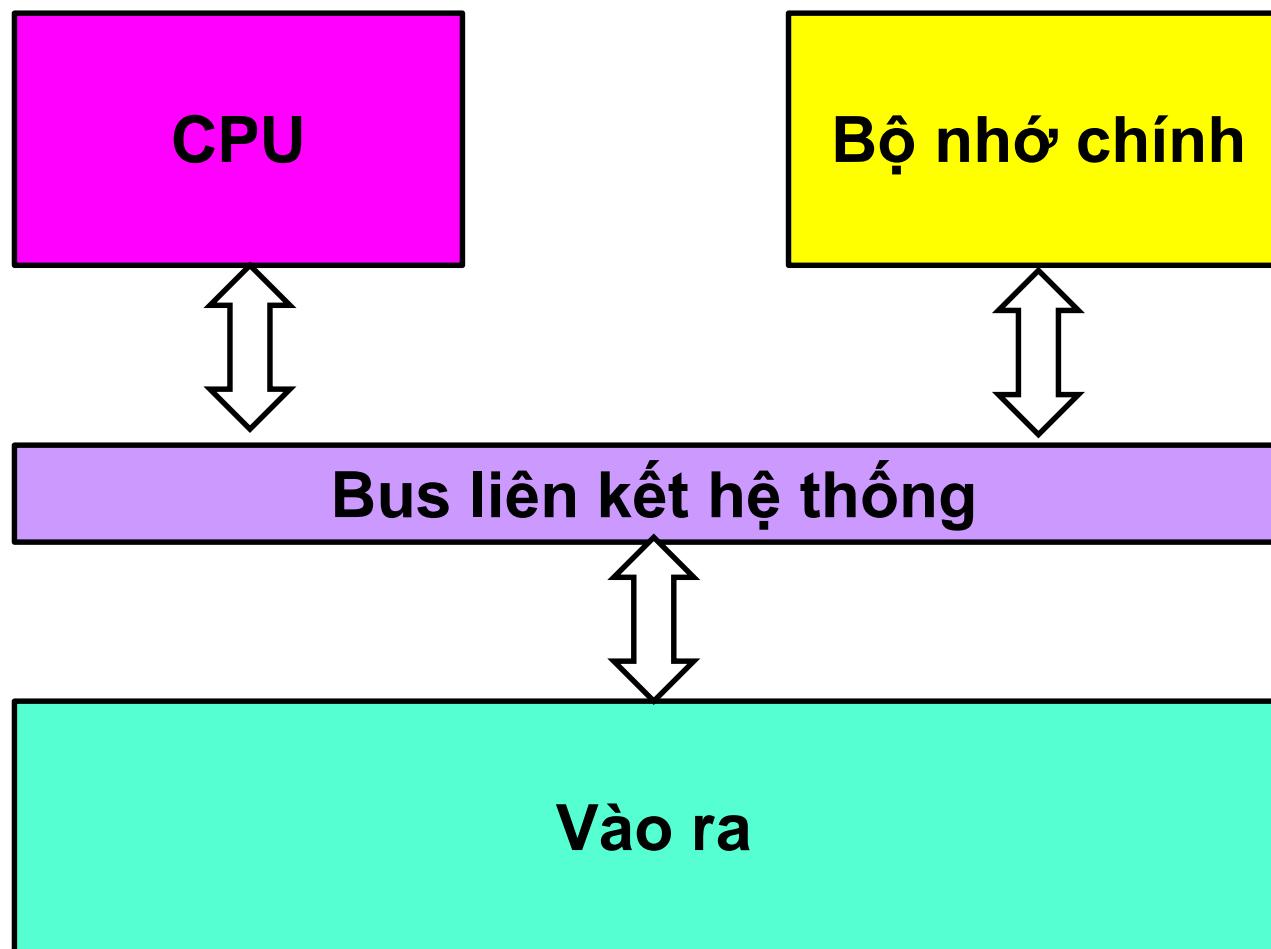


Kiến trúc tập lệnh

Kiến trúc tập lệnh của máy tính bao gồm:

- **Tập lệnh:** tập hợp các chuỗi số nhị phân mã hoá cho các thao tác mà máy tính có thể thực hiện
- **Các kiểu dữ liệu:** các kiểu dữ liệu mà máy tính có thể xử lý

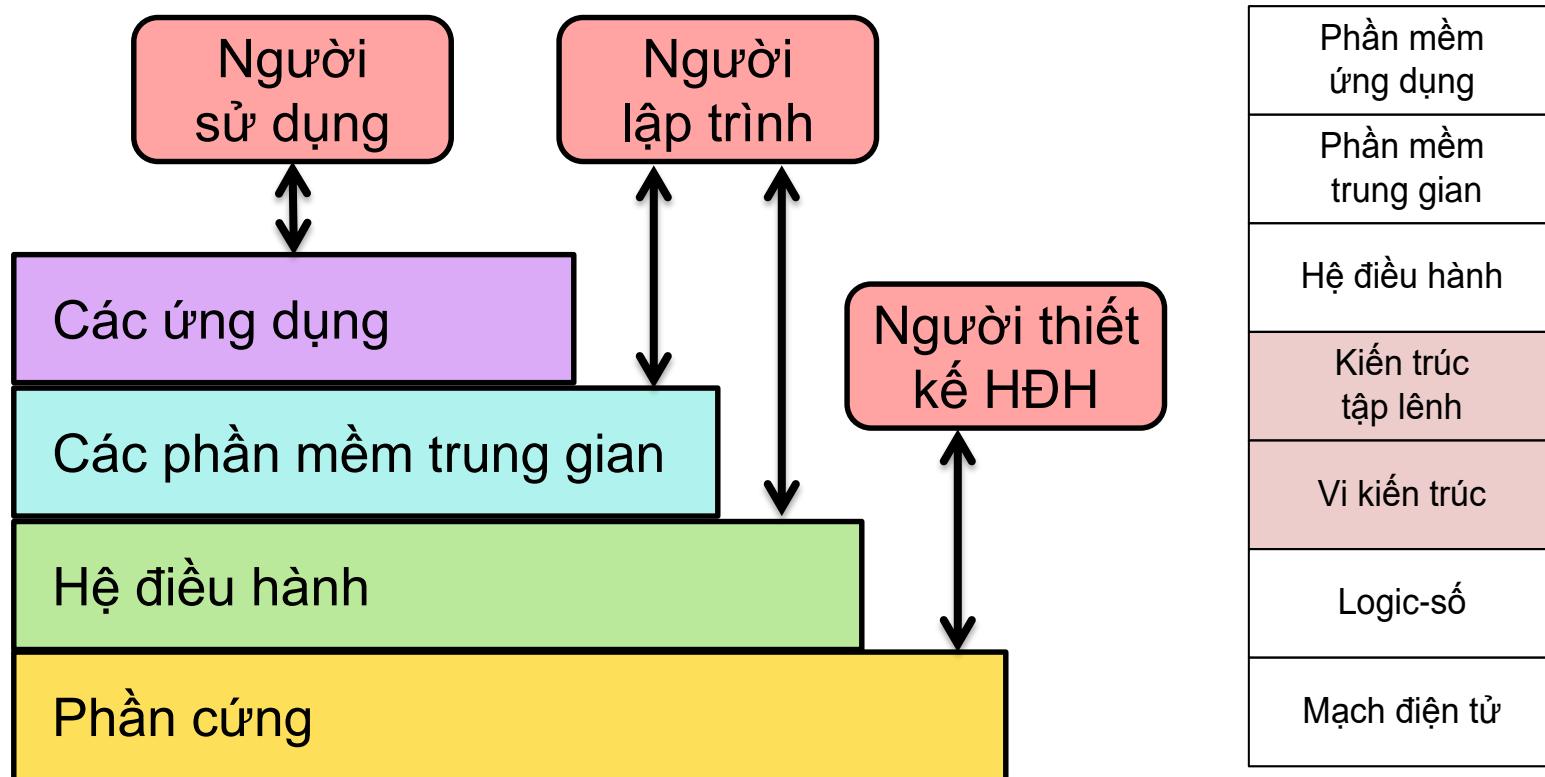
Cấu trúc cơ bản của máy tính



Các thành phần cơ bản của máy tính

- **Bộ xử lý trung tâm** (Central Processing Unit): Điều khiển hoạt động của máy tính và xử lý dữ liệu.
- **Bộ nhớ chính** (Main Memory): Chứa các chương trình và dữ liệu đang được sử dụng.
- **Hệ thống vào-ra** (Input/Output System): Trao đổi thông tin giữa máy tính với bên ngoài.
- **Bus liên kết hệ thống** (System Interconnection Bus): Kết nối và vận chuyển thông tin giữa các thành phần với nhau.

Mô hình phân lớp của máy tính



- **Phần cứng (Hardware):** hệ thống vật lý của máy tính.
- **Phần mềm (Software):** các chương trình và dữ liệu.

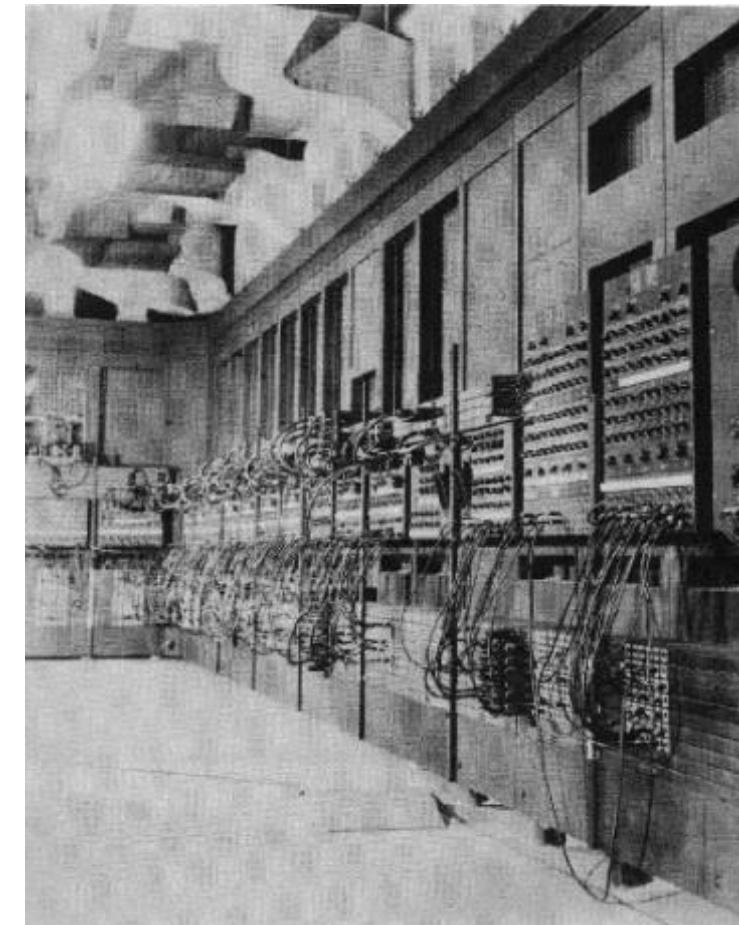
1.3. Sự tiến hóa của máy tính

Các thế hệ máy tính

- Thế hệ thứ nhất: Máy tính dùng đèn điện tử chân không (1950s)
- Thế hệ thứ hai: Máy tính dùng transistor (1960s)
- Thế hệ thứ ba: Máy tính dùng vi mạch SSI, MSI và LSI (1970s)
- Thế hệ thứ tư: Máy tính dùng vi mạch VLSI (1980s)
- Thế hệ thứ năm: Máy tính dùng vi mạch ULSI, SoC (1990s-nay)

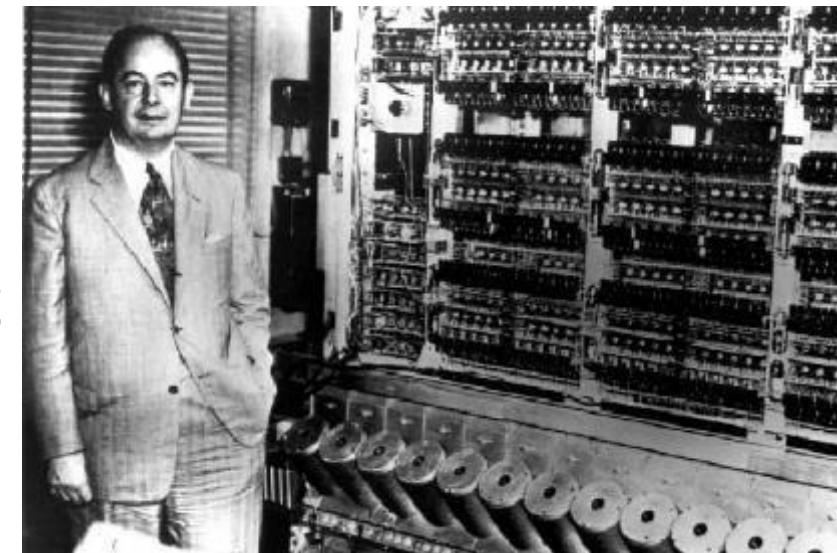
ENIAC – Máy tính điện tử đầu tiên

- Electronic Numerical Intergator and Computer
- Dự án của Bộ Quốc phòng Mỹ
- Do John Mauchly và John Presper Eckert ở Đại học Pennsylvania thiết kế.
- Bắt đầu từ 1943, hoàn thành 1946
- Nặng 30 tấn
- 18000 đèn điện tử và 1500 rơle
- 5000 phép cộng/giây
- Xử lý theo số thập phân
- Bộ nhớ chỉ lưu trữ dữ liệu
- Lập trình bằng cách thiết lập vị trí của các chuyển mạch và các cáp nối.



Máy tính von Neumann

- Chính là máy tính IAS (thực hiện tại Princeton Institute for Advanced Studies)
- Bắt đầu 1947, hoàn thành 1952
- Do John von Neumann thiết kế
- Được xây dựng theo ý tưởng “chương trình được lưu trữ” - (*stored-program concept*) của von Neumann/Turing (1945)
- Trở thành mô hình cơ bản của máy tính



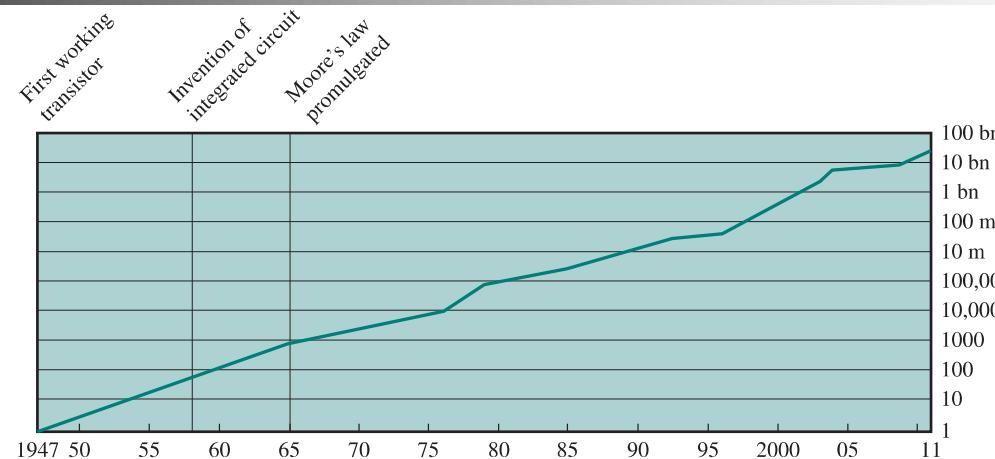
Vi mạch

- Vi mạch hay là mạch tích hợp (Integrated Circuit - IC): mạch điện tử gồm nhiều transistors và các linh kiện khác được tích hợp trên một chip bán dẫn.
- Phân loại vi mạch theo qui mô tích hợp:
 - SSI - Small Scale Integration
 - MSI - Medium Scale Integration
 - LSI - Large Scale Integration
 - VLSI - Very Large Scale Integration
 - ULSI - Ultra Large Scale Integration

Một số vi mạch số điển hình

- **Bộ vi xử lý (Microprocessors)**: CPU được chế tạo trên một chip.
- **Vi mạch điều khiển tổng hợp (Chipset)**: một hoặc một vài vi mạch thực hiện được nhiều chức năng điều khiển và nối ghép.
- **Bộ nhớ bán dẫn (Semiconductor Memory)**: ROM, RAM, Flash
- **Hệ thống trên chip (SoC – System on Chip)**
- **Các bộ vi điều khiển (Microcontrollers)**

Luật Moore



- Gordon Moore – người đồng sáng lập Intel
- Số transistors trên chip sẽ gấp đôi sau 18 tháng
- Giá thành của chip hầu như không thay đổi
- Mật độ cao hơn, do vậy đường dẫn ngắn hơn
- Kích thước nhỏ hơn dẫn tới độ phức tạp tăng lên
- Điện năng tiêu thụ ít hơn
- Hệ thống có ít các chip liên kết với nhau, do đó tăng độ tin cậy

Sự phát triển của 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
- 1985: các bộ xử lý 32-bit
- 2001: các bộ xử lý 64-bit
- 2006: các bộ xử lý đa lõi (multicores)



Máy tính ngày nay



1.4. Hiệu năng máy tính

- Định nghĩa hiệu năng P(Performance):

$$P = \frac{1}{T}$$

trong đó: t là thời gian thực hiện

- “Máy tính A nhanh hơn máy B n lần”

$$\text{Hieu nang}_A / \text{Hieu nang}_B = P_A / P_B$$

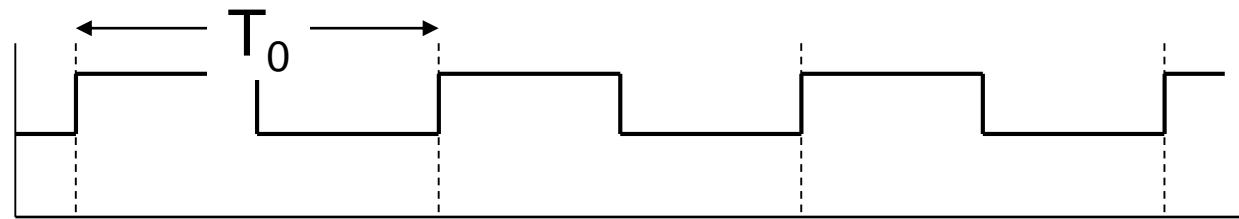
$$= \text{Thoi gian thuc hien}_B / \text{Thoi gian thuc hien}_A = T_B / T_A = n$$

- 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

Xung nhịp của CPU

- Hoạt động của CPU được điều khiển bởi xung nhịp có tần số xác định



- Chu kỳ xung nhịp T_0 (Clock period): thời gian của một chu kỳ
- Tần số xung nhịp f_0 (Clock rate): số chu kỳ trong 1 giây.
 - $f_0 = 1/T_0$
- VD: Bộ xử lý có $f_0 = 4\text{GHz} = 4000\text{MHz} = 4 \times 10^9\text{Hz}$
 $T_0 = 1/(4 \times 10^9) = 0.25 \times 10^{-9}\text{s} = 0.25\text{ns}$

Thời gian CPU (t_{CPU})

- Thời gian thực hiện của CPU
= <số chu kỳ xung nhịp> * <Chu kỳ xung nhịp>

$$T_{CPU} = n * T_0 = n / f_0$$

- trong đó: n là số chu kỳ xung nhịp
- Hiệu năng được tăng lên bằng cách:
 - Giảm số chu kỳ xung nhịp n
 - Tăng tần số xung nhịp f_0

Ví dụ

- Máy tính A:
 - Tần số xung nhịp: $f_A = 2\text{GHz}$
 - Thời gian của CPU: $t_A = 10\text{s}$
- Máy tính B
 - Thời gian của CPU: $t_B = 6\text{s}$
 - Số chu kỳ xung nhịp của B = 1.2 x Số chu kỳ xung nhịp của A
- Xác định tần số xung nhịp của máy B (f_B)?

Ví dụ

- Máy tính A:
 - Tần số xung nhịp: $f_A = 2\text{GHz}$
 - Thời gian của CPU: $t_A = 10\text{s}$
- Máy tính B
 - Thời gian của CPU: $t_B = 6\text{s}$
 - Số chu kỳ xung nhịp của B = $1.2 \times$ Số chu kỳ xung nhịp của A
- Xác định tần số xung nhịp của máy B (f_B)?
- Giải:

$$T_{CPU(A)} = \frac{n(A)}{f_0(A)} = 10\text{s} = \frac{n(A)}{2\text{GHz}}$$

$$T_{CPU(B)} = \frac{n(B)}{f_0(B)} = 6\text{s} = \frac{1.2 * n(A)}{f_0(B)}$$

$$\frac{T_{CPU(A)}}{T_{CPU(B)}} = \frac{10\text{s}}{6\text{s}} = \frac{\frac{n(A)}{2\text{GHz}}}{\frac{1.2 * n(A)}{f_0(B)}} \rightarrow f_0(B) = 4\text{GHz}$$

Số lệnh và số chu kỳ trên một lệnh

- Số chu kỳ = Số lệnh x Số chu kỳ trên một lệnh

$$n = IC \times CPI$$

n - số chu kỳ, IC - số lệnh (Instruction Count), CPI - số chu kỳ trên một lệnh (Clock Cycles per Instruction)

- Thời gian thực hiện của CPU:

<Thời gian thực hiện> = <Số lệnh> * <CPI_{trung bình}> * <Chu kì xung nhịp>

$$T_{CPU} = n * T_0 = IC * CPI * T_0 = \frac{IC * CPI}{f_0}$$

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

Ví dụ

- Máy tính A: $T_A = 250\text{ps}$, $CPI_A = 2.0$
- Máy tính B: $T_B = 500\text{ps}$, $CPI_B = 1.2$
- Cùng kiến trúc tập lệnh (ISA)
- Máy nào nhanh hơn và nhanh hơn bao nhiêu ?

Ví dụ

- Máy tính A: $T_A = 250\text{ps}$, $CPI_A = 2.0$
- Máy tính B: $T_B = 500\text{ps}$, $CPI_B = 1.2$
- Cùng kiến trúc tập lệnh (ISA)
- Máy nào nhanh hơn và nhanh hơn bao nhiêu ?

$$\begin{aligned}t_A &= IC \times CPI_A \times T_A \\&= IC \times 2.0 \times 250\text{ps} = IC \times 500\text{ps}\end{aligned}$$

$$\begin{aligned}t_B &= IC \times CPI_B \times T_B \\&= IC \times 1.2 \times 500\text{ps} = IC \times 600\text{ps}\end{aligned}$$

$$\frac{t_B}{t_A} = \frac{IC \times 600\text{ps}}{IC \times 500\text{ps}} = 1.2$$

Vậy:
A nhanh hơn B 1.2 lần

Chi tiết hơn về CPI

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

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

- CPI trung bình:

$$CPI_{TB} = \frac{\langle \text{Tổng số chu kỳ} \rangle}{\langle \text{Tổng số lệnh} \rangle} = \frac{n}{IC} = \sum_{i=1}^n \left(CPI_i \times \frac{IC_i}{IC} \right)$$

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	2	1	2
IC trong dãy lệnh 2	4	1	1

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	2	1	2
IC trong dãy lệnh 2	4	1	1

- Dãy lệnh 1: IC = 5
 - Số chu kỳ
 $= 2 \times 1 + 1 \times 2 + 2 \times 3$
 $= 10$
 - $CPI_{TB} = 10/5 = 2.0$
- Dãy lệnh 2: IC = 6
 - Số chu kỳ
 $= 4 \times 1 + 1 \times 2 + 1 \times 3$
 $= 9$
 - $CPI_{TB} = 9/6 = 1.5$

Tóm tắt về Hiệu năng

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

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

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

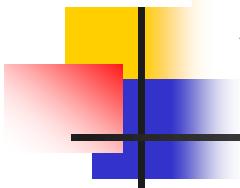
MIPS như là thước đo hiệu năng

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

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

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

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

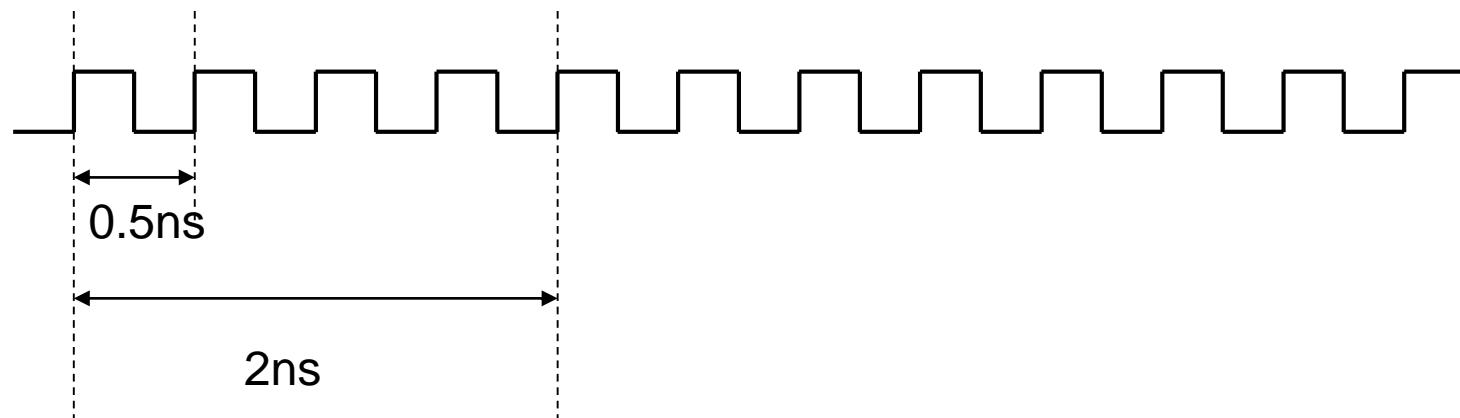


Ví dụ

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



$$1 \text{ chu kỳ} = 1/(2 \times 10^9) = 0,5\text{ns}$$

CPI = 4 → thời gian thực hiện 1 lệnh:

$$4 \times 0,5\text{ns} = 2\text{ns}$$

Vậy bộ xử lý thực hiện được 500 MIPS

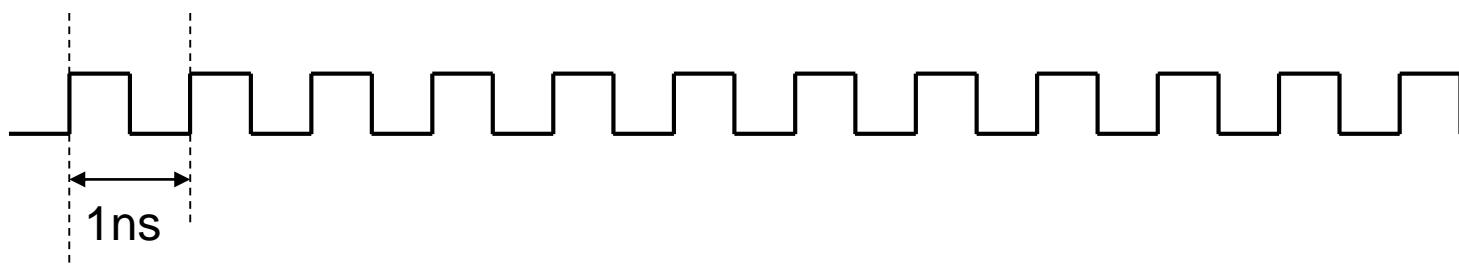


Ví dụ

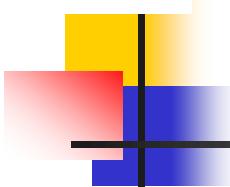
Tính CPI của bộ xử lý với:
clock rate = 1GHz và 400 MIPS?

Ví dụ

Tính CPI của bộ xử lý với:
clock rate = 1GHz và 400 MIPS?



- 4x10⁸ lệnh thực hiện trong 1s
→ 1 lệnh thực hiện trong $1/(4 \times 10^8)$ s = 2,5ns
→ CPI = 2,5



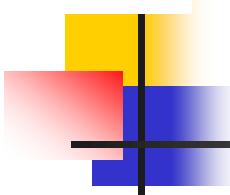
MFLOPS

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})

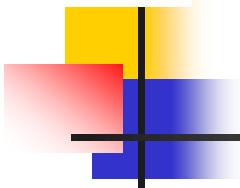


Hết chương 1

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

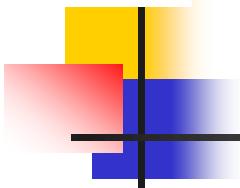
Chương 5. Kiến trúc tập lệnh

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 dây

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 mươi 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

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

Giá trị của A được hiểu như sau:

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

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

Ví dụ số thập phân

$$472.38 = 4 \times 10^2 + 7 \times 10^1 + 2 \times 10^0 + 3 \times 10^{-1} + 8 \times 10^{-2}$$

- 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:

- $0.38 \times 10 = 3.8$ phần nguyên = 3
- $0.8 \times 10 = 8.0$ phần nguyên = 8



2. Hệ nhị phân

- Cơ số 2
- 2 chữ số nhị phân: 0 và 1
- Chữ số nhị phân 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:
 - $00\dots000 = 0$
 - $11\dots111 = 2^n - 1$

Bits, Bytes, Nibbles...

■ Bits

10010110

most significant bit least significant bit

■ Bytes & Nibbles

byte
10010110
nibble

■ Bytes

CEBF9AD7

most significant byte least significant byte

Lũy thừa hai

- $2^{10} = 1 \text{ Ki}$ $\approx 1000 \text{ (1024)}$
- $2^{20} = 1 \text{ Mi}$ $\approx 1 \text{ triệu (1,048,576)}$
- $2^{30} = 1 \text{ Gi}$ $\approx 1 \text{ tỷ (1,073,741,824)}$
- $2^{40} = 1 \text{ Ti}$ $\approx 1000 \text{ tỷ}$
- $2^{50} = 1 \text{ Pi}$ $\approx 1 \text{ triệu tỷ}$

Dạng tổng quát của số nhị phân

Có một số nhị phân A như sau:

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

Giá trị của A được tính như sau:

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

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

Ví dụ số nhị phân

$$\begin{array}{ccccccccc} 1 & 1 & 0 & 1 & 0 & 0 & 1 & . & 1 & 0 & 1 & 1 \\ & 6 & 5 & 4 & 3 & 2 & 1 & 0 & -1 & -2 & -3 & -4 \end{array} _{(2)} =$$

$$\begin{aligned} &= 1 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + \\ &1 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} + 1 \times 2^{-4} \\ &= 2^6 + 2^5 + 2^3 + 2^0 + 2^{-1} + 2^{-3} + 2^{-4} \\ &= 64 + 32 + 8 + 1 + 0.5 + 0.125 + 0.0625 \\ &= 105.6875_{(10)} \end{aligned}$$

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	đư	1
■	$52 : 2 =$	26	đư	0
■	$26 : 2 =$	13	đư	0
■	$13 : 2 =$	6	đư	1
■	$6 : 2 =$	3	đư	0
■	$3 : 2 =$	1	đư	1
■	$1 : 2 =$	0	đư	1



- 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: $17000_{(10)} = 16384 + 512 + 64 + 32 + 8$
 $= 2^{14} + 2^9 + 2^6 + 2^5 + 2^3$

$$17000_{(10)} = 0100\ 0010\ 0110\ 1000_{(2)}$$

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

Chuyển đổi số lẻ thập phân sang nhị phân

■ Ví dụ 1: chuyển đổi $0.6875_{(10)}$

- $0.6875 \times 2 = 1.375$ phần nguyên = 1
- $0.375 \times 2 = 0.75$ phần nguyên = 0
- $0.75 \times 2 = 1.5$ phần nguyên = 1
- $0.5 \times 2 = 1.0$ phần nguyên = 1



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

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

- Ví dụ 2: chuyển đổi $0.81_{(10)}$

- $0.81 \times 2 = 1.62$ phần nguyên = 1
 - $0.62 \times 2 = 1.24$ phần nguyên = 1
 - $0.24 \times 2 = 0.48$ phần nguyên = 0
 - $0.48 \times 2 = 0.96$ phần nguyên = 0
 - $0.96 \times 2 = 1.92$ phần nguyên = 1
 - $0.92 \times 2 = 1.84$ phần nguyên = 1
 - $0.84 \times 2 = 1.68$ phần nguyên = 1

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

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

■ Ví dụ 3: chuyển đổi $0.2_{(10)}$

- $0.2 \times 2 = 0.4$ phần nguyên = 0
- $0.4 \times 2 = 0.8$ phần nguyên = 0
- $0.8 \times 2 = 1.6$ phần nguyên = 1
- $0.6 \times 2 = 1.2$ phần nguyên = 1
- $0.2 \times 2 = 0.4$ phần nguyên = 0
- $0.4 \times 2 = 0.8$ phần nguyên = 0
- $0.8 \times 2 = 1.6$ phần nguyên = 1
- $0.6 \times 2 = 1.2$ phần nguyên = 1



- $0.2_{(10)} \approx 0.00110011_{(2)}$

3. Hệ mươi 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

4-bit	Chữ số Hexa
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

Ví dụ chuyển đổi số nhị phân → số Hexa:

- $1011\ 0011_2 = B3_{16}$
- $0000\ 0000_2 = 00_{16}$
- $0010\ 1101\ 1001\ 1010_2 = 2D9A_{16}$
- $1111\ 1111\ 1111\ 1111_2 = FFFF_{16}$

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)
- Phép toán logic cơ bản là AND, OR và NOT với ký hiệu như sau:
 - A AND B : $A \cdot B$
 - A OR B : $A + B$
 - NOT A : \bar{A}
- Thứ tự ưu tiên: NOT > AND > OR

Các phép toán logic (tiếp)

■ Các phép toán NAND, NOR, XOR:

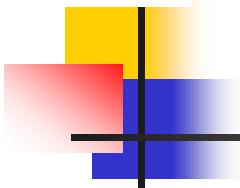
■ A NAND B: $\overline{A \bullet B}$

■ 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

A	B	NOT A \bar{A}	A AND B $A \cdot B$	A OR B $A+B$	A NAND B $\bar{A \cdot B}$	A NOR B $\bar{A+B}$	A XOR B $A \oplus B$
0	0	1	0	0	1	1	0
0	1	1	0	1	1	0	1
1	0	0	0	1	1	0	1
1	1	0	1	1	0	0	0



Các đồng nhất thức của đại số Boole

$$A \cdot B = B \cdot A$$

$$A \cdot (B + C) = (A \cdot B) + (A \cdot C)$$

$$1 \cdot A = A$$

$$A \cdot \overline{A} = 0$$

$$0 \cdot A = 0$$

$$A \cdot A = A$$

$$A \cdot (B \cdot C) = (A \cdot B) \cdot C$$

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

$$A + B = B + A$$

$$A + (B \cdot C) = (A + B) \cdot (A + C)$$

$$0 + A = A$$

$$A + \overline{A} = 1$$

$$1 + A = 1$$

$$A + A = A$$

$$A + (B + C) = (A + B) + C$$

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

2.3. Các cổng logic (Logic Gates)

- Thực hiện các hàm logic:
 - NOT, AND, OR, NAND, NOR, etc.
- Cổng logic một đầu vào:
 - Cổng NOT, bộ đệm (buffer)
- Cổng hai đầu vào:
 - AND, OR, XOR, NAND, NOR, XNOR
- Cổng nhiều đầu vào

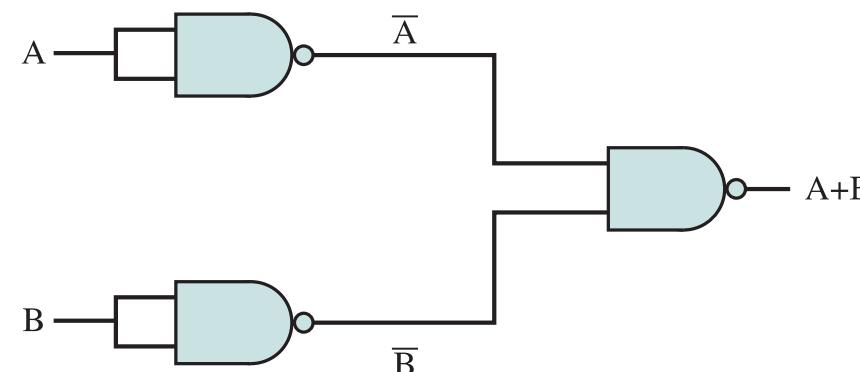
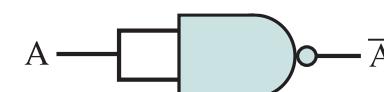
Các công logic

Name	Graphical Symbol	Algebraic Function	Truth Table															
AND		$F = A \cdot B$ or $F = AB$	<table border="1"> <thead> <tr> <th>A</th><th>B</th><th>F</th></tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>0</td></tr> <tr><td>1</td><td>0</td><td>0</td></tr> <tr><td>1</td><td>1</td><td>1</td></tr> </tbody> </table>	A	B	F	0	0	0	0	1	0	1	0	0	1	1	1
A	B	F																
0	0	0																
0	1	0																
1	0	0																
1	1	1																
OR		$F = A + B$	<table border="1"> <thead> <tr> <th>A</th><th>B</th><th>F</th></tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>0</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>1</td></tr> </tbody> </table>	A	B	F	0	0	0	0	1	1	1	0	1	1	1	1
A	B	F																
0	0	0																
0	1	1																
1	0	1																
1	1	1																
NOT		$F = \bar{A}$ or $F = A'$	<table border="1"> <thead> <tr> <th>A</th><th>F</th></tr> </thead> <tbody> <tr><td>0</td><td>1</td></tr> <tr><td>1</td><td>0</td></tr> </tbody> </table>	A	F	0	1	1	0									
A	F																	
0	1																	
1	0																	
NAND		$F = \overline{AB}$	<table border="1"> <thead> <tr> <th>A</th><th>B</th><th>F</th></tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>1</td></tr> <tr><td>0</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>0</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>0</td></tr> </tbody> </table>	A	B	F	0	0	1	0	1	1	1	0	1	1	1	0
A	B	F																
0	0	1																
0	1	1																
1	0	1																
1	1	0																
NOR		$F = \overline{A + B}$	<table border="1"> <thead> <tr> <th>A</th><th>B</th><th>F</th></tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>1</td></tr> <tr><td>0</td><td>1</td><td>0</td></tr> <tr><td>1</td><td>0</td><td>0</td></tr> <tr><td>1</td><td>1</td><td>0</td></tr> </tbody> </table>	A	B	F	0	0	1	0	1	0	1	0	0	1	1	0
A	B	F																
0	0	1																
0	1	0																
1	0	0																
1	1	0																
XOR		$F = A \oplus B$	<table border="1"> <thead> <tr> <th>A</th><th>B</th><th>F</th></tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>0</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>0</td></tr> </tbody> </table>	A	B	F	0	0	0	0	1	1	1	0	1	1	1	0
A	B	F																
0	0	0																
0	1	1																
1	0	1																
1	1	0																

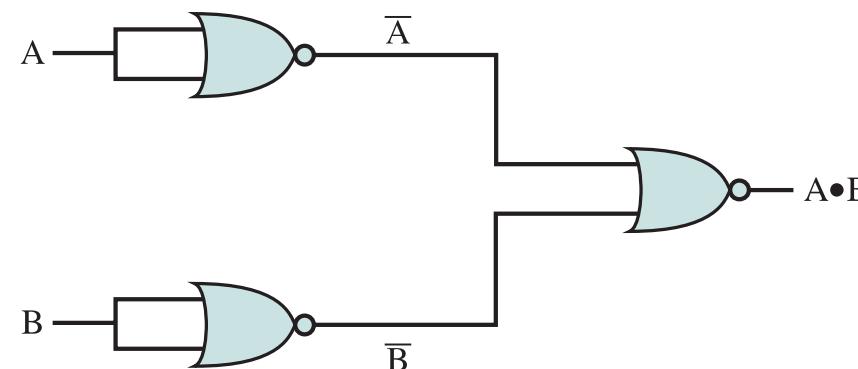
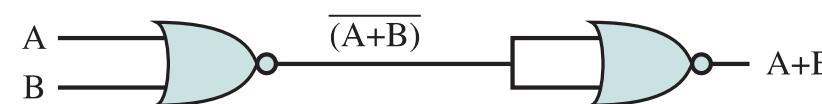
Tập đầy đủ

- 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}

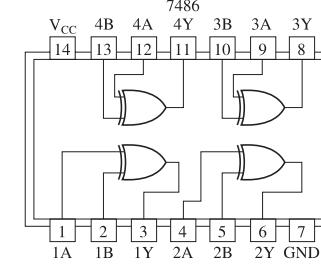
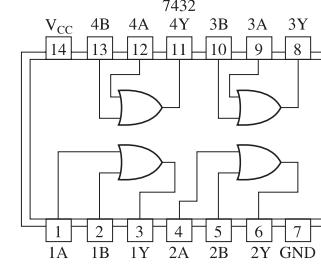
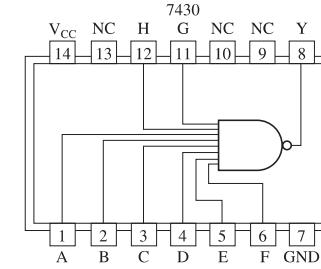
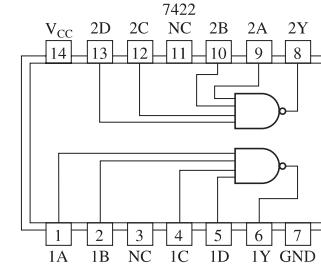
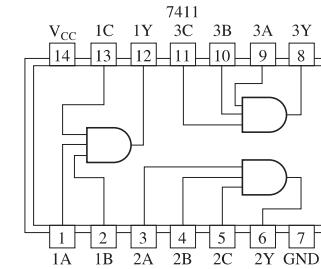
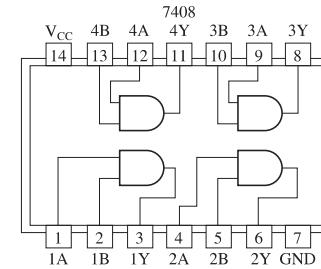
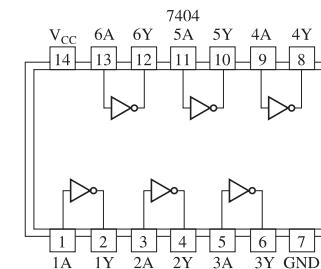
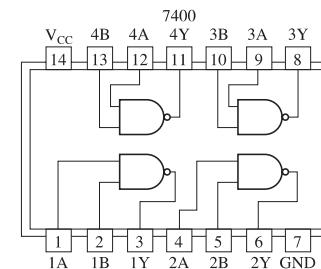
Sử dụng cổng NAND



Sử dụng cổng NOR



Một số ví dụ vi mạch logic



2.4. Mạch tổ hợp

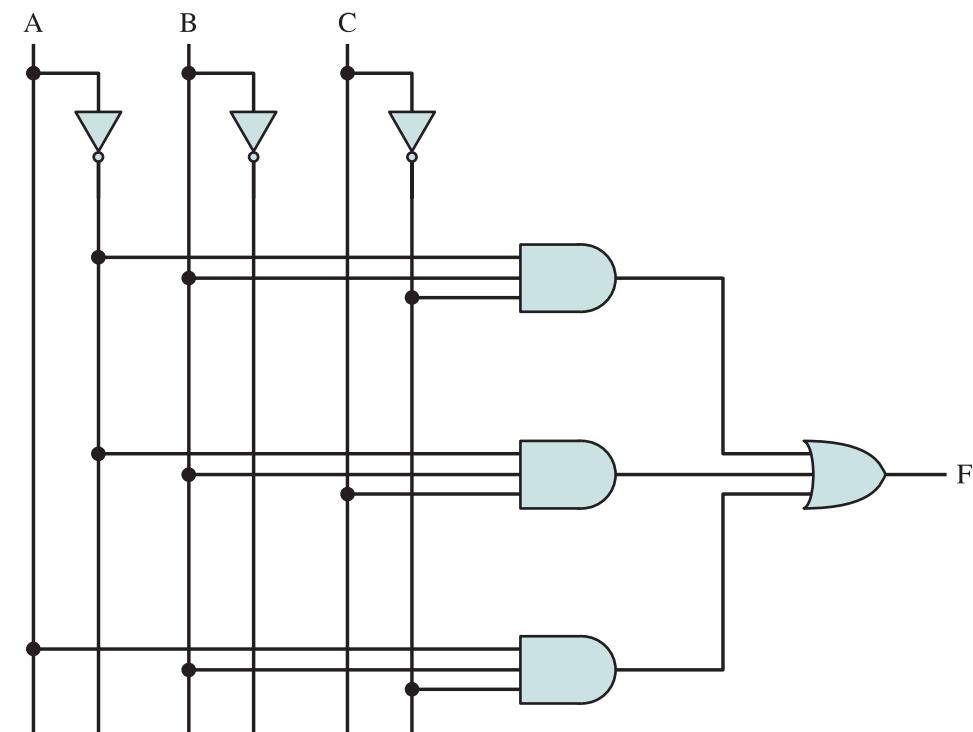
- Mạch logic là mạch bao gồm:
 - Các đầu vào (Inputs)
 - Các đầu ra (Outputs)
 - Đặc tả chức năng (Functional specification)
 - Đặc tả thời gian (Timing specification)
- Các kiểu mạch logic:
 - Mạch logic tổ hợp (Combinational Logic)
 - Mạch không nhớ
 - Đầu ra được xác định bởi các giá trị hiện tại của đầu vào
 - Mạch logic dây (Sequential Logic)
 - 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 cơ bản
- Mạch tổ hợp có thể được định nghĩa theo ba cách:
 - Bảng thật
 - Dạng sơ đồ
 - Phương trình Boole

Ví dụ

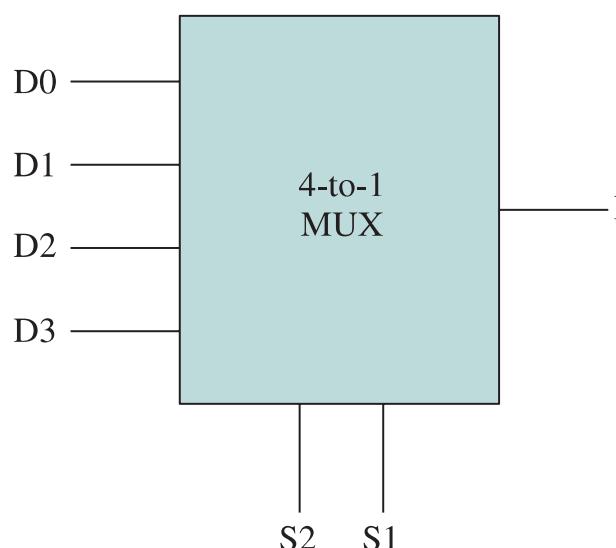
A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0



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

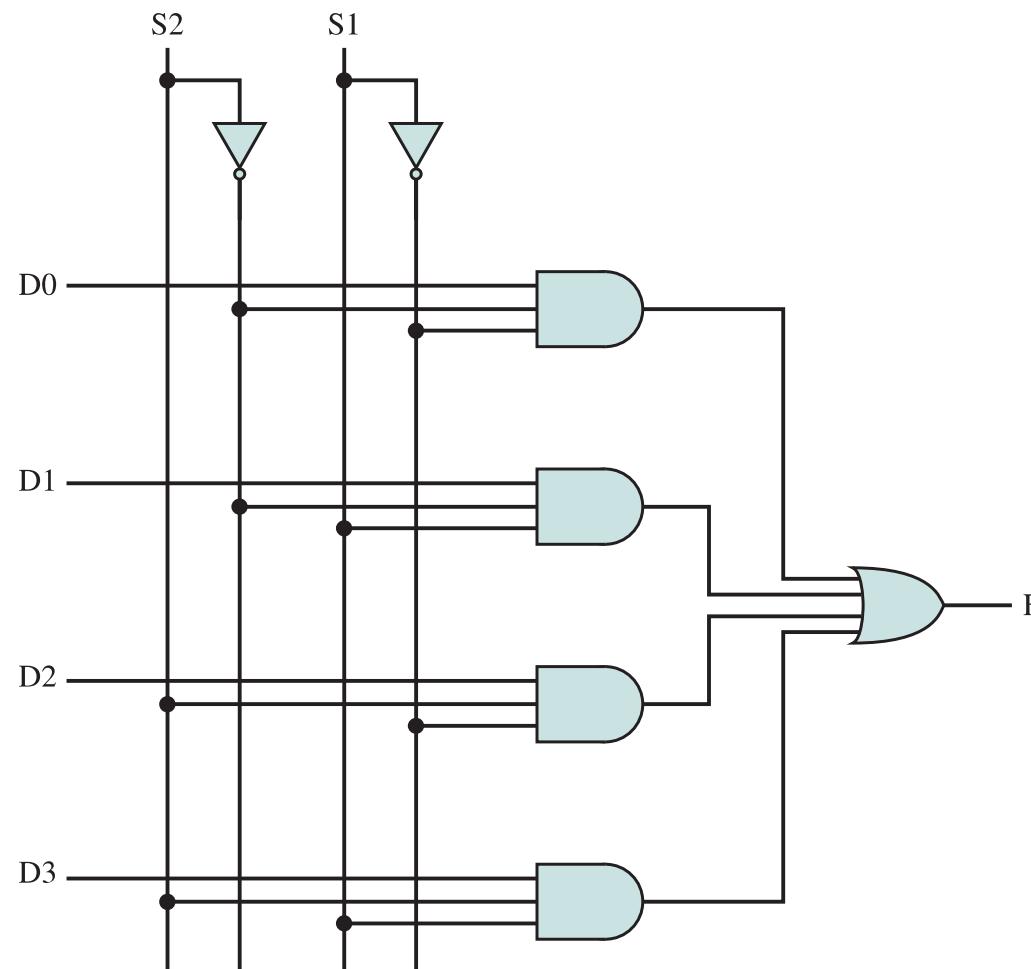
Bộ dồn kênh (Multiplexer-MUX)

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



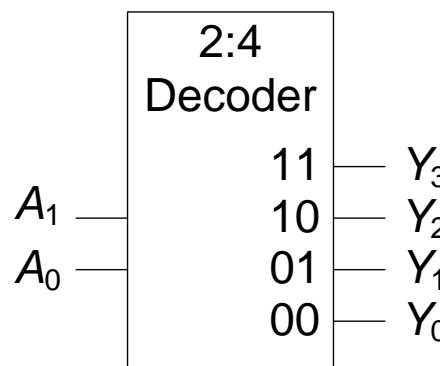
S2	S1	F
0	0	D0
0	1	D1
1	0	D2
1	1	D3

Thực hiện MUX bốn đầu vào

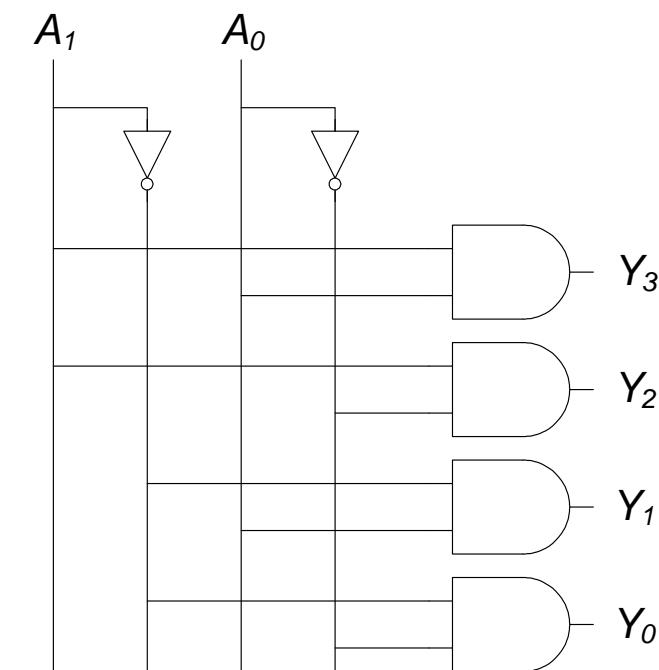


Bộ giải mã (Decoder)

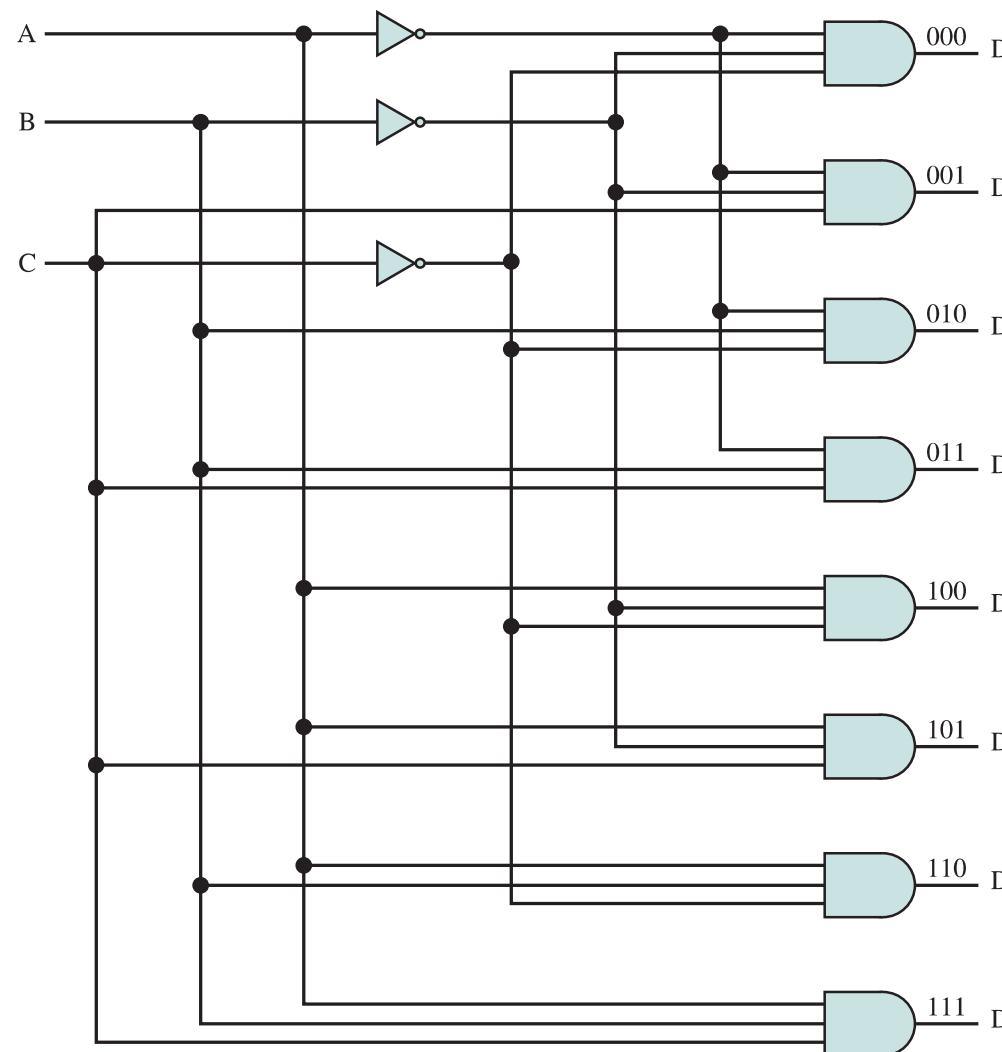
- N đầu vào, 2^N đầu ra
- Chỉ có một đầu ra tích cực (được chọn) tương ứng với một tổ hợp của N đầu vào.



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

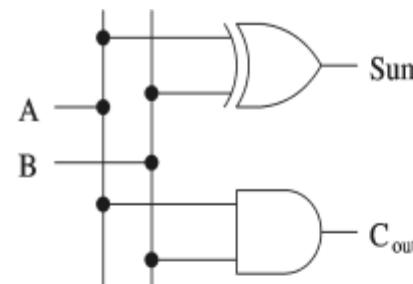


Bộ cộng (Adder)

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

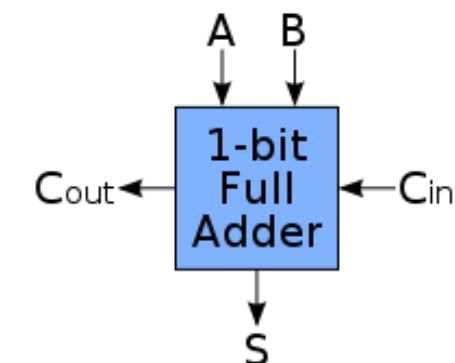
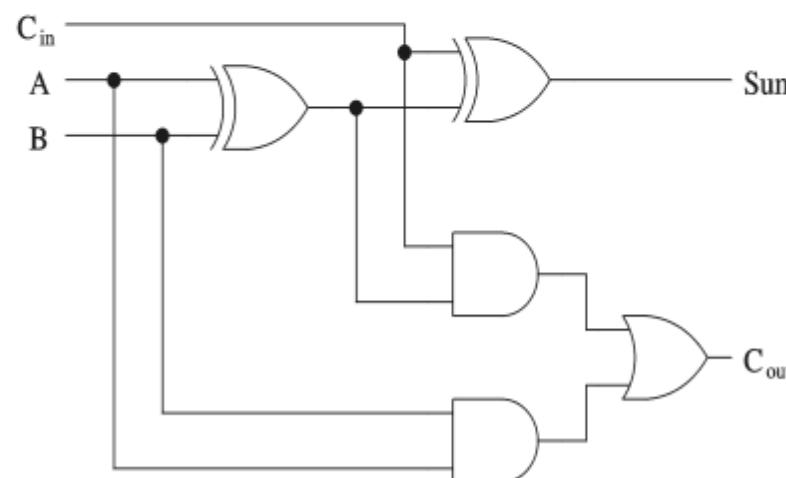
Bộ cộng (tiếp)

A	B	Sum	C _{out}
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



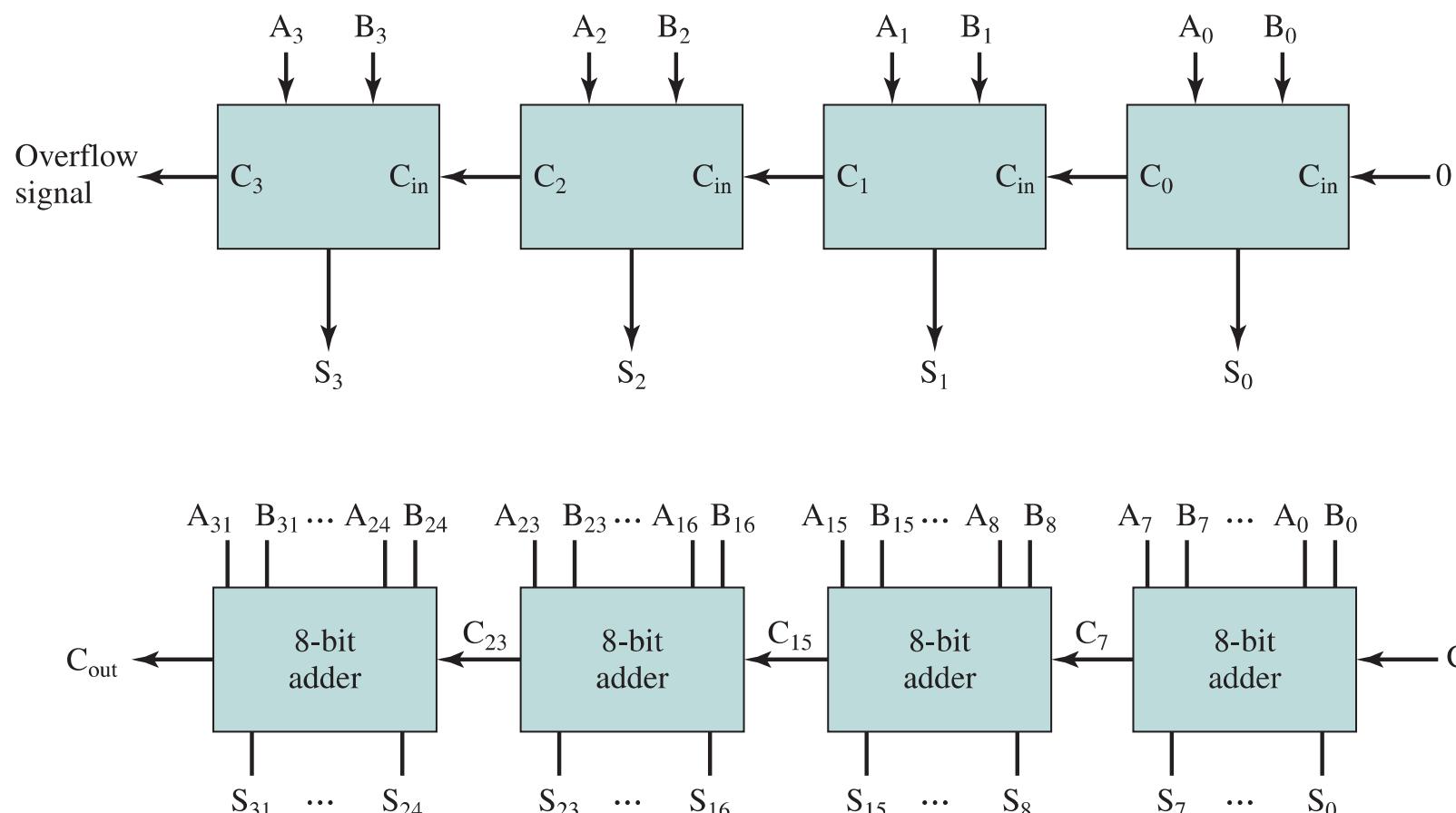
(a) Half-adder truth table and implementation

A	B	C _{in}	Sum	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



(b) Full-adder truth table and implementation

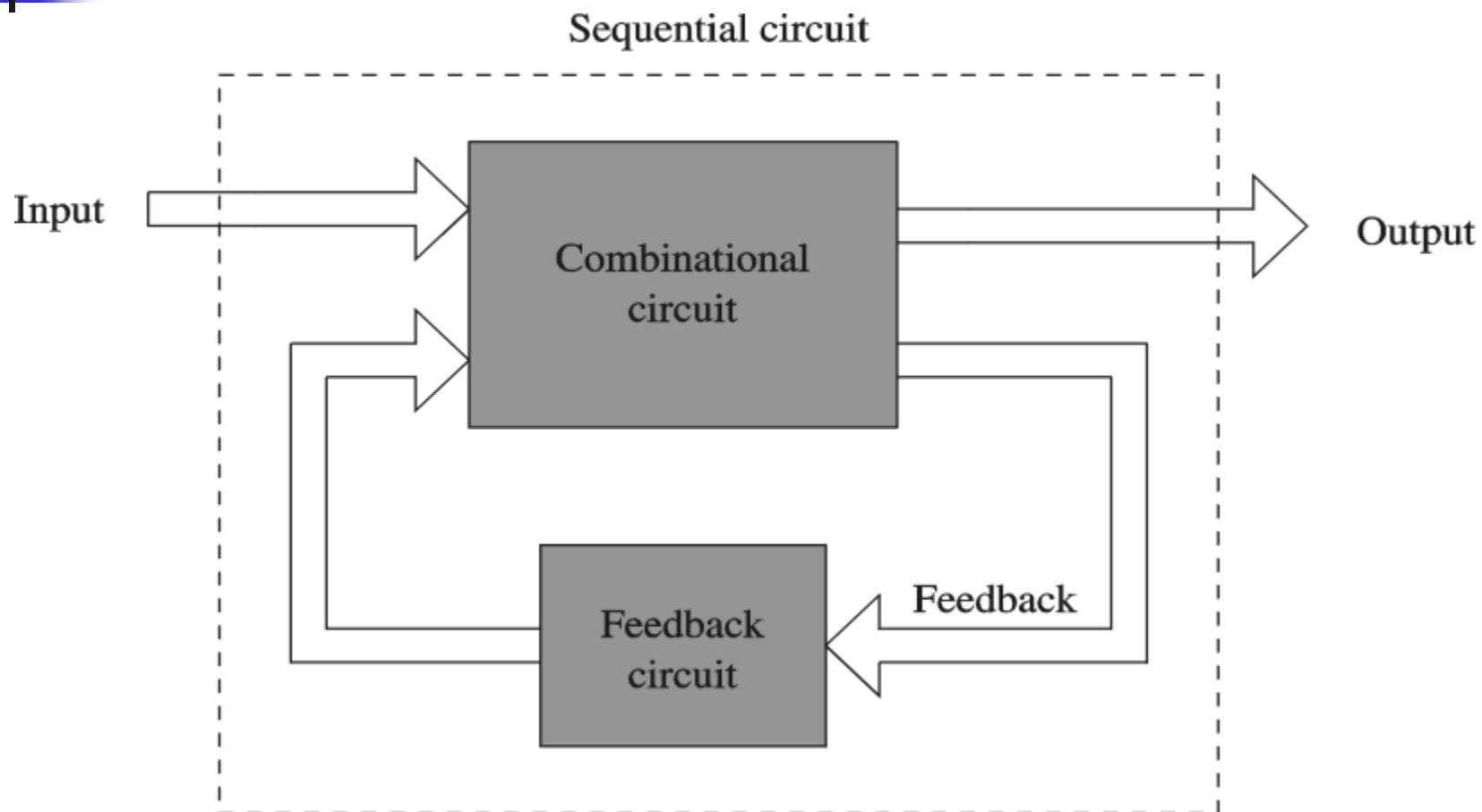
Bộ cộng 4-bit và bộ cộng 32-bit



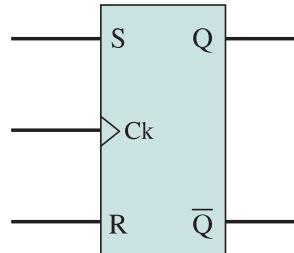
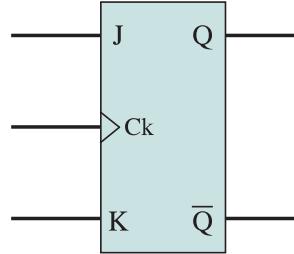
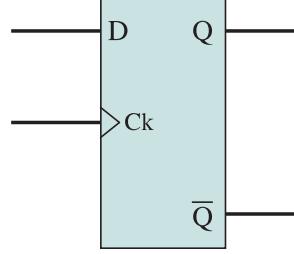
2.5. Mạch dây

- Mạch dây là mạch logic trong đó đầu ra phụ thuộc giá trị đầu vào ở thời điểm hiện tại và 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 cơ bản
- Mạch dây bao gồm:
 - Mạch tổ hợp
 - Mạch hồi tiếp

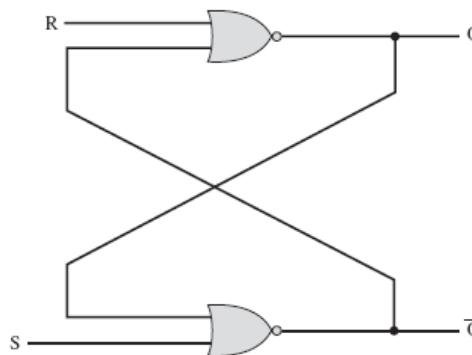
Các thành phần chính của mạch dãy



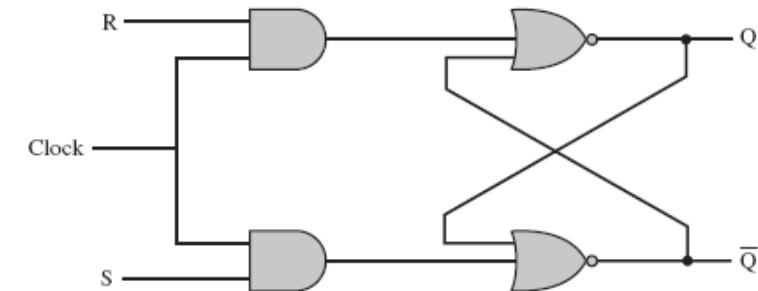
Các Flip-Flop cơ bản

Name	Graphical Symbol	Truth Table															
S-R		<table border="1"> <thead> <tr> <th>S</th> <th>R</th> <th>Q_{n+1}</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>Q_n</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>-</td> </tr> </tbody> </table>	S	R	Q_{n+1}	0	0	Q_n	0	1	0	1	0	1	1	1	-
S	R	Q_{n+1}															
0	0	Q_n															
0	1	0															
1	0	1															
1	1	-															
J-K		<table border="1"> <thead> <tr> <th>J</th> <th>K</th> <th>Q_{n+1}</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>Q_n</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>$\overline{Q_n}$</td> </tr> </tbody> </table>	J	K	Q_{n+1}	0	0	Q_n	0	1	0	1	0	1	1	1	$\overline{Q_n}$
J	K	Q_{n+1}															
0	0	Q_n															
0	1	0															
1	0	1															
1	1	$\overline{Q_n}$															
D		<table border="1"> <thead> <tr> <th>D</th> <th>Q_{n+1}</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> </tr> </tbody> </table>	D	Q_{n+1}	0	0	1	1									
D	Q_{n+1}																
0	0																
1	1																

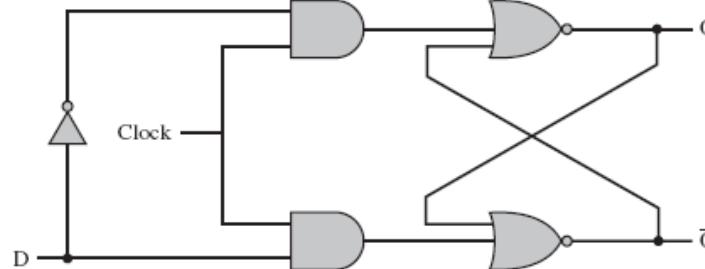
R-S Latch và các Flip-Flop



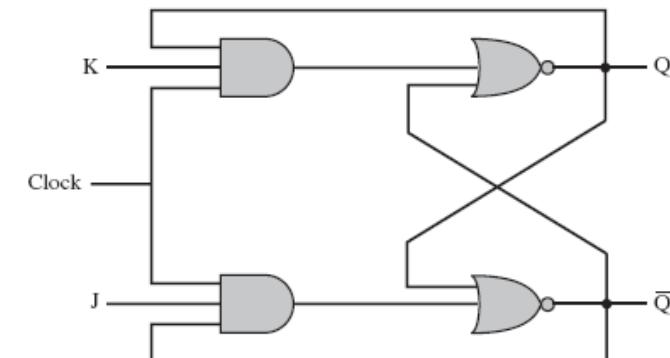
R-S Latch



R-S Flip Flop

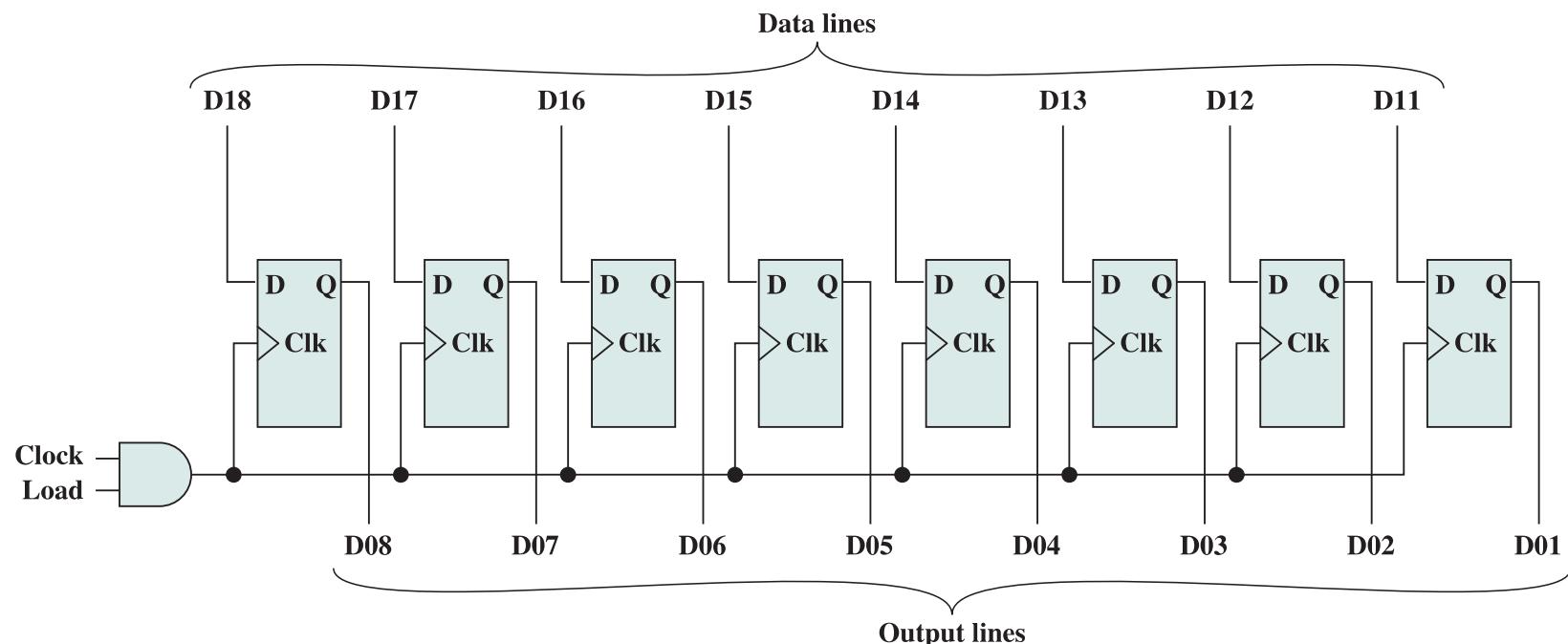


D Flip Flop

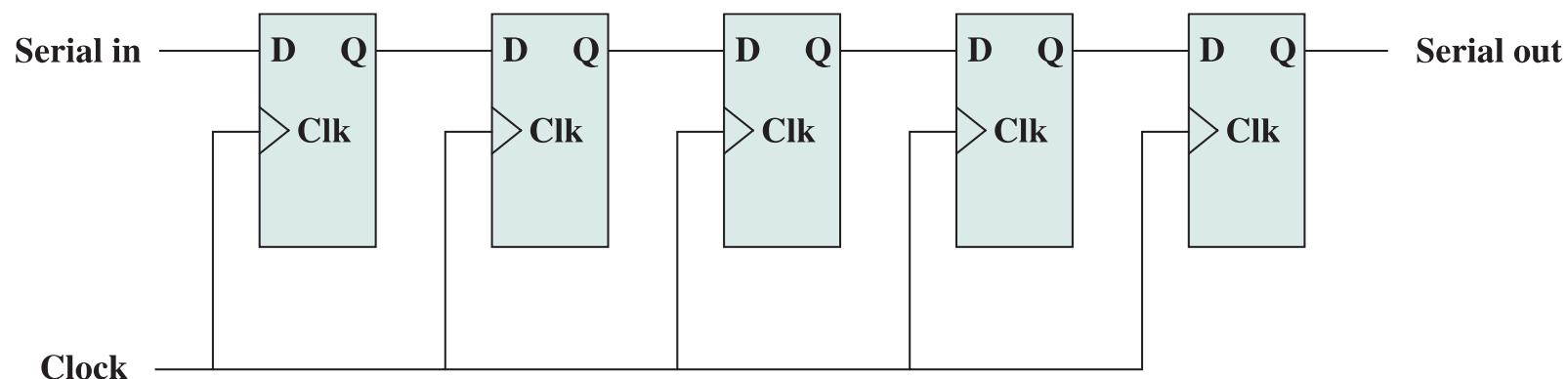


J-K Flip Flop

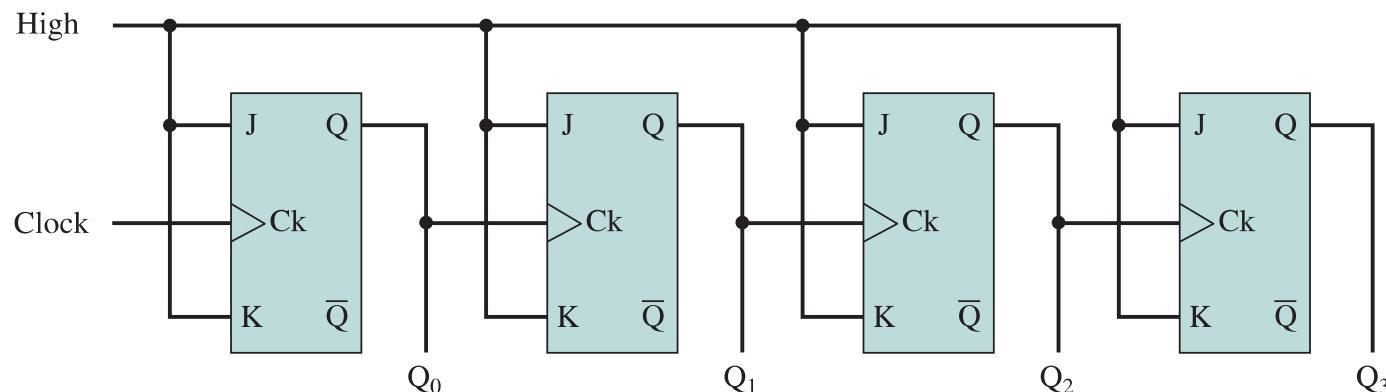
Thanh ghi 8-bit song song



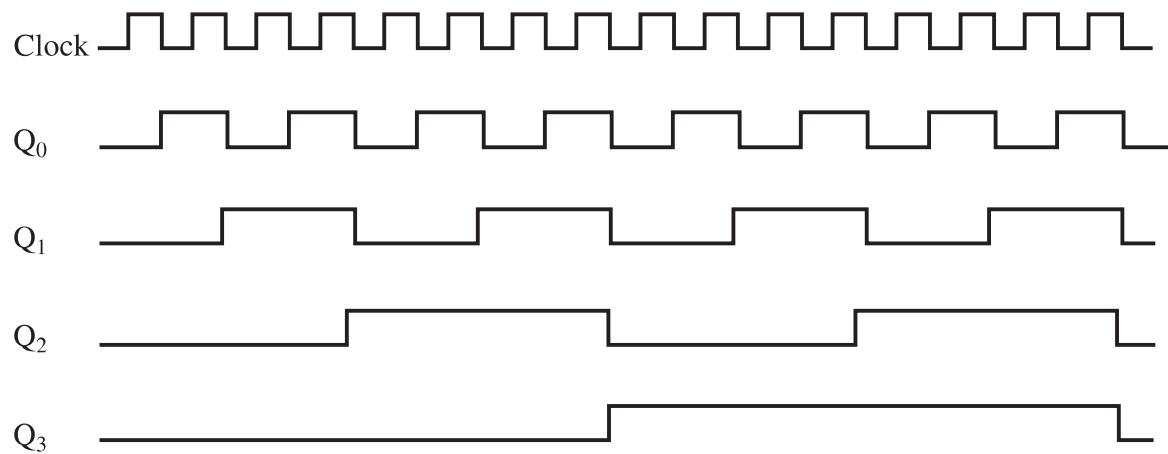
Thanh ghi dịch 5-bit



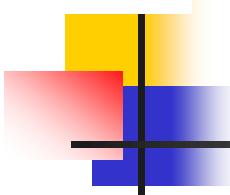
Bộ đếm 4-bit



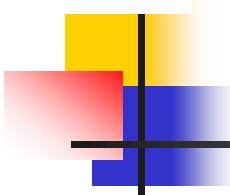
(a) Sequential circuit



(b) Timing diagram



Hết chương 2

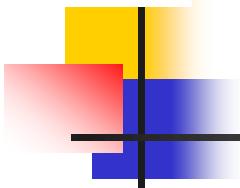


Kiến trúc máy tính

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

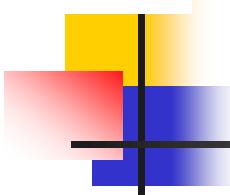
Chương 5. Kiến trúc tập lệnh

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 (CPU)
- Bộ nhớ (Memory)
- Hệ thống vào-ra (Input/Output System)
- Bus liên kết hệ thống (System Interconnection Bus)

1. Bộ xử lý trung tâm (CPU)

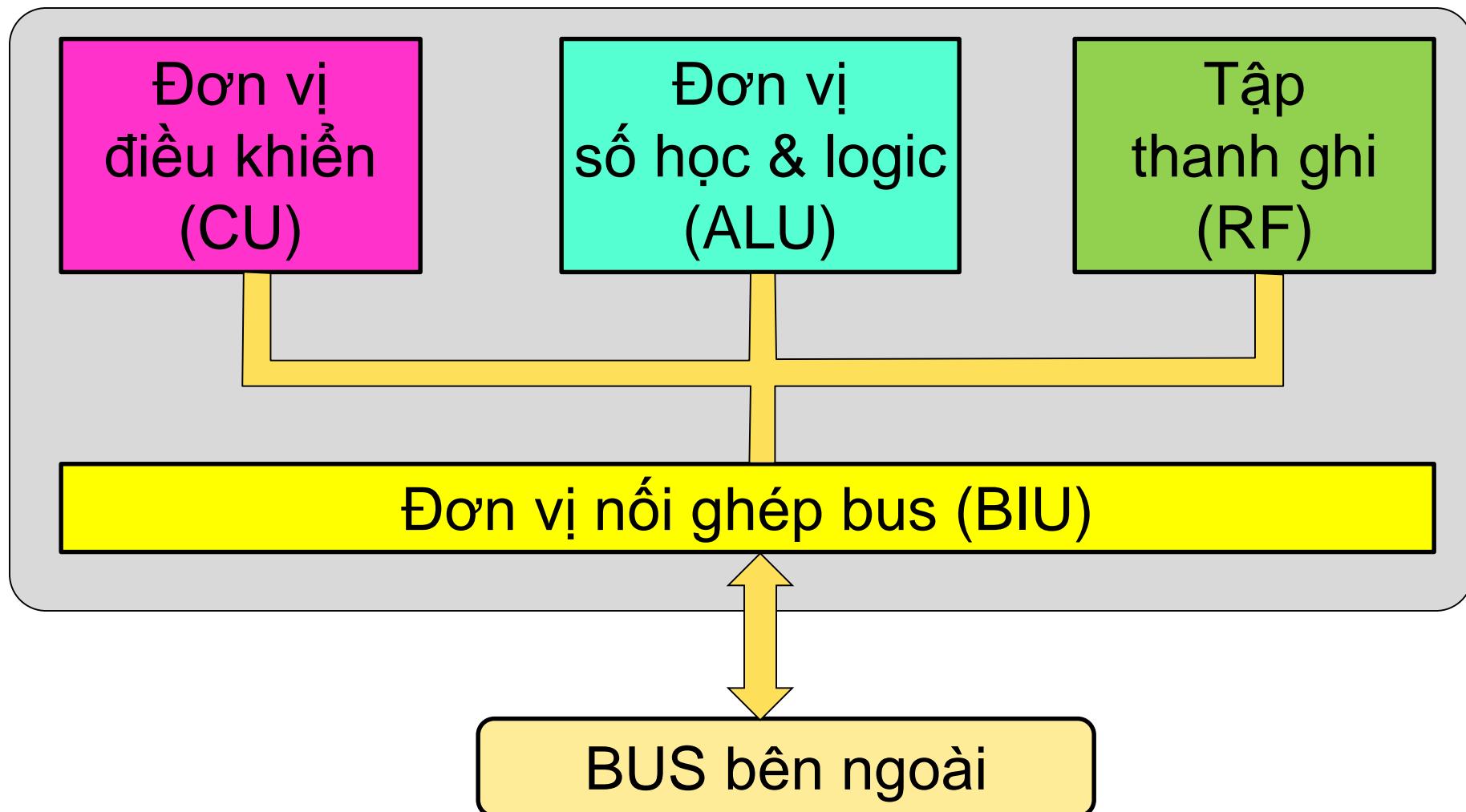
- Chức năng:

- 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.

Cấu trúc cơ bản của CPU



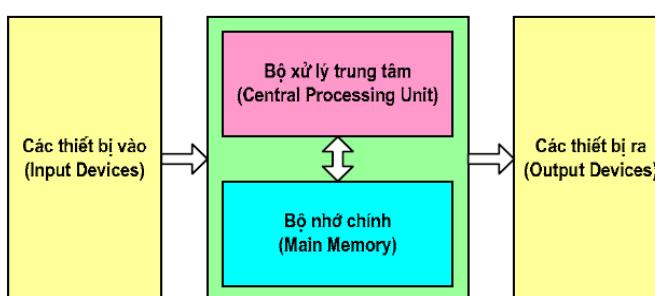
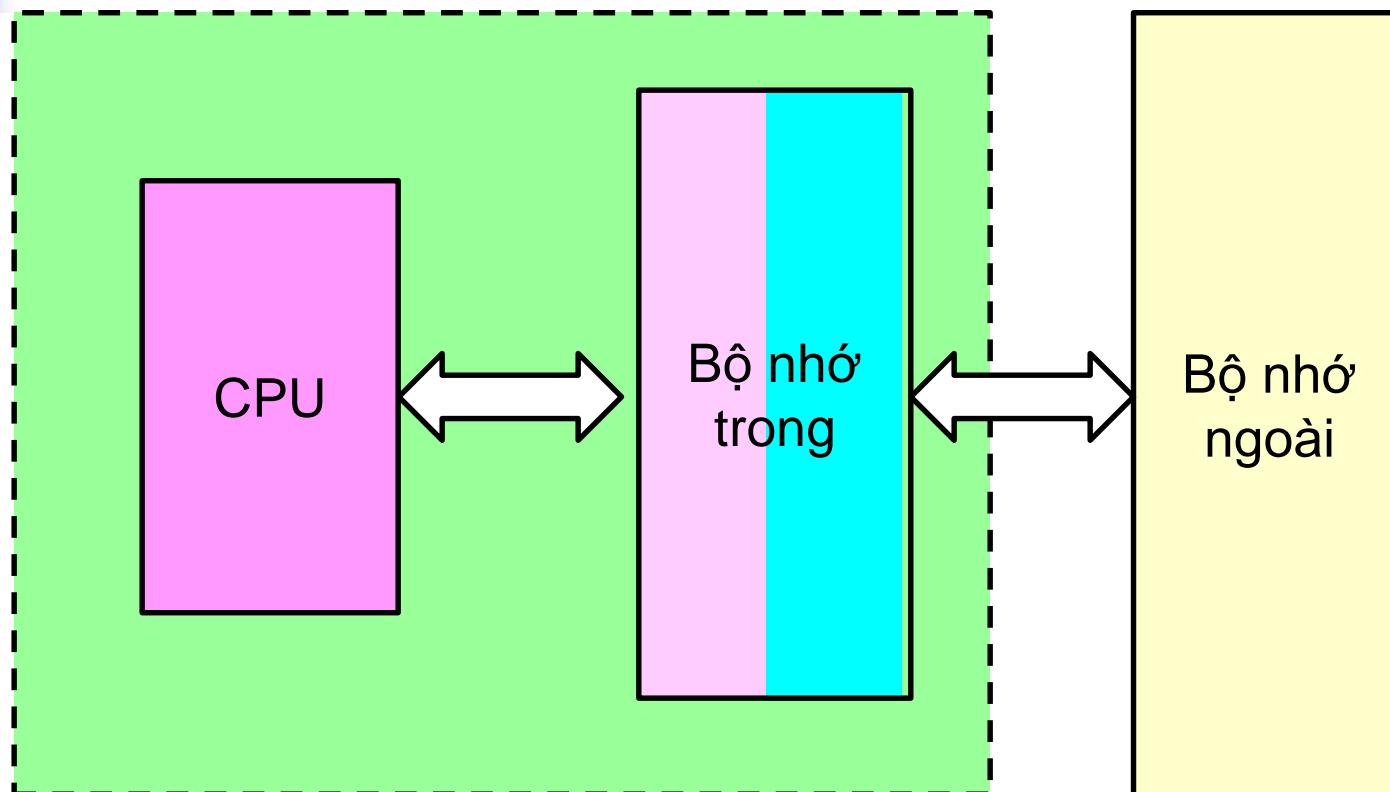
Các thành phần cơ bản của CPU

- **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*): lưu giữ các thông tin tạm thời phục vụ cho hoạt động của CPU.
- **Đơn vị nối ghép bus** (*Bus Interface Unit - BIU*) kết nối và trao đổi thông tin giữa bus bên trong (*internal bus*) và bus bên ngoài (*external bus*).

2. Bộ nhớ máy tính

- Chức năng: lưu trữ chương trình và dữ liệu.
- Các thao tác cơ bản với bộ nhớ:
 - Thao tác ghi (Write)
 - Thao tác đọc (Read)
- Các thành phần chính:
 - Bộ nhớ trong (Internal Memory)
 - Bộ nhớ ngoài (External Memory)

Các thành phần của bộ nhớ máy tính



Bộ nhớ trong

BN trong =
BN chính+cache

- Chức năng và đặc điểm:
 - Chứa các thông tin mà CPU có thể trao đổi trực tiếp
 - Tốc độ rất nhanh
 - Dung lượng không lớn
 - Sử dụng bộ nhớ bán dẫn: ROM và RAM
- Các loại bộ nhớ trong:
 - Bộ nhớ chính
 - Bộ nhớ cache (bộ nhớ đệm)

Bộ nhớ chính (Main Memory)

BN trong =
BN chính+cache

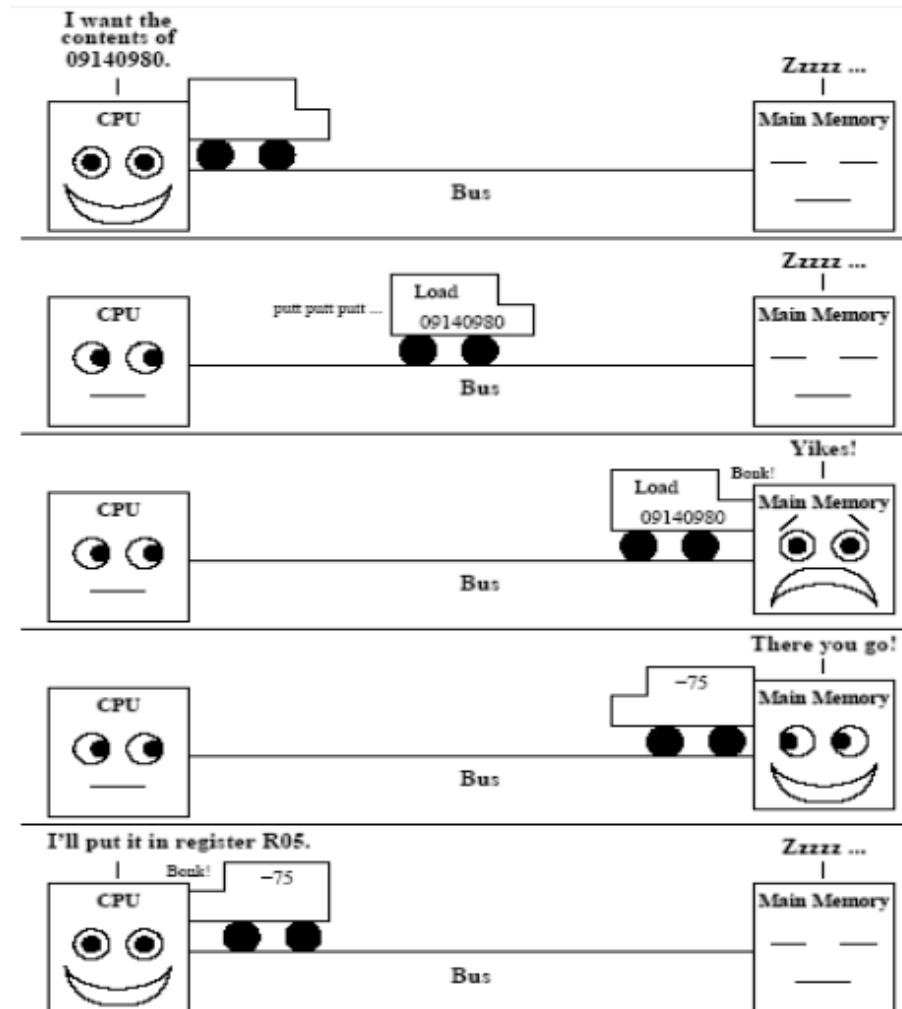
- Chứa các **chương trình** và **dữ liệu** đang được CPU sử dụng.
- Tổ chức thành **các ngăn nhớ** được **đánh địa chỉ**.
- **Ngăn nhớ** thường được **tổ chức theo byte**.
- **Nội dung của ngăn nhớ** có thể **thay đổi**, song **địa chỉ** vật lý của ngăn nhớ luôn **cố định**.



Nội dung	Địa chỉ
1011 0010	0000
1110 0010	0001
0001 1111	0010
1010 1011	0011
0000 1000	0100
1111 1111	0101
0011 1100	0110
1000 1111	0111
1111 0001	1000
0011 1101	1001
1000 1111	1010
0011 0011	1011
1100 1101	1100
0101 1010	1101
1000 1101	1110
1111 0000	1111

Bộ nhớ chính (Main Memory)

BN trong =
BN chính+cache



Hardware Lesson
CS1313 Spring 2014

Bộ nhớ cache

BN trong =
BN chính+cache

- 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 ($\approx 1/4\text{MiB} - 64\text{MiB}$)
- Tốc độ nhanh hơn ($\approx 5\text{-}100\%$ tốc độ của thanh ghi)
- Cache thường được chia thành một số mức
- Cache có thể được tích hợp trên cùng chip bộ xử lý.
- Cache có thể có hoặc không

Bộ nhớ ngoài (External Memory)

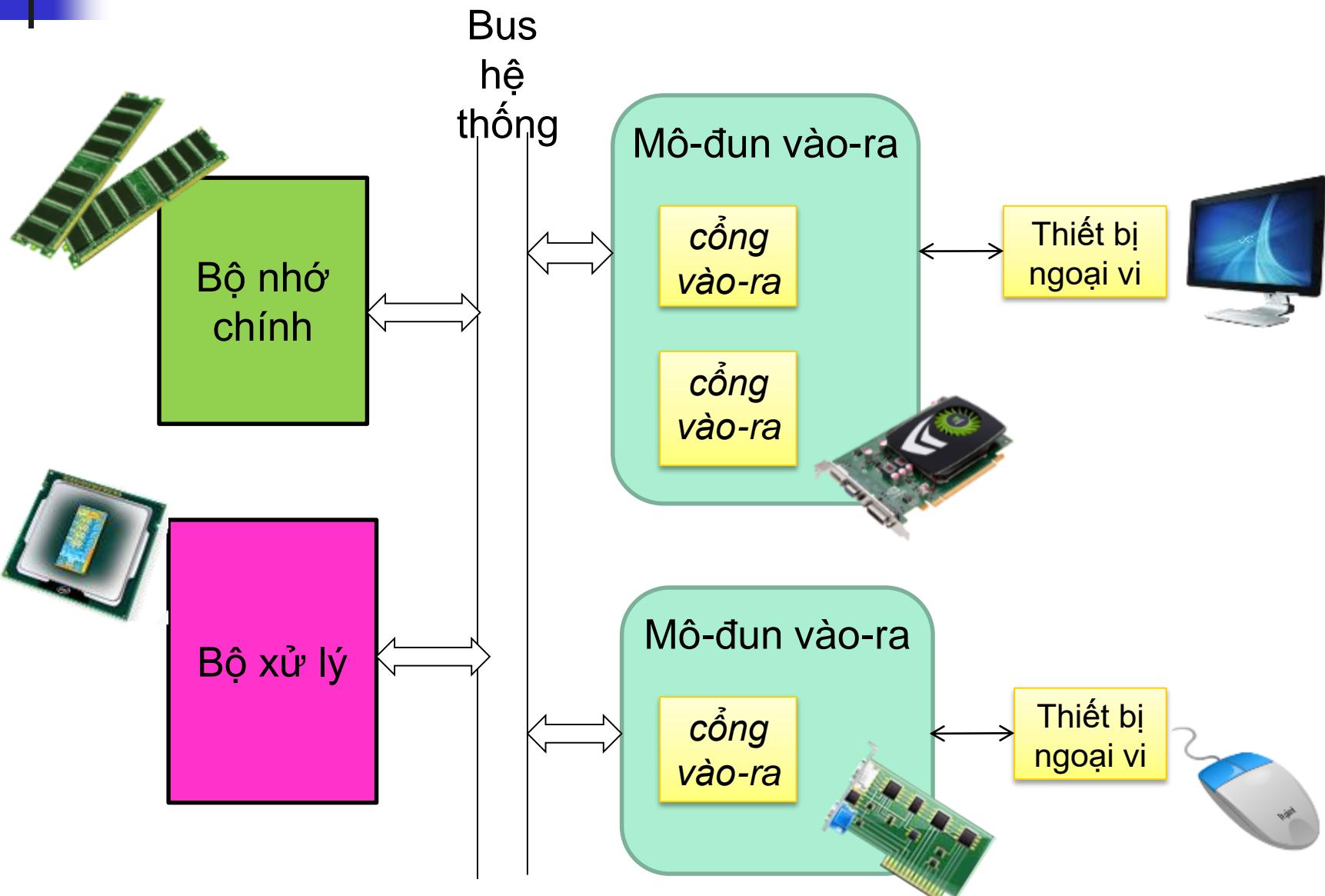
Chức năng và đặc điểm

- 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 bộ nhớ ngoài
 - Bộ nhớ từ: ổ đĩa cứng
 - Bộ nhớ quang: đĩa CD, DVD
 - Bộ nhớ bán dẫn: Ổ nhớ flash, thẻ nhớ, ổ SSD

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 ngoại vi (Peripheral Devices)
 - Các mô-đun vào-ra (IO Modules)

Cấu trúc cơ bản của hệ thống vào-ra



Các thiết bị ngoại vi

- Chức năng: chuyển đổi dữ liệu giữa bên trong và bên ngoài máy tính
- Các loại thiết bị ngoại vi cơ bản
 - Thiết bị vào: bàn phím, chuột, máy quét ...
 - Thiết bị ra: màn hình, máy in ...
 - Thiết bị nhớ: các ổ đĩa ...
 - Thiết bị truyền thông: MODEM ...

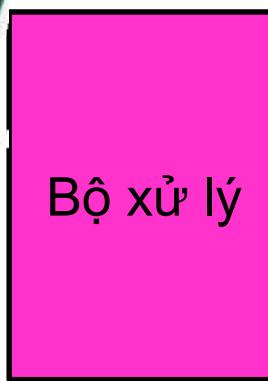
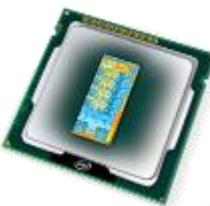
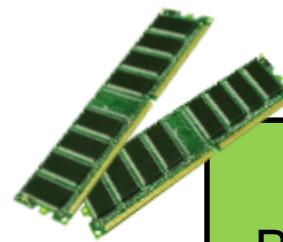
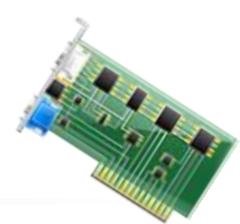




Mô-đun vào-ra

- Chức năng: nối ghép các thiết bị ngoại vi 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 ngoại vi được kết nối và trao đổi dữ liệu với máy tính thông qua các cổng vào-ra.

Mô-đun vào-ra



Bus
hệ
thống

Mô-đun vào-ra

1	1	1	1	1	1
0	0	1	0	0	0
0	0	1	0	0	0
0	0	1	0	0	0
0	0	1	0	0	0
0	0	1	0	0	0
0	0	1	0	0	0



TBNV



Mô-đun vào-ra

cổng
vào-ra

T



TBNV

3.2. Hoạt động cơ bản của máy tính

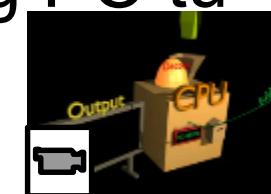
- Thực hiện chương trình
- Hoạt động ngắt
- Hoạt động vào-ra

1. Thực hiện chương trình

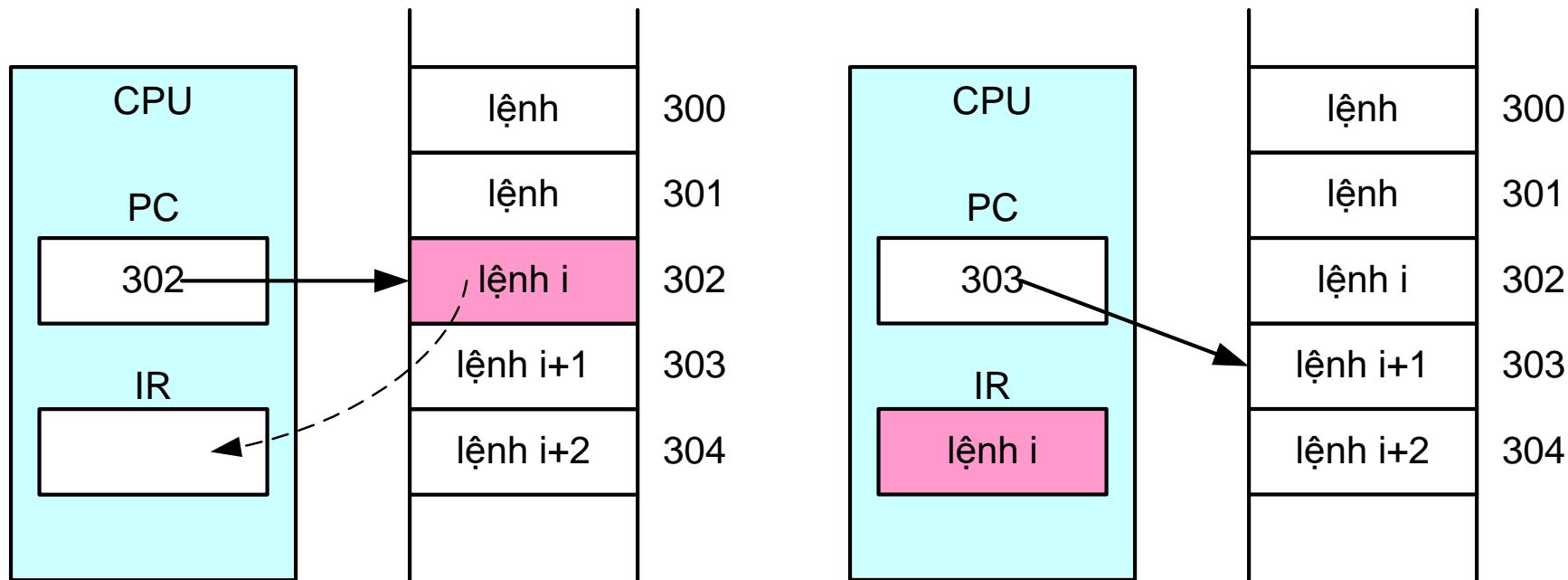
- Là hoạt động cơ bản của máy tính
- Máy tính lặp đi lặp lại hai bước:
 - Nhận lệnh
 - Thực hiện lệnh
- Thực hiện chương trình bị dừng nếu thực hiện lệnh bị lỗi hoặc gặp lệnh dừng.

Nhận lệnh

- Bắt đầu mỗi chu trình lệnh, CPU nhận lệnh từ bộ nhớ chính.
- Bộ đếm chương trình PC (Program Counter) của CPU giữ địa chỉ của lệnh sẽ được nhận. (hoặc tên khác là *IP, Instruction Pointer*)
- CPU nhận lệnh từ ngăn nhớ được trỏ bởi PC.
- Lệnh được nạp vào thanh ghi lệnh IR (Instruction Register).
- Sau khi lệnh được nhận vào, nội dung PC tự động tăng để trỏ sang lệnh kế tiếp.



Minh họa quá trình nhận lệnh



Trước khi nhận lệnh i

Sau khi nhận lệnh i

Thực hiện lệnh

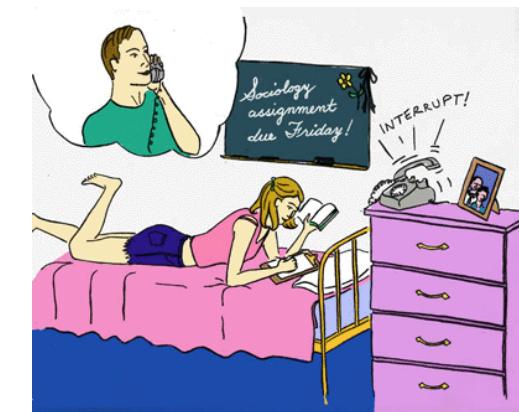
- Bộ xử lý giải mã lệnh đã được nhận và phát tín hiệu điều khiển thực hiện thao tác mà lệnh yêu cầu.
- Các kiểu thao tác của lệnh:
 - Trao đổi dữ liệu giữa CPU và bộ nhớ chính
 - Trao đổi dữ liệu giữa CPU và mô-đun vào-ra
 - Xử lý dữ liệu: thực hiện các phép toán số học hoặc phép toán logic với các dữ liệu.
 - Điều khiển rẽ nhánh
 - Kết hợp các thao tác trên.

2. Hoạt động 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 khác, gọi là *chương trình con phục vụ ngắt*.
- Các loại ngắt:
 - Ngắt do lỗi khi thực hiện chương trình, ví dụ: tràn số, chia cho 0.
 - Ngắt do lỗi phần cứng, ví dụ lỗi bộ nhớ RAM.
 - Ngắt do mô-đun vào-ra phát tín hiệu ngắt đến CPU yêu cầu trao đổi dữ liệu.
 - Ngắt do bộ định thời trong chế độ đa chương trình



Joe Hobbyist, stumped on his timing loops.

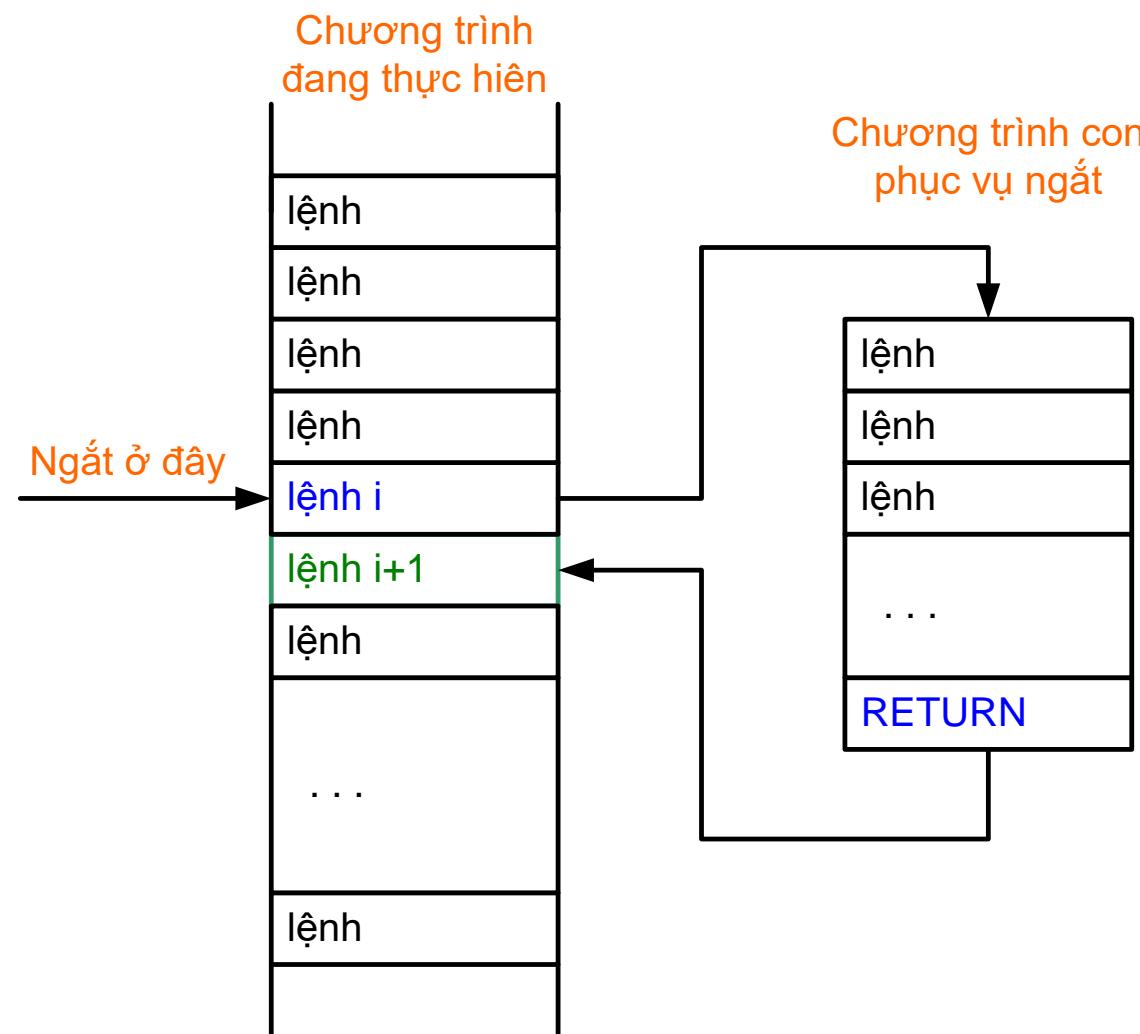


Suddenly more data becomes available for the assignment.

Hoạt động ngắt (tiếp)

- 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 chương trình đang thực hiện
 - Cắt ngũ cảnh (các thông tin liên quan đến chương trình bị ngắt)
 - Thiết lập PC trả đến chương trình con phục vụ ngắt
 - Chuyển sang thực hiện chương trình con phục vụ ngắt
 - Cuối chương trình con phục vụ ngắt, khôi phục ngũ cảnh và tiếp tục chương trình đang bị tạm dừng

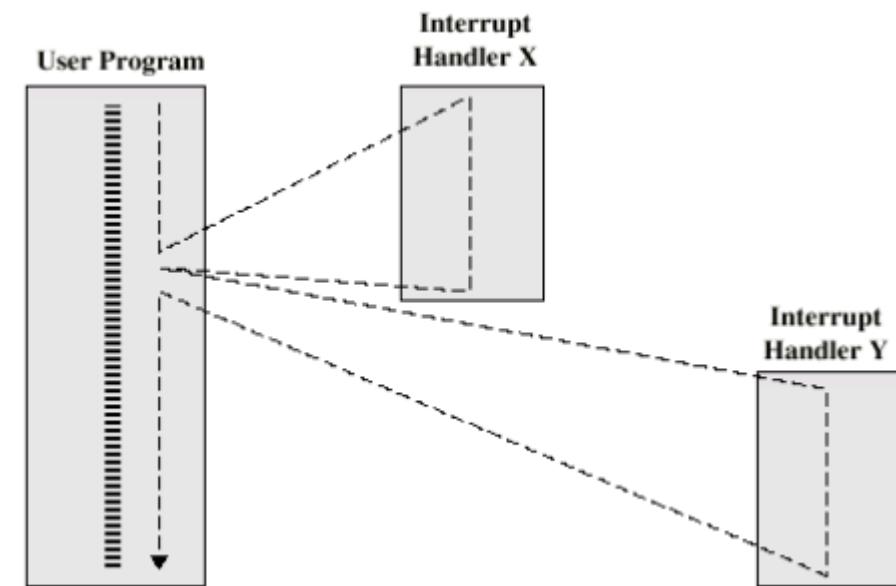
Hoạt động ngắt (tiếp)



Xử lý với nhiều tín hiệu yêu cầu ngắt

Xử lý ngắt tuần tự

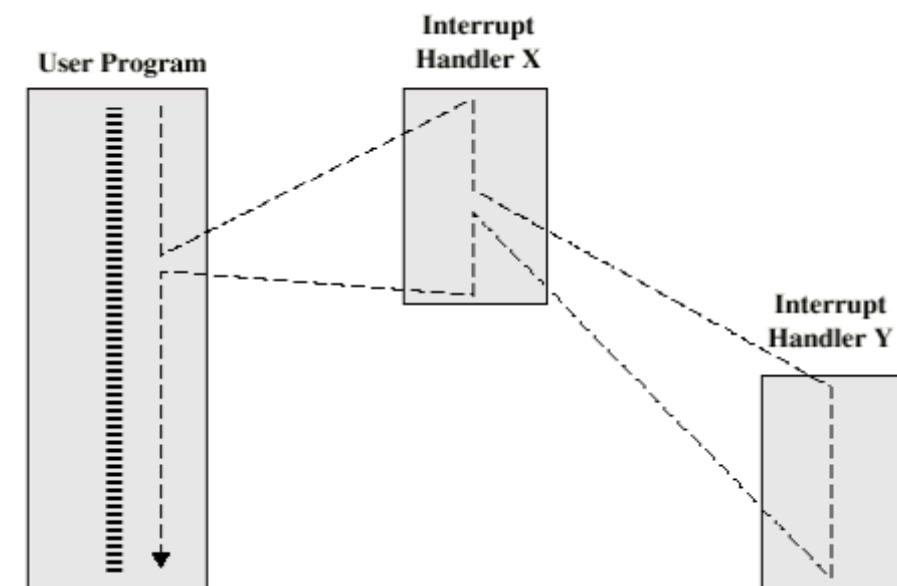
- Khi một ngắt đang được thực hiện, các ngắt khác sẽ bị cấm.
- Bộ xử lý sẽ bỏ qua các ngắt tiếp theo trong khi đang xử lý một ngắt
- Các yêu cầu ngắt vẫn đang đợi và được kiểm tra sau khi ngắt đầu tiên được xử lý xong
- Các ngắt được thực hiện tuần tự



Xử lý với nhiều tín hiệu yêu cầu ngắt...

■ 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 ưu tiên cao hơn
- Xảy ra ngắt lồng nhau



3. Hoạt động vào-ra

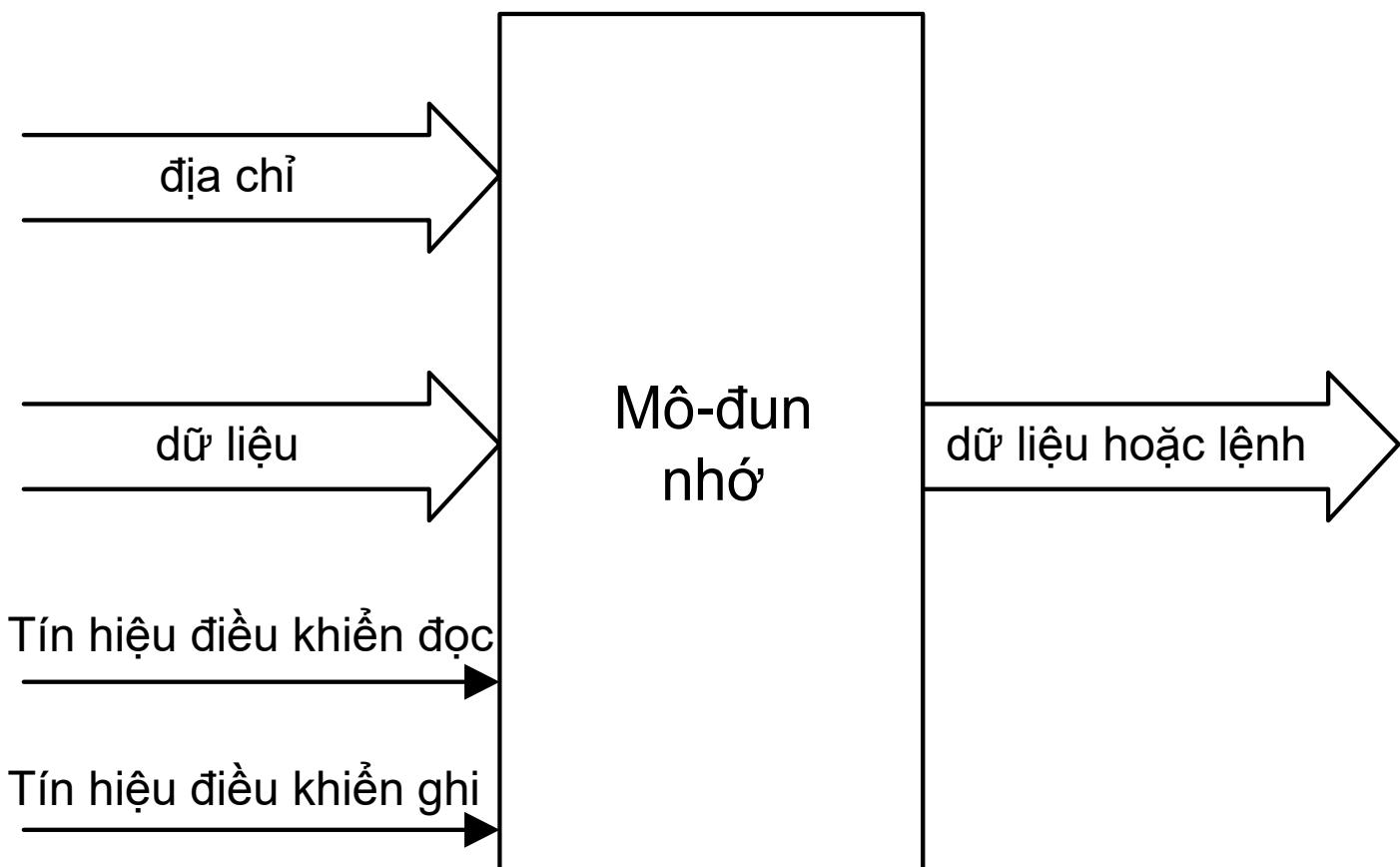
- **Hoạt động vào-ra:** là hoạt động trao đổi dữ liệu giữa mô-đun vào-ra với bên trong máy tính.
- **Các kiểu hoạt động vào-ra:**
 - CPU trao đổi dữ liệu với mô-đun vào-ra
 - Mô-đun vào-ra trao đổi dữ liệu trực tiếp với bộ nhớ chính (DMA- Direct Memory Access).

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

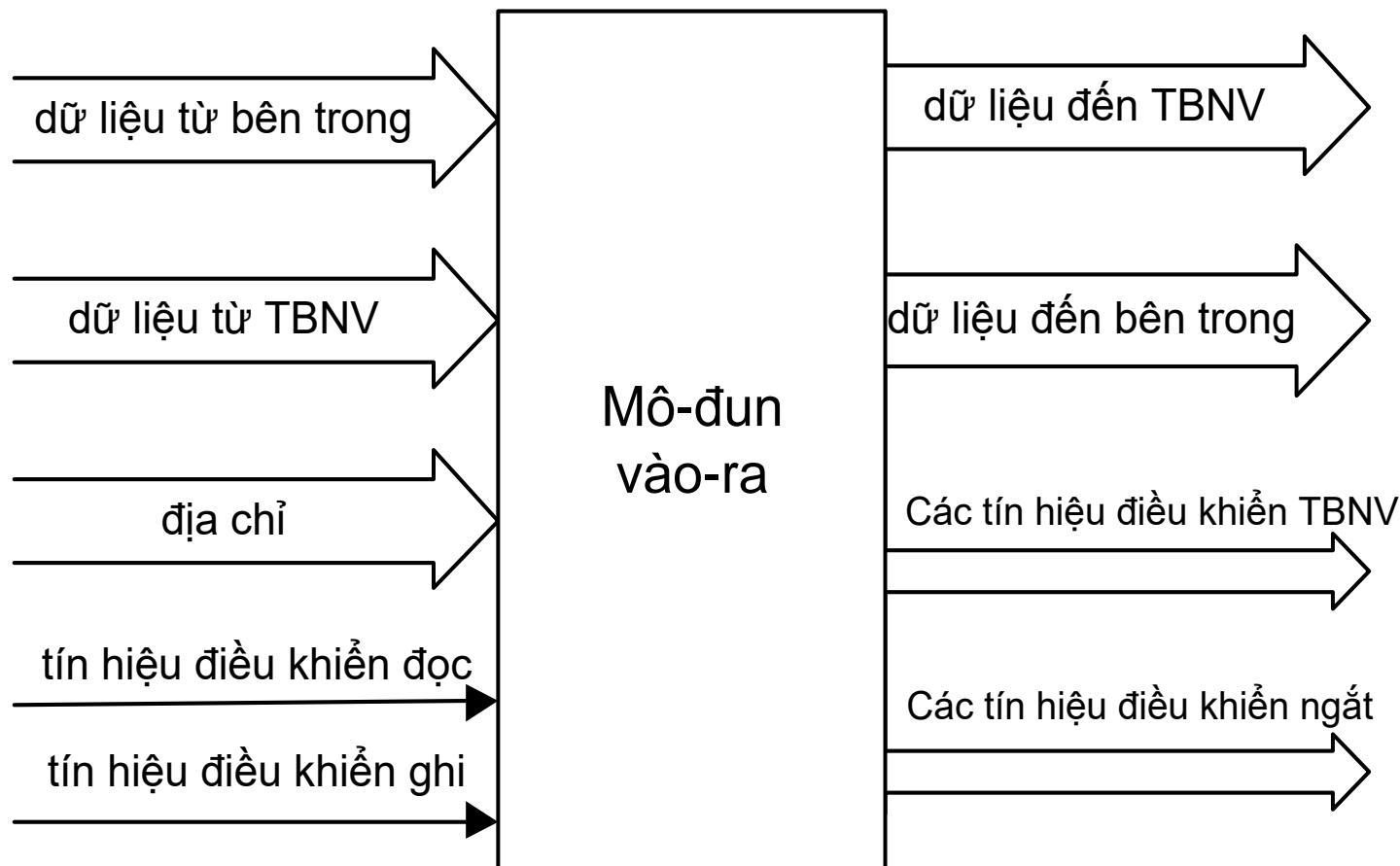
Kết nối mô-đun nhớ



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
(lưu ý: bộ nhớ không phân biệt lệnh và dữ liệu)
- Nhận các tín hiệu điều khiển:
 - Điều khiển đọc (Read)
 - Điều khiển ghi (Write)

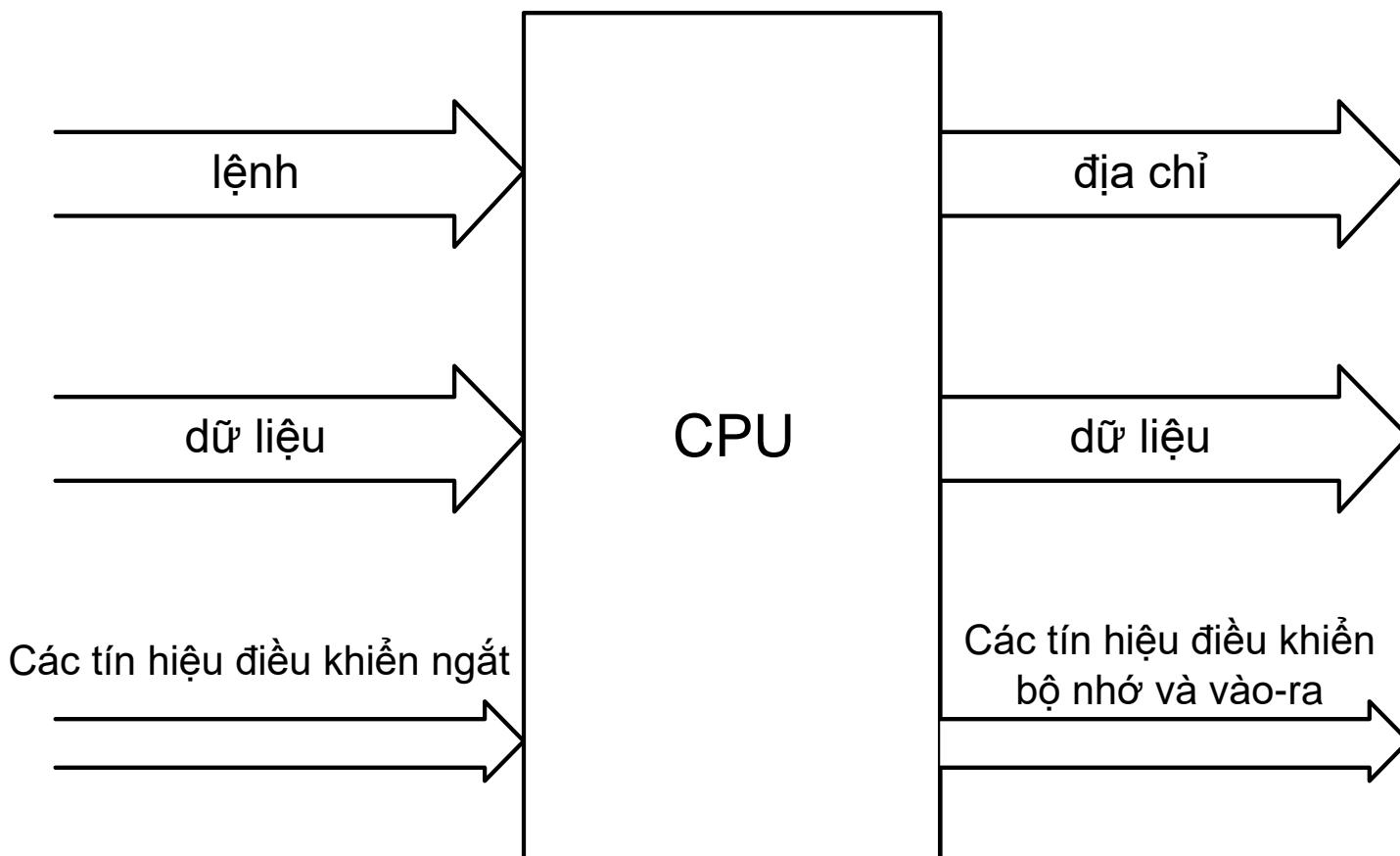
Kết nối mô-đun vào-ra



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 CPU hoặc bộ nhớ chính
 - Đưa dữ liệu ra thiết bị ngoại vi
- Vào dữ liệu (Input)
 - Nhận dữ liệu từ thiết bị ngoại vi
 - Đưa dữ liệu vào CPU hoặc bộ nhớ chính
- Nhận các tín hiệu điều khiển từ CPU
- Phát các tín hiệu điều khiển đến thiết bị ngoại vi
- Phát các tín hiệu ngắn đến CPU

Kết nối CPU



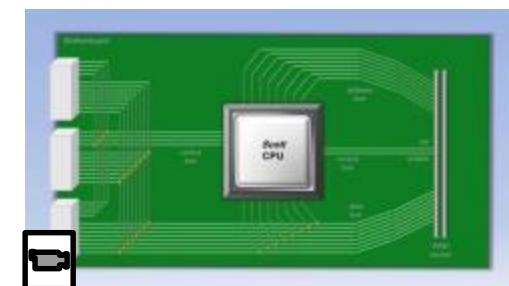
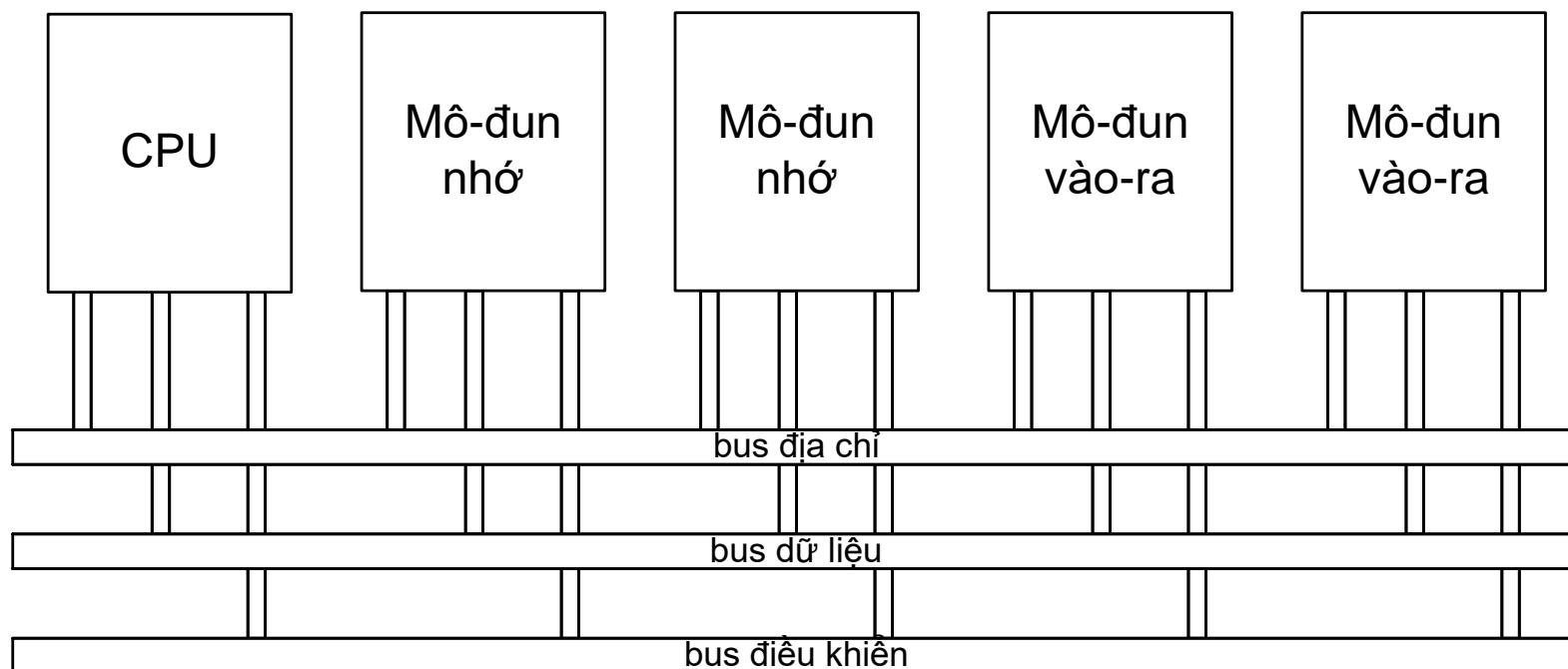
Kết nối CPU (tiếp)

- Phát địa chỉ đến các mô-đun nhớ hay các mô-đun vào-ra
- Đọc lệnh và dữ liệu
- Đưa dữ liệu ra (sau khi xử lý)
- 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 dùng để vận chuyển thông tin giữa các mô-đun của máy tính với nhau.
- **Các bus chức năng:**
 - Bus địa chỉ
 - Bus dữ liệu
 - Bus điều khiển
- **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 **ngăn nhớ** hay **cổng vào-ra**
- **Độ rộng bus địa chỉ:** cho biết số lượng ngăn nhớ tối đa được đánh địa chỉ.
 - N bit: $A_{N-1}, A_{N-2}, \dots, A_2, A_1, A_0$
→ có thể đánh địa chỉ tối đa cho 2^N ngăn nhớ (không gian địa chỉ bộ nhớ)
- **Ví dụ:**
 - Bộ xử lý Pentium có bus địa chỉ 32 bit
→ có khả năng đánh địa chỉ cho 2^{32} bytes nhớ (4GiBytes) (ngăn nhớ tổ chức theo byte)

Bus dữ liệu

■ **Chức năng:**

- vận chuyển lệnh từ bộ nhớ đến CPU
- vận chuyển dữ liệu giữa CPU, mô đun nhớ, mô đun vào-ra với nhau

■ **Độ rộng bus dữ liệu:** Xác định số bit dữ liệu có thể được trao đổi đồng thời.

- M bit: $D_{M-1}, D_{M-2}, \dots, D_2, D_1, D_0$
- M thường là 8, 16, 32, 64, 128 bit.

■ **Ví dụ:** Các bộ xử lý Pentium có bus dữ liệu 64 bit

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 ngắn
 - Các tín hiệu điều khiển bus

Một số tín hiệu điều khiển hiển thị hình

- Các tín hiệu (phát ra từ CPU) điều khiển đọc-ghi:
 - *Memory Read (MEMR)*: điều khiển đọc dữ liệu từ một ngăn nhớ có địa chỉ xác định lên bus dữ liệu.
 - *Memory Write (MEMW)*: điều khiển ghi dữ liệu có sẵn trên bus dữ liệu đến một ngăn nhớ có địa chỉ xác định.
 - *I/O Read (IOR)*: điều khiển đọc dữ liệu từ một cổng vào-ra có địa chỉ xác định lên bus dữ liệu.
 - *I/O Write (IOW)*: 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 điện 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) hay là *Hold*: Tín hiệu từ mô-đun điều khiển vào-ra gửi đến yêu cầu CPU chuyển nhượng quyền sử dụng bus.
 - *Bus Grant* (BGT) hay là *Hold Acknowledge* (HLDA): Tín hiệu phát ra từ CPU chấp nhận chuyển nhượng quyền sử dụng bus.
 - *Lock/ Unlock*: Tín hiệu cấm/cho-phép xin chuyển nhượng bus

Đặc điểm của cấu trúc đơn bus

- Bus hệ thống chỉ phục vụ được một yêu cầu trao đổi dữ liệu tại một thời điểm
- Bus hệ thống phải có tốc độ bằng tốc độ bus của mô-đun nhanh nhất trong hệ thống
- Bus hệ thống phụ thuộc vào cấu trúc bus (các tín hiệu) của bộ xử lý → các mô-đun nhớ và các mô-đun vào-ra cũng phụ thuộc vào bộ xử lý.
- Khắc phục: phân cấp bus → cấu trúc đa bus

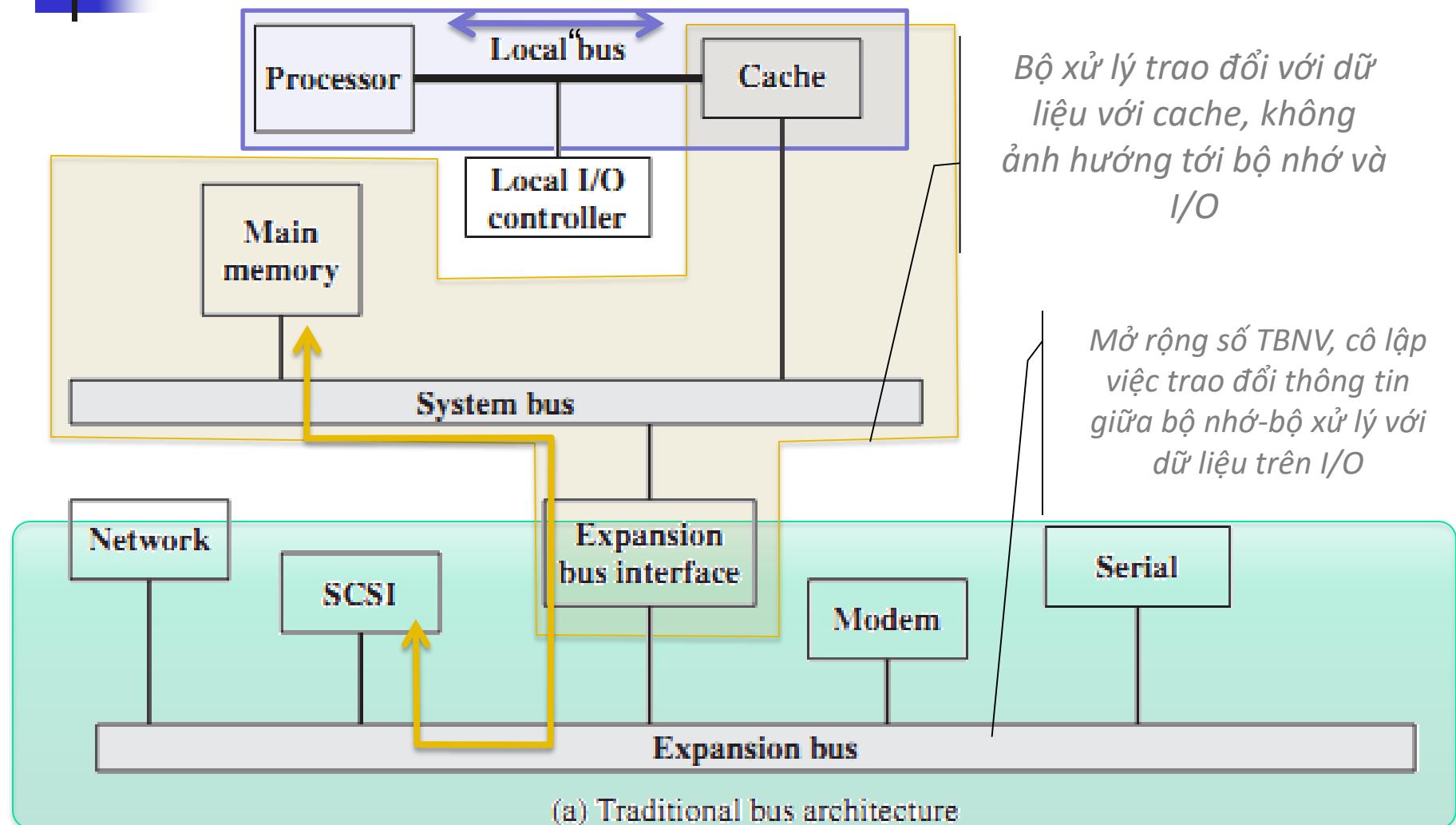
3. Phân cấp bus trong máy tính

- Tổ chức thành nhiều bus trong hệ thống máy tính
 - Cho các thành phần khác nhau:
 - Bus của bộ xử lý
 - Bus của bộ nhớ chính
 - Các bus vào-ra
 - Các bus khác nhau về tốc độ
- Bus bộ nhớ chính và các bus vào-ra không phụ thuộc vào bộ xử lý cụ thể.

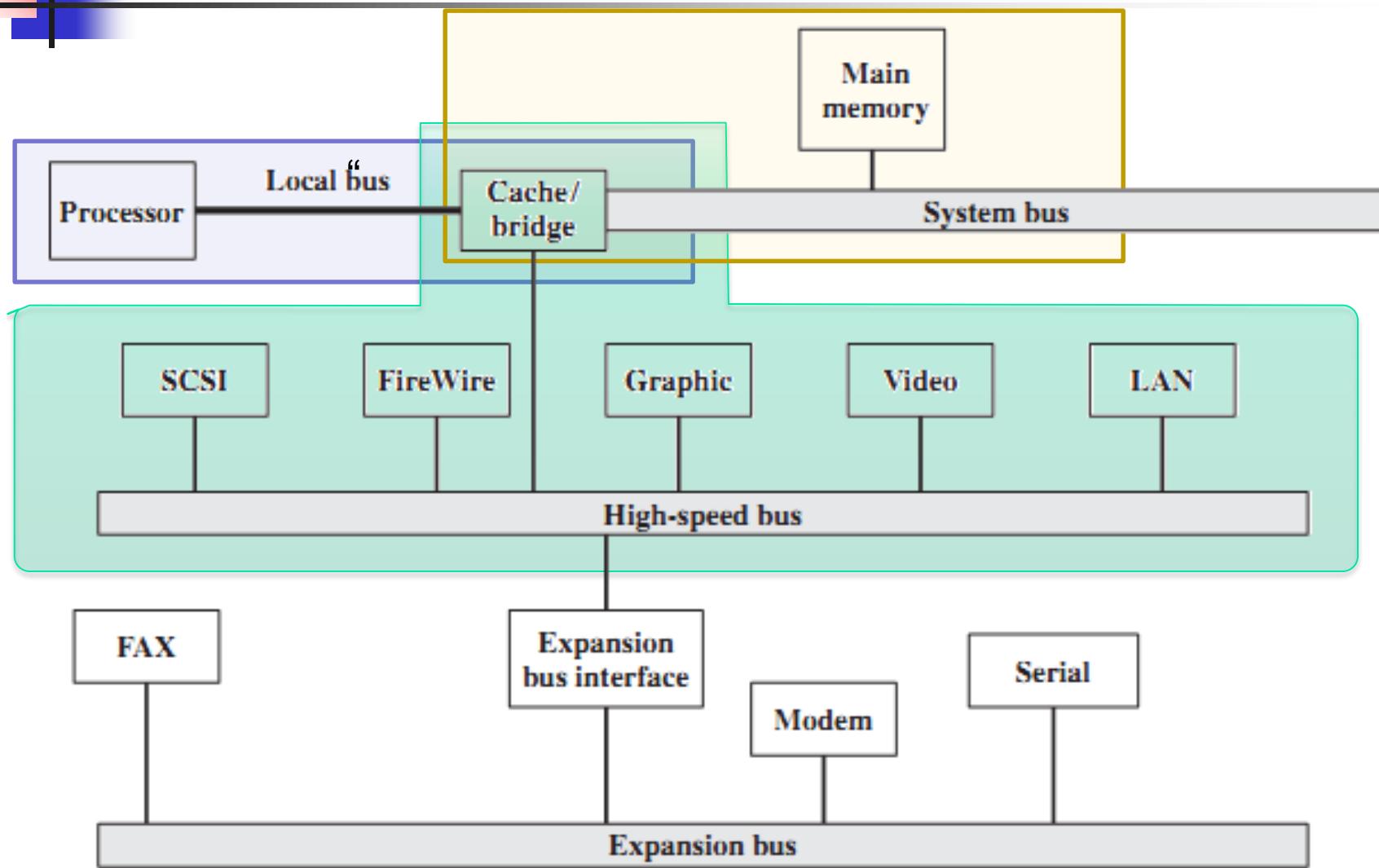
Một số bus điển hình trong máy tính

- Bus của bộ xử lý: có tốc độ nhanh nhất
- Bus của bộ nhớ chính (nối ghép với các mô-đun RAM)
- PCI Express bus (Peripheral Component Interconnect): nối ghép với các thiết bị ngoại vi có tốc độ trao đổi dữ liệu nhanh.
- 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

Kiến trúc phân cấp bus kiểu cổ điển

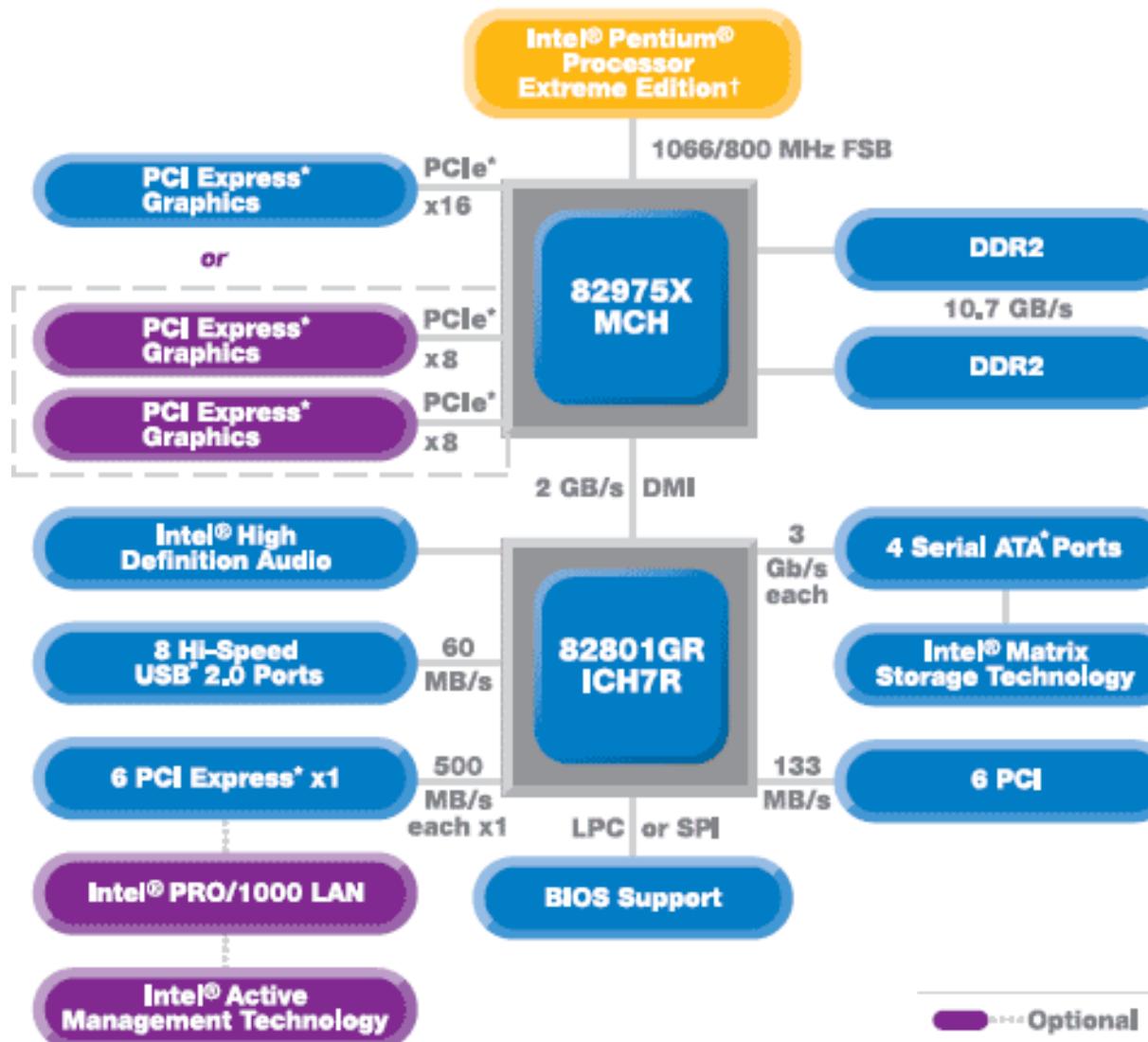


Kiến trúc phân cấp bus hiệu năng cao

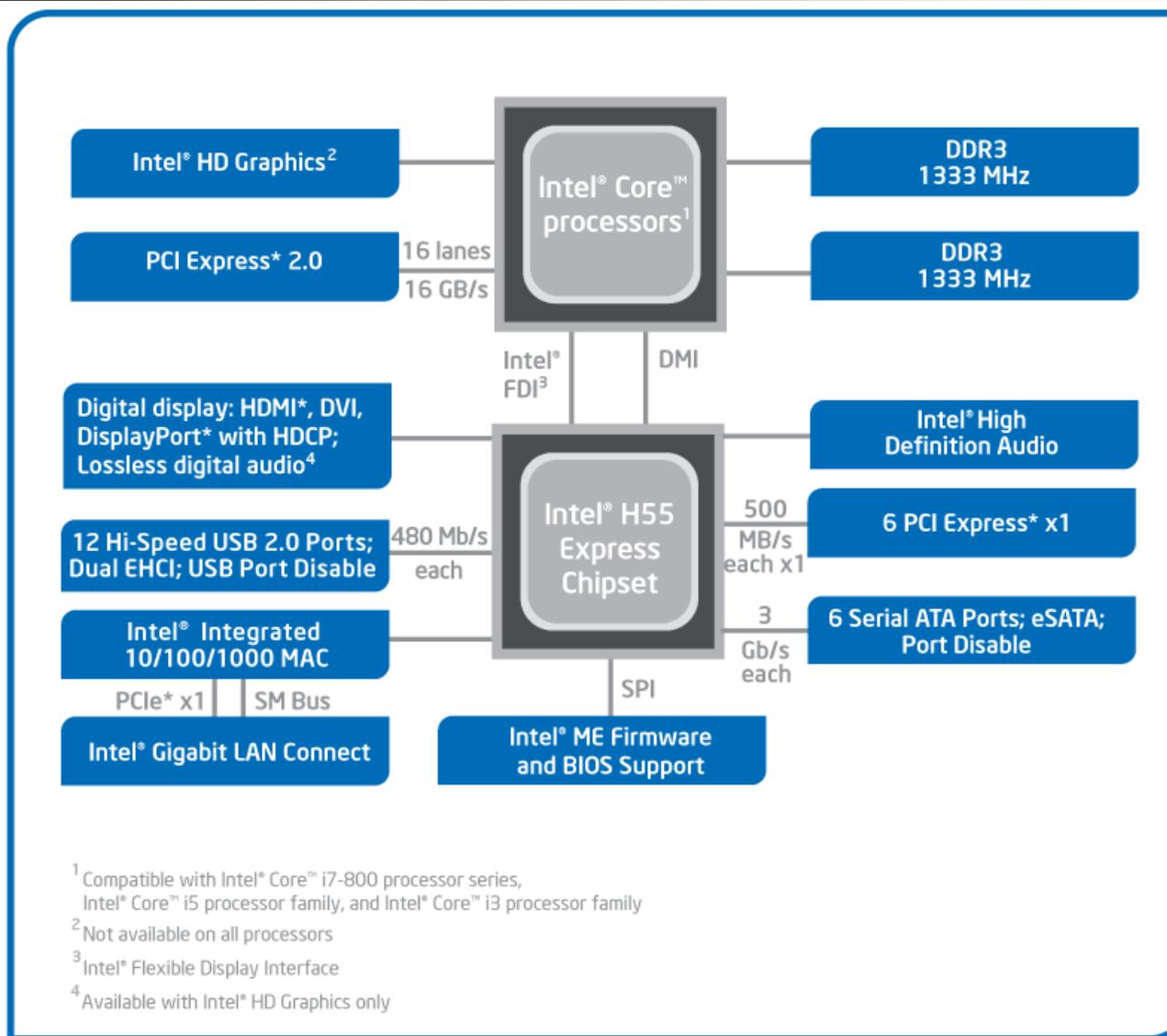


(b) High-performance architecture

Ví dụ bus trong máy tính



Ví dụ bus trong máy tính (tiếp)



Intel® H55 Express Chipset Platform Block Diagram
 Computer Architecture

Ví dụ về bo mạch chính

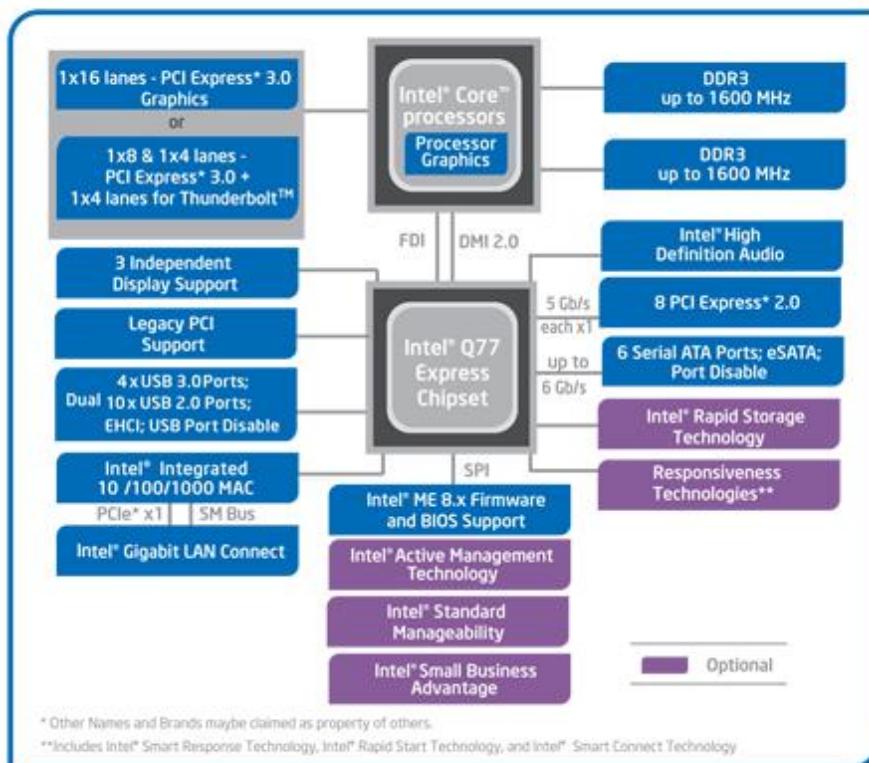


Ví dụ về bo mạch chính GA-P34-DS4



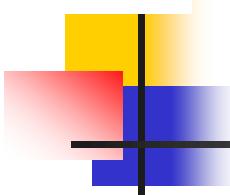
Hướng dẫn sử dụng

Ví dụ bo mạch chính trong máy tính để bàn



Intel Q77 Express Chipset Platform Block Diagram



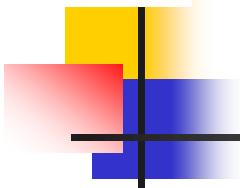


Hết chương 3

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:

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

Giá trị của A được tính như sau:

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

Dải biểu diễn của A: từ 0 đến $2^n - 1$

Các ví dụ

- Ví dụ 1. Biểu diễn các số nguyên không dấu sau đây bằng 8-bit:

$$A = 41 ; \quad B = 150$$

Giải:

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

$$41 = 0010\ 1001$$

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

$$150 = 1001\ 0110$$

Các ví dụ (tiếp)

- 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

$$0000\ 0000 = 0$$

Chú ý:

$$0000\ 0001 = 1$$

$$1111\ 1111$$

$$0000\ 0010 = 2$$

$$+ \underline{0000\ 0001}$$

$$0000\ 0011 = 3$$

$$\textcolor{red}{1}\ 0000\ 0000$$

...

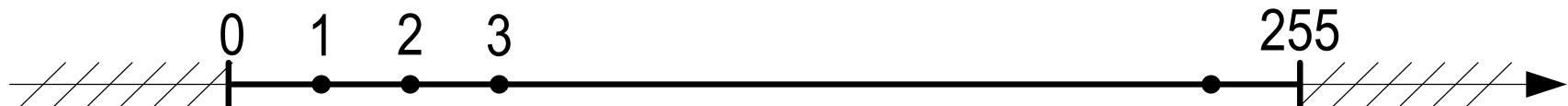
$$1111\ 1111 = 255$$

Vậy: $255 + 1 = 0$?

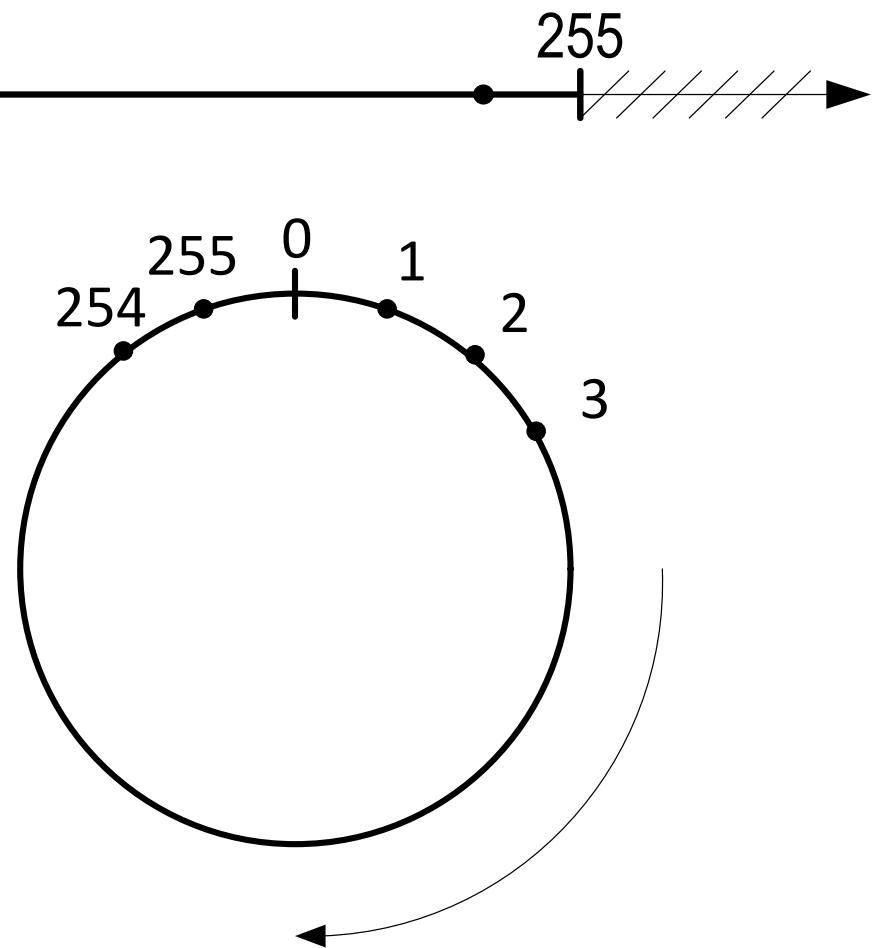
→ do tràn nhớ ra
ngoài

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 chín và Số bù mươi

- Cho một số thập phân A được biểu diễn bằng n chữ số thập phân, ta có:
 - Số bù chín của $A = (10^n - 1) - A$
 - Số bù mươi của $A = 10^n - A$
- Số bù mươi của $A = (\text{Số bù chín của } A) + 1$

Số bù chín và Số bù mươi (tiếp)

- Ví dụ: với $n=4$, cho $A = 3265$

- Số bù chín của A :

$$\begin{array}{r} 9999 \\ - \underline{3265} \\ \hline 6734 \end{array} \quad (10^4 - 1)$$

- Số bù mươi của A :

$$\begin{array}{r} 10000 \\ - \underline{3265} \\ \hline 6735 \end{array} \quad (10^4)$$

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 $A = (\text{Số bù một của } A) + 1$

Số bù một và Số bù hai (tiếp)

Ví dụ: với n = 8 bit, cho A = 0010 0101

- Số bù một của A được tính như sau:

$$\begin{array}{r} 1111\ 1111 \quad (2^8-1) \\ - \underline{0010\ 0101} \quad (A) \\ \hline 1101\ 1010 \\ \rightarrow \text{đảo các bit của A} \end{array}$$

- Số bù hai của A được tính như sau:

$$\begin{array}{r} 1\ 0000\ 0000 \quad (2^8) \\ - \underline{0010\ 0101} \quad (A) \\ \hline 1101\ 1011 \\ \rightarrow \text{thực hiện khó khăn} \end{array}$$

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 $= \begin{array}{r} 1101\ 1010 \\ + \quad \quad \quad 1 \\ \hline 1101\ 1011 \end{array}$
- Số bù hai $= \begin{array}{r} 1101\ 1011 \\ + \quad \quad \quad 1 \\ \hline 1101\ 1011 \end{array}$

- Nhận xét:

$$\begin{array}{rcl} A & = & 0010\ 0101 \\ \text{Số bù hai} & = & + \begin{array}{r} 1101\ 1011 \\ \hline 1\ 0000\ 0000 \end{array} = 0 \end{array}$$

(bỏ qua bit nhớ ra ngoài)

\rightarrow Số bù hai của A = -A

Biểu diễn số nguyên có dấu bằng mã bù hai

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

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

- **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 bằng số bù hai của số dương tương ứng, vì vậy bit $a_{n-1} = 1$

Biểu diễn số dương

- Dạng tổng quát của số dương A:

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

- Giá trị của số dương A:

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

- Dải biểu diễn cho số dương: 0 đến $2^{n-1}-1$

Biểu diễn số âm

- Dạng tổng quát của số âm A:

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

- Giá trị của số âm A:

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

- Dải biểu diễn cho số âm: -1 đến -2^{n-1}

Biểu diễn tổng quát cho số nguyên có dấu

- Dạng tổng quát của số nguyên A:

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

- 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: từ $-(2^{n-1})$ đến $+(2^{n-1}-1)$

Các ví dụ

- Ví dụ 1. Biểu diễn các số nguyên có dấu sau đây bằng 8-bit:

$$A = +58 ; \quad B = -80$$

Giải:

$$A = +58 = 0011\ 1010$$

$$B = -80$$

$$\text{Ta có: } +80 = 0101\ 0000$$

$$\text{Số bù một} = 1010\ 1111$$

$$\begin{array}{r} & & & + & 1 \\ & & & \hline & 1011\ 0000 \end{array}$$

Số bù hai

$$\text{Vậy: } B = -80 = 1011\ 0000$$

Các ví dụ

- Ví dụ 2. Hãy xác định giá trị của các số nguyên có dấu được biểu diễn dưới đây:
 - $P = 0110\ 0010$
 - $Q = 1101\ 1011$

Giải:

- $P = 0110\ 0010 = -0*2^7 + 64 + 32 + 2 = +98$
- $Q = 1101\ 1011 = -128 + 64 + 16 + 8 + 2 + 1 = -37$

Với $n = 8$ bit

Biểu diễn được các giá trị từ -128 đến +127

$$0000\ 0000 = 0$$

$$0000\ 0001 = +1$$

$$0000\ 0010 = +2$$

$$0000\ 0011 = +3$$

...

$$0111\ 1111 = +127$$

$$1000\ 0000 = -128$$

$$1000\ 0001 = -127$$

...

$$1111\ 1110 = -2$$

$$1111\ 1111 = -1$$

Chú ý:

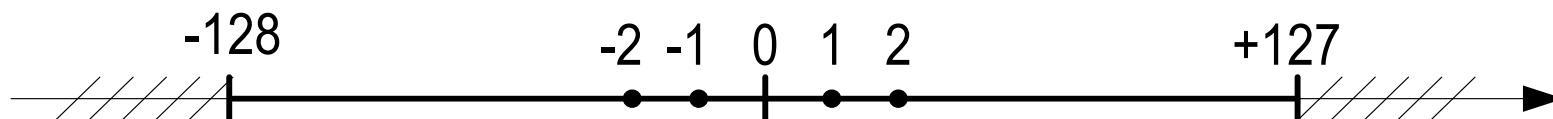
$$+127 + 1 = -128$$

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

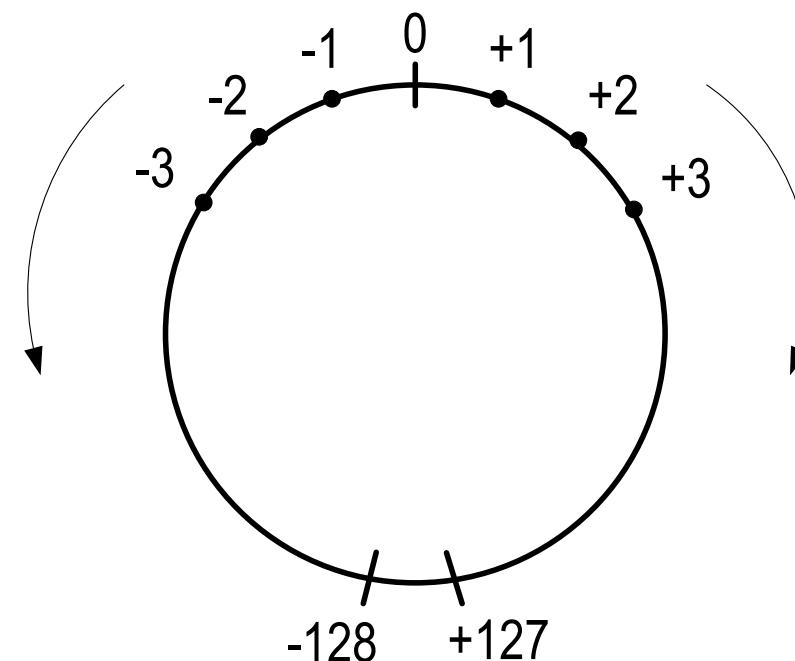
→ do tràn xảy ra

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

Chuyển đổi dữ liệu từ 8 bit thành 16 bit

- Đối với số dương:

+19 = 0001 0011 (8bit)

+19 = 0000 0000 0001 0011 (16bit)

→ thêm 8 bit 0 bên trái

- Đối với số âm:

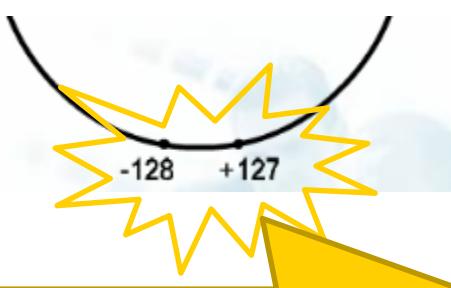
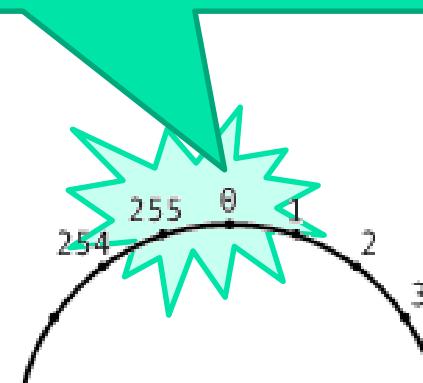
- 19 = 1110 1101 (8bit)

- 19 = 1111 1111 1110 1101 (16bit)

→ thêm 8 bit 1 bên trái

Phân biệt giữa tràn và tràn số học

Số nguyên **không** dấu:
Tràn số. Carryout

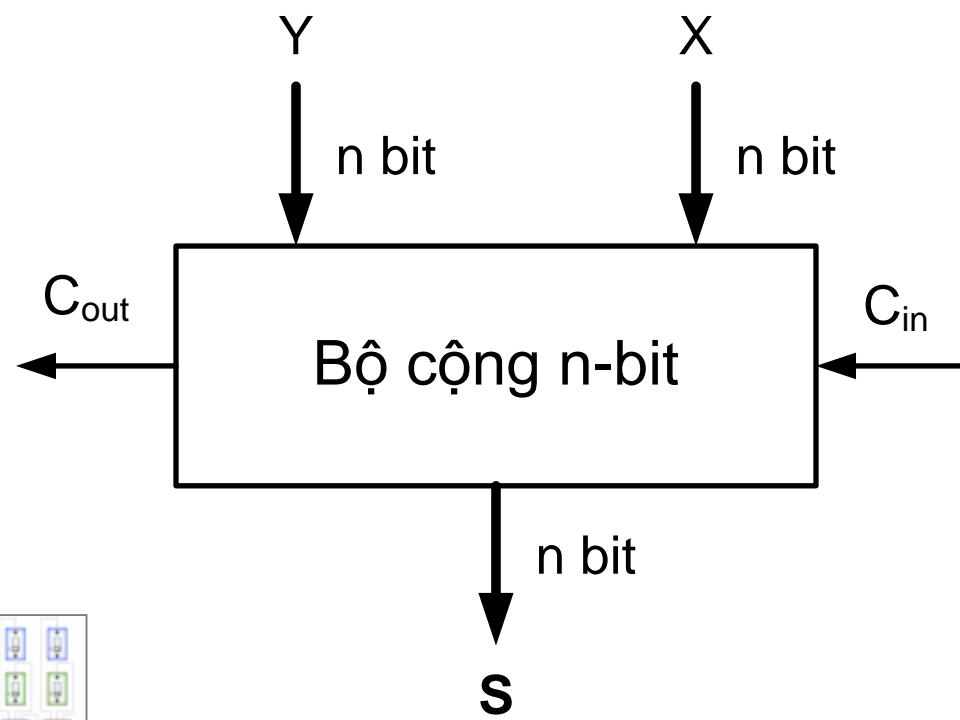


Số nguyên **có** dấu:
Tràn số học, Overflow

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 $C_{out}=0 \rightarrow$ nhận được kết quả đúng.
- Nếu $C_{out}=1 \rightarrow$ nhận được kết quả sai, do tràn nhớ ra ngoài (*Carry Out*).
- Tràn nhớ ra ngoài khi: $tổng > (2^n - 1)$

Ví dụ cộng số nguyên không dấu

- $$\begin{array}{rcl} 57 & = & 0011\ 1001 \\ + \underline{34} & = + & \underline{0010\ 0010} \\ 91 & & 0101\ 1011 = 64+16+8+2+1=91 \rightarrow \text{đúng} \end{array}$$

- $$\begin{array}{rcl} 209 & = & 1101\ 0001 \\ + \underline{73} & = + & \underline{0100\ 1001} \\ 282 & & \textcolor{red}{1}\ 0001\ 1010 \\ & & 0001\ 1010 = 16+8+2=26 \rightarrow \text{sai} \\ & & \rightarrow \text{có tràn nhớ ra ngoài } (\textcolor{red}{C}_{\text{out}}=1) \end{array}$$

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

$$\begin{array}{rcl} 209 & = & 0000\ 0000\ 1101\ 0001 \\ + 73 & = + & \underline{0000\ 0000\ 0100\ 1001} \\ & & 0000\ 0001\ 0001\ 1010 = 256+16+8+2 = 282 \end{array}$$

2. Phép đảo dấu

- Ta có:

$$\begin{array}{rcl}
 + 37 & = & 0010\ 0101 \\
 \text{bù một} & = & 1101\ 1010 \\
 & & + \frac{1}{} \\
 \text{bù hai} & = & 1101\ 1011 = -37
 \end{array}$$

- Lấy bù hai của số âm:

$$\begin{array}{rcl}
 - 37 & = & 1101\ 1011 \\
 \text{bù một} & = & 0010\ 0100 \\
 & & + \frac{1}{} \\
 \text{bù hai} & = & 0010\ 0101 = +37
 \end{array}$$

- 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}** .

- Cộng hai số khác dấu: **kết quả luôn luôn đúng.**
- Cộng hai số cùng dấu:
 - nếu dấu kết quả cùng dấu với các số hạng thì **kết quả là đúng.**
 - nếu kết quả có dấu ngược lại, khi đó có **tràn** xảy ra (**Overflow**) và **kết quả bị sai.**
- 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

- $$\begin{array}{r} (+ 70) \\ + \underline{(+ 42)} \\ \hline + 112 \end{array} = \begin{array}{r} 0100\ 0110 \\ \underline{0010\ 1010} \\ 0111\ 0000 \end{array} = +112$$

- $$\begin{array}{r} (+ 97) \\ + \underline{(- 52)} \\ \hline + 45 \end{array} = \begin{array}{r} 0110\ 0001 \\ \underline{1100\ 1100} \\ 1\ 0010\ 1101 \end{array} = +45$$
 (+52=0011 0100)

- $$\begin{array}{r} (- 90) \\ + \underline{(+36)} \\ \hline - 54 \end{array} = \begin{array}{r} 1010\ 0110 \\ \underline{0010\ 0100} \\ 1100\ 1010 \end{array} = -54$$
 (+90=0101 1010)

- $$\begin{array}{r} (- 74) \\ + \underline{(- 30)} \\ \hline -104 \end{array} = \begin{array}{r} 1011\ 0110 \\ \underline{1110\ 0010} \\ 1\ 1001\ 1000 \end{array} = -104$$
 (+74=0100 1010)
(+30=0001 1110)

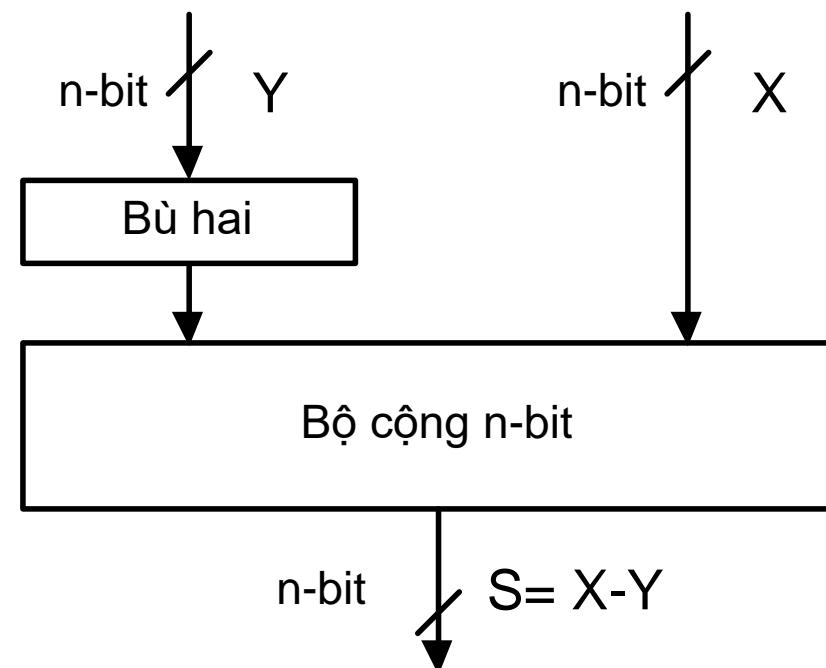
Ví dụ cộng số nguyên có dấu bị tràn

- $(+ 75) = 0100\ 1011$
- $+ (\underline{+ 82}) = \underline{0101\ 0010}$
- $+ 157 \quad \quad \quad 1001\ 1101$
- $= - 128 + 16 + 8 + 4 + 1 = - 99 \rightarrow \text{sai}$

- $(- 104) = 1001\ 1000 \quad (+104=0110\ 1000)$
- $+ (-\underline{43}) = \underline{1101\ 0101} \quad (+43=0010\ 1011)$
- $- 147 \quad \quad \quad 1\ 0110\ 1101$
- $= 64 + 32 + 8 + 4 + 1 = +109 \rightarrow \text{sai}$
- 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

$$\begin{array}{r} 1011 \\ \times \underline{1101} \\ \hline 1011 \\ 0000 \\ 1011 \\ 1011 \\ \hline 10001111 \end{array}$$

Số bị nhân (11)
Số nhân (13)

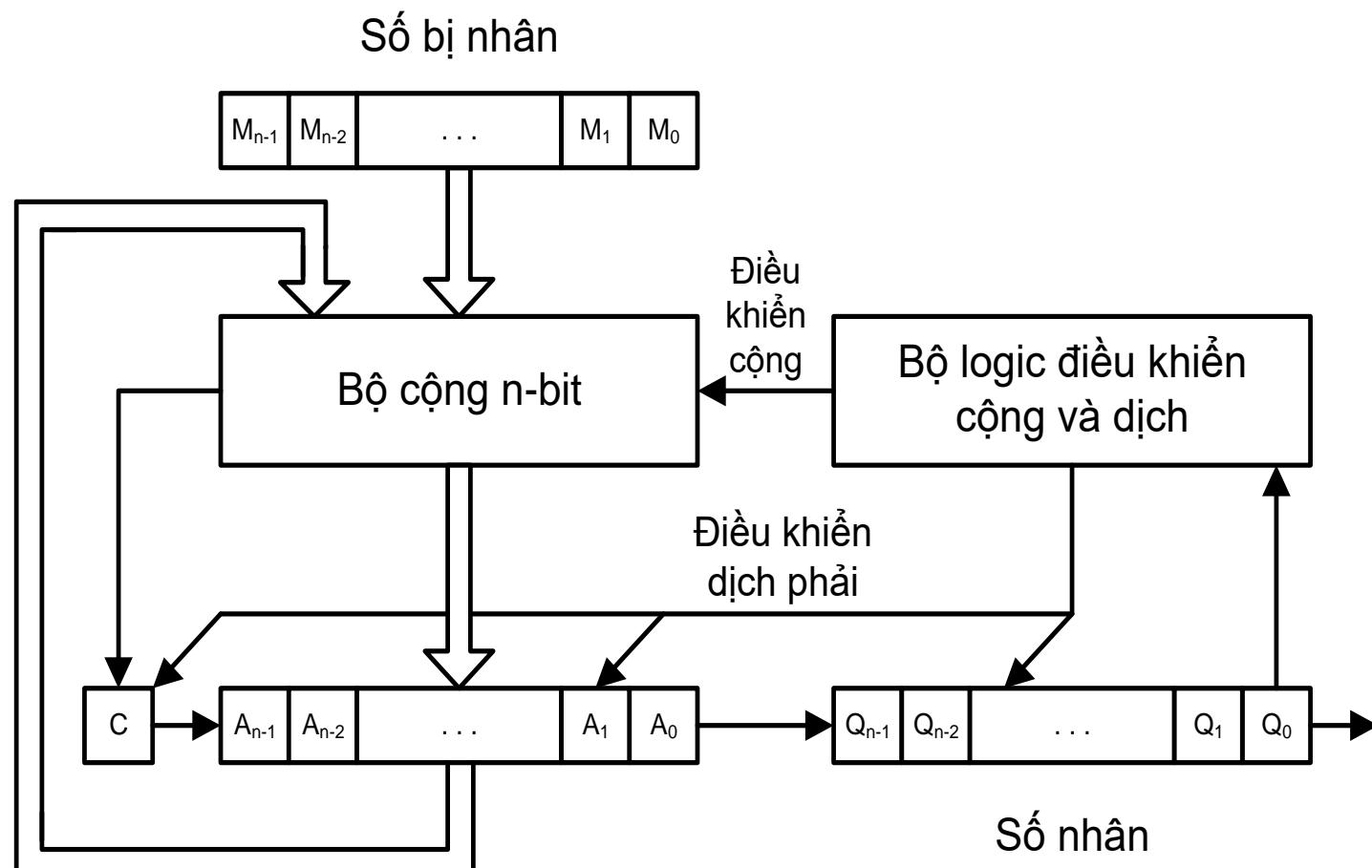
Các tích riêng phần

Tích (143)

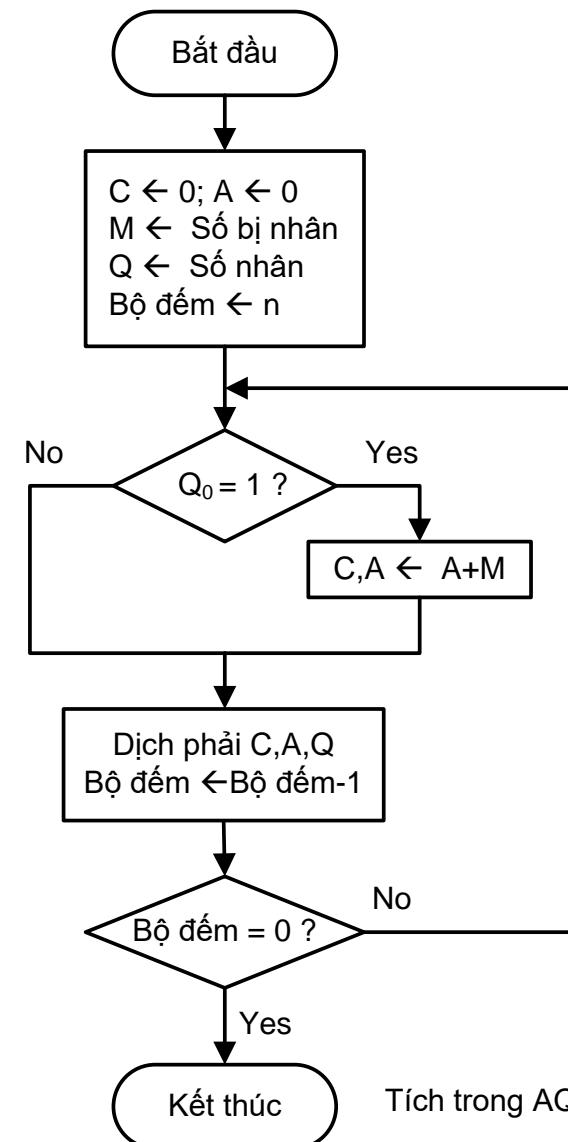
Nhân số nguyên không dấu (tiếp)

- 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 \leftarrow A + M
■	0	0101	1110	Dịch phải
■	0	0010	1111	Dịch phải
		+ 1011		
■	0	1101	1111	A \leftarrow A + M
■	0	0110	1111	Dịch phải
		+ 1011		
■	1	0001	1111	A \leftarrow A + M
■	0	1000	1111	Dịch phải

Ví dụ nhân số nguyên không dấu

- Số bị nhân M = 0010 (2)
- Số nhân Q = 0011 (3)
- Tích = 0000 0110 (6)

- | | C | A | Q | |
|---|---|-------------|--------------|----------------------|
| ■ | 0 | 0000 | 001 1 | Các giá trị khởi đầu |
| | + | <u>0010</u> | | |
| | 0 | 0010 | 0011 | A $\leftarrow A + M$ |
| ■ | 0 | 0001 | 000 1 | Dịch phải |
| | + | <u>0010</u> | | |
| | 0 | 0011 | 0001 | |
| ■ | 0 | 0001 | 100 0 | Dịch phải |
| ■ | 0 | 0000 | 110 0 | Dịch phải |
| ■ | 0 | 0000 | 0110 | Dịch phải |

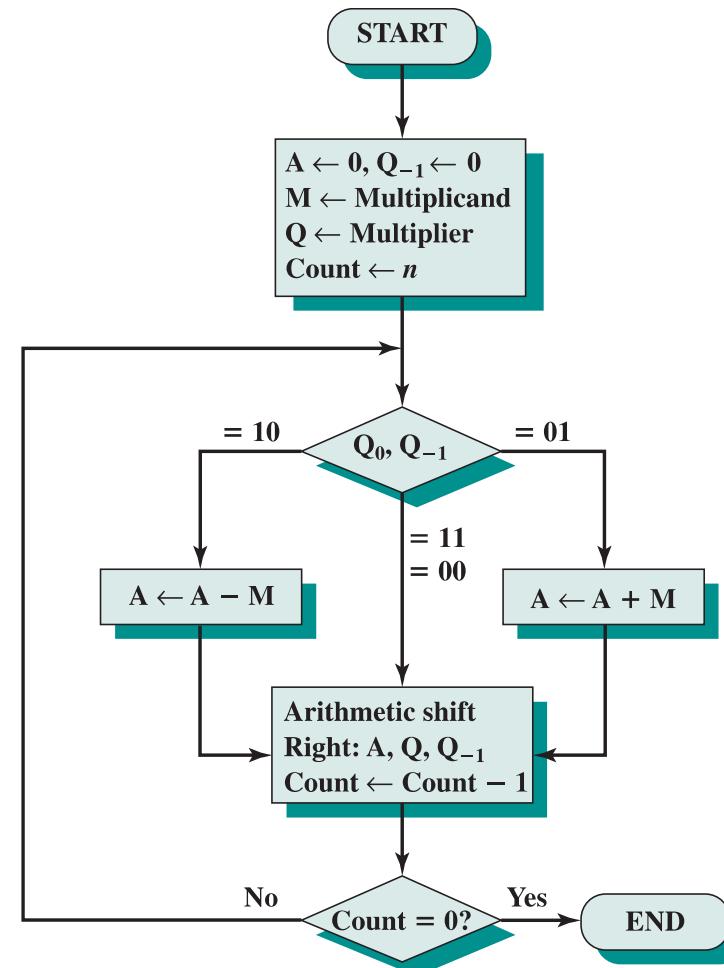
2. Nhân số nguyên có dấu

- 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).

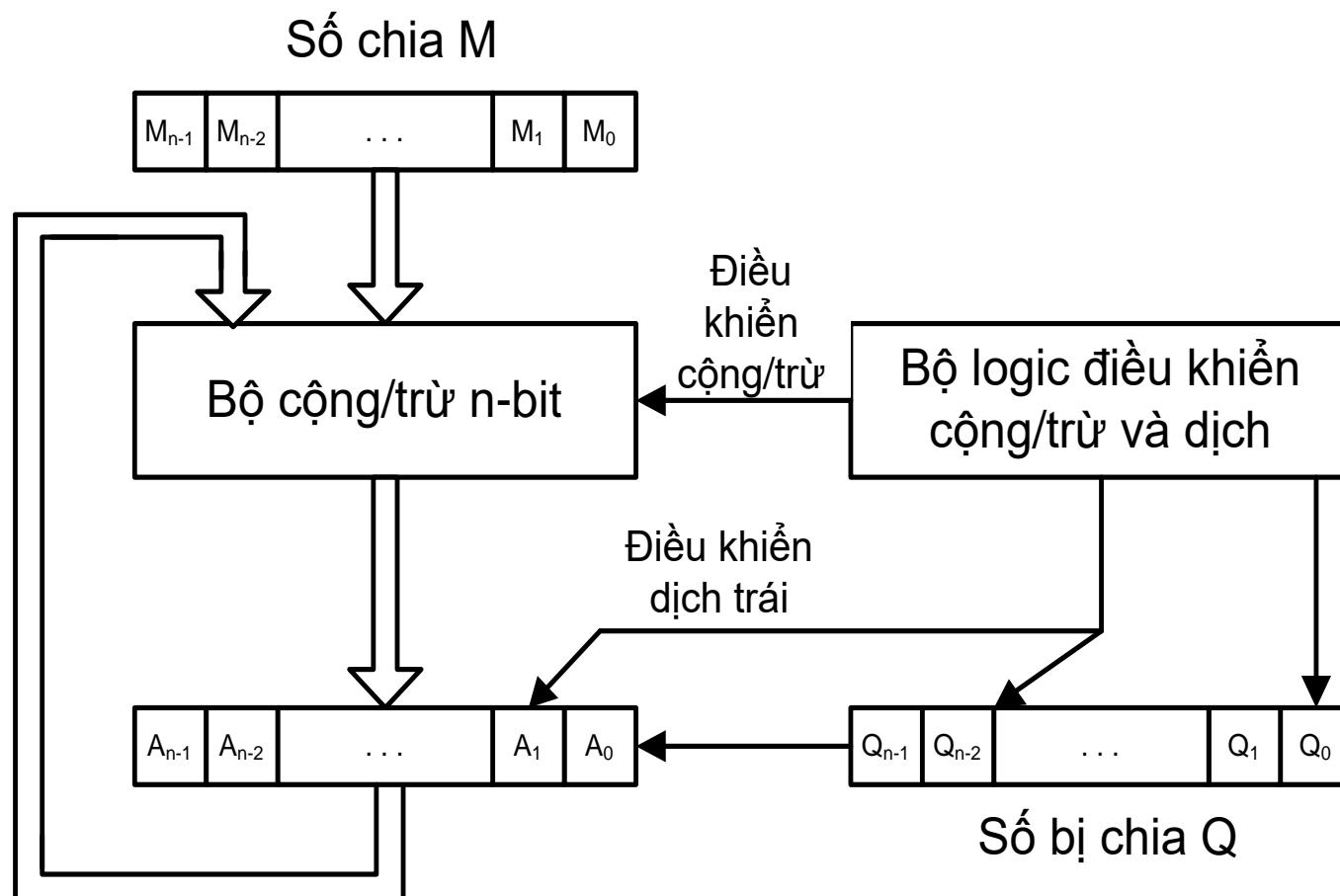
Thuật giải Booth (tham khảo COA-[1])



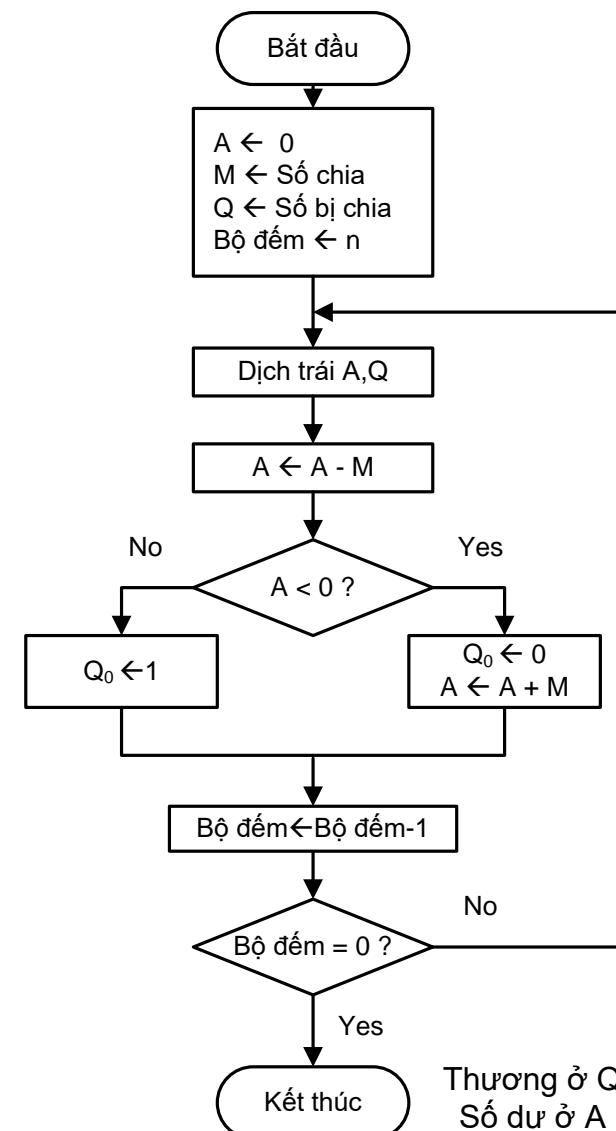
3. Chia số nguyên không dấu

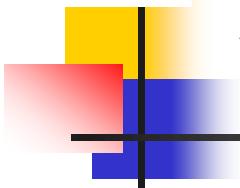
Số bị chia	10010011		Số chia
	- <u>1011</u>	00001101	Thương
	001110		
	- <u>1011</u>		
	001111		
	- <u>1011</u>		
	100		Phần dư

Bộ chia số nguyên không dấu



Lưu đồ chia số nguyên không dấu





Ví dụ

A				

A 0000	Q 0111	Initial value
0000	1110	Shift
<u>1101</u>		Use twos complement of 0011 for subtraction
<u>1101</u>		Subtract
0000	1110	Restore, set $Q_0 = 0$
0001	1100	Shift
<u>1101</u>		Subtract
<u>1110</u>		Restore, set $Q_0 = 0$
0001	1100	
0011	1000	Shift
<u>1101</u>		

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)

cùng dấu

Số bị chia	Số chia	Thương	Phần 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:

$$X = M * R^E$$

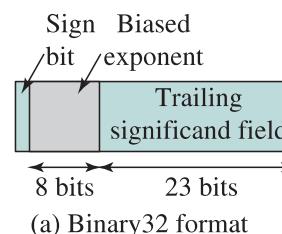
- M là phần định trị (Mantissa),
- R là cơ số (Radix),
- E là phần mũ (Exponent).

2. Chuẩn IEEE754-2008

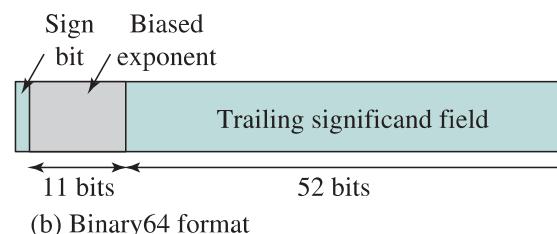
- Cơ số $R = 2$

- Các dạng:

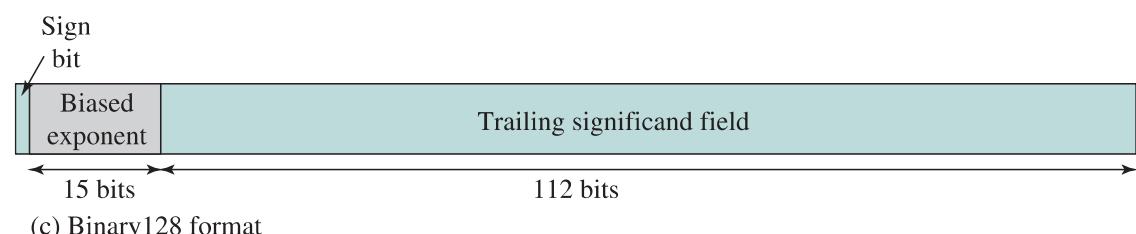
- Dạng 32-bit



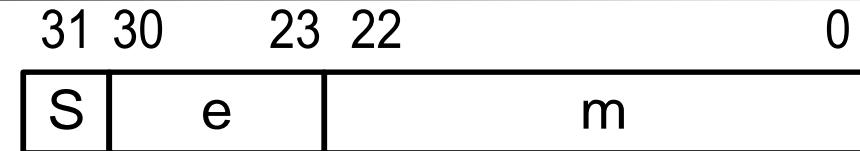
- Dạng 64-bit



- Dạng 128-bit



Dạng 32 bit



- **S** là bit dấu:
 - $S = 0 \rightarrow$ số dương
 - $S = 1 \rightarrow$ số âm
- **e** (8 bit) là mã *excess-127* của phần mũ E:
 - $e = E + 127 \rightarrow E = e - 127$
 - giá trị 127 gọi là độ lệch (bias)
- **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 số thực được biểu diễn bằng 32-bit như sau:

- **1100 0001 0101 0110 0000 0000 0000 0000**

- $S = 1 \rightarrow$ số âm

- $e = 1000\ 0010_2 = 130 \rightarrow E = 130 - 127 = 3$

Vậy

$$X = -1.\textcolor{red}{10101100} * 2^3 = -1101.011 = -13.375$$

- **0011 1111 1000 0000 0000 0000 0000 0000 = ?**

$$= +1.0$$

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

Ví dụ 2

Biểu diễn số thực $X = 83.75$ về dạng số dấu phẩy động IEEE754 32-bit

Giải:

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

- $X = 83.75_{(10)} = 1010011.11_{(2)} = (-1)^0 \cdot 1.01001111 \times 2^6$
- Ta có:
 - $S = 0$ vì đây là số dương
 - $E = e - 127 = 6 \rightarrow e = 127 + 6 = 133_{(10)} = 1000\ 0101_{(2)}$
- Vậy:
 $X = 0100\ 0010\ 1010\ 0111\ 1000\ 0000\ 0000\ 0000$

Ví dụ 3

Biểu diễn số thực $X = -0,2$ về dạng số dấu phẩy động IEEE754 32-bit

Giải:

- $X = -0,2_{(10)} = -0.00110011..0011..._{(2)} = -1.100110011..0011... \times 2^{-3}$
- Ta có:
 - $S = 1$ vì đây là số âm
 - $E = e - 127 = -3 \rightarrow e = 127 - 3 = 124_{(10)} = 0111\ 1100_{(2)}$
- Vậy:
 $X = 1011\ 1110\ 0100\ 1100\ 1100\ 1100\ 1100$

Bài tập

Biểu diễn các số thực sau đây về dạng
số dấu phẩy động IEEE754 32-bit:

$$X = -27.0625; \quad Y = 1/32$$

Các qui ước đặc biệt

- Các bit của e bằng 0, các bit của m bằng 0, thì $X = \pm 0$
 $x000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000 \rightarrow X = \pm 0$
- Các bit của e bằng 1, các bit của m bằng 0, thì $X = \pm \infty$
 $x111\ 1111\ 1000\ 0000\ 0000\ 0000\ 0000\ 0000 \rightarrow X = \pm \infty$
- 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): mã *excess-1023* của phần mũ E → $E = e - 1023$
- m (52 bit): phần lẻ của phần định trị M
- Giá trị số thực:

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

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

Dạng 128-bit

- S là bit dấu
- e (15 bit): mã *excess-16383* của phần mũ E → $E = e - 16383$
- m (112 bit): phần lẻ của phần định trị M
- Giá trị số thực:
$$X = (-1)^S \cdot 1.m \cdot 2^{e-16383}$$
- Dải giá trị biểu diễn: 10^{-4932} đến 10^{+4932}

3. Thực hiện phép toán số dấu phẩy động

- $X_1 = M_1 * R^{E_1}$
- $X_2 = M_2 * R^{E_2}$
- Ta có
 - $X_1 * X_2 = (M_1 * M_2) * R^{E_1+E_2}$
 - $X_1 / X_2 = (M_1 / M_2) * R^{E_1-E_2}$
 - $X_1 \pm X_2 = (M_1 * R^{E_1-E_2} \pm M_2) * R^{E_2}$, với $E_2 \geq E_1$

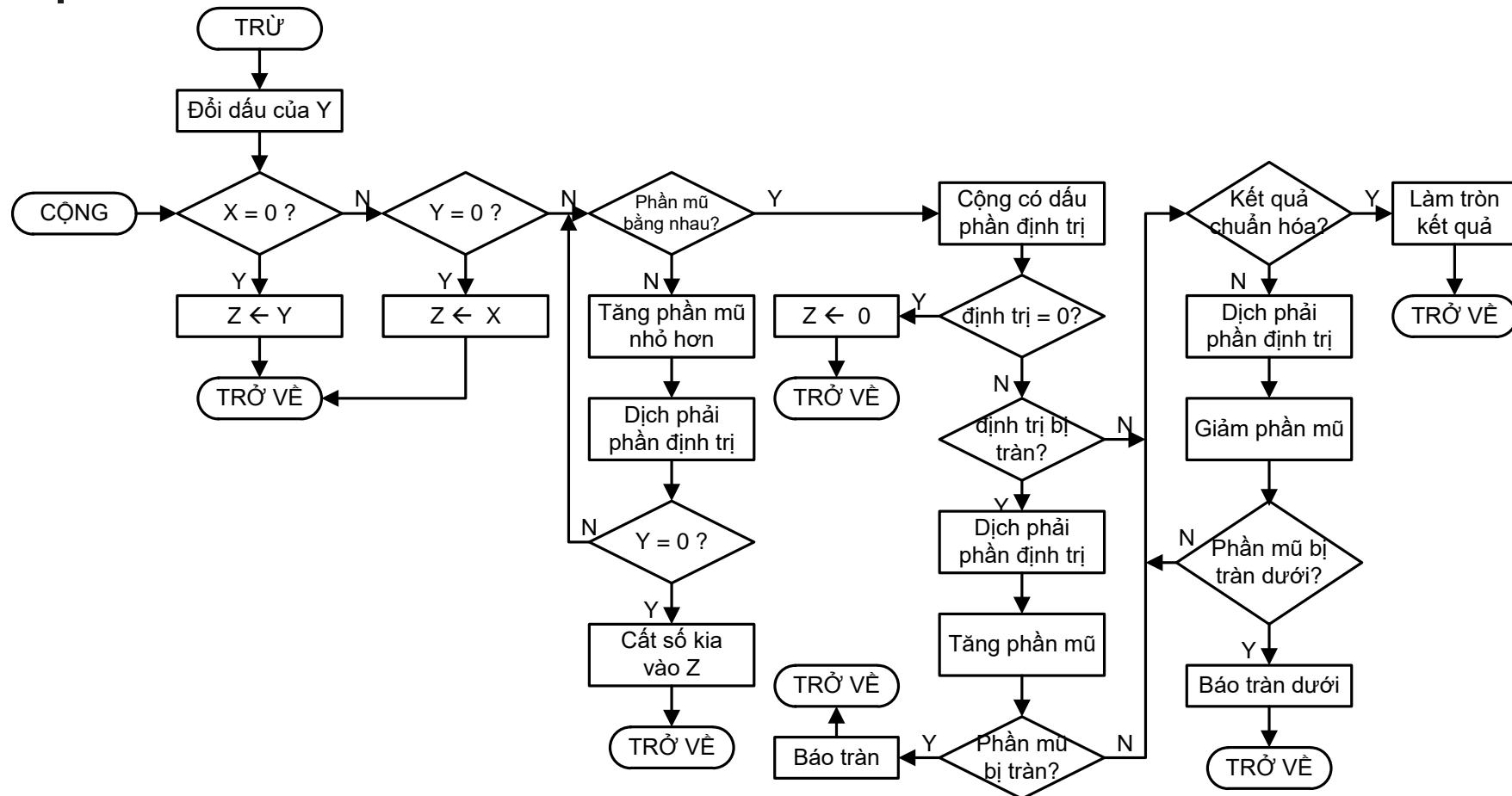
Các khả năng tràn số

- Tràn trên số mũ (Exponent Overflow): mũ dương vượt ra khỏi giá trị cực đại của số mũ dương có thể. ($\rightarrow \infty$)
- Tràn dưới số mũ (Exponent Underflow): mũ âm vượt ra khỏi giá trị cực đại của số mũ âm có thể ($\rightarrow 0$).
- Tràn trên phần định trị (Mantissa Overflow): cộng hai phần định trị có cùng dấu, kết quả bị nhór 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.

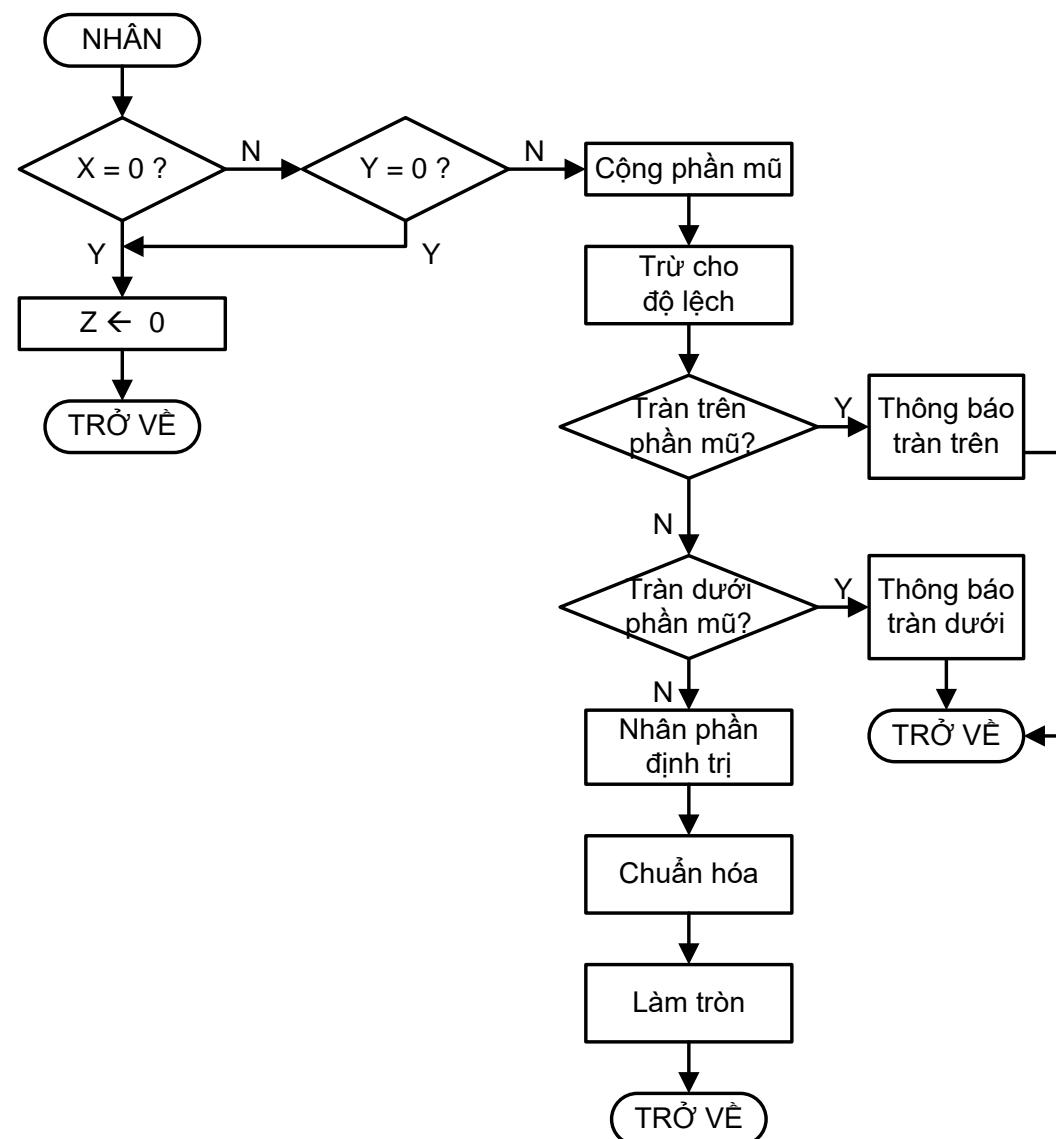
Phép cộng và phép trừ

- 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

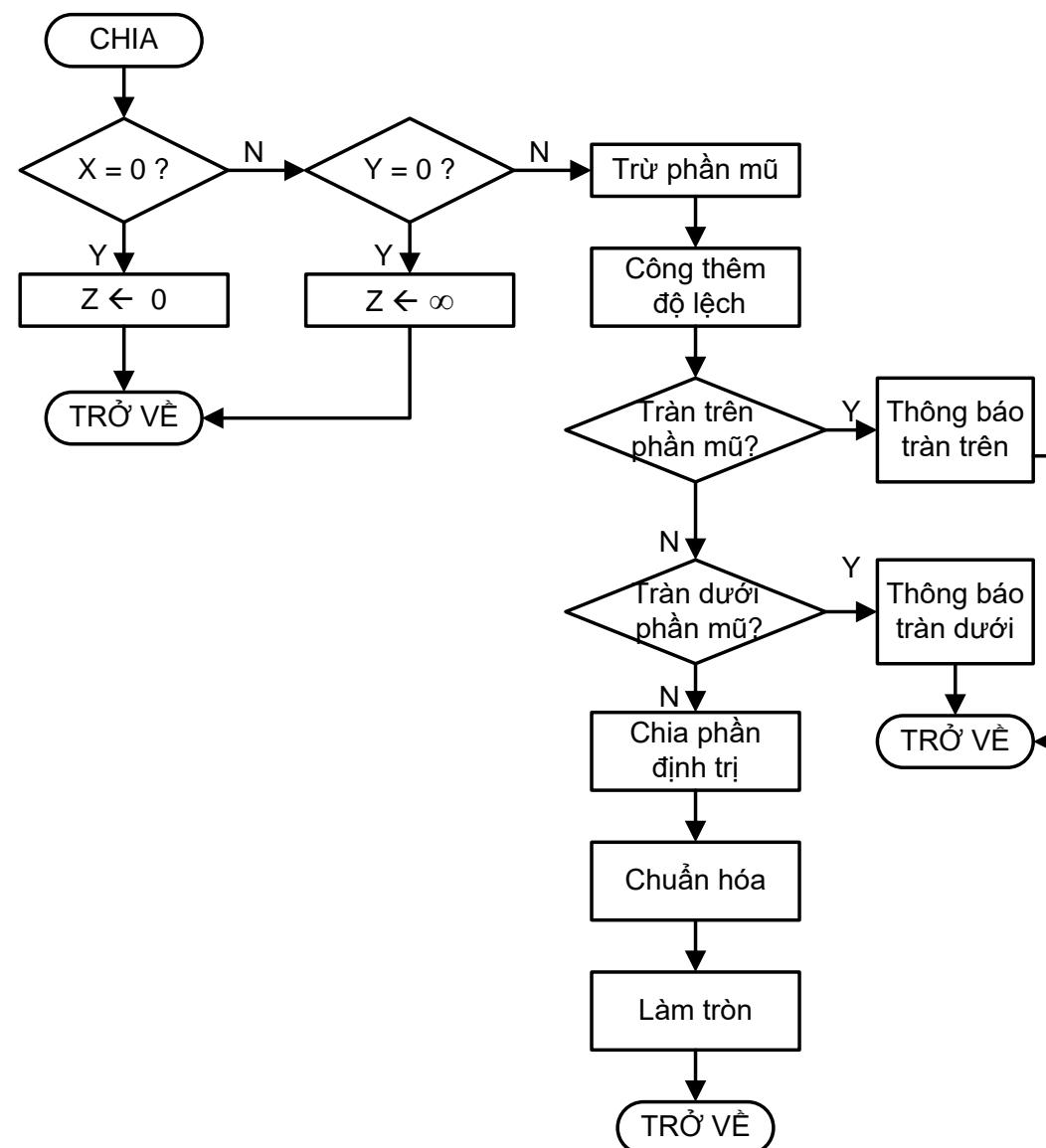
Thuật toán cộng/trừ số dấu phẩy động

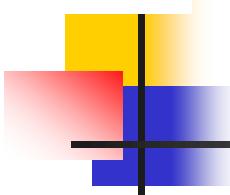


Thuật toán nhân số dấu phẩy động



Thuật toán chia số dấu phẩy động





Hết chương 4

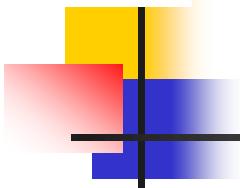
Chương 5

KIẾN TRÚC TẬP LỆNH

(Instruction Set Architecture)

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

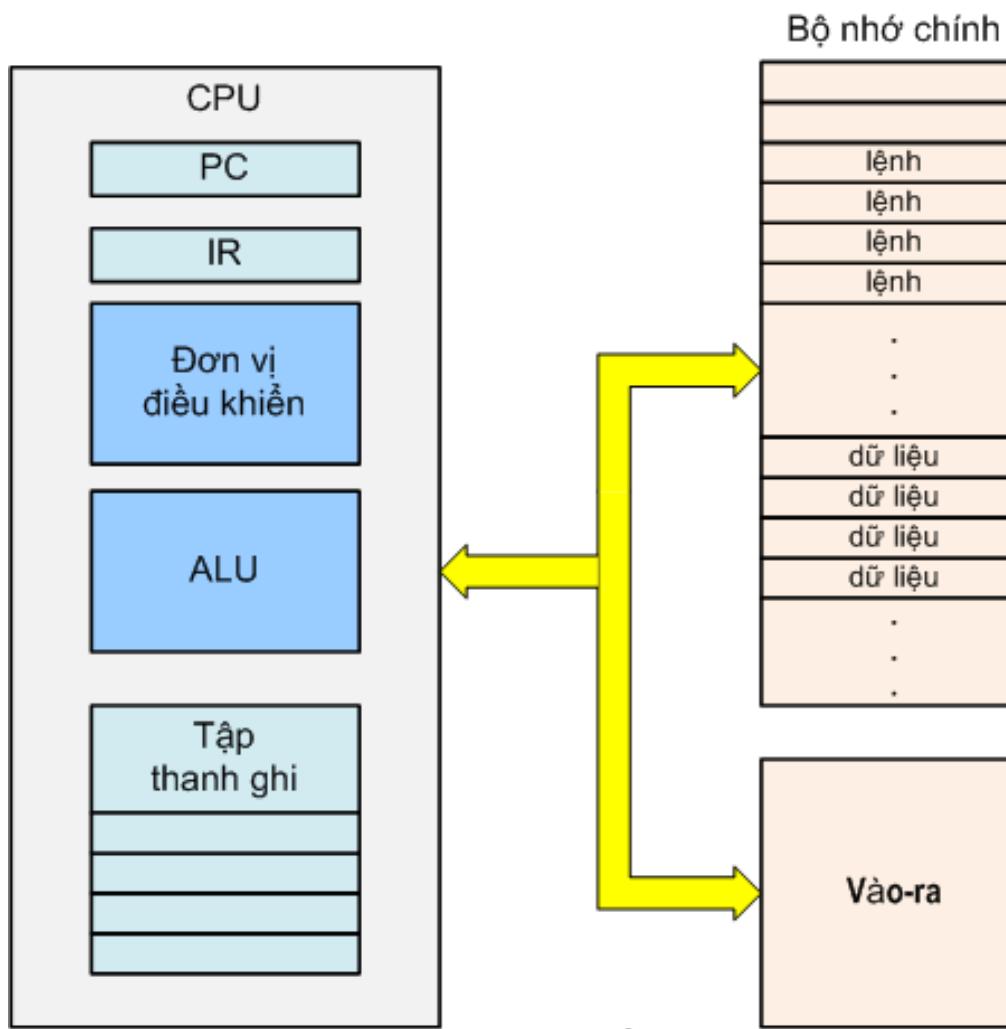
Chương 9. Các kiến trúc song song

Nội dung của chương 5

- 5.1. Giới thiệu chung về kiến trúc tập lệnh
- 5.2. Lệnh hợp ngũ và toán hạng
- 5.3. Mã máy
- 5.4. Cơ bản về lập trình hợp ngũ
- 5.5. Các phương pháp định địa chỉ
- 5.6. Dịch và chạy chương trình hợp ngũ

5.1. Giới thiệu chung về kiến trúc tập lệnh

1. Mô hình lập trình của máy tính



PC: Program Counter

IR: Instruction Register

Tập thanh ghi

- Chứa các thông tin (dữ liệu, địa chỉ, trạng thái) cho hoạt động điều khiển và xử lý dữ liệu của CPU ở thời điểm hiện tại
- Được coi là mức đầu tiên của hệ thống nhớ
- Số lượng thanh ghi nhiều → tăng hiệu năng của CPU
- Có hai loại thanh ghi:
 - Các thanh ghi lập trình được
 - Các thanh ghi không lập trình được

Một số thanh ghi điển hình

- Các thanh ghi địa chỉ
 - Bộ đếm chương trình PC (Program Counter)
 - Con trỏ dữ liệu DP (Data Pointer)
 - Con trỏ ngăn xếp SP (Stack Pointer)
 - Thanh ghi cơ sở và Thanh ghi chỉ số (Base Register & Index Register)
- Các thanh ghi dữ liệu
- Thanh ghi trạng thái

Ý nghĩa của các thanh ghi địa chỉ

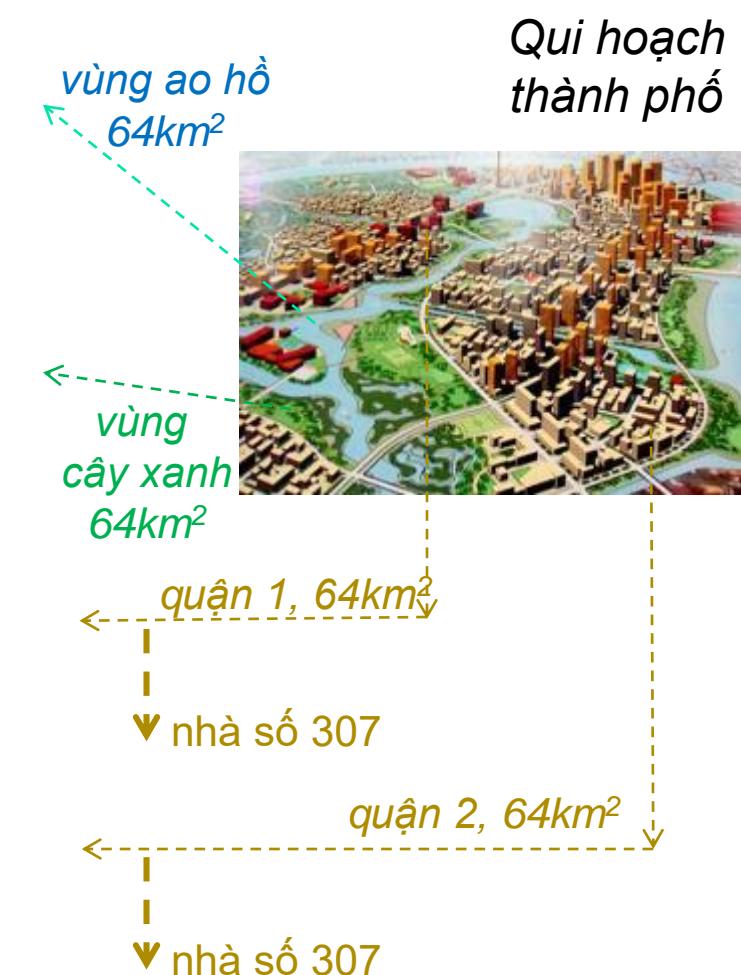
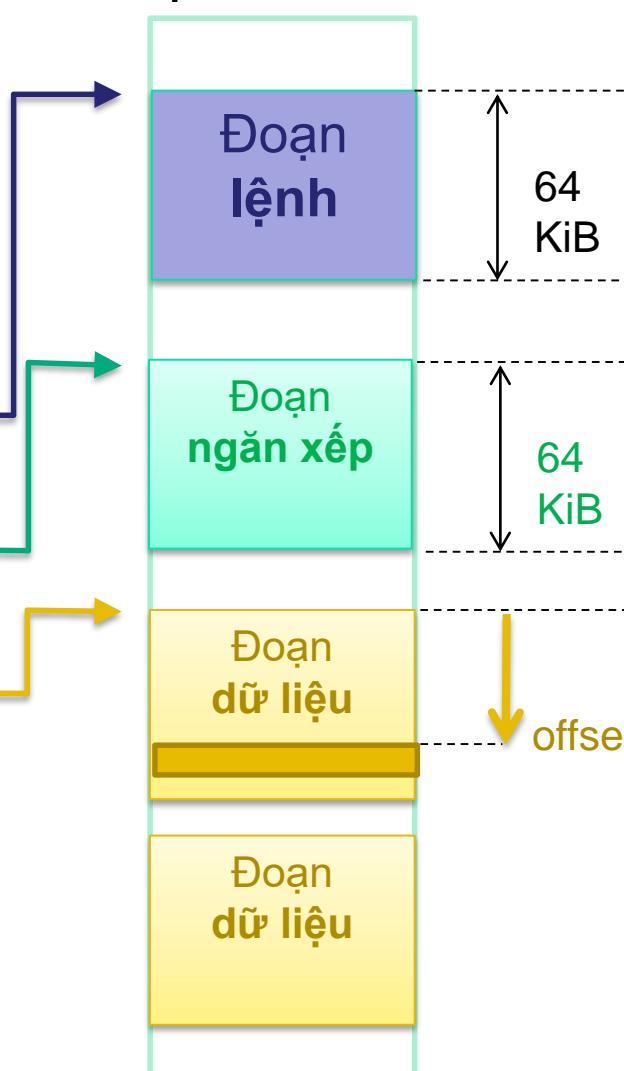
Bộ nhớ chính

Thanh ghi
địa chỉ

PC

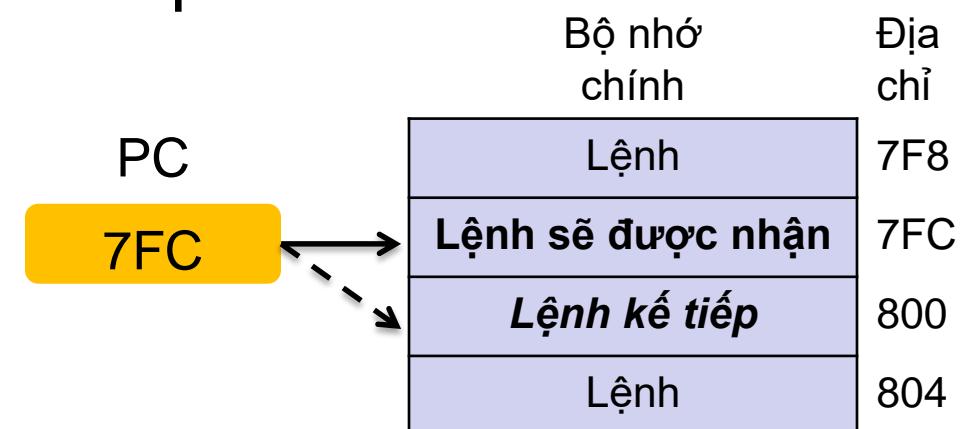
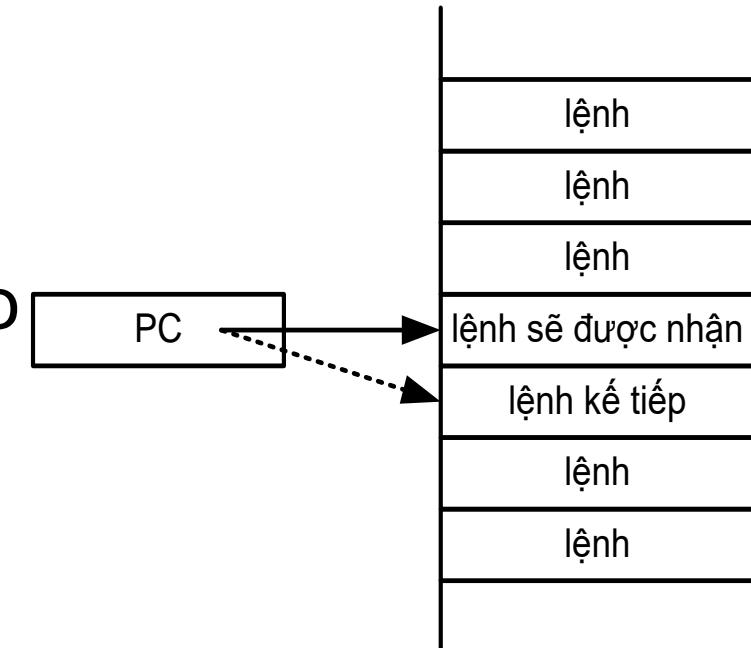
SP

DP



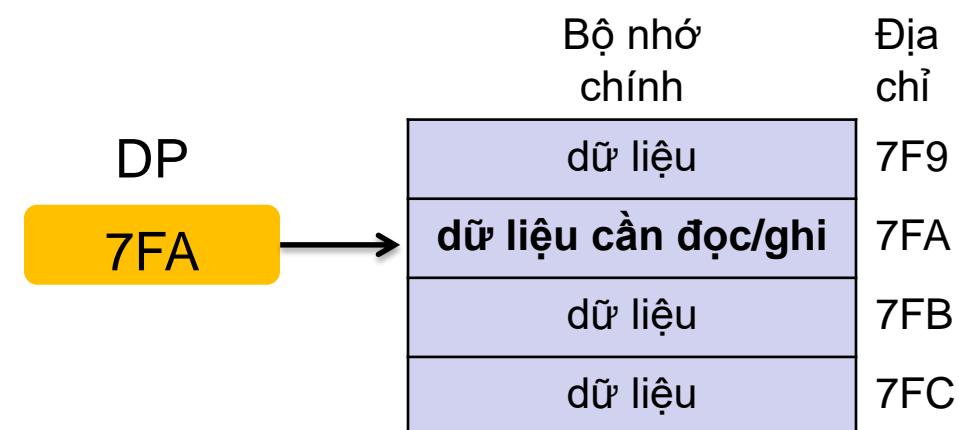
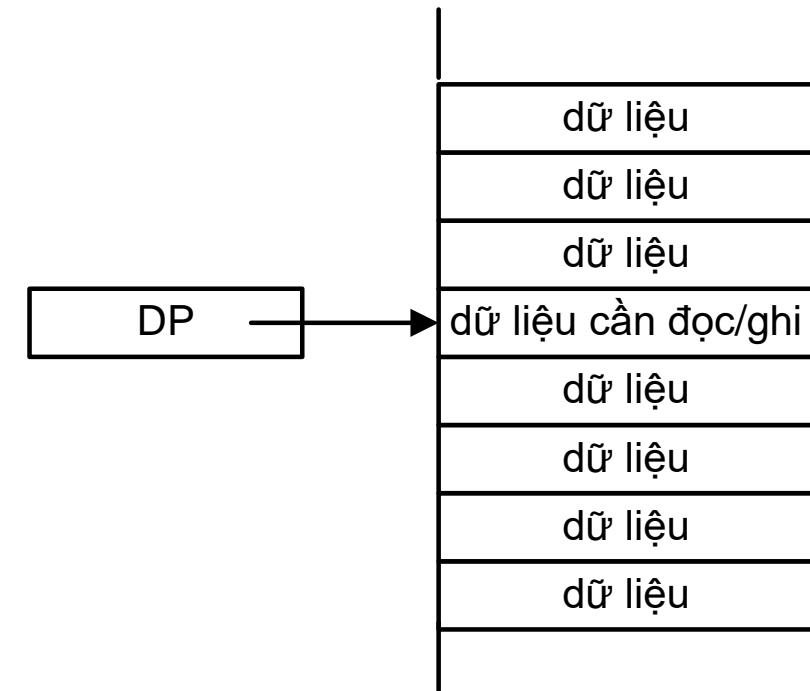
Bộ đếm chương trình PC

- Còn được gọi là con trỏ lệnh IP (Instruction Pointer)
- Giữ địa chỉ của lệnh tiếp theo sẽ được nhận vào.
- Sau khi một lệnh được nhận vào, nội dung PC **tự động tăng** để trỏ sang lệnh kế tiếp.
- PC tăng bao nhiêu?



Thanh ghi con trả dữ liệu

- Chứa địa chỉ của ngăn nhớ dữ liệu mà CPU muốn truy nhập

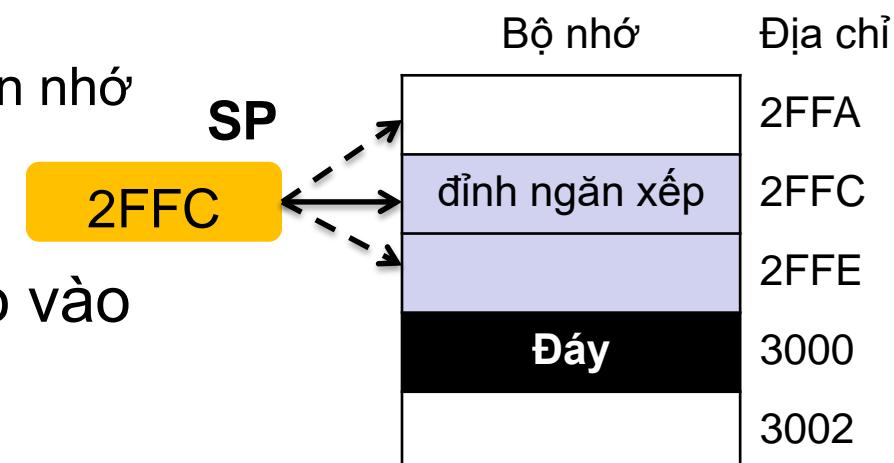
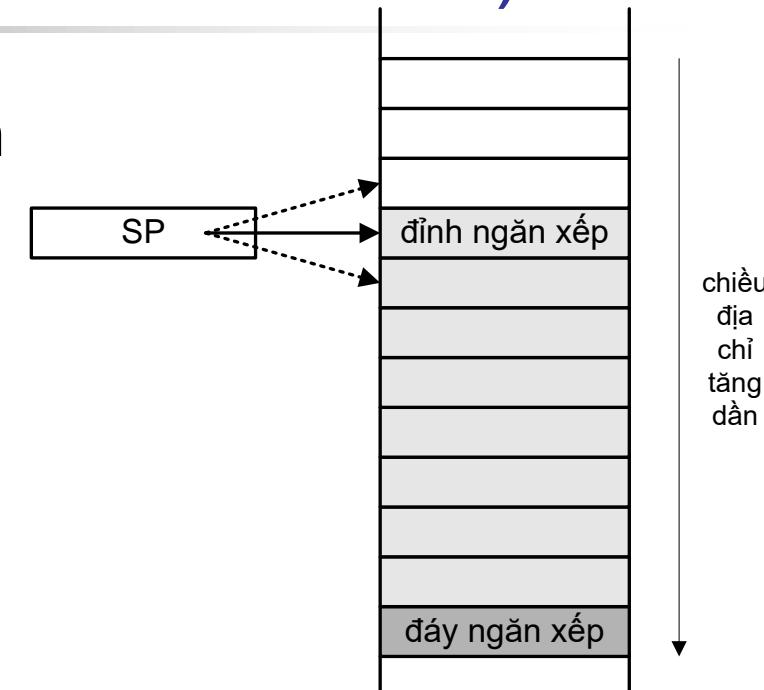


Ngăn xếp (Stack)

- Ngăn xếp là vùng nhớ có cấu trúc LIFO (Last In - First Out → vào sau – ra trước)
- Ngăn xếp thường dùng để phục vụ cho chương trình con
- Đây 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

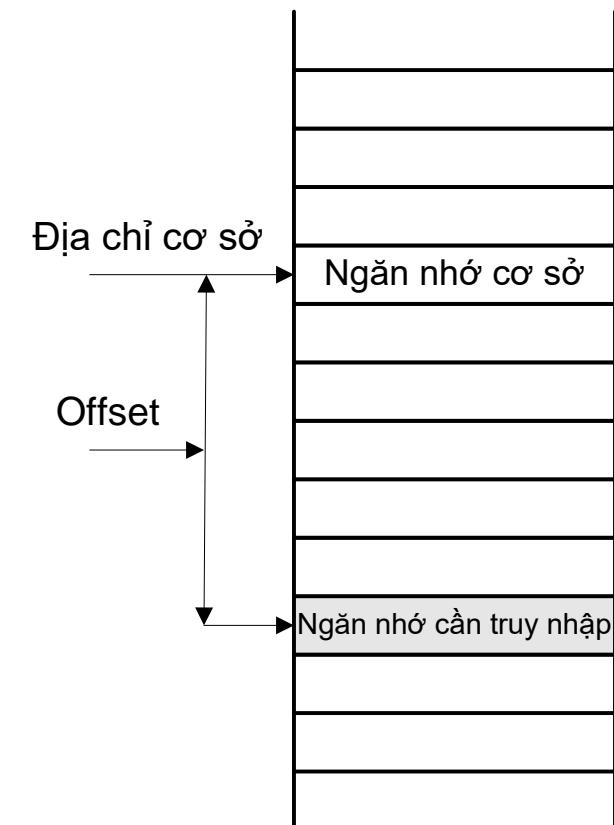
Con trỏ ngăn xếp SP (Stack Pointer)

- Chứa địa chỉ của ngăn nhớ đĩnh ngăn xếp
- Khi cất một thông tin vào ngăn xếp:
 - Nội dung của SP giảm
 - Thông tin được cất vào ngăn nhớ được trả bởi SP
- Khi lấy một thông tin ra khỏi ngăn xếp:
 - Thông tin được đọc từ ngăn nhớ được trả bởi SP
 - Nội dung của SP tăng
- Khi ngăn xếp rỗng, SP trở vào đáy



Thanh ghi cơ sở và thanh ghi chỉ số

- Để truy nhập một ngăn nhớ có thể sử dụng hai tham số:
 - Địa chỉ cơ sở (base address)
 - Phần dịch chuyển địa chỉ (offset)
 - Địa chỉ của ngăn nhớ cần truy nhập = địa chỉ cơ sở + offset
- Có thể sử dụng các thanh ghi để quản lý các tham số này:
 - Thanh ghi cơ sở: chứa địa chỉ cơ sở
 - Thanh ghi chỉ số: chứa phần dịch chuyển địa chỉ



Các thanh ghi dữ liệu

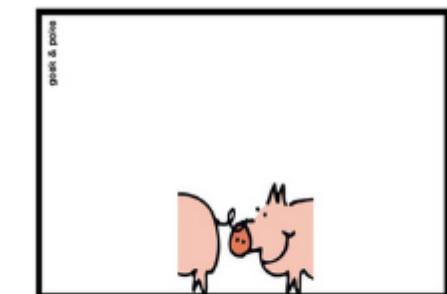
- Chứa các dữ liệu tạm thời hoặc các kết quả trung gian
- Cần có nhiều thanh ghi dữ liệu
- Các thanh ghi số nguyên: 8, 16, 32, 64 bit
- Các thanh ghi số dấu phẩy động

Thanh ghi trạng thái (Status Register)

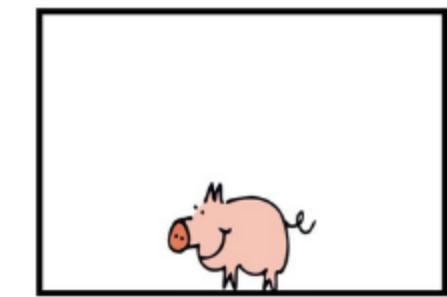
- Được sử dụng trên một số kiến trúc cụ thể
- Còn gọi là thanh ghi cờ (Flag Register)
- Chứa các thông tin trạng thái của CPU
 - Các cờ phép toán: báo hiệu trạng thái của kết quả phép toán
 - Các cờ điều khiển: biểu thị trạng thái điều khiển của CPU

2. Thứ tự lưu trữ các byte trong bộ nhớ chính

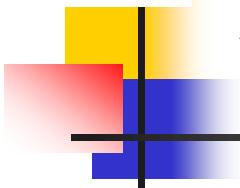
- Bộ nhớ chính thường đánh địa chỉ theo byte
- Hai cách lưu trữ thông tin nhiều byte:
 - **Đầu nhỏ (Little-endian)**: Byte có ý nghĩa thấp được lưu trữ ở ngăn nhớ có địa chỉ nhỏ, byte có ý nghĩa cao được lưu trữ ở ngăn nhớ có địa chỉ lớn.
 - **Đầu to (Big-endian)**: Byte có ý nghĩa cao được lưu trữ ở ngăn nhớ có địa chỉ nhỏ, byte có ý nghĩa thấp được lưu trữ ở ngăn nhớ có địa chỉ lớn.



LITTLE-ENDIAN

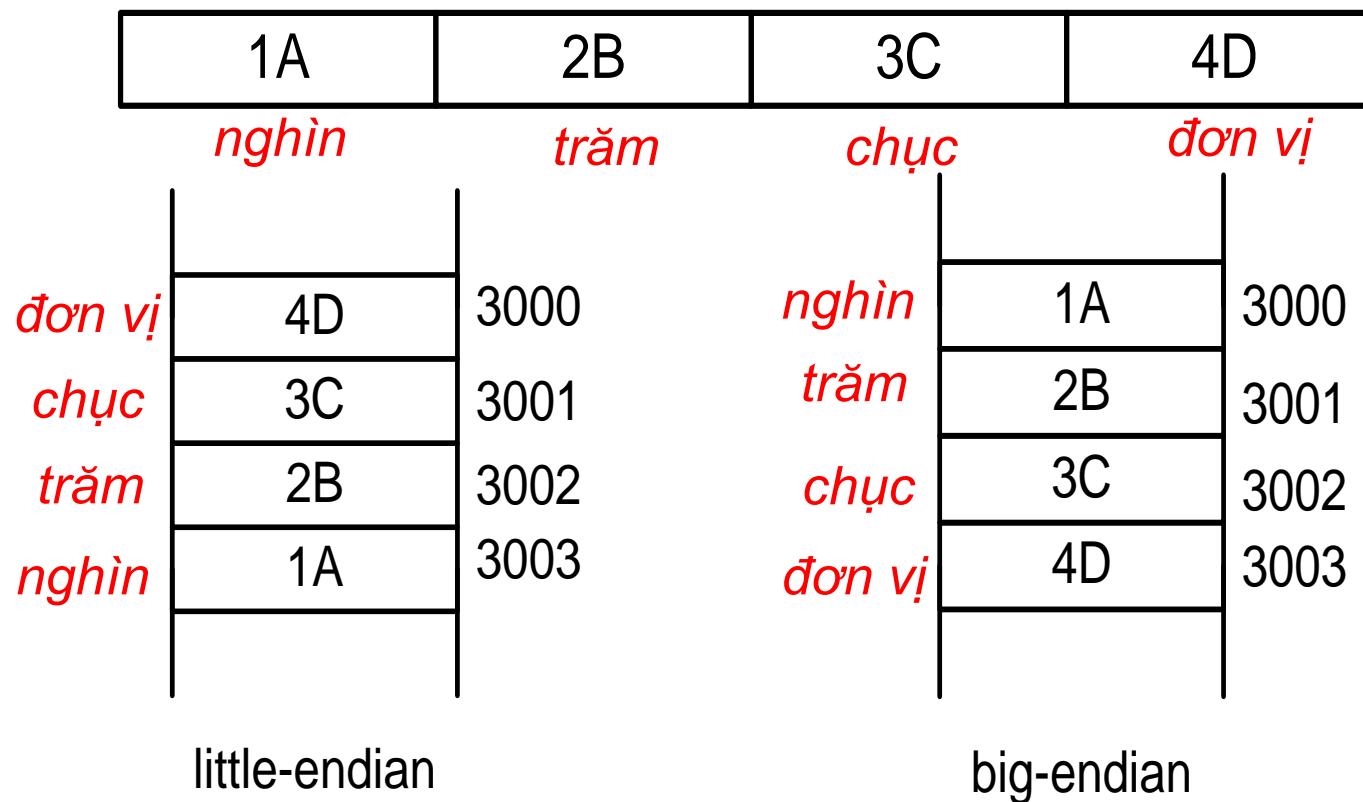


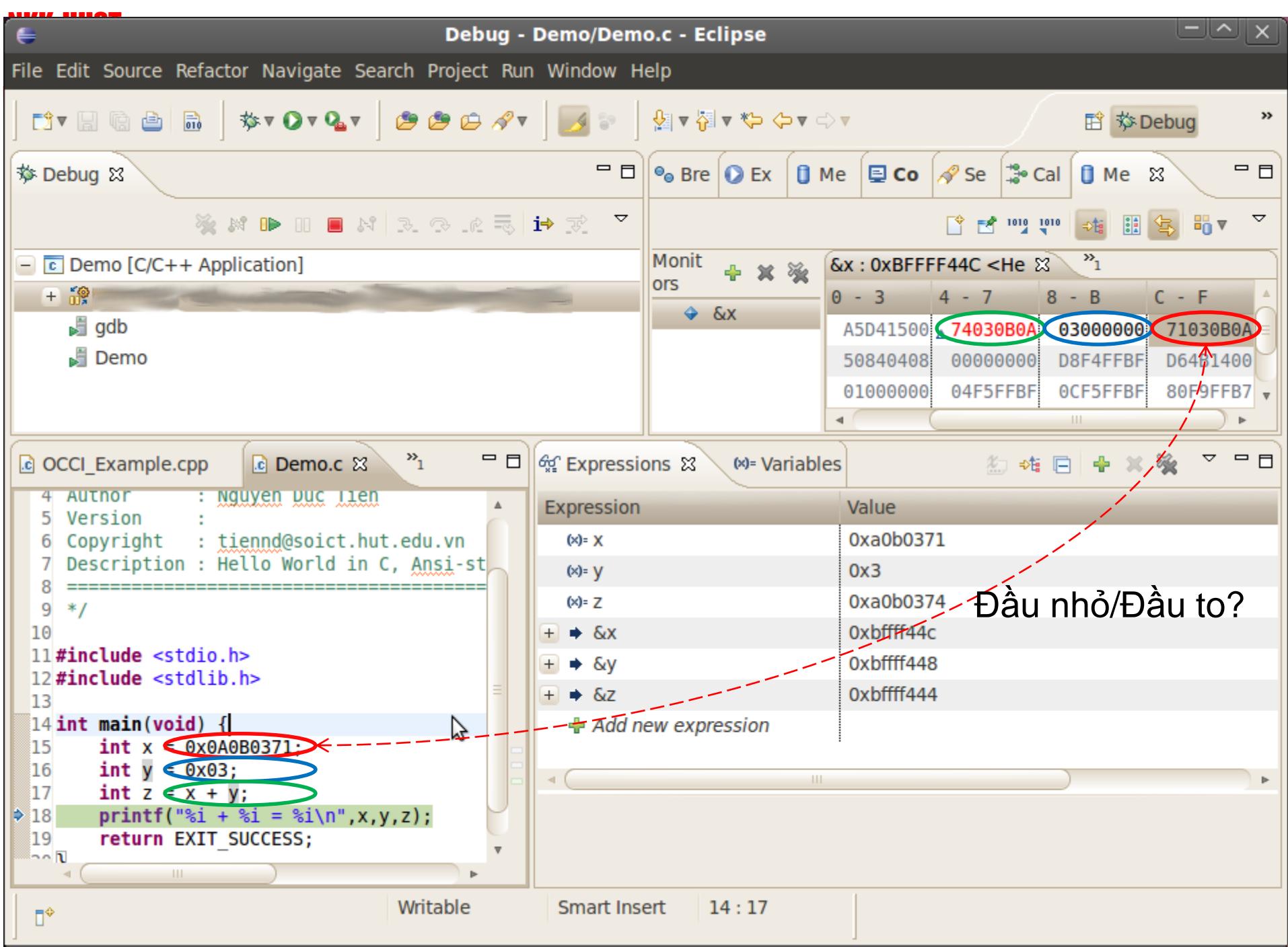
BIG-ENDIAN



Ví dụ lưu trữ dữ liệu 32-bit

0001 1010 0010 1011 0011 1100 0100 1101







- Tương Ứng C – Asm

Demo.cpp Disassembly

```

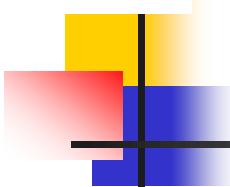
File Edit View Terminal Help

#include <stdio.h>
#include <stdlib.h>

int main(void) {
    80483e4:      55                      push   %ebp
    80483e5:      89 e5                  mov    %esp,%ebp
    80483e7:      83 e4 f0              and    $0xffffffff0,%esp
    80483ea:      83 ec 20              sub    $0x20,%esp
    int x = 0xA0B0371;
    80483ed:      c7 44 24 1c 71 03 0b  movl   $0xa0b0371,0x1c(%esp)
    80483f4:      0a
    int y = 0x03;
    80483f5:      c7 44 24 18 03 00 00  movl   $0x3,0x18(%esp)
    80483fc:      00
    int z = x + y;
    80483fd:      8b 44 24 18          mov    0x18(%esp),%eax
    8048401:      8b 54 24 1c          mov    0x1c(%esp),%edx
    8048405:      8d 04 02          lea    (%edx,%eax,1),%eax
    8048408:      89 44 24 14          mov    %eax,0x14(%esp)
    printf("%i + %i = %i\n",x,y,z);
    804840c:      b8 00 85 04 08  mov    $0x8048500,%eax
    8048411:      8b 54 24 14  mov    0x14(%esp),%edx
    8048415:      89 54 24 0c  mov    %edx,0xc(%esp)
    8048419:      8b 54 24 18  mov    0x18(%esp),%edx
    804841d:      89 54 24 08  mov    %edx,0x8(%esp)
    8048421:      8b 54 24 1c  mov    0x1c(%esp),%edx
    8048425:      89 54 24 04  mov    %edx,0x4(%esp)
    8048429:      89 04 24  mov    %eax,(%esp)
    804842c:      e8 eb fe ff ff  call   804831c <printf@plt>
    return EXIT_SUCCESS;
    8048431:      b8 00 00 00 00  mov    $0x0,%eax
}
8048436:      c9                      leave

```

Linux cmd: \$ objdump -hS



Lưu trữ của các bộ xử lý điển hình

- Intel x86: little-endian
- Motorola 680x0, MIPS, SunSPARC: big-endian
- Power PC, Itanium: bi-endian

3. Giới thiệu chung về tập lệnh

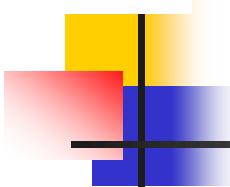
- Mỗi bộ xử lý có một tập lệnh xác định
- Tập lệnh thường có hàng chục đến hàng trăm lệnh
- Mỗi lệnh là một chuỗi số nhị phân mà bộ xử lý hiểu được để thực hiện một thao tác xác định.
- Các lệnh được mô tả bằng các ký hiệu gợi nhớ dạng text → chính là các lệnh của hợp ngữ (assembly language)

Các thành phần của lệnh máy

Mã thao tác

Địa chỉ của các toán hạng

- Mã thao tác (operation code → opcode): mã hóa cho thao tác mà bộ xử lý phải thực hiện
- Địa chỉ toán hạng: chỉ ra nơi chứa các toán hạng mà thao tác sẽ tác động
 - Toán hạng nguồn (source operand): dữ liệu vào của thao tác
 - Toán hạng đích (destination operand): dữ liệu ra của thao tác

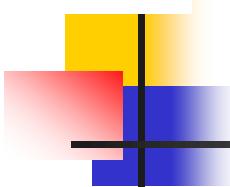


Các kiểu thao tác thông dụng của tập lệnh

- Các lệnh chuyển dữ liệu
- Các lệnh xử lý số học
- Các lệnh xử lý logic
- Các lệnh chuyển điều khiển (rẽ nhánh, nhảy)

Định địa chỉ toán hạng

- Toán hạng của lệnh có thể là:
 - Một giá trị cụ thể nằm ngay trong lệnh
 - Nội dung của thanh ghi
 - Nội dung của ngăn nhớ hoặc cổng vào-ra
- Phương pháp định địa chỉ (addressing modes) là cách thức địa chỉ hóa trong trường địa chỉ của lệnh để xác định nơi chứa toán hạng



Các phương pháp định địa chỉ thông dụng

- Định địa chỉ tức thì
- Định địa chỉ thanh ghi
- Định địa chỉ trực tiếp
- Định địa chỉ gián tiếp qua thanh ghi
- Định địa chỉ dịch chuyển

Định địa chỉ tức thì

Mã thao tác

Toán hạng

- Toán hạng là hằng số nằm ngay trong lệnh
- Chỉ có thể là toán hạng nguồn
- Ví dụ:

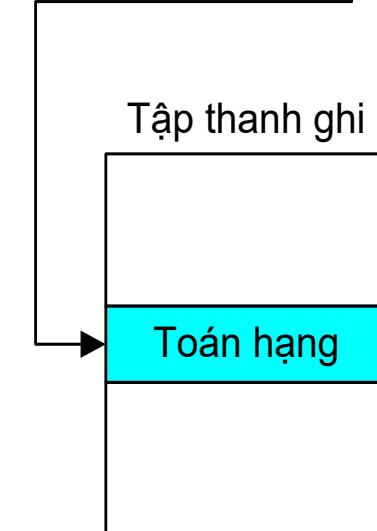
ADD R1, 5 # R1 \leftarrow R1+5

- Không tham chiếu bộ nhớ
- Truy nhập toán hạng rất nhanh
- Dải giá trị của toán hạng bị hạn chế

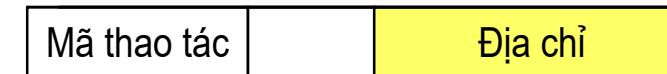
Định địa chỉ thanh ghi

Mã thao tác		Tên thanh ghi
-------------	--	---------------

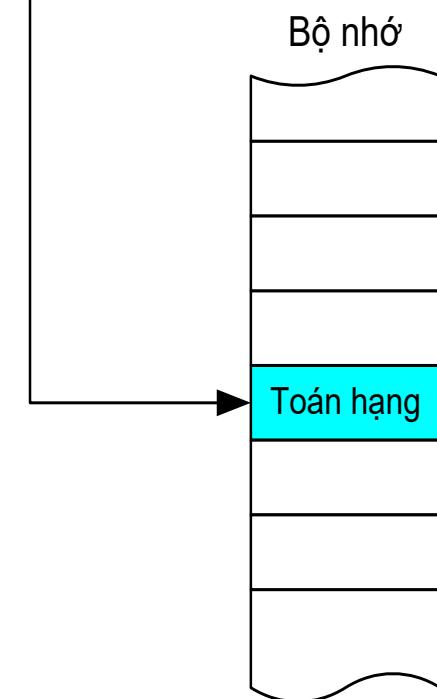
- Toán hạng nằm trong thanh ghi có tên được chỉ ra trong lệnh
- Ví dụ:
ADD R1, R2 # R1 \leftarrow R1+R2
- Số lượng thanh ghi ít \rightarrow Trường địa chỉ toán hạng chỉ cần ít bit
- Không tham chiếu bộ nhớ
- Truy nhập toán hạng nhanh
- Tăng số lượng thanh ghi \rightarrow hiệu quả hơn



Định địa chỉ trực tiếp



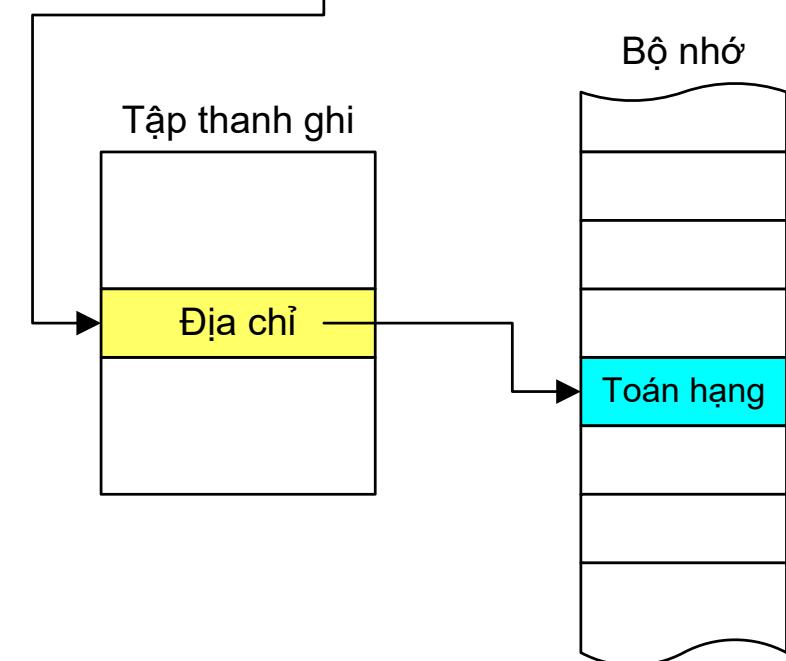
- Toán hạng là ngăn nhớ có địa chỉ được cho trực tiếp trong lệnh
- Ví dụ:
 - ADD R1, A $\#R1 \leftarrow R1 + (A)$
 - Cộng nội dung thanh ghi R1 với nội dung của ngăn nhớ có địa chỉ là A
 - Tìm toán hạng trong bộ nhớ ở địa chỉ A
 - CPU tham chiếu bộ nhớ một lần để truy nhập dữ liệu



Định địa chỉ gián tiếp qua thanh ghi

Mã thao tác		Tên thanh ghi
-------------	--	---------------

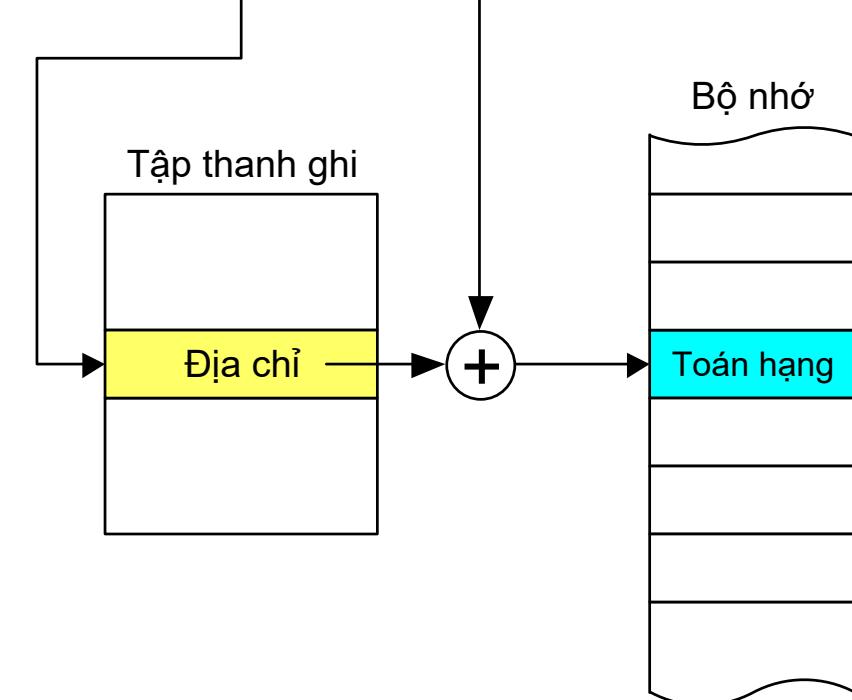
- Toán hạng nằm ở ngăn nhớ có địa chỉ trong thanh ghi
- Trường địa chỉ toán hạng cho biết tên thanh ghi đó
- Thanh ghi có thể là ngầm định
- Thanh ghi này được gọi là thanh ghi con trỏ
- Vùng nhớ có thể được tham chiếu là lớn (2^n), (với n là độ dài của thanh ghi)



Định địa chỉ dịch chuyển



- Để xác định toán hạng, Trường địa chỉ chứa hai thành phần:
 - Tên thanh ghi
 - Hằng số (offset)
- Địa chỉ của toán hạng = nội dung thanh ghi + hằng số
- Thanh ghi có thể được ngầm định



Số lượng địa chỉ toán hạng trong lệnh (1)

■ Ba địa chỉ toán hạng:

- 2 toán hạng nguồn, 1 toán hạng đích
- Phù hợp với dạng: $c = a + b$
- add r1, r2, r3 # $r1 \leftarrow r2 + r3$
- Từ lệnh dài vì phải mã hoá địa chỉ cho cả ba toán hạng
- Được sử dụng trên các bộ xử lý tiên tiến

Số lượng địa chỉ toán hạng trong lệnh (2)

■ Hai địa chỉ toán hạng:

- Một toán hạng vừa là toán hạng nguồn vừa là toán hạng đích; toán hạng còn lại là toán hạng nguồn
- $a = a + b$
- add r1, r2 # $r1 \leftarrow r1 + r2$
- Giá trị cũ của 1 toán hạng nguồn bị mất vì phải chứa kết quả
- Rút gọn độ dài từ lệnh
- Phổ biến

Số lượng địa chỉ toán hạng trong lệnh (3)

■ Một địa chỉ toán hạng:

- Một toán hạng được chỉ ra trong lệnh
- Một toán hạng là ngầm định → thường là thanh ghi (thanh chứa –accumulator)
- add r1 # Acc \leftarrow Acc + r1
- Được sử dụng trên các máy ở các thế hệ trước

Số lượng địa chỉ toán hạng trong lệnh (4)

■ 0 địa chỉ toán hạng:

- Các toán hạng đều được ngầm định
- Sử dụng Stack
- Ví dụ:

push a

push b

add

pop c

có nghĩa là : $c = a+b$

- không thông dụng

4. CISC và RISC

- CISC: Complex Instruction Set Computer:
 - Máy tính với tập lệnh phức tạp
 - Các bộ xử lý truyền thống: 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 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
- Thời gian thực hiện lệnh là một chu kỳ máy
- Các lệnh có độ dài cố định (32 bit)
- Số lượng dạng lệnh ít (≤ 4)
- CPU có tập thanh ghi lớn
- Có ít phương pháp định địa chỉ toán hạng (≤ 4)
- Hỗ trợ các thao tác của ngôn ngữ bậc cao

5.2. Kiến trúc tập lệnh MIPS

5.1. Giới thiệu chung

- MIPS- Microprocessor without Interlocked Pipeline Stages
 - Được phát triển ở đại học Stanford, sau đó được thương mại hóa bởi Công ty MIPS Technologies
 - Năm 2012: MIPS Technologies được bán cho Imagination Technologies (imgtech.com)
 - Kiến trúc RISC
 - Chiếm thị phần lớn trong các sản phẩm nhúng
 - Diễn hình cho nhiều kiến trúc tập lệnh hiện đại
- 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. Phép hợp ngũ 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
\$at	1	assembler temporary
\$v0-\$v1	2-3	procedure return values
\$a0-\$a3	4-7	procedure arguments
\$t0-\$t7	8-15	temporaries
\$s0-\$s7	16-23	saved variables
\$t8-\$t9	24-25	more temporaries
\$k0-\$k1	26-27	OS temporaries
\$gp	28	global pointer
\$sp	29	stack pointer
\$fp	30	frame pointer
\$ra	31	procedure return address

Toán hạng thanh ghi

- Lệnh add, lệnh sub (subtract) chỉ thao tác với toán hạng thanh ghi
 - $\text{add rd, rs, rt} \# (\text{rd}) = (\text{rs}) + (\text{rt})$
 - $\text{sub rd, rs, rt} \# (\text{rd}) = (\text{rs}) - (\text{rt})$
- 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)	Dữ liệu hoặc lệnh	Địa chỉ word (theo Hexa)
byte (8-bit)	0x0000 0000	word (32-bit)	0x0000 0000
byte	0x0000 0001	word	0x0000 0004
byte	0x0000 0002	word	0x0000 0008
byte	0x0000 0003	word	0x0000 000C
byte	0x0000 0004	word	0x0000 0010
byte	0x0000 0005	word	0x0000 0014
byte	0x0000 0006	word	0x0000 0018
byte	0x0000 0007	.	
.		.	
.		.	
.		.	
byte	0xFFFF FFFF	word	0xFFFF FFF4
byte	0xFFFF FFFC	word	0xFFFF FFF8
byte	0xFFFF FFFD	word	0xFFFF FFFC
byte	0xFFFF FFFE		
byte	0xFFFF FFFF		

2^{30} words

2^{32} bytes

Lệnh load và lệnh store

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

$$\text{lw rt, imm(rs)} \quad \# (rt) = \text{mem}[(rs)+imm]$$
 - 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) $\# \text{mem}[(rs)+imm] = (rt)$

- 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

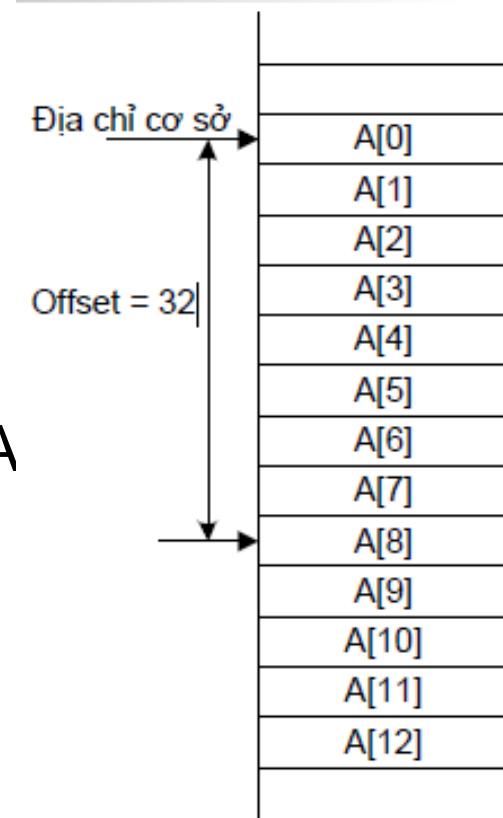
Ví dụ toán hạng bộ nhớ

■ Mã C:

// A là mảng các phần tử 32-bit

g = h + A[8];

- 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];

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

- Mã hợp ngữ MIPS:

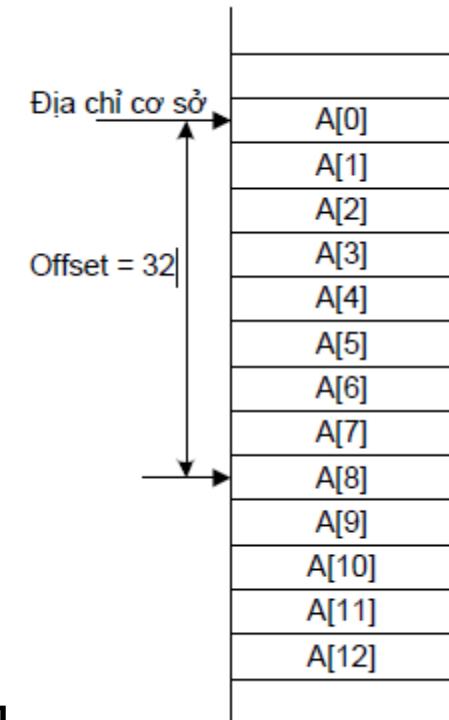
- # Chỉ số 8, do đó offset = 32

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

offset

base register

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

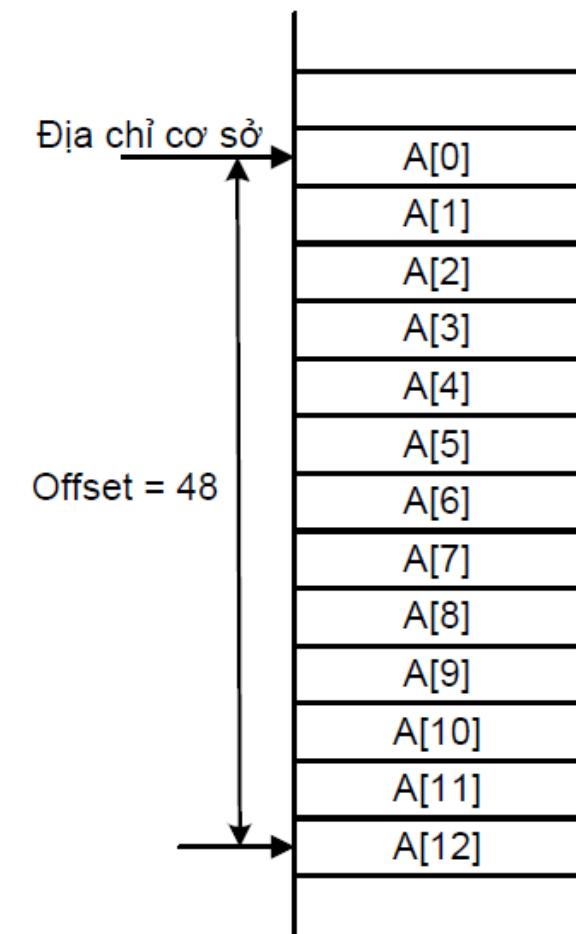


Ví dụ toán hạng bộ nhớ (tiếp)

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)

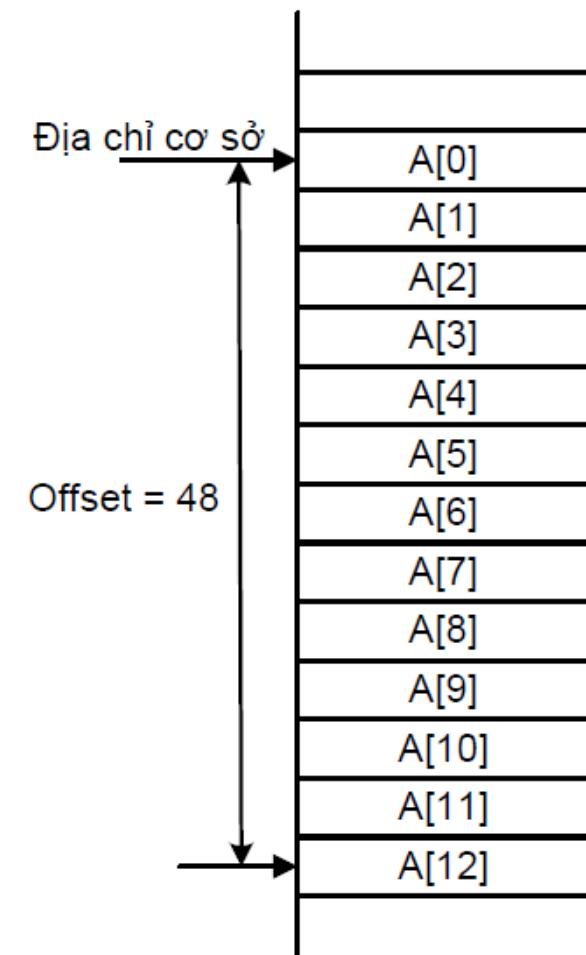
- Mã C:

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

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

- Mã hợp ngữ MIPS:

<code>lw \$t0, 32(\$s3)</code>	# $\$t0 = A[8]$
<code>add \$t0, \$s2, \$t0</code>	# $t0 = h+A[8]$
<code>sw \$t0, 48(\$s3)</code>	# $A[12]=h+A[8]$



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 \leftarrow \$s3+4

- Không có lệnh trừ (subi) với giá trị tức thì

- Sử dụng hằng số âm để thực hiện phép trừ

addi \$s2, \$s1, -1 # \$s2 \leftarrow \$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**, **subi** dành cho số nguyên có dấu
- Số nguyên không dấu:
 - Với n bit, dải biểu diễn: $[0, 2^n - 1]$
 - Các lệnh **addu**, **subu** dành cho số nguyên không dấu
- Qui ước biểu diễn hàng 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
 - \$t0 – \$t7 là các thanh ghi 8 – 15
 - \$t8 – \$t9 là các thanh ghi 24 – 25
 - \$s0 – \$s7 là các thanh ghi 16 – 23

Các dạng lệnh của MIPS

Lệnh kiểu R

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

Lệnh kiểu I

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

Lệnh kiểu J

op	address
6 bits	26 bits

Lệnh dạng R (Register)



- 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 (extends opcode): 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 dạng I (Immediate)

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

- Dùng cho các lệnh số học 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 # $(rt) = (rs) + \text{SignExtImm}$

lw rt, imm(rs) # $(rt) = \text{mem}[(rs) + \text{SignExtImm}]$

sw rt, imm(rs) # $\text{mem}[(rs) + \text{SignExtImm}] = (rt)$

(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): opcode = 000010
 - jal (jump and link): opcode = 000011

J-Type

op	addr
6 bits	26 bits

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

5.4.1. Các lệnh logic

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

Phép toán logic	Toán tử trong C	Toán tử trong Java	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

and \$s3, \$s1, \$s2

\$s3								
------	--	--	--	--	--	--	--	--

or \$s4, \$s1, \$s2

\$s4								
------	--	--	--	--	--	--	--	--

xor \$s5, \$s1, \$s2

\$s5								
------	--	--	--	--	--	--	--	--

nor \$s6, \$s1, \$s2

\$s6								
------	--	--	--	--	--	--	--	--

Kết quả thanh ghi đích

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

and \$s3, \$s1, \$s2

Kết quả thanh ghi đích

\$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

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

andi \$s2,\$s1,0xFA34

\$s2

--	--	--	--	--	--	--	--	--

Kết quả thanh ghi đích

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 khác 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
 - $a \text{ NOR } b == \text{NOT} (a \text{ OR } b)$

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

Lệnh logic dịch bit

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

- *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 ngũ:

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	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 slr

- Lệnh hợp ngũ:

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]
------	------	------	------	------	------	------	------	------	----------------

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

5.4.2 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 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
------	------	------	------	------	------	------	------

Ví dụ khởi tạo thanh ghi 32-bit

- Nạp vào các 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

\$s0	0010	0001	1010	0000	0000	0000	0000	0000
	0000	0000	0000	0000	0100	0000	0011	1011

or

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

Các lệnh rẽ nhánh và lệnh nhảy

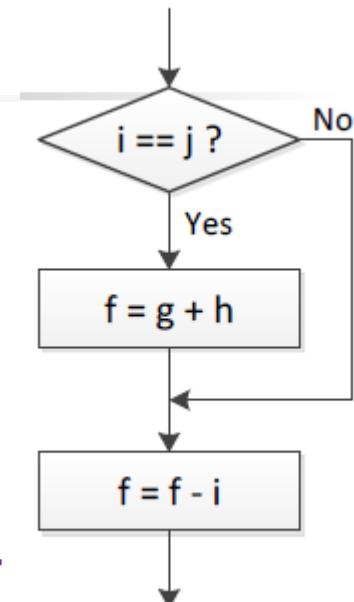
- Các lệnh rẽ nhánh: beq, bne
 - Rẽ nhánh đến lệnh được đánh nhãn nếu điều kiện là đúng, ngược lại, thực hiện tuần tự
 - **beq rs, rt, L1**
 - branch on equal
 - nếu ($rs == rt$) rẽ nhánh đến lệnh ở nhãn L1;
 - **bne rs, rt, L1**
 - branch on not equal
 - nếu ($rs != rt$) rẽ nhánh đến lệnh ở nhãn L1;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

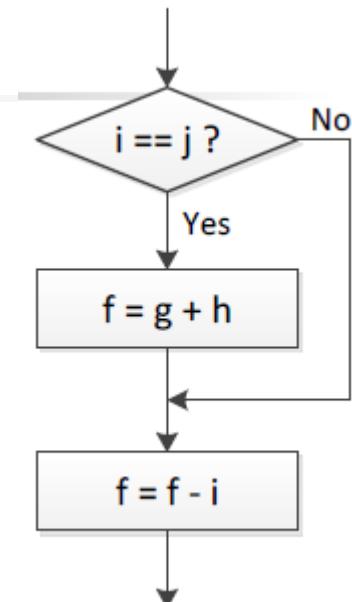


Dịch câu lệnh If

- Mã C:

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

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



- Mã 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
```

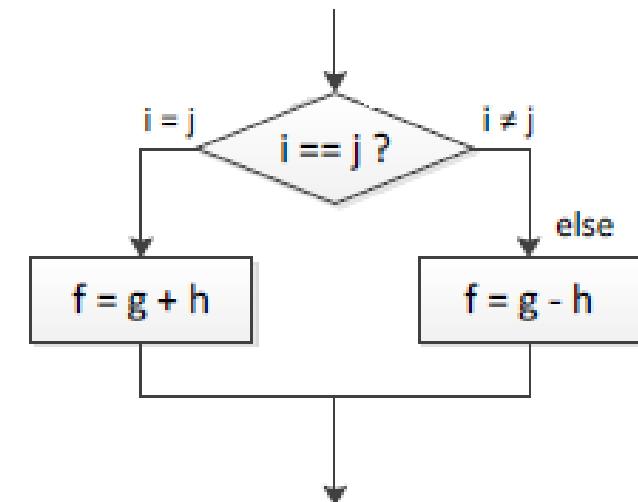
```
L1: sub $s0, $s0, $s3    # f=f-i
```

Dịch câu lệnh If/else

- Mã C:

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

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



Dịch câu lệnh If/else

- Mã C:

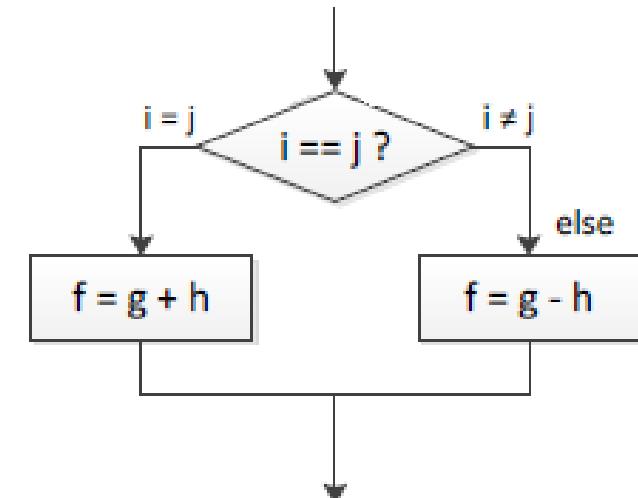
```
if (i==j)
    f = g+h;
```

```
else
    f = g-h;
```

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

- Mã 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 khác
	# thì $f=g-h$



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

bne \$s0, \$t0, **case50**

addi \$s1, \$0, 2

j **done**

case50: addi \$t0, \$0, 50

bne \$s0, \$t0, **case100**

addi \$s1, \$0, 3

j **done**

case100: addi \$t0, \$0, 100

bne \$s0, \$t0, **default**

addi \$s1, \$0, 5

j **done**

default: add \$s1 , \$0, \$0

done:

\$t0 = 20

amount == 20? if not, skip to case50

if so, fee = 2

and break out of case

\$t0 = 50

amount == 50? if not, skip to case100

if so, fee = 3

and break out of case

\$t0 = 100

amount == 100? if not, skip to default

if so, fee = 5

and break out of case

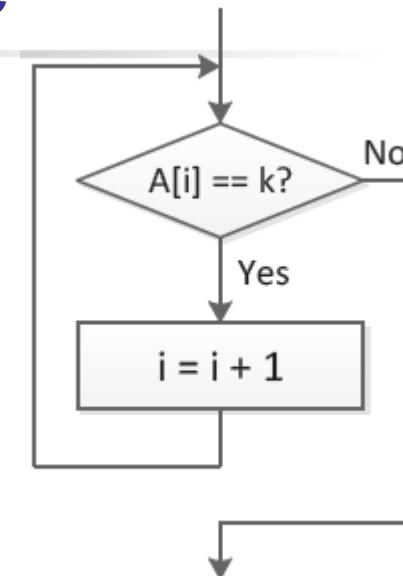
fee = 0

Dịch câu lệnh vòng lặp While

■ Mã C:

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

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

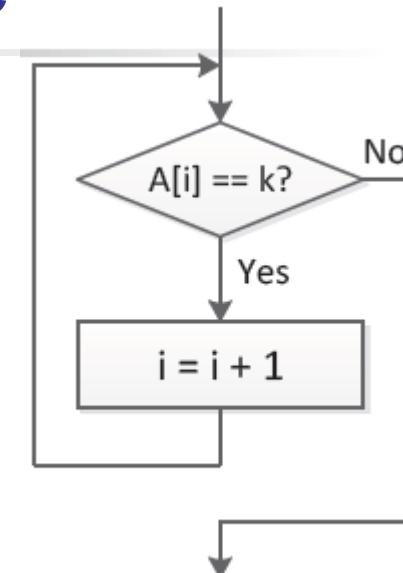


Dịch câu lệnh vòng lặp While

Mã C:

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

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



Mã MIPS được dịch:

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

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 = i+1) {
    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 = i+1) {
    sum = sum + i;
}
```

■ Mã MIPS được dịch:

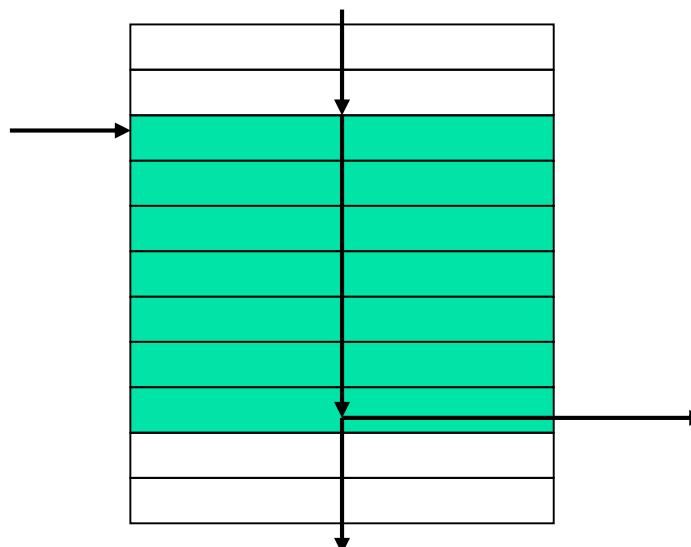
```
# $s0 = i, $s1 = sum
        addi $s1, $0, 0          # sum = 0
        add  $s0, $0, $0          # i = 0
        addi $t0, $0, 10          # $t0 = 10
for:   beq  $s0, $t0, done      # Nếu i = 10, thoát
        add  $s1, $s1, $s0        # sum = sum + i
        addi $s0, $s0, 1          # i = i+1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

done:
```

Khối lệnh cơ sở

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 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 ($rs < \text{constant}$) 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 1111
 - \$s1 = 0000 0000 0000 0000 0000 0000 0000 0000 0001
 - slt \$t0, \$s0, \$s1 # signed
 - $-1 < +1$ $\rightarrow \$t0 = 1$
 - sltu \$t0, \$s0, \$s1 # unsigned
 - $+4,294,967,295 > +1$ $\rightarrow \$t0 = 0$

Ví dụ mã lệnhslt

- Mã C

```
int sum = 0;  
int i;  
for (i=1; i < 101; i = i*2) {  
    sum = sum + i;  
}
```

Ví dụ mã 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

0x12348000	array[0]
0x12348004	array[1]
0x12348008	array[2]
0x1234800C	array[3]
0x12348010	array[4]

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
done: j loop                      # repeat

```

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 (các thanh ghi 4 – 7)
- \$v0, \$v1: giá trị kết quả (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 trả về (thanh ghi 31)

Các lệnh gọi thủ tục

- Gọi thủ tục: jump and link

jal ProcedureLabel

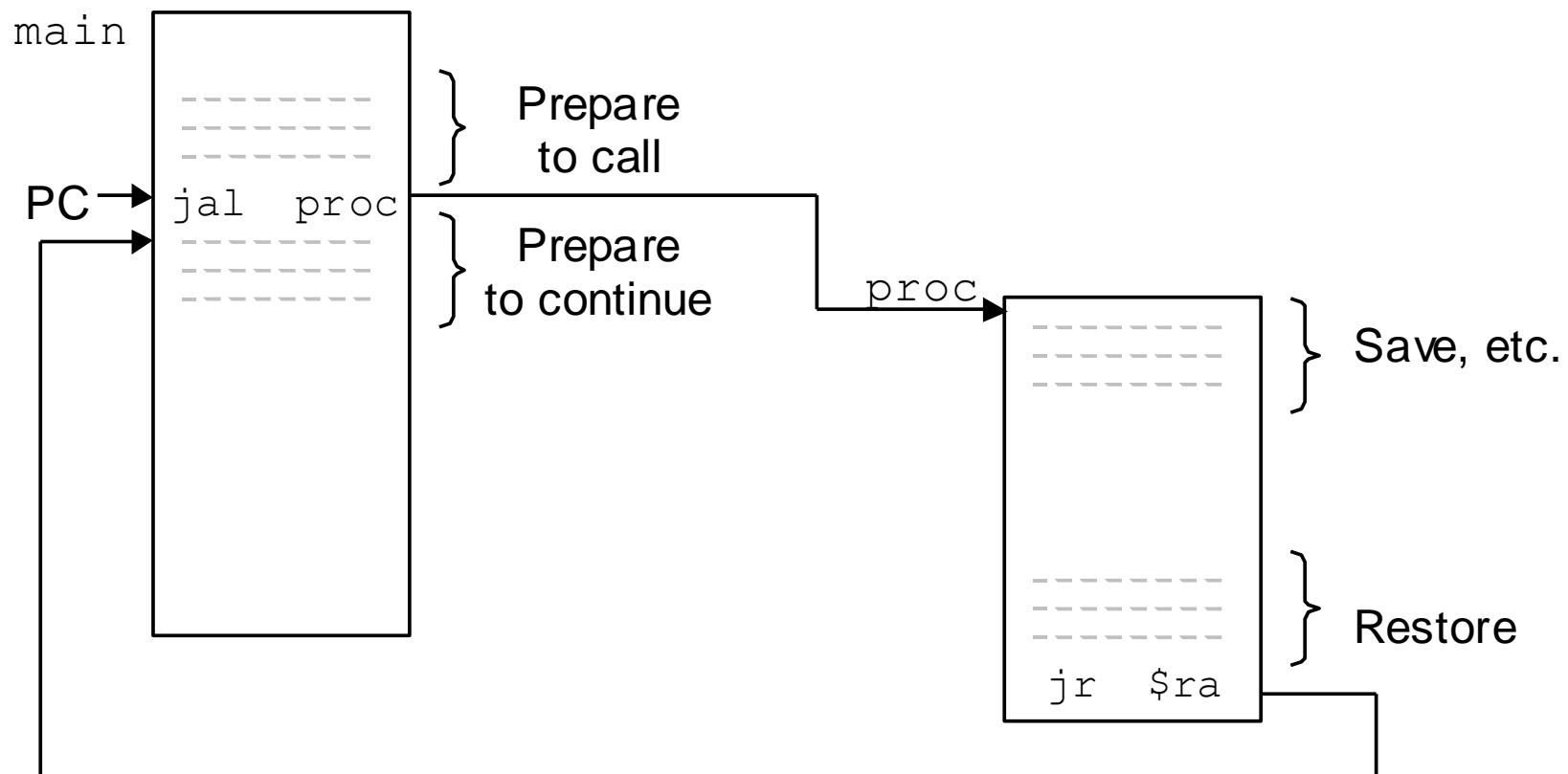
- Địa chỉ của lệnh kế tiếp (địa chỉ trở về) được cất ở thanh ghi \$ra
- Nhảy đến địa chỉ của thủ tục

- Trở về từ thủ tục: jump register

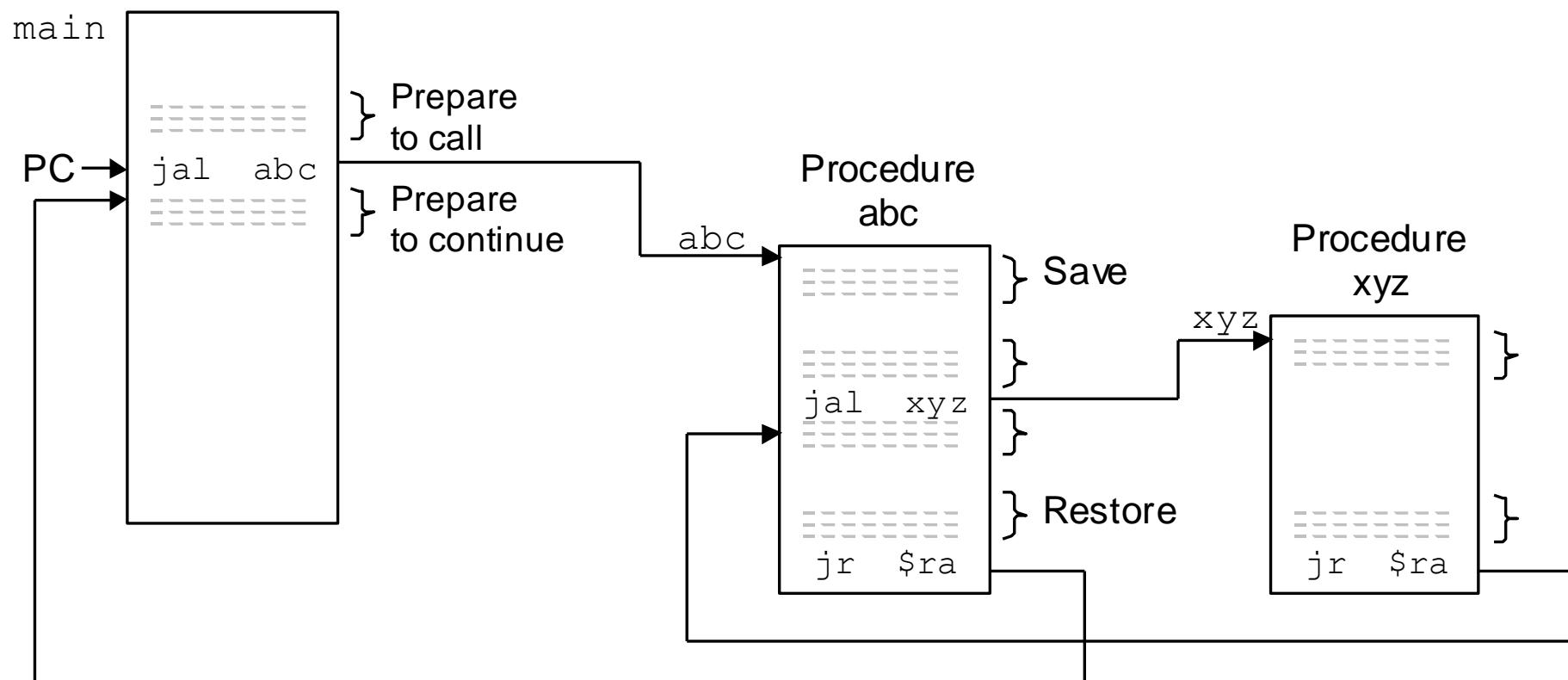
jr \$ra

- 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 sử dụng để chứa các giá trị tạm thời , cũng cần cất trước khi sử dụng.
- 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 trả 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 trả 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 trả về
addi \$sp, \$sp, 8	# lấy 2 mục từ stack
mul \$v0, \$a0, \$v0	# nhân để nhận kết quả
jr \$ra	# và trả về

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);
}
```

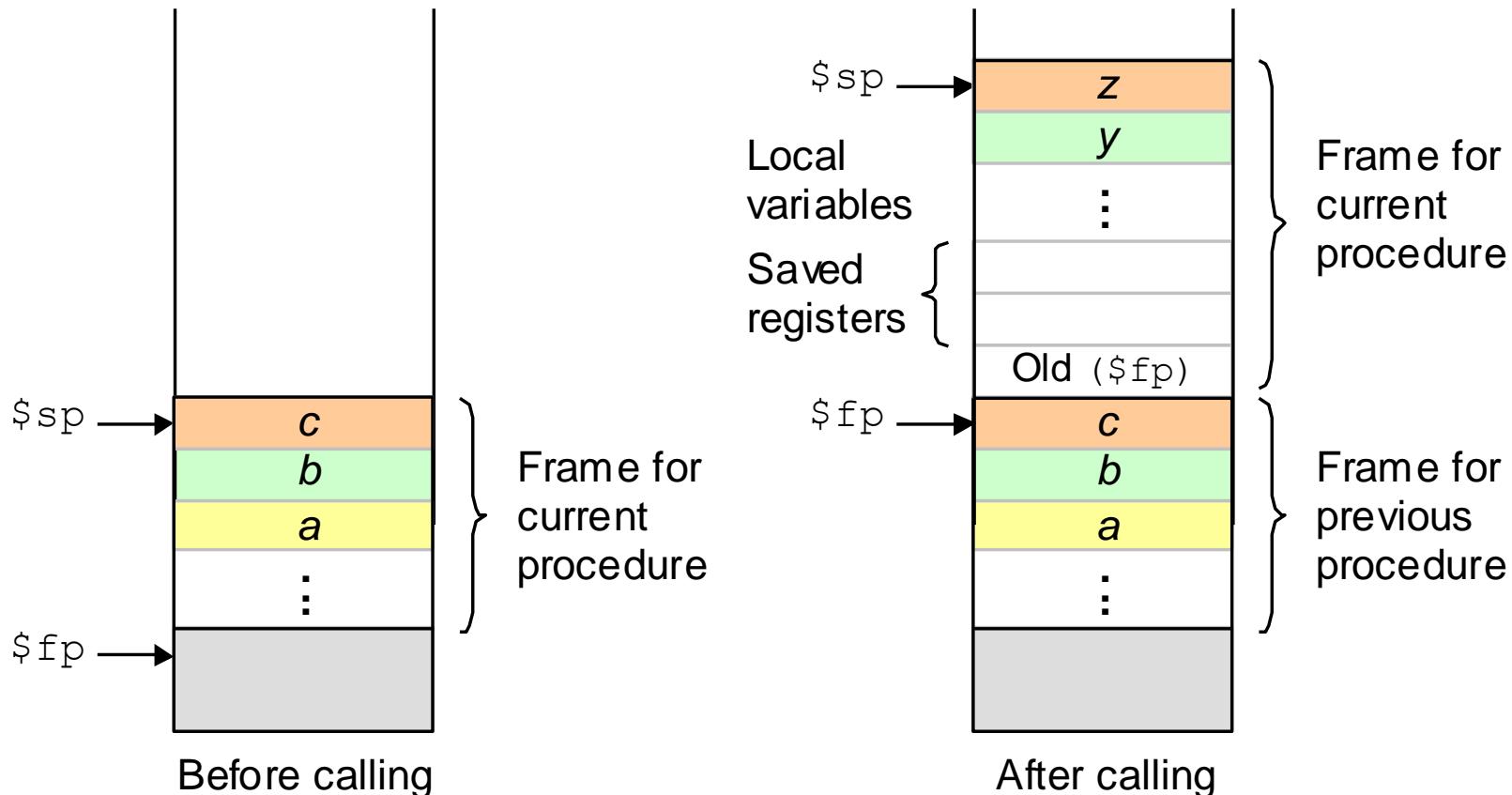
- Tham số n ở \$a0
- Kết quả ở \$v0

Ví dụ Thủ tục cành (tiếp)

- Mã MIPS:

fact:	
	addi \$sp, \$sp, -8 # dành stack cho 2 mục
	sw \$ra, 4(\$sp) # cát địa chỉ trả về
	sw \$a0, 0(\$sp) # cát 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 trả 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 trả về
	addi \$sp, \$sp, 8 # lấy 2 mục từ stack
	mul \$v0, \$a0, \$v0 # nhân để nhận kết quả
	jr \$ra # và trả 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 16-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
- $lb\ rt, \text{offset}(rs)$ $lh\ rt, \text{offset}(rs)$
 - Mở rộng dấu thành 32 bits trong rt
- $lbu\ rt, \text{offset}(rs)$ $lhu\ rt, \text{offset}(rs)$
 - Mở rộng zero thành 32 bits trong rt
- $sb\ rt, \text{offset}(rs)$ $sh\ rt, \text{offset}(rs)$
 - 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

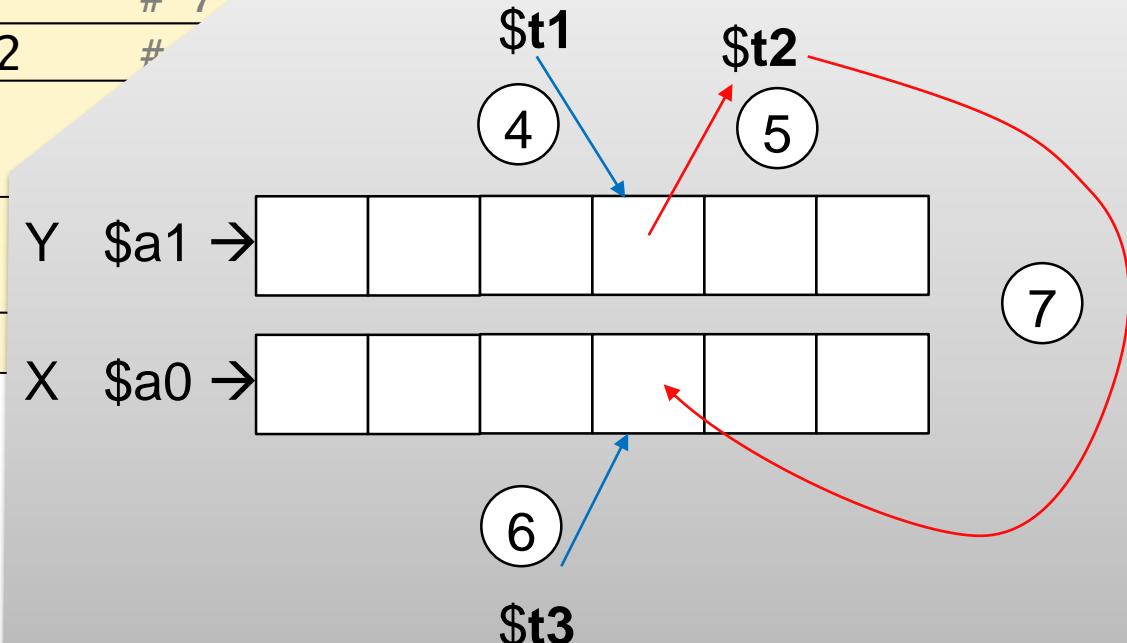
MIPS code:

strcpy:

```

    addi $sp, $sp, -4      # 1. adjust stack for 1 item
    sw   $s0, 0($sp)       # 2. save $s0
    add  $s0, $zero, $zero # 3. i = 0
L1: add  $t1, $s0, $a1    # 4. addr of y[i] in $t1
    lbu $t2, 0($t1)        # 5. $t2 = y[i]
    add  $t3, $s0, $a0    # 6. addr of x[i] in $t3
    sb   $t2, 0($t3)        # 7
    beq $t2, $zero, L2    #
    addi $s0, $s0, 1
    j    L1
L2: lw   $s0, 0($sp)
    addi $sp, $sp, 4
    jr  $ra

```



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 # Thanh ghi rd \leftarrow Hi
 - mflo rd # Thanh ghi rd \leftarrow Lo

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, offset
- 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 = $PC + \text{hằng số} \times 4$
 - 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

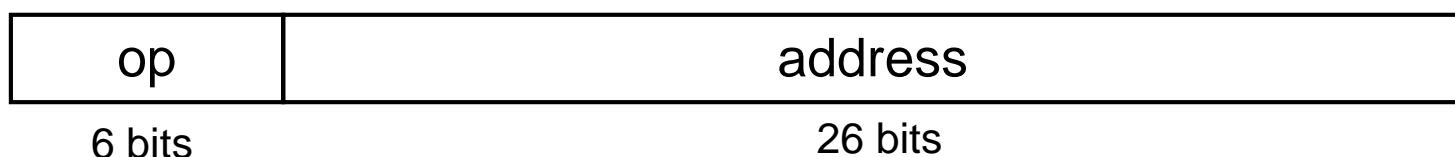
Lệnh mã máy

beq	000100	10000	10001	0000	0000	0000	0110
-----	--------	-------	-------	------	------	------	------

bne	000101	10000	10001	0000	0000	0000	0110
-----	--------	-------	-------	------	------	------	------

Địa chỉ hóa cho lệnh Jump

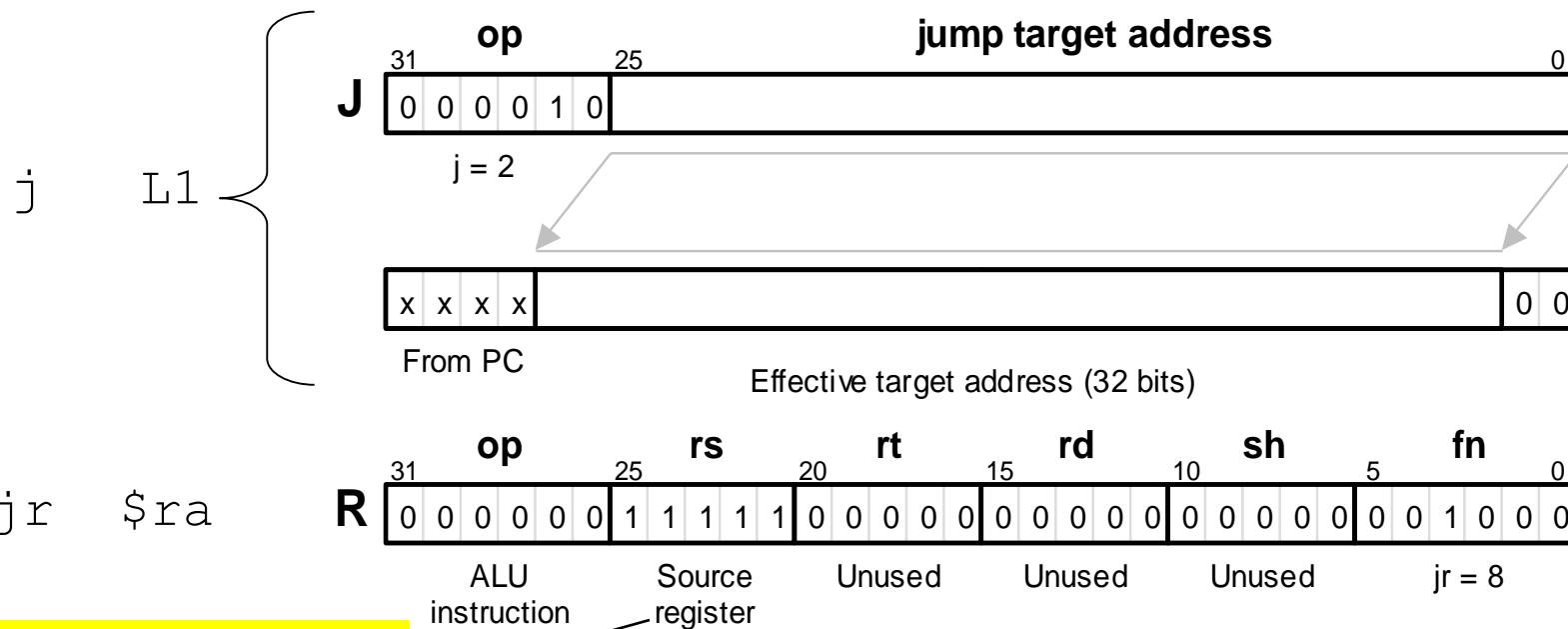
- Đích của lệnh Jump (j và ja1) có thể là bất kỳ chỗ nào trong chương trình
 - Cần mã hóa đầy đủ địa chỉ trong lệnh



- Định địa chỉ nhảy (giả) trực tiếp
(Pseudo)Direct jump addressing
 - Địa chỉ đích = $PC_{31\dots28} : (\text{address} \times 4)$
 $= PC_{31\dots28} : \text{address}_{27..2} : 00_{1..0}$

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 may hold a return address



- \$ra là thanh ghi \$31
(return address)

Ví dụ mã hóa lệnh

Loop:	sll	\$t1, \$s3, 2	8000
	add	\$t1, \$t1, \$s6	8004
	lw	\$t0, 0(\$t1)	8008
	bne	\$t0, \$s5, Exit	8012
	addi	\$s3, \$s3, 1	8016
	j	Loop	8020
Exit:	...		8024

0	0	19	9	2	0
0	9	22	9	0	32
35	9	8		0	
5	8	21			2
8	19	19		1	
2				0x2000	

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)

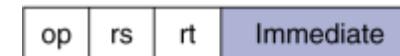
...

L1:

Tóm tắt về các phương pháp định địa chỉ

1. Định địa chỉ tức thì

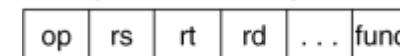
1. Immediate addressing



addi \$t1,\$t2, 6

2. Định địa chỉ thanh ghi

2. Register addressing

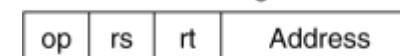


Registers

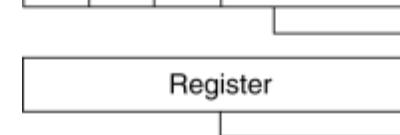
Register

3. Định địa chỉ cơ sở

3. Base addressing



Memory

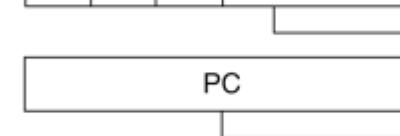


4. Định địa chỉ tương đối với PC

4. PC-relative addressing



Memory

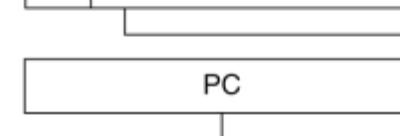


5. Định địa chỉ giả trực tiếp

5. Pseudodirect addressing

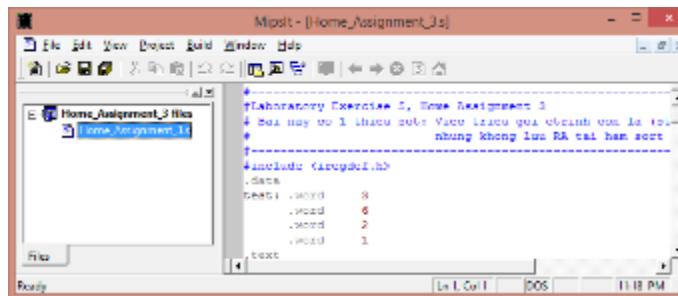


Memory

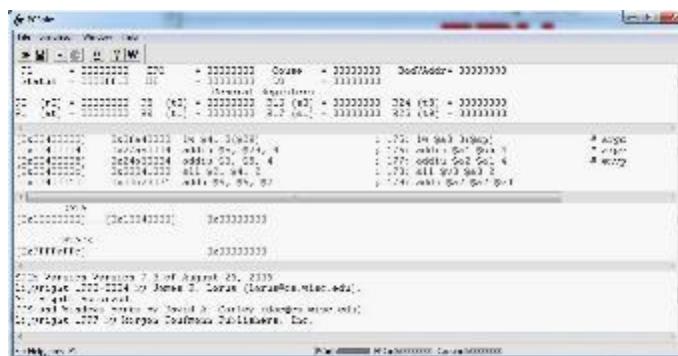


5.6. Dịch và chạy chương trình hợp ngữ

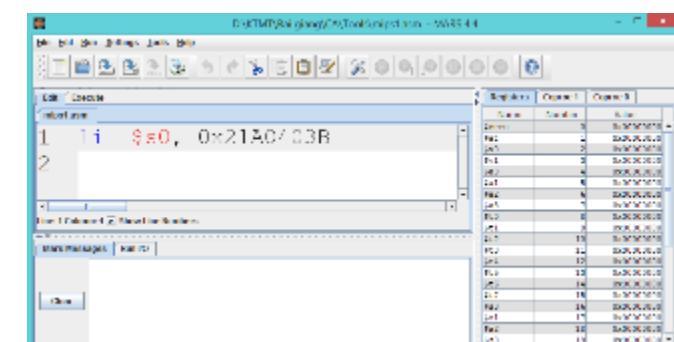
■ Các phần mềm lập trình hợp ngữ MIPS:



MipsIT



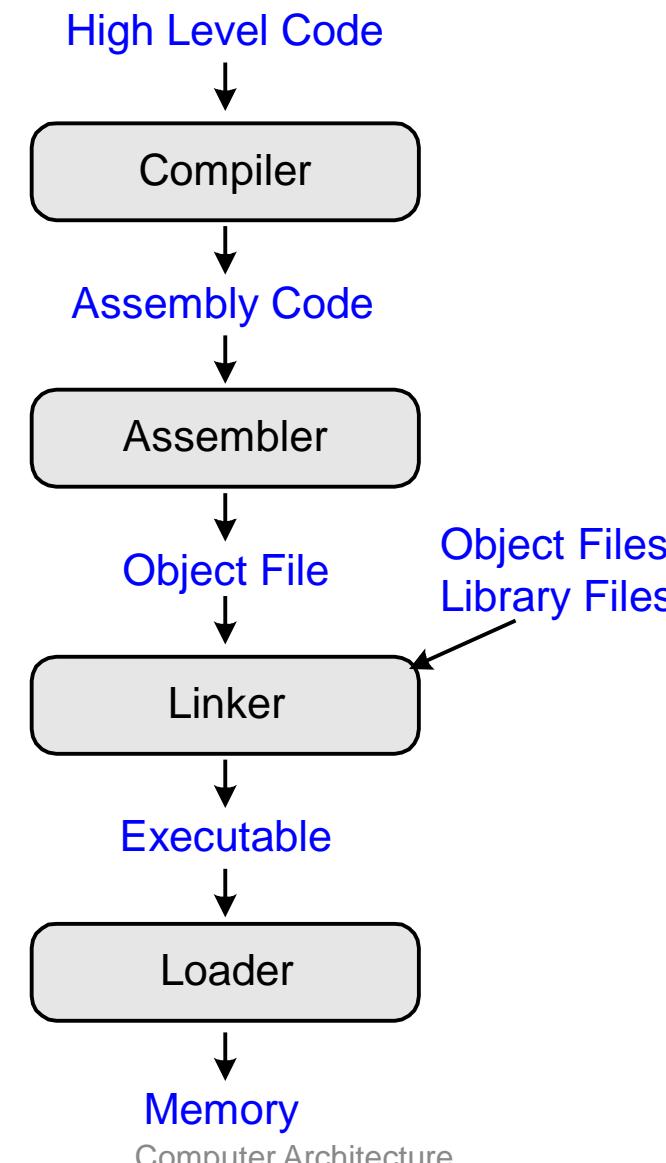
PCSpim



MARS

■ MIPS Reference Data

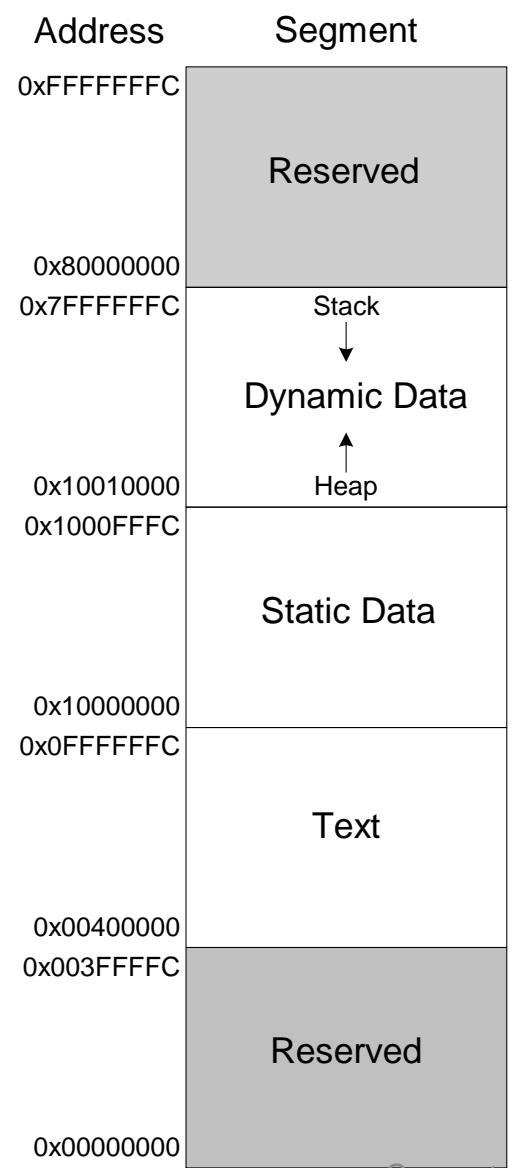
Dịch và chạy ứng dụng



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:
 - $2^{32} = 4$ gigabytes (4 GiB)
 - Từ địa chỉ 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

```
.data
f:
g:
y:
.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
    jr   $ra                  # return to OS
sum:
    add  $v0, $a0, $a1        # $v0 = a + b
    jr   $ra                  # return
```

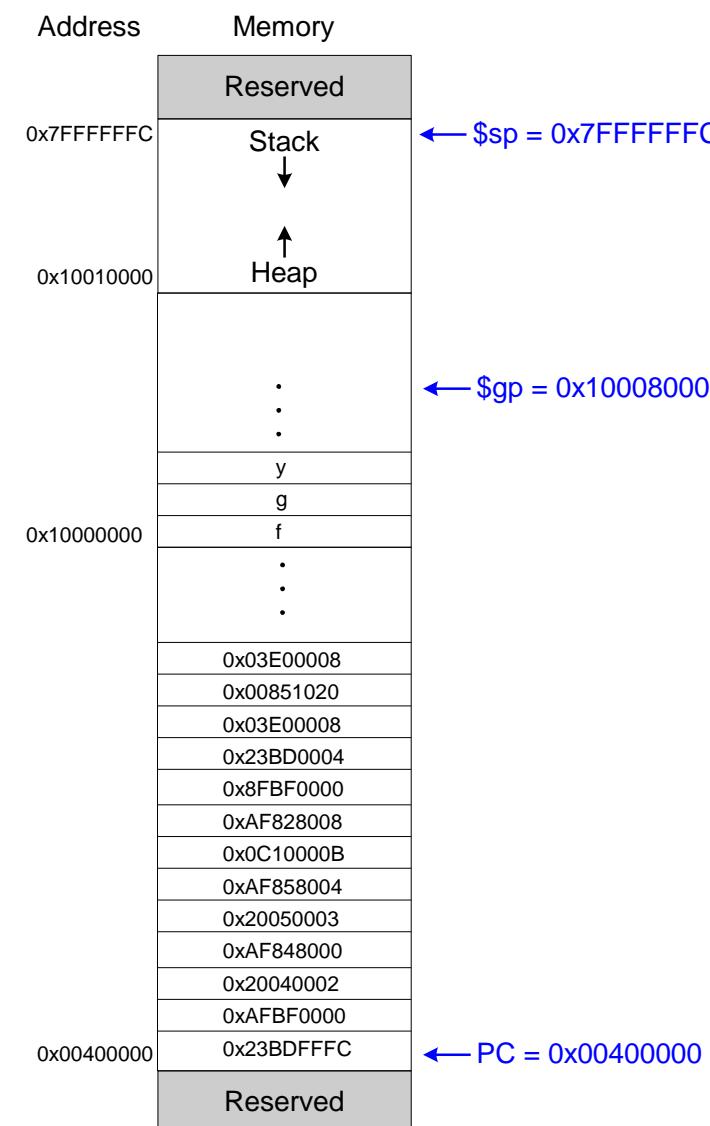
Bảng ký hiệu

Ký hiệu	Địa chỉ
f	0x10000000
g	0x10000004
y	0x10000008
main	0x00400000
sum	0x0040002C

Chương trình thực thi

Executable file header	Text Size	Data Size	
	0x34 (52 bytes)	0xC (12 bytes)	
Text segment	Address	Instruction	
	0x00400000	0x23BDFFFFC	addi \$sp, \$sp, -4
	0x00400004	0xAFBF0000	sw \$ra, 0 (\$sp)
	0x00400008	0x20040002	addi \$a0, \$0, 2
	0x0040000C	0xAF848000	sw \$a0, 0x8000 (\$gp)
	0x00400010	0x20050003	addi \$a1, \$0, 3
	0x00400014	0xAF858004	sw \$a1, 0x8004 (\$gp)
	0x00400018	0x0C10000B	jal 0x0040002C
	0x0040001C	0xAF828008	sw \$v0, 0x8008 (\$gp)
	0x00400020	0x8FBF0000	lw \$ra, 0 (\$sp)
	0x00400024	0x23BD0004	addi \$sp, \$sp, -4
	0x00400028	0x03E00008	jr \$ra
	0x0040002C	0x00851020	add \$v0, \$a0, \$a1
	0x00400030	0x03E00008	jr \$ra
Data segment	Address	Data	
	0x10000000	f	
	0x10000004	g	
	0x10000008	y	

Chương trình trong bộ nhớ



Ví dụ lệnh giả (Pseudoinstruction)

Pseudoinstruction	MIPS Instructions
li \$s0, 0x1234AA77	lui \$s0, 0x1234 ori \$s0, 0xAA77
mul \$s0, \$s1, \$s2	mult \$s1, \$s2 mflo \$s0
clear \$t0	add \$t0, \$0, \$0
move \$s1, \$s2	add \$s2, \$s1, \$0
nop	sll \$0, \$0, 0

Phụ lục: Kiến trúc tập lệnh Intel x86(*)

- Sự tiến hóa của các bộ xử lý Intel
 - 8080 (1974): 8-bit microprocessor
 - Accumulator, plus 3 index-register pairs
 - 8086 (1978): 16-bit extension to 8080
 - Complex instruction set (CISC)
 - 8087 (1980): floating-point coprocessor
 - Adds FP instructions and register stack
 - 80286 (1982): 24-bit addresses, MMU
 - Segmented memory mapping and protection
 - 80386 (1985): 32-bit extension (now IA-32)
 - Additional addressing modes and operations
 - Paged memory mapping as well as segments

Kiến trúc tập lệnh Intel x86

- i486 (1989): pipelined, on-chip caches and FPU
 - Compatible competitors: AMD, Cyrix, ...
- Pentium (1993): superscalar, 64-bit datapath
 - Later versions added MMX (Multi-Media eXtension) instructions
 - The infamous FDIV bug
- Pentium Pro (1995), Pentium II (1997)
 - New microarchitecture
- Pentium III (1999)
 - Added SSE (Streaming SIMD Extensions) and associated registers
- Pentium 4 (2001)
 - New microarchitecture
 - Added SSE2 instructions
- Intel Core (2006)
 - Added SSE4 instructions, virtual machine support

Các thanh ghi cơ bản của x86

Name	Use
31	0
EAX	GPR 0
ECX	GPR 1
EDX	GPR 2
EBX	GPR 3
ESP	GPR 4
EBP	GPR 5
ESI	GPR 6
EDI	GPR 7
CS	Code segment pointer
SS	Stack segment pointer (top of stack)
DS	Data segment pointer 0
ES	Data segment pointer 1
FS	Data segment pointer 2
GS	Data segment pointer 3
EIP	Instruction pointer (PC)
EFLAGS	Condition codes

Các phương pháp định địa chỉ cơ bản

- Hai toán hạng của lệnh

Source/dest operand	Second source operand
Register	Register
Register	Immediate
Register	Memory
Memory	Register
Memory	Immediate

- Các phương pháp định địa chỉ

- Address in register
- Address = $R_{base} + \text{displacement}$
- Address = $R_{base} + 2^{\text{scale}} \times R_{index}$ (scale = 0, 1, 2, or 3)
- Address = $R_{base} + 2^{\text{scale}} \times R_{index} + \text{displacement}$

Mã hóa lệnh x86

a. JE EIP + displacement

4	4	8
JE	Condition	Displacement

b. CALL

8	32
CALL	Offset

c. MOV EBX, [EDI + 45]

6	1	1	8	8
MOV	d	w	r/m Postbyte	Displacement

d. PUSH ESI

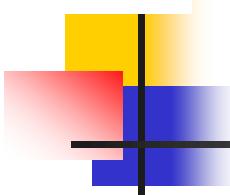
5	3
PUSH	Reg

e. ADD EAX, #6765

4	3	1	32
ADD	Reg	w	Immediate

f. TEST EDX, #42

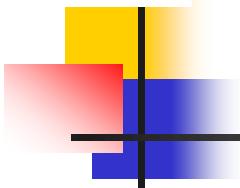
7	1	8	32
TEST	w	Postbyte	Immediate



Hết chương 5

Chương 6 BỘ XỬ LÝ (Processor)

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

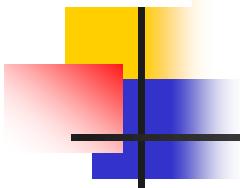
Chương 5. Kiến trúc tập lệnh

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

- 6.1. Tổ chức của bộ xử lý
- 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**

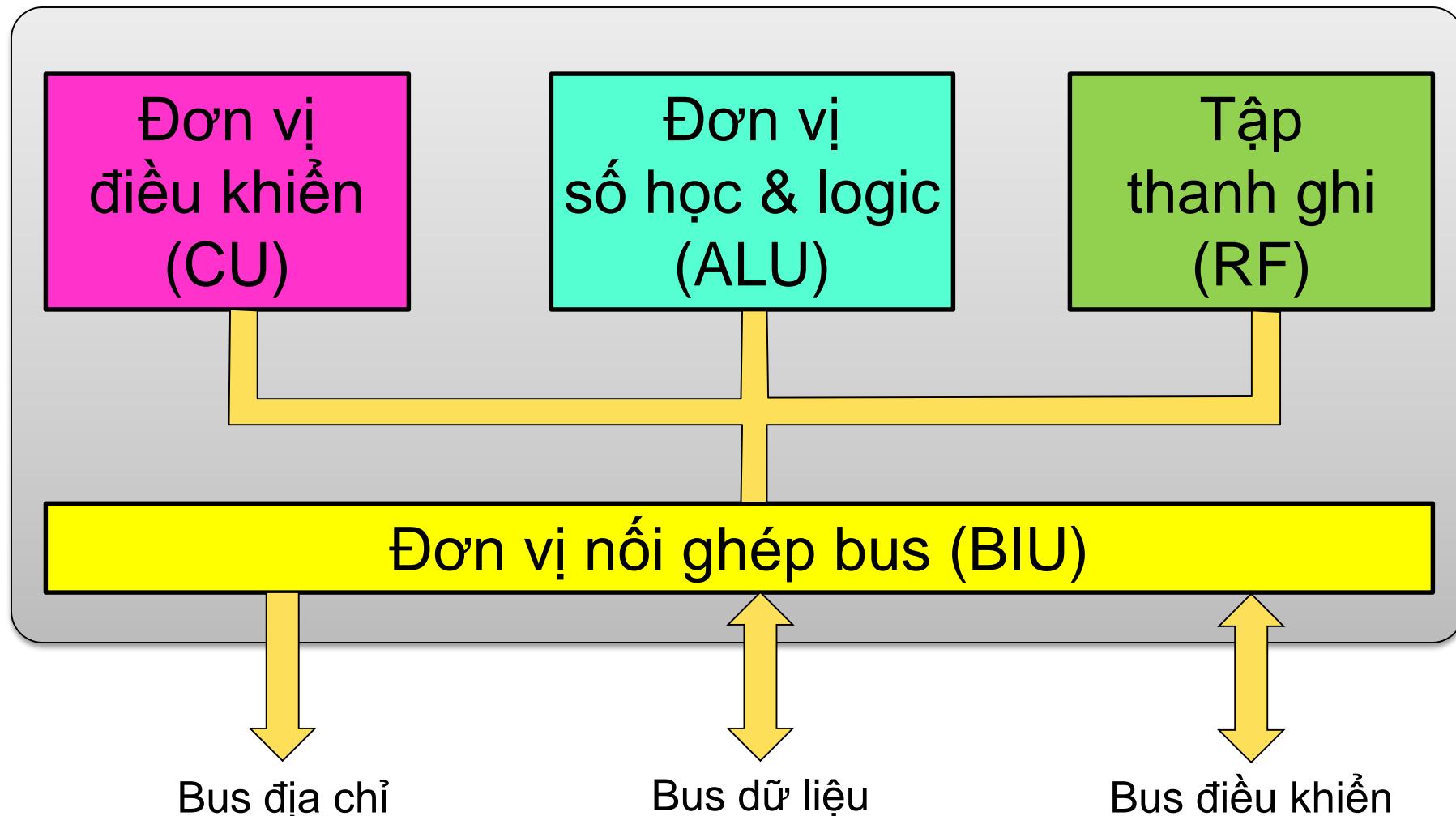
6.1. Tổ chức của CPU

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

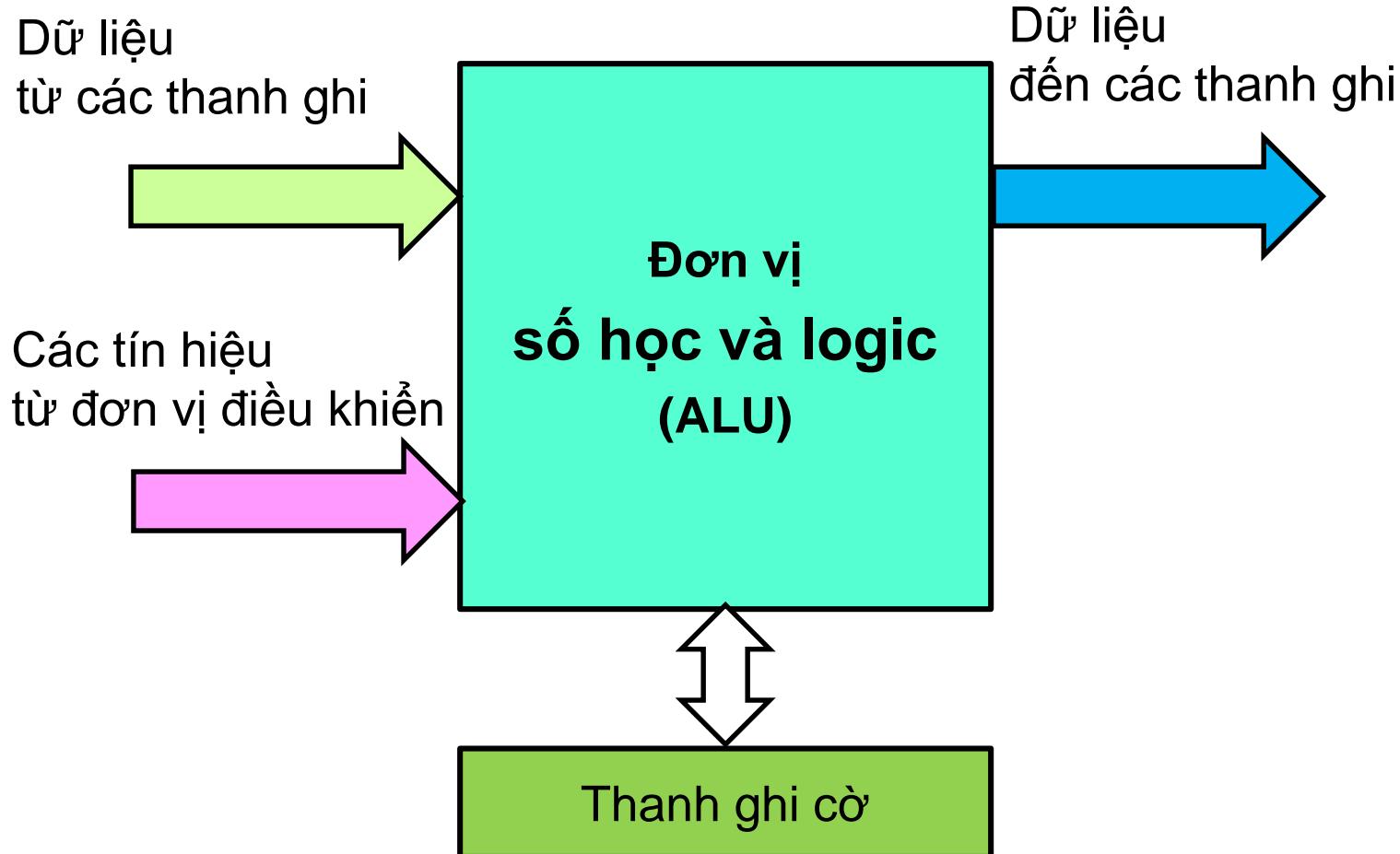
Sơ đồ cấu trúc cơ bản của CPU



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, tăng, giảm, đảo dấu
 - Logic: AND, OR, XOR, NOT, phép dịch bit.

Mô hình kết nối ALU

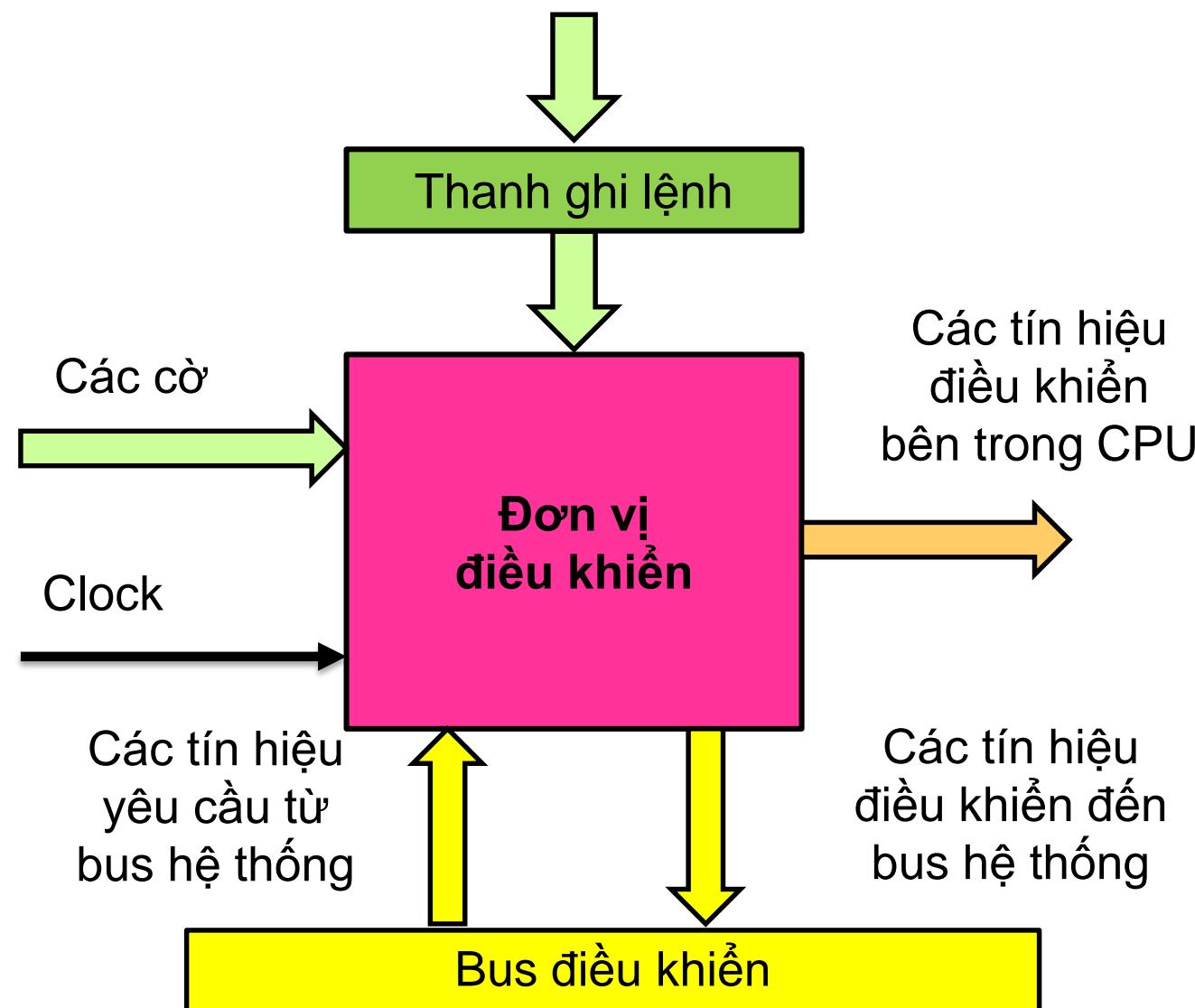


3. Đơn vị điều khiển

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.
- Mã 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

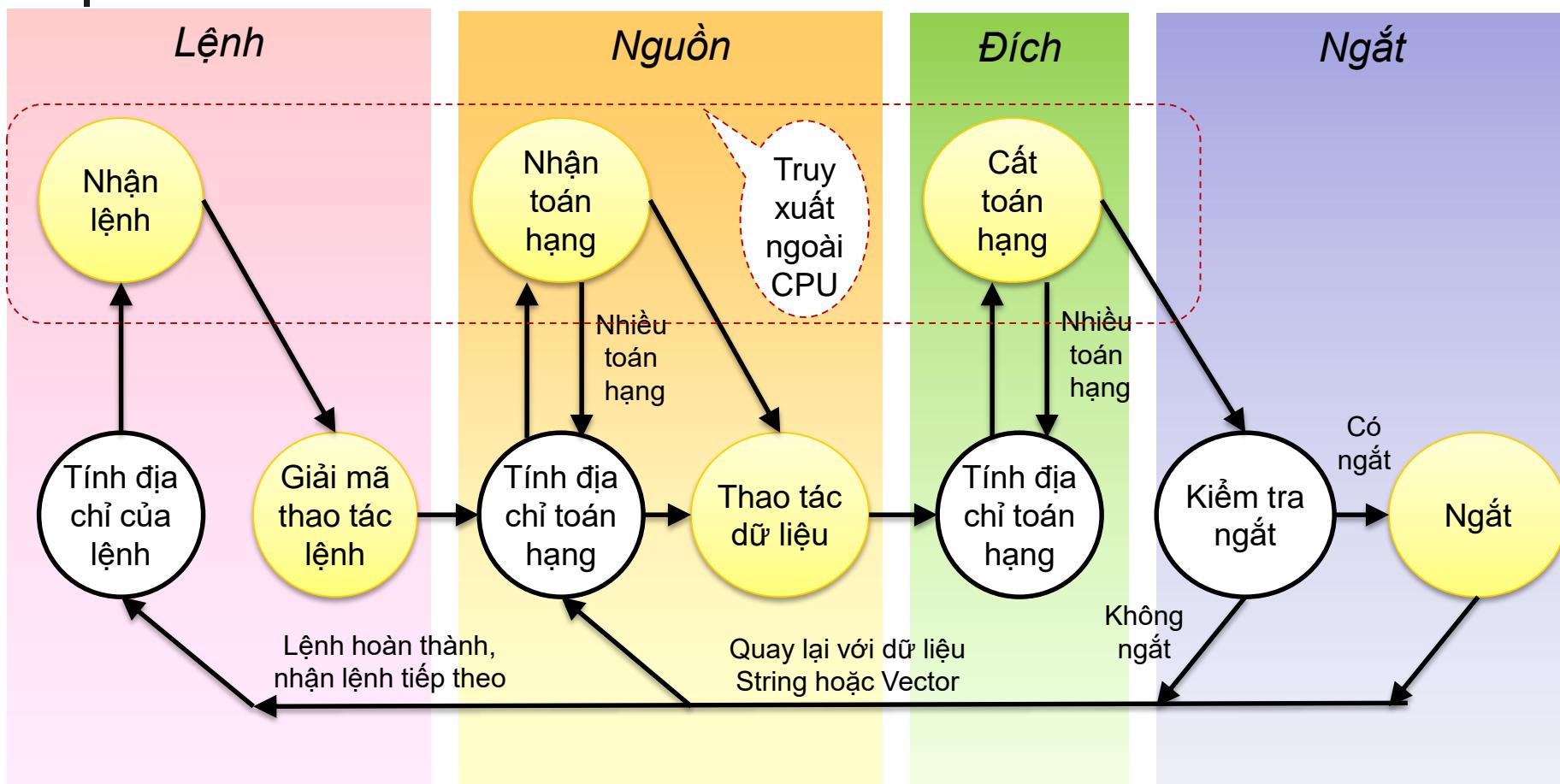
- Các tín hiệu điều khiển bên trong CPU:
 - Điều khiển các thanh ghi
 - Điều khiển ALU
- Các tín hiệu điều khiển bên ngoài CPU:
 - Điều khiển bộ nhớ
 - Điều khiển các mô-đun vào-ra

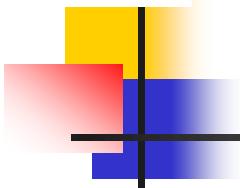
4. Hoạt động của chu trình lệnh

Chu trình lệnh

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

Giản đồ trạng thái chu trình lệnh

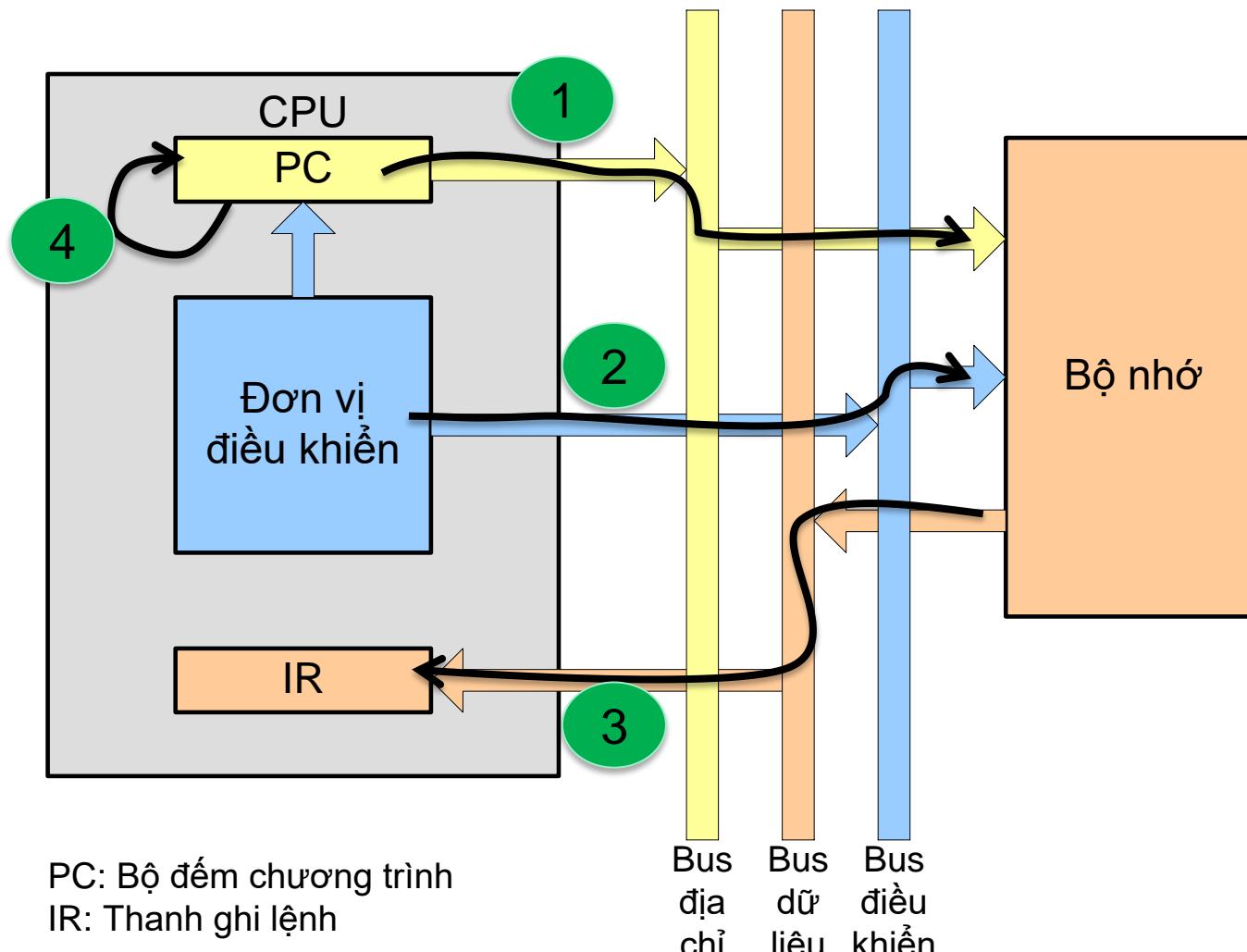




Nhận lệnh

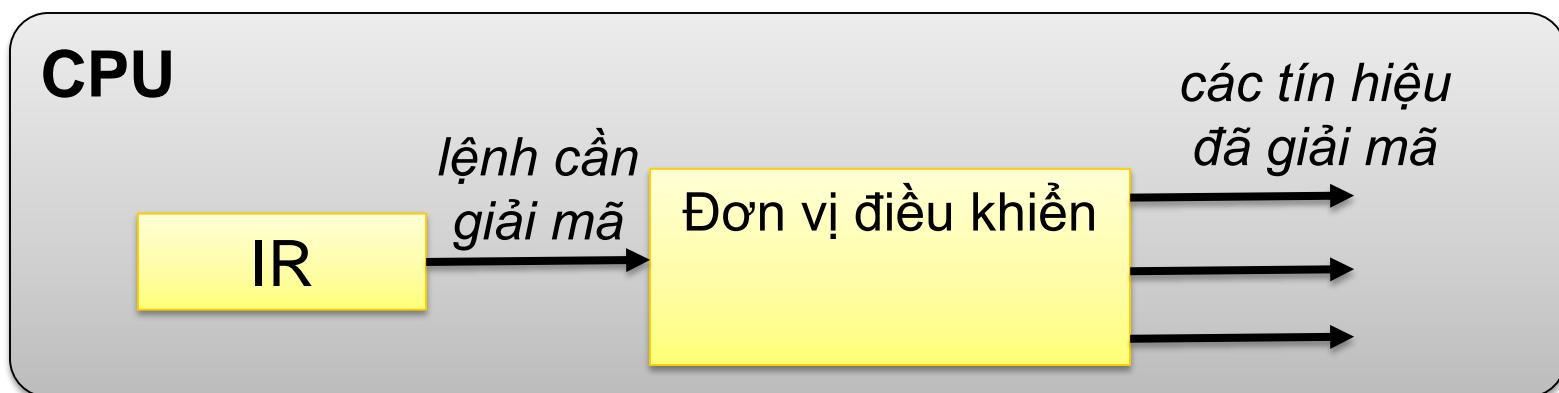
1. CPU đưa địa chỉ của lệnh cần nhận từ bộ đếm chương trình PC ra bus địa chỉ
2. CPU phát tín hiệu điều khiển đọc bộ nhớ
3. Lệnh từ bộ nhớ được đặt lên bus dữ liệu và được CPU copy vào thanh ghi lệnh IR
4. CPU tăng nội dung PC để trả sang lệnh kế tiếp

Sơ đồ mô tả quá trình nhận lệnh



Giải mã lệnh

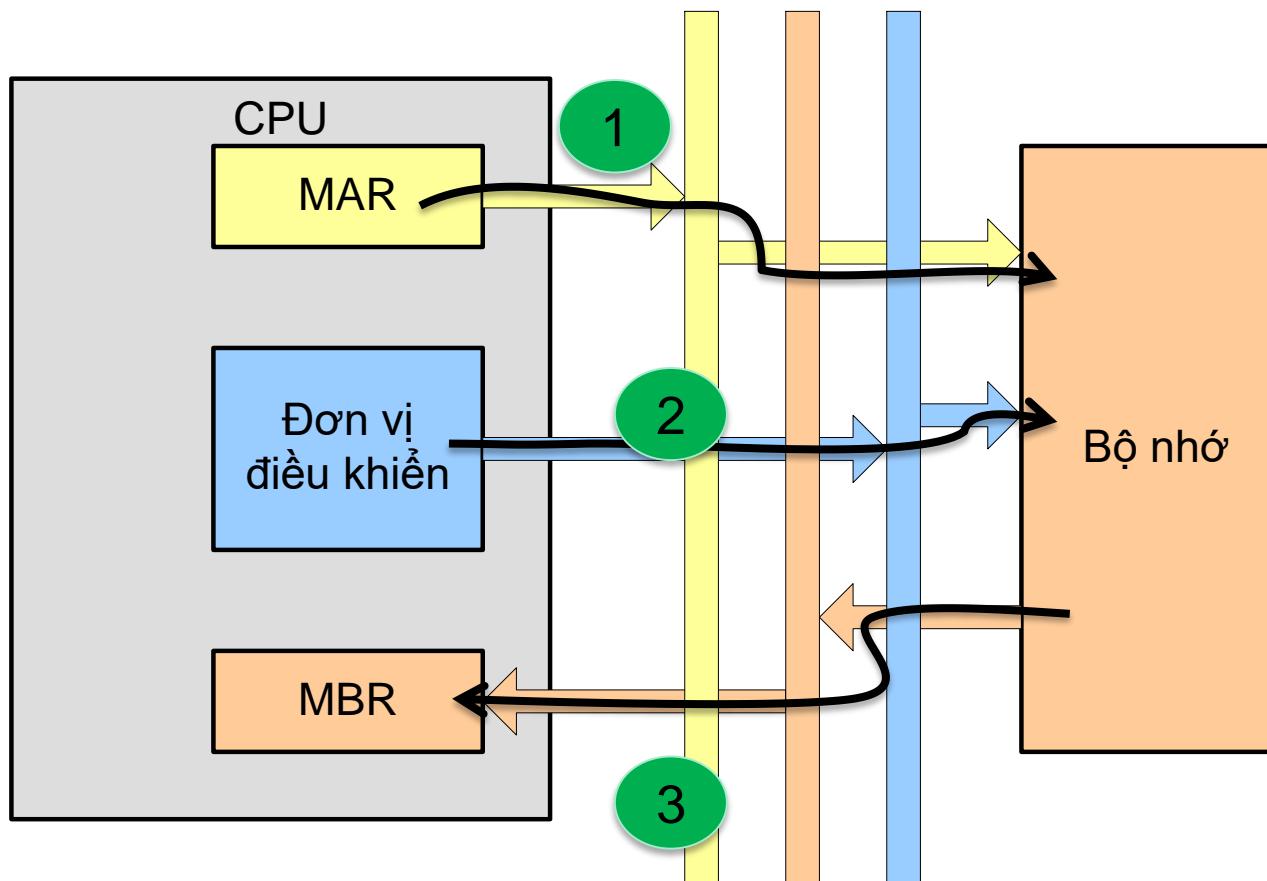
- Lệnh từ thanh ghi lệnh IR được đưa đến đơn vị điều khiển
- Đơn vị điều khiển tiến hành giải mã lệnh để xác định thao tác phải thực hiện
- Giải mã lệnh xảy ra bên trong CPU



Nhận dữ liệu từ bộ nhớ

1. CPU đưa địa chỉ của toán hạng ra bus địa chỉ
 2. CPU phát tín hiệu điều khiển đọc
 3. 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

Bus
địa
chi
Bus
dữ
liệu
Bus
điều
khiển

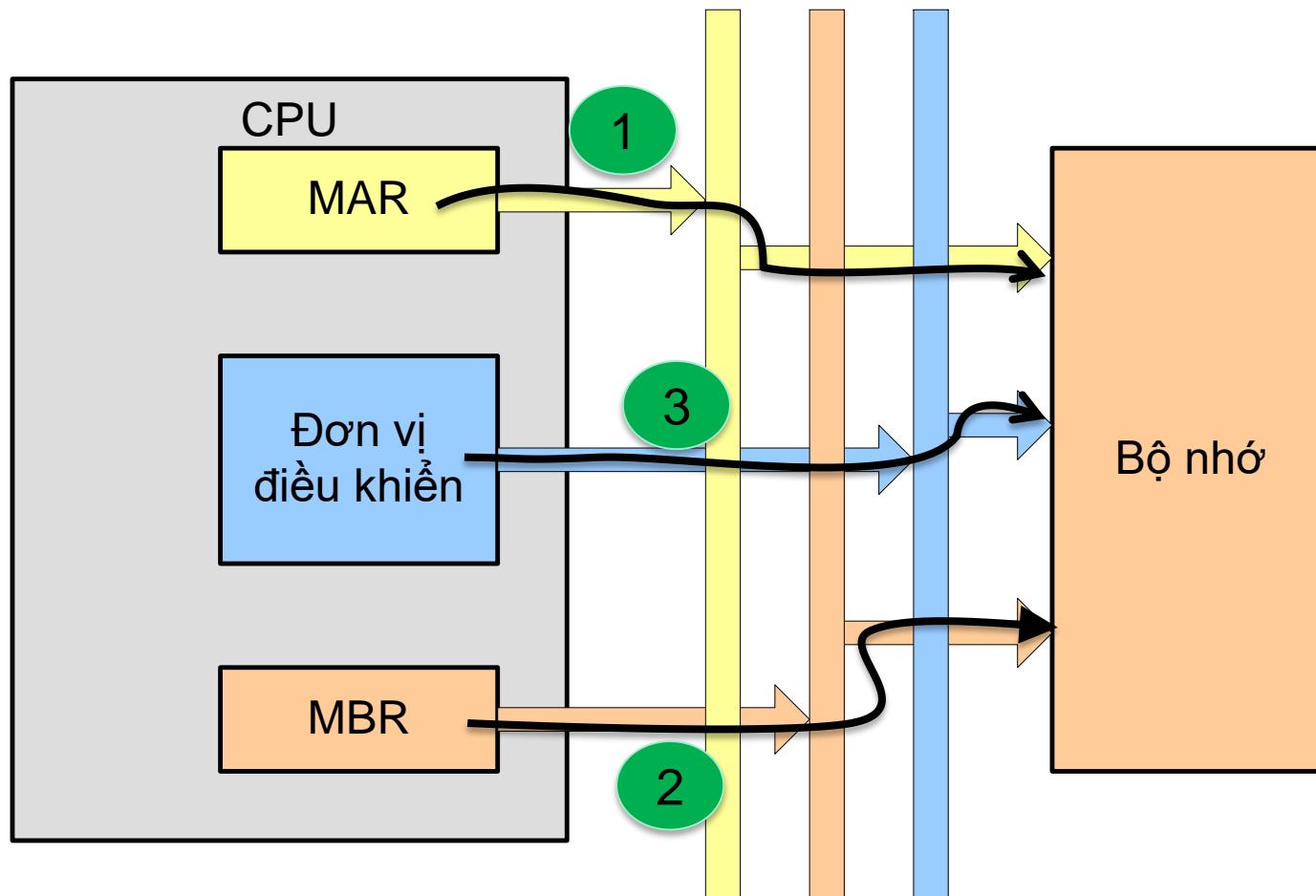
Thực hiện lệnh

- Có nhiều dạng tùy thuộc vào lệnh
- Có thể là:
 - Đọc/Ghi bộ nhớ
 - Vào/Ra
 - Chuyển giữa các thanh ghi
 - Thao tác 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

Bus
địa
chi
Bus
dữ
liệu
Bus
điều
khiển

Ngắt

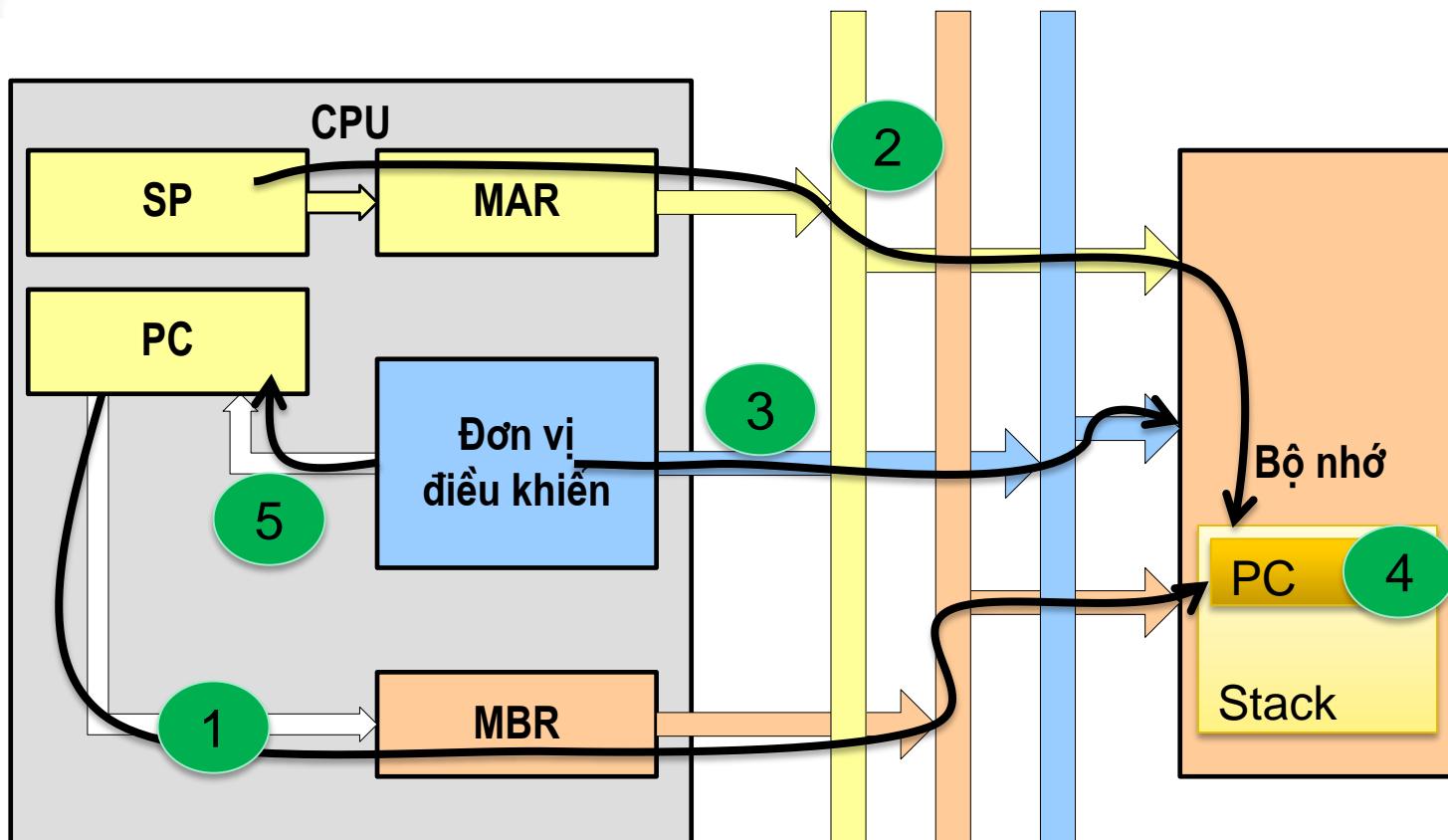
Cắt ngũ cảnh trở về

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

Chuyển tới Ch.trình con ngắt

- 5. 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 $\ddot{a}a$ ch \ddot{b} nh \square

MBR: Thanh ghi $\ddot{a}\square m$ b \square nh \square

PC: B $\ddot{a}\square m$ ch- \square ng tr \ddot{a} nh

SP: Con tr \ddot{a} ngon x \ddot{a} p

Bus $\ddot{a}a$

Bus d \ddot{e}

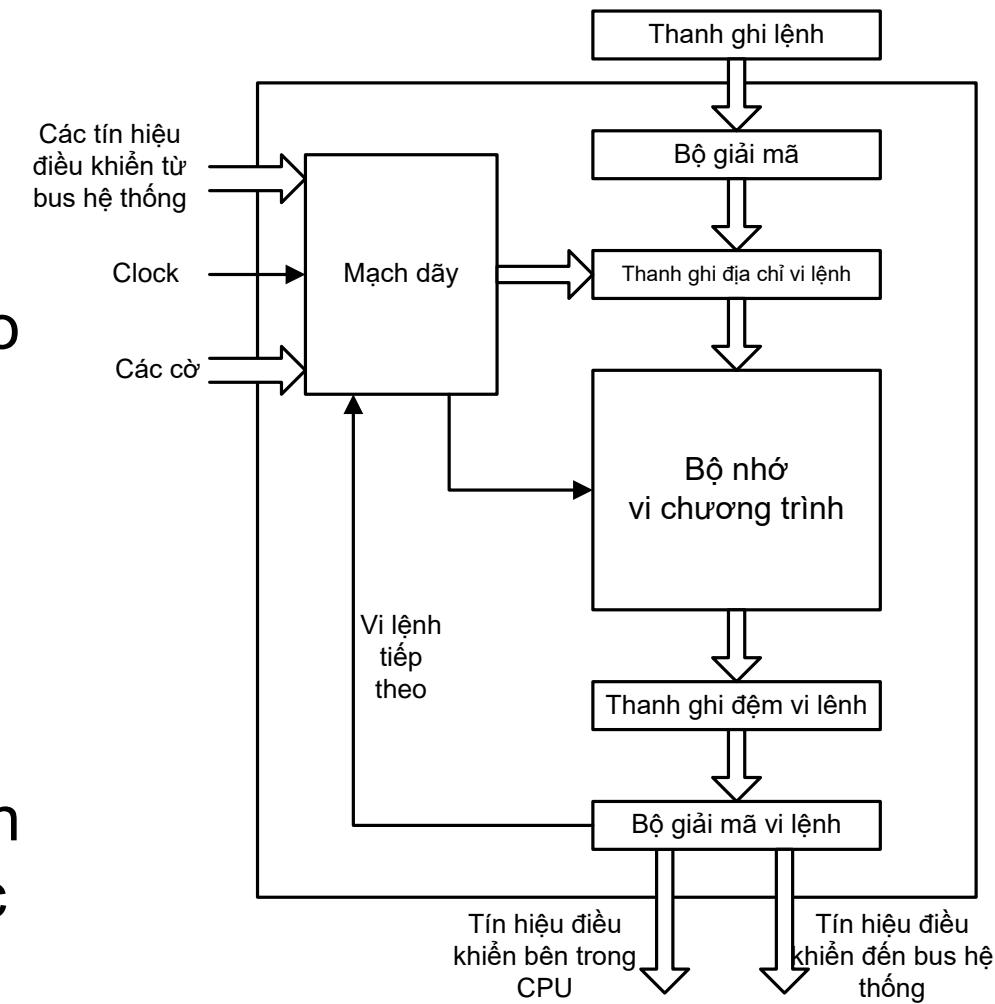
Bus $\ddot{a}i\ddot{u}$

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)

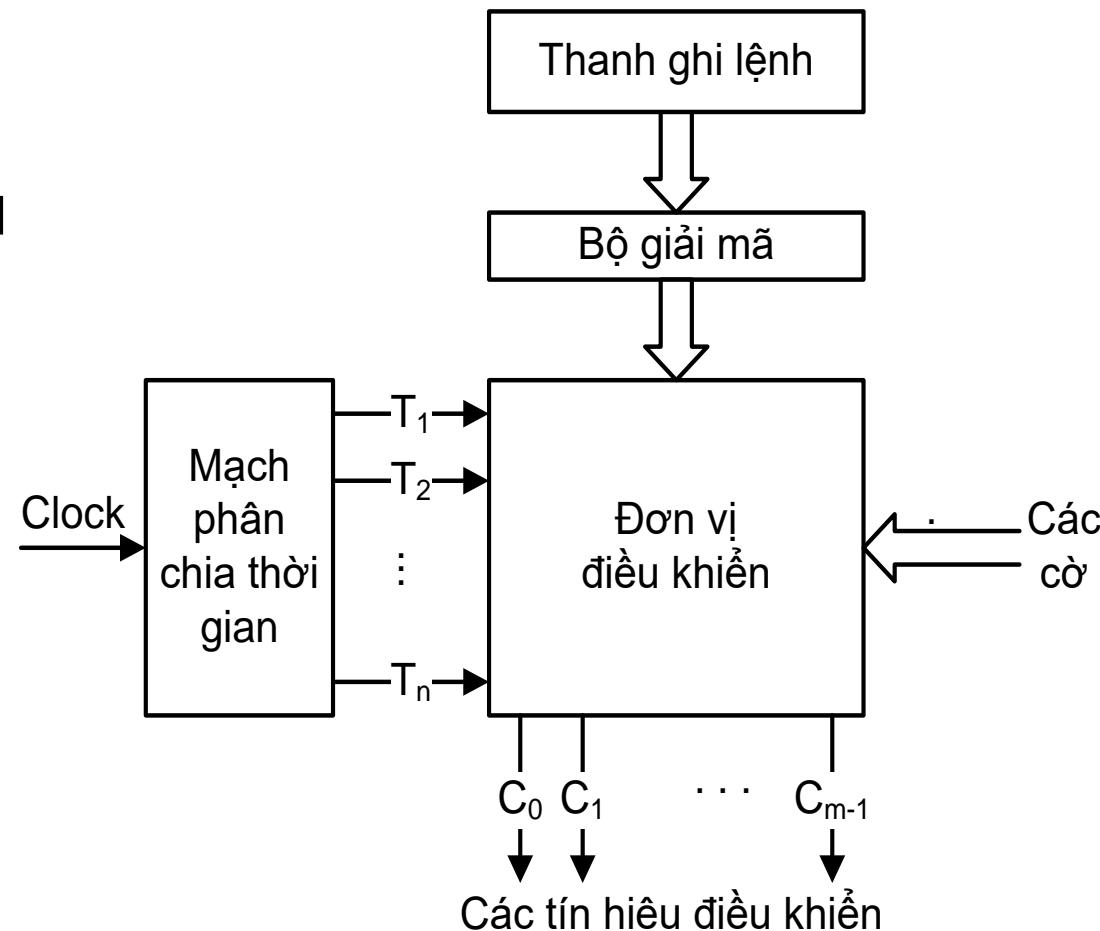
1. Đơn vị điều khiển vi chương trình

- 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 hóa 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 và song song mức lệnh

- Kỹ thuật đường ống lệnh (Instruction Pipelining): Chia chu trình lệnh thành các công đoạn và cho phép thực hiện gối lén nhau (như dây chuyền lắp ráp)
- Chẳng hạn bộ xử lý MIPS có 5 công đoạn:
 - **IF:** Instruction Fetch from memory – Nhận lệnh từ bộ nhớ
 - **ID:** Instruction Decode & register read-Giải mã lệnh & đọc thanh ghi
 - **EX:** Execute Operation or calculate address - Thực hiện thao tác hoặc tính toán địa chỉ (*địa chỉ bộ nhớ chính*)
 - **MEM:** Access MEMORY Operand – Truy nhập toán hạng bộ nhớ
 - **WB:** Write result Back to register – Ghi kết quả về thanh ghi

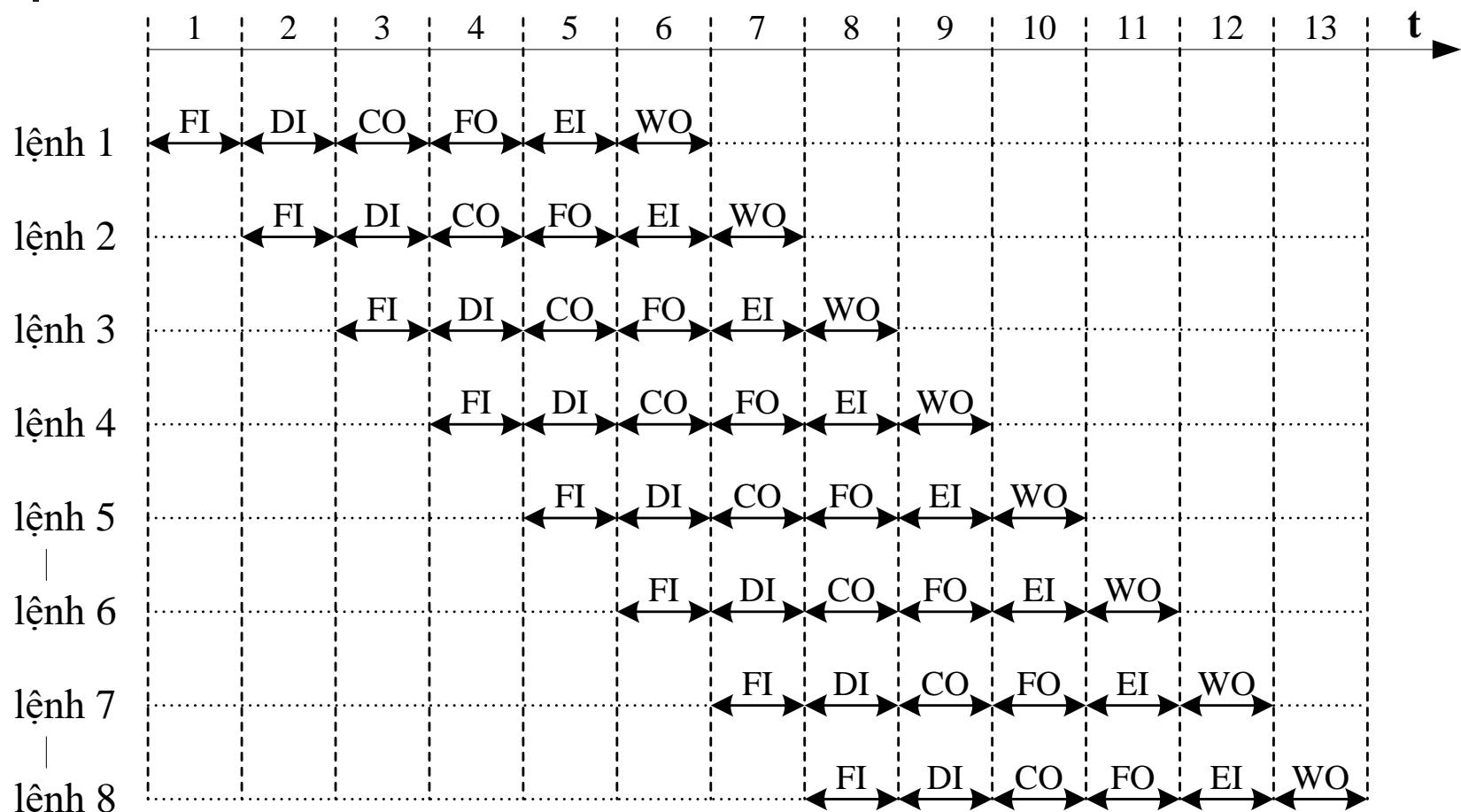
Ford Model T –
100 Years Later



Verry harrd



Biểu đồ thời gian của đường ống lệnh



n lệnh
p: công đoạn

→ nếu o có pipeline → $T = n * p$
→ có pipeline → $T = n + p - 1$

Các Hazard (trở ngại) của đường ống lệnh

- **Hazard:** Tình huống ngăn cản bắt đầu của lệnh tiếp theo ở chu kỳ tiếp theo.





Hazard về cấu trúc

■ Khắc phục:

- nhân tài nguyên để tránh xung đột
- Làm trễ

Ví dụ:

- Bus dữ liệu: truyền lệnh và dữ liệu
→ Bus lệnh riêng, bus dữ liệu riêng (cache lệnh và cache dữ liệu)

Ví dụ Hazard về cấu trúc



conflict on arithmetic unit

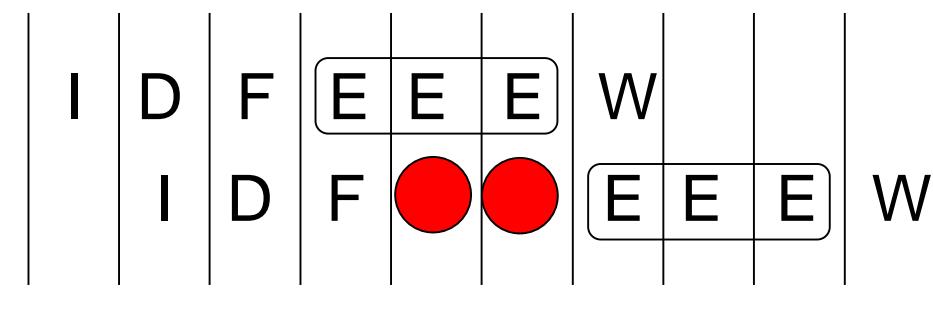
MULT A,B,C

MULT D,E,F

cache miss

TLB miss

3 clocks necessary for multiplication



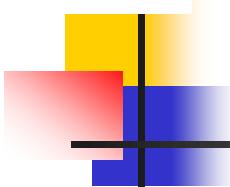
stall



Hazard về dữ liệu

- Các dạng:

- **RAW** (Read After Write)
- **WAR** (Write After Read)
- **WAW** (Write After Write)



Sự phụ thuộc về dữ liệu



RAW	ADD A,B,C ADD E,A,D	Write-A must be earlier than Read-A
WAR	ADD A,B,C ADD B,D,E	Read-B must be earlier than Write-B
WAW	ADD A,B,C ADD A,D,E	First Write-A must be earlier Than second Write-A

WAR and WAW

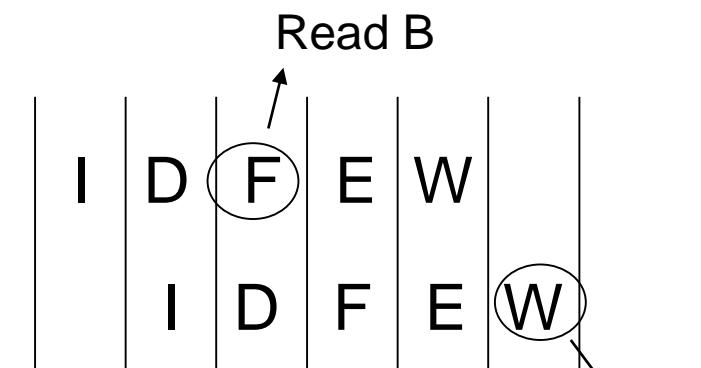


WAR

ADD A,B,C

ADD B,D,E

Read-B is earlier than Write-B

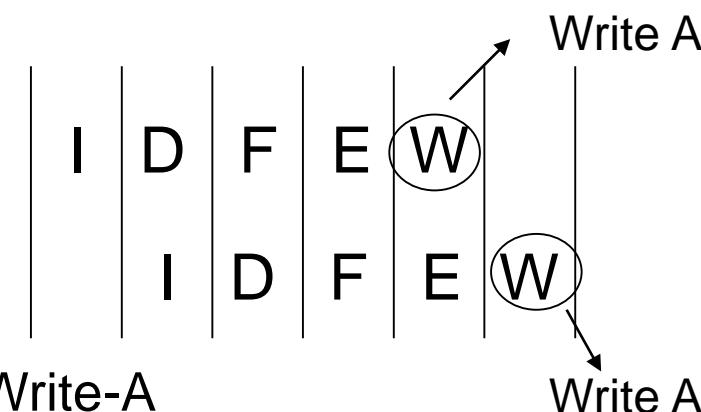


WAW

ADD A,B,C

ADD A,D,E

first Write-A is earlier than second Write-A



no conflict at in-order pipeline
conflict at out-of-order pipeline



RAW

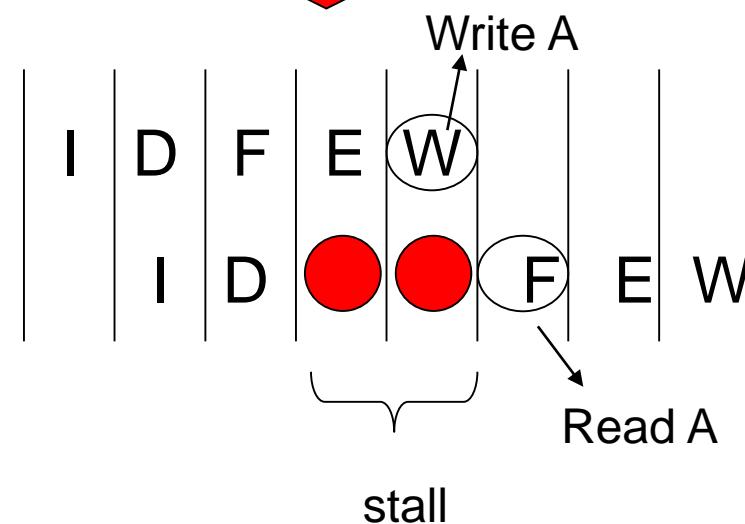
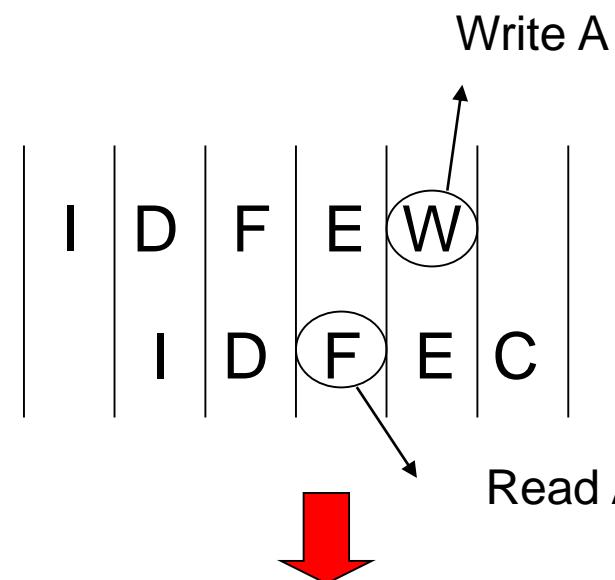


ADD A,B,C

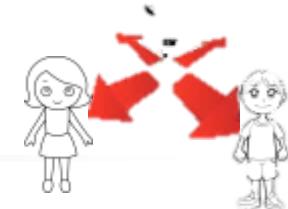
ADD E,A,D

Write-A must be earlier

Than Read-A



Hazard điều khiển



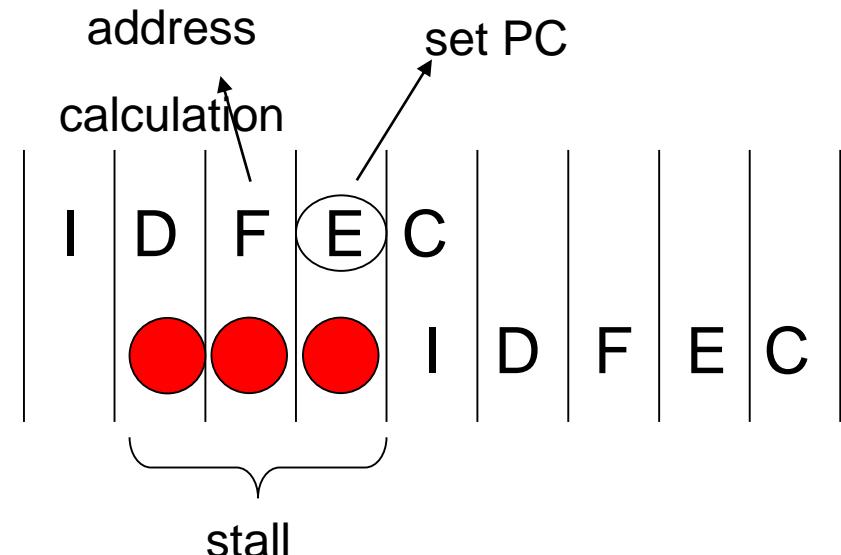
Wait for branch

BEQ A, B, Label

LOAD C, X

...

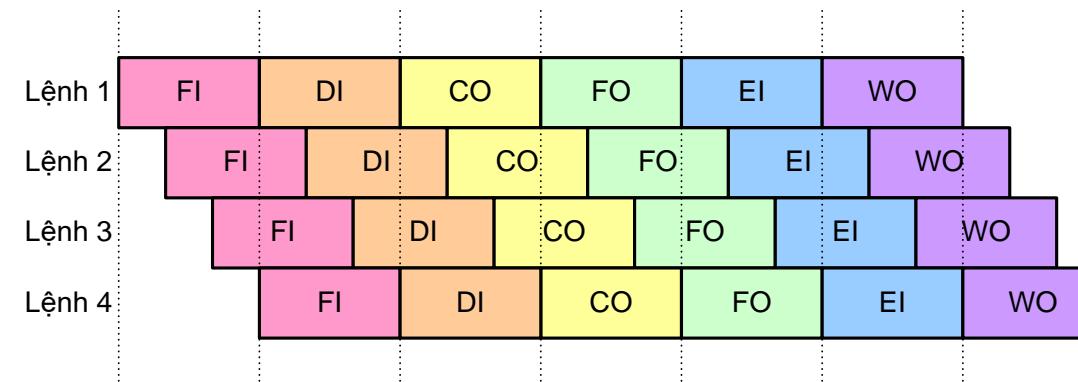
Label: LOAD C, Y



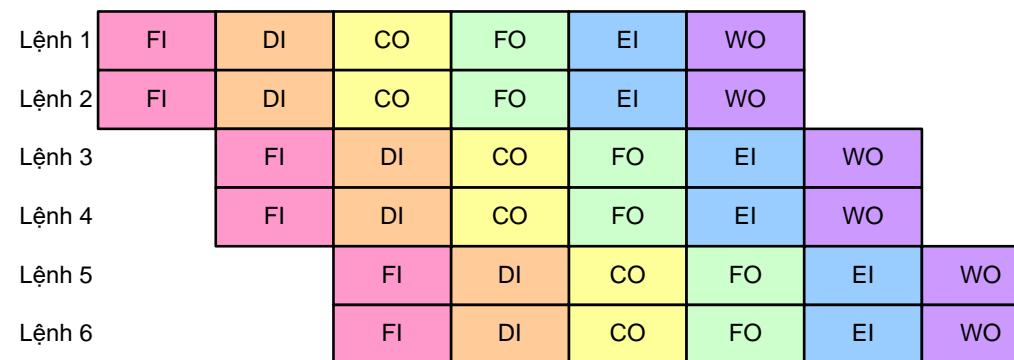
Lệnh tiếp theo sau lệnh branch sẽ không được nạp cho tới khi xác định rõ điều kiện rẽ nhánh và cập nhật thanh ghi PC

Các kiến trúc song song mức lệnh

- Siêu đường ống (Superpipeline & Hyperpipeline)

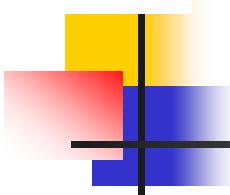


- Siêu vô hướng (Superscalar)

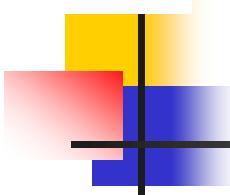


6.4. Thiết kế bộ xử lý theo kiến trúc MIPS*

Bộ xử lý MIPS – Chapter 4 – [2]



Hết chương 6

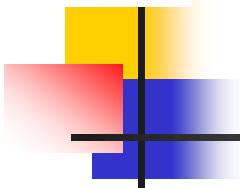


Kiến trúc máy tính

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

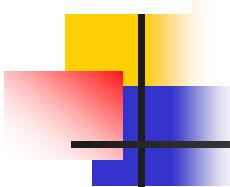
Chương 5. Kiến trúc tập lệnh

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

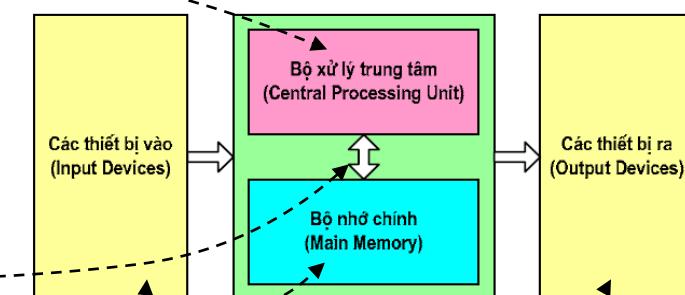
- 7.1. Phân cấp bộ nhớ
- 7.2. Bộ nhớ chính
- 7.3. Bộ nhớ cache
- 7.4. Bộ nhớ ngoài
- 7.5. Bộ nhớ ảo

7.1. Phân cấp bộ nhớ máy tính

1. Các đặc trưng của bộ nhớ máy tính

Vị trí

- Bên trong CPU:
 - **tập thanh ghi**
- Bộ nhớ trong:
 - bộ nhớ cache
 - **bộ nhớ chính**
- Bộ nhớ ngoài: **các thiết bị nhớ**



Dung lượng

- Độ dài từ nhớ (tính bằng bit)
- Số lượng từ nhớ



Các đặc trưng của hệ thống nhớ (tiếp)

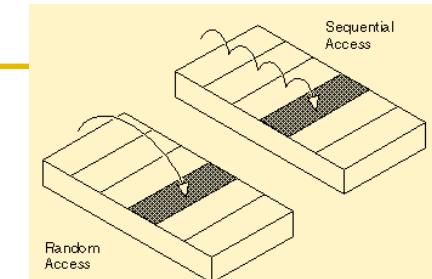
Đơn vị truyền

- Tùy nhớ
- Khối nhớ



Phương pháp truy nhập

- Truy nhập tuần tự (băng tùy)
- Truy nhập trực tiếp (các loại đĩa)
- Truy nhập ngẫu nhiên (bộ nhớ bán dẫn)
- Truy nhập liên kết (cache)



Các đặc trưng của hệ thống nhớ (tiếp)

Hiệu năng (performance)

- Thời gian truy nhập
- Chu kỳ nhớ
- Tốc độ truyền

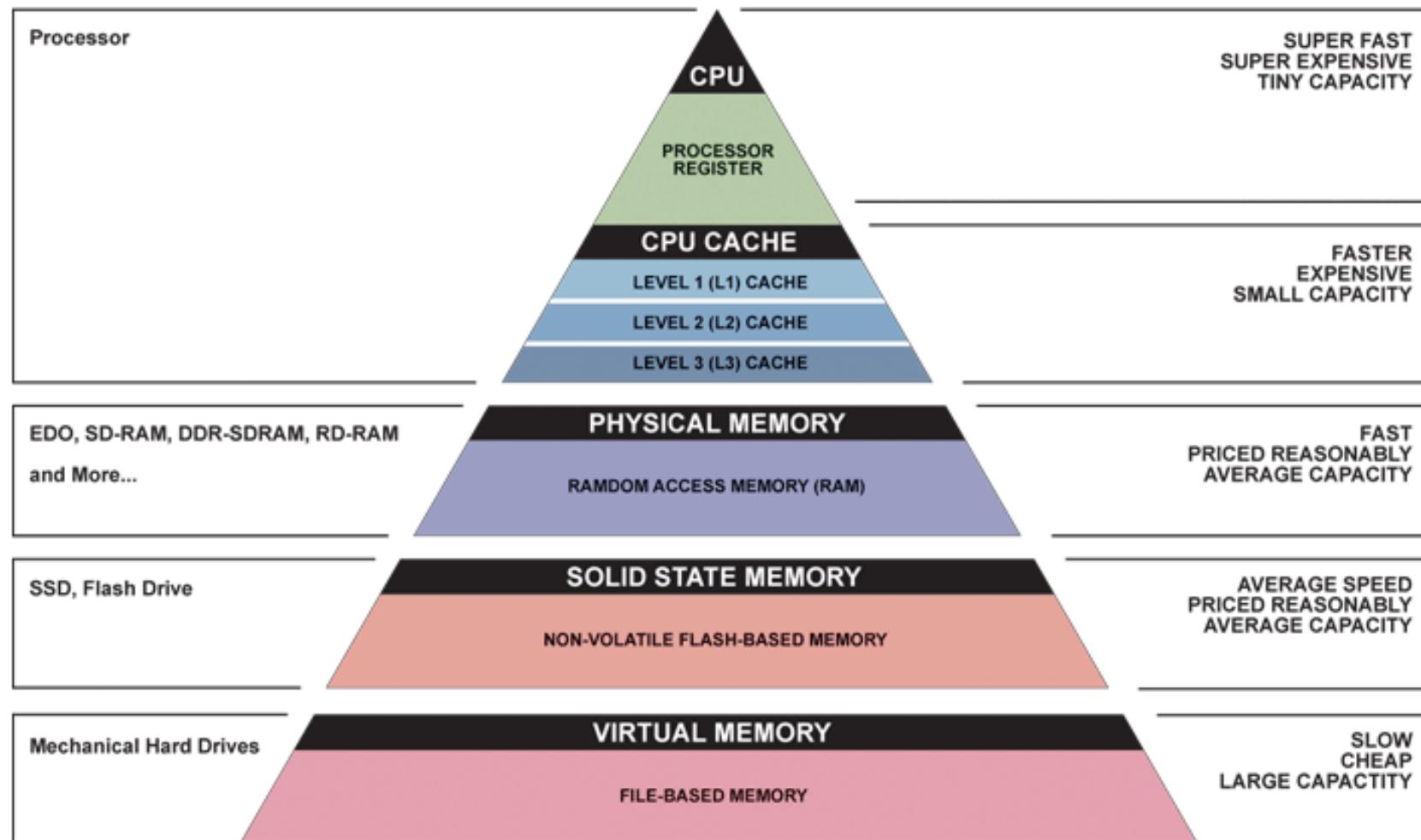
Kiểu vật lý

- Bộ nhớ bán dẫn
- Bộ nhớ từ
- Bộ nhớ quang

Các đặc trưng của hệ thống nhớ (tiếp)

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

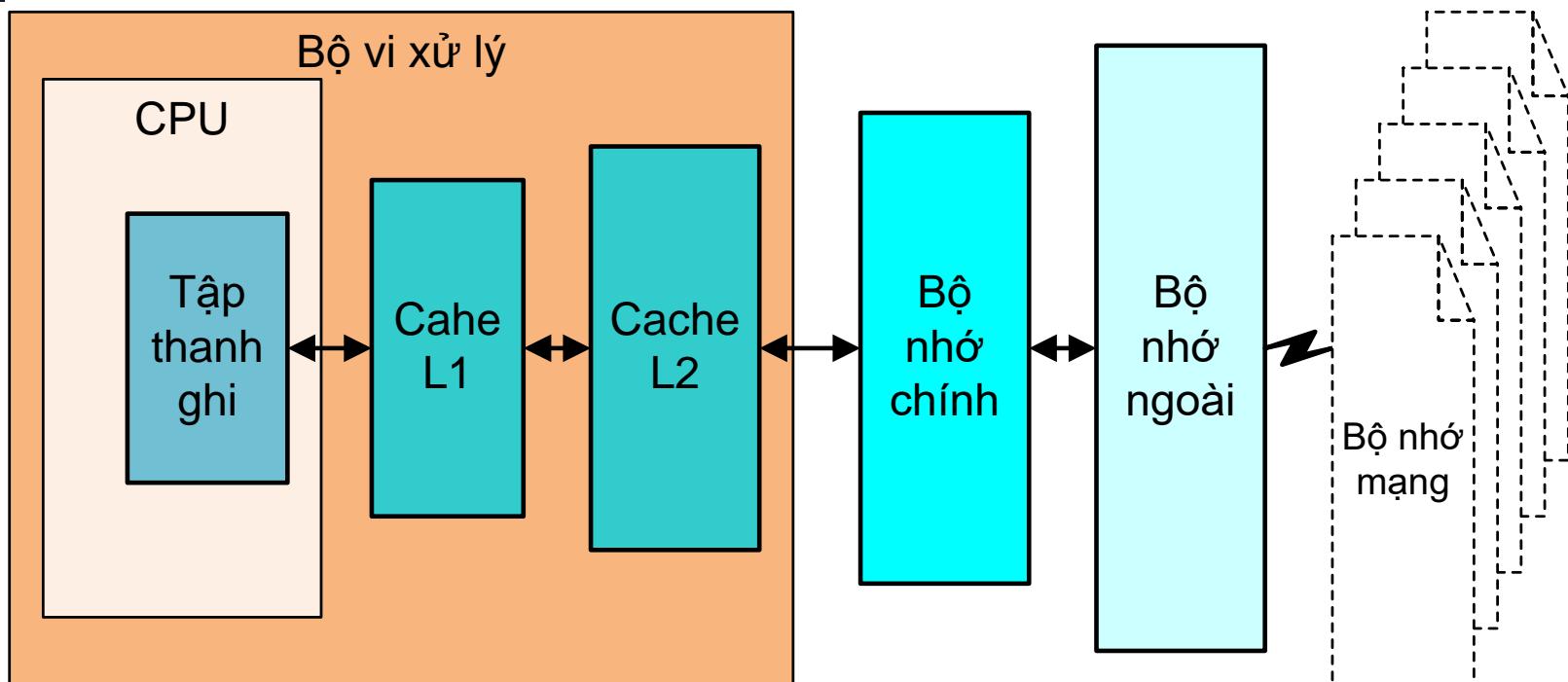
2. Phân cấp bộ nhớ



▲ Simplified Computer Memory Hierarchy



Ví dụ hệ thống nhớ thông dụng



Từ trái sang phải:

- dung lượng tăng dần
- tốc độ giảm dần
- giá thành/1bit 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	5.000 – 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

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	
Programmable ROM (PROM)				
Erasable PROM (EPROM)	Bộ nhớ hầu như chỉ đọc	bằng tia cực tím, cả chip		Không khả biến
Electrically Erasable PROM (EEPROM)		bằng điện, mức từng byte	Bằng điện	
Flash memory	Bộ nhớ	bằng điện, từng khối		
Random Access Memory (RAM)	đọc-ghi	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
 - rất đắt
- PROM (Programmable ROM)
 - Cần thiết bị chuyên dụng để ghi bằng chương trình → chỉ ghi được một lần
- EPROM (Erasable PROM)
 - Cần thiết bị chuyên dụng để ghi bằng chương trình → ghi được nhiều lần
 - Trước khi ghi lại, xóa bằng tia cực tím
- EEPROM (Electrically Erasable PROM)
 - Có thể ghi theo từng byte
 - Xóa bằng điện

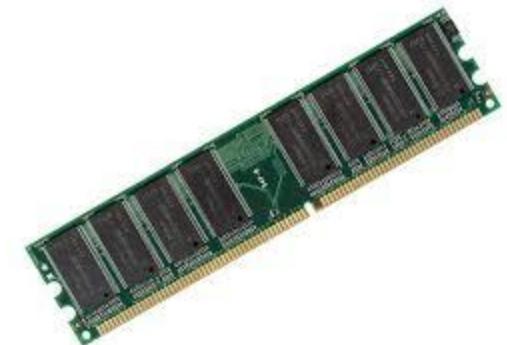
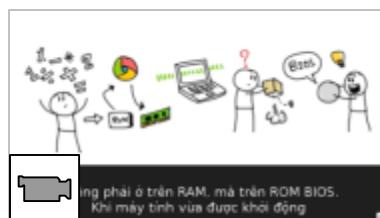
Bộ nhớ Flash

- Flash memory (Bộ nhớ cực nhanh)
 - Ghi theo khối
 - Xóa bằng điện
 - Dung lượng lớn
 - Sáng chế: Dr. Fujio Masuoka, 1980



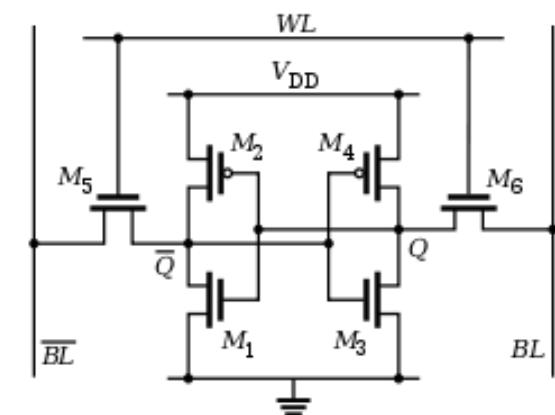
RAM (Random Access Memory)

- 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

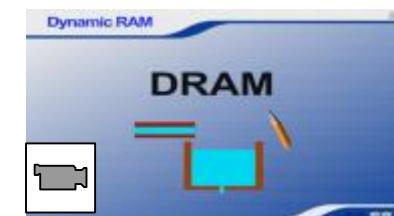
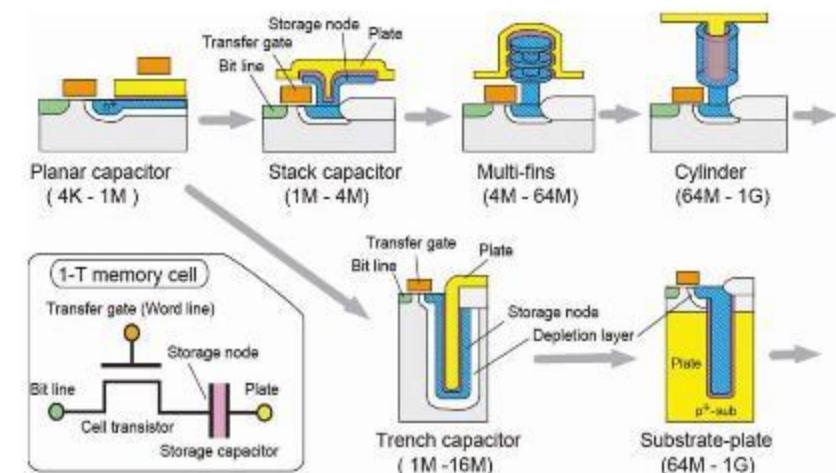
- Các bit được lưu trữ bằng các Flip-Flop
→ thông tin ổn định
- Cấu trúc phức tạp
- Dung lượng chip nhỏ
- Tốc độ nhanh
- Đắt tiền
- Dùng làm bộ nhớ cache



■ 1bit SRAM 6T

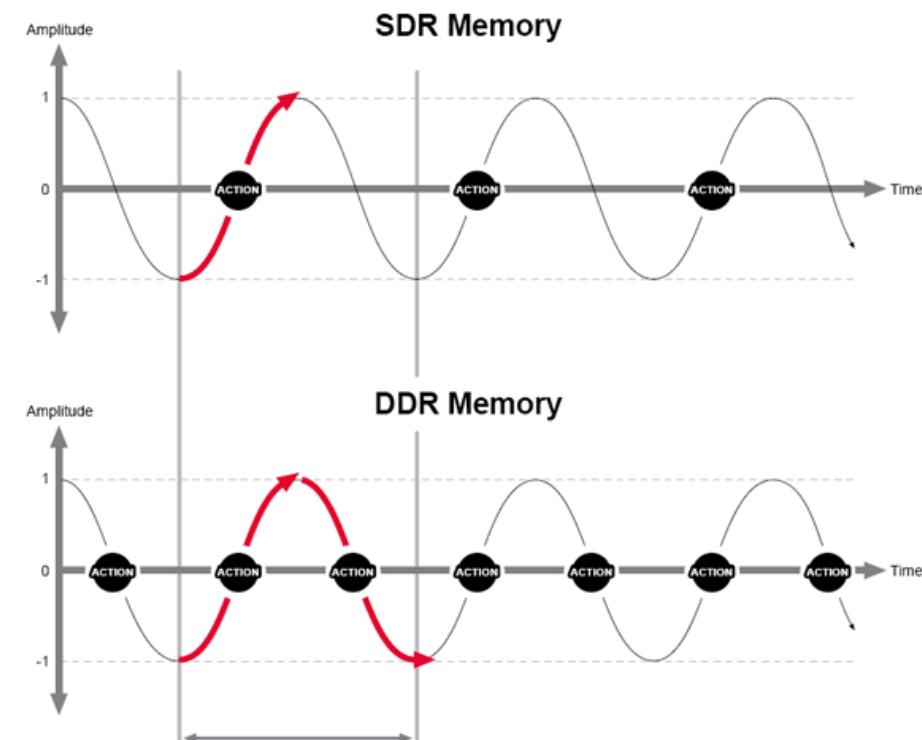
DRAM (Dynamic) – RAM động

- 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



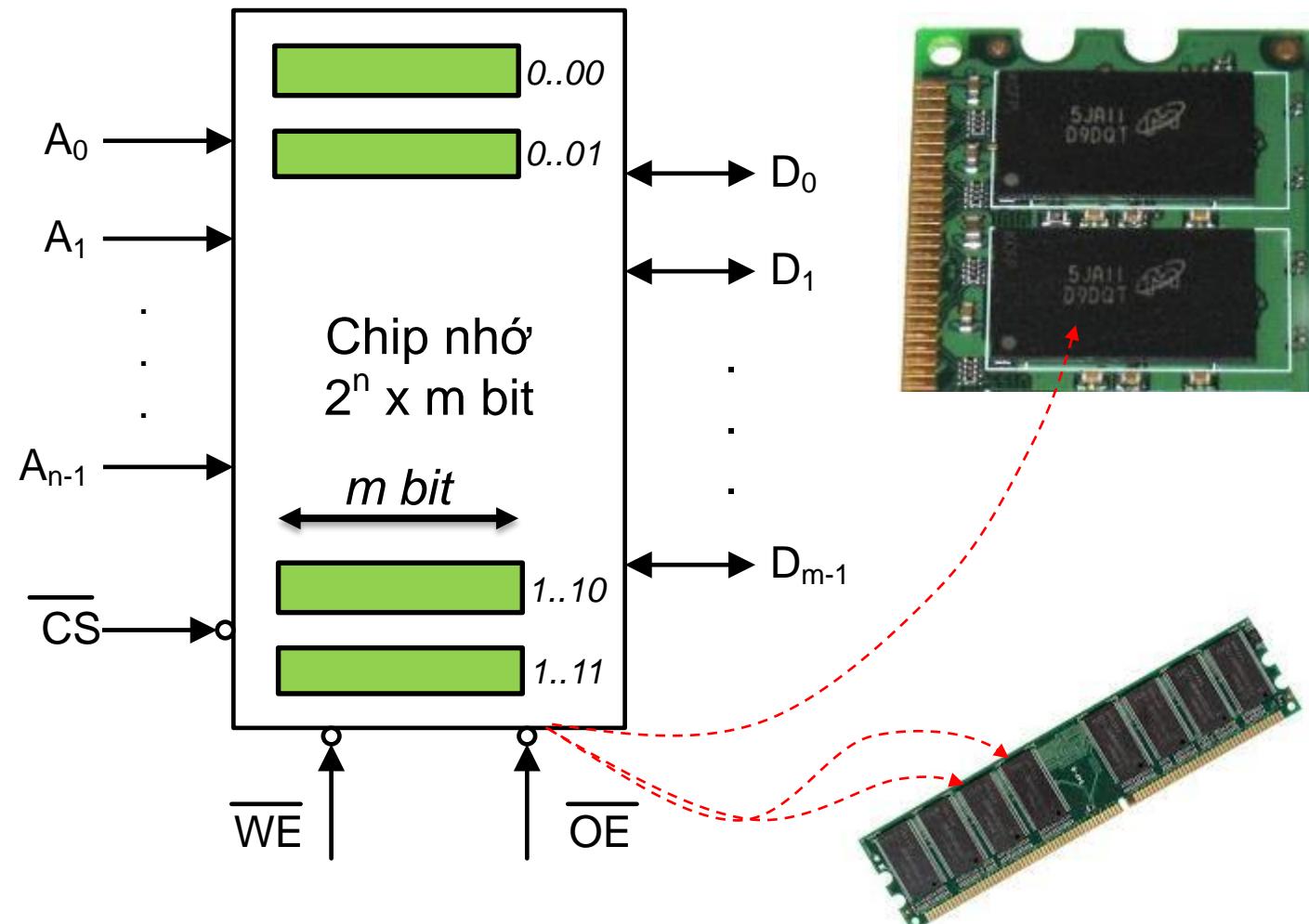
Các DRAM tiên tiến

- Enhanced DRAM
- Cache DRAM
- 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



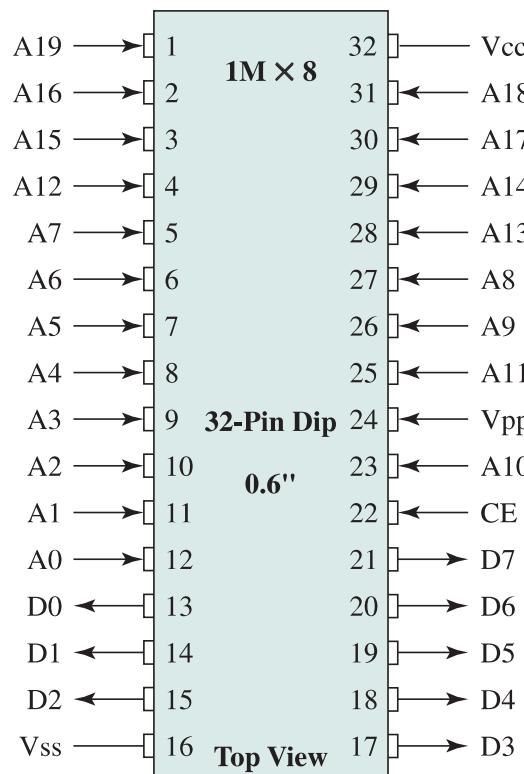
Các tín hiệu của chip nhớ

- Các đường địa chỉ: $A_{n-1} \div A_0 \rightarrow$ có 2^n từ nhớ
- Các đường dữ liệu: $D_{m-1} \div D_0 \rightarrow$ độ dài từ nhớ = m bit
- Dung lượng chip nhớ = **$2^n \times m$ 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)

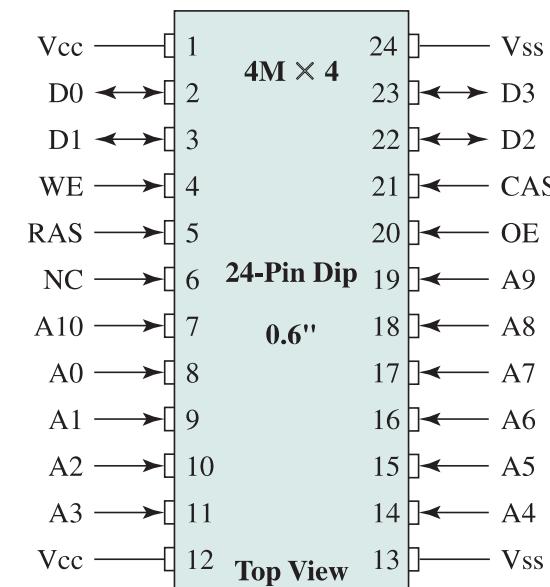
Tổ chức của DRAM

- Dùng n đường địa chỉ dồn kênh → cho phép truyền 2^n 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= $2^{2n} \times m$ bit

Ví dụ chip nhớ



(a) 8-Mbit EPROM



(b) 16-Mbit DRAM

Thiết kế mô-đun nhớ bán dẫn

- Dung lượng chip nhớ $2^n \times m$ 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

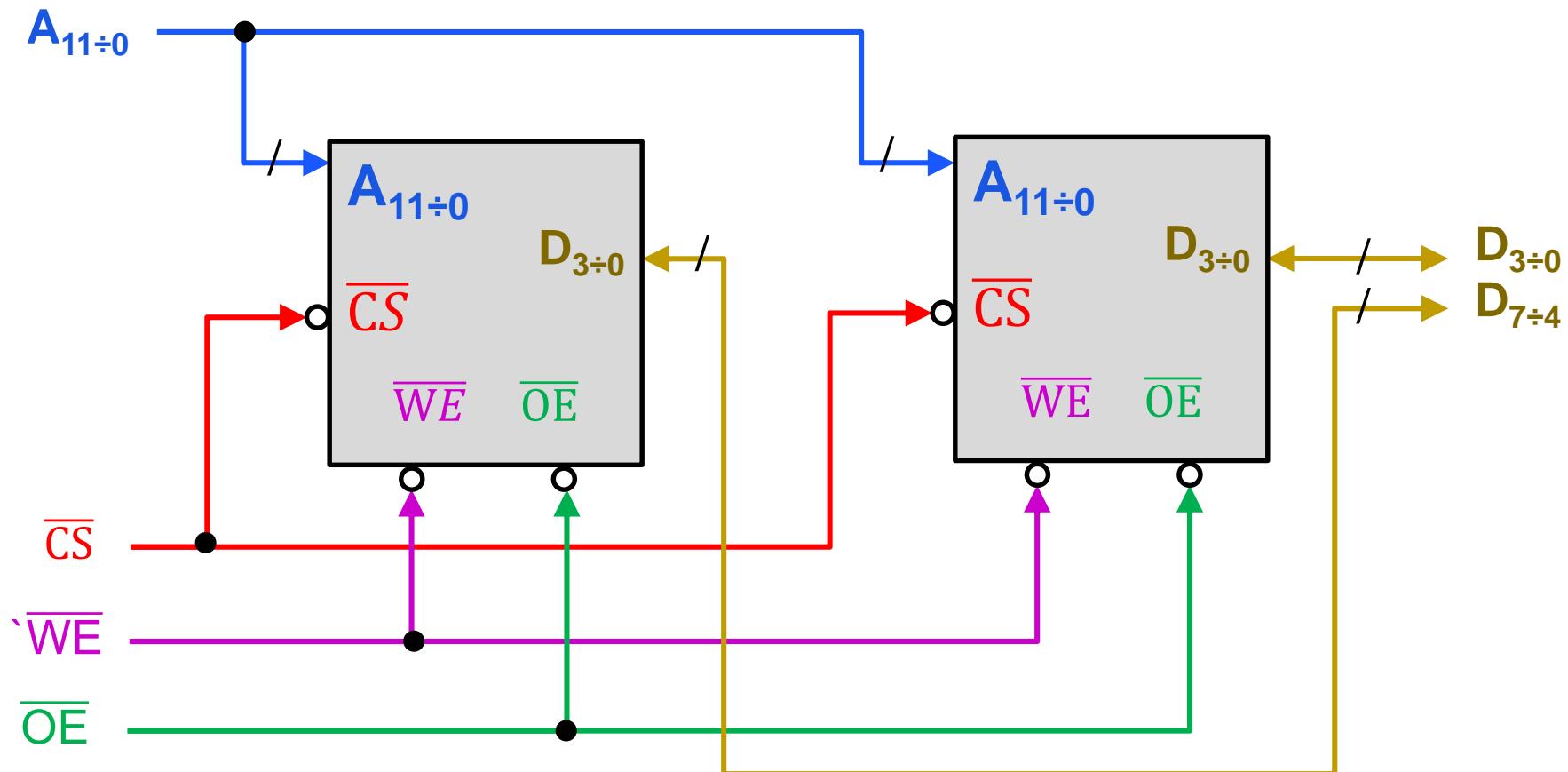
VD1:

- Cho chip nhớ SRAM $4K \times 4$ bit
- Thiết kế mô-đun nhớ $4K \times 8$ bit

Giải:

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

Ví dụ tăng độ dài từ nhớ



Bài toán tăng độ dài từ nhớ tổng quát

- Cho chip nhớ $2^n \times \text{mbit}$
- Thiết kế mô-đun nhớ $2^n \times (k.m)$ bit
- Dùng k chip nhớ

Tăng số lượng từ nhớ

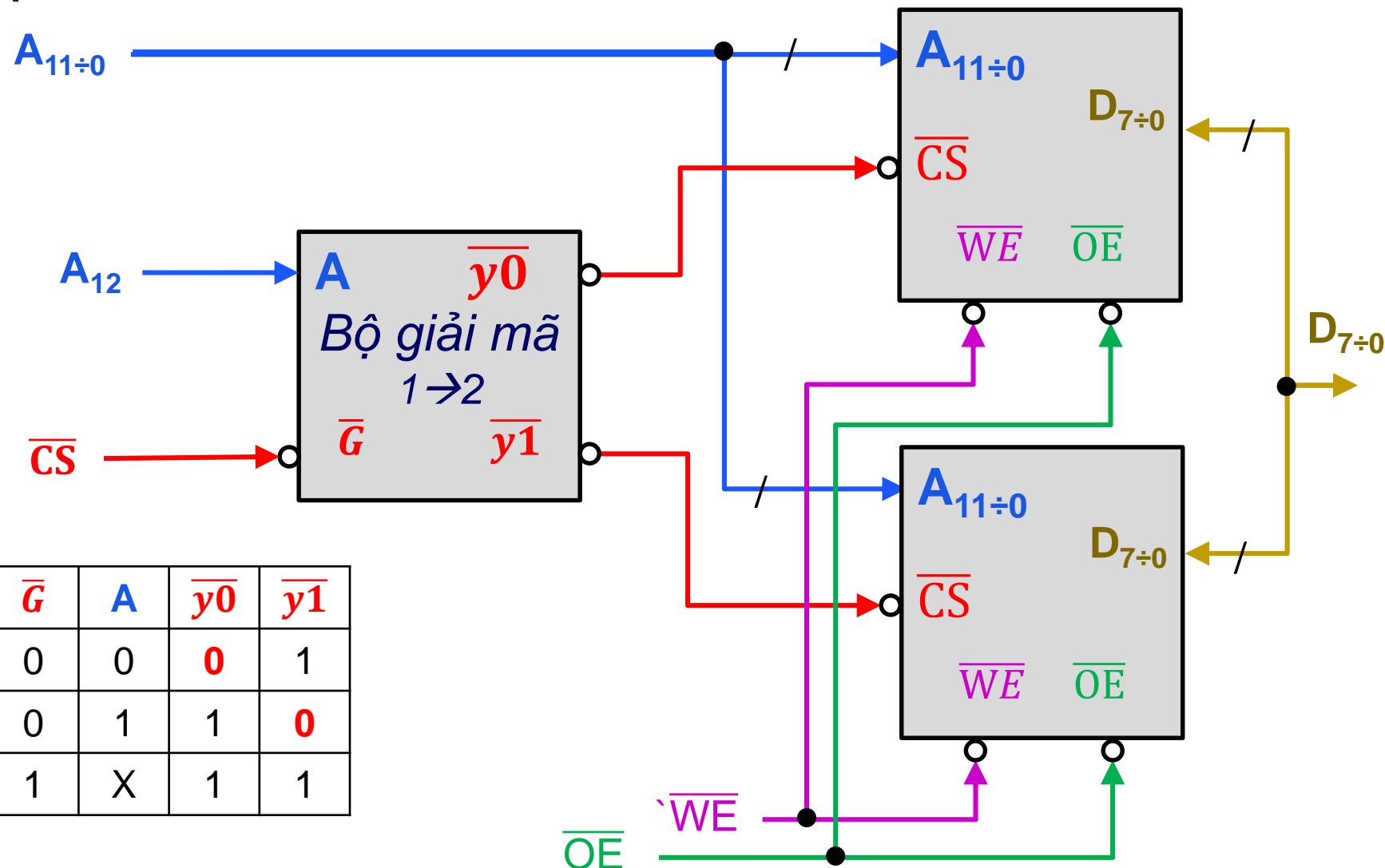
VD2:

- Cho chip nhớ SRAM $4K \times 8$ bit
- Thiết kế mô-đun nhớ $8K \times 8$ bit

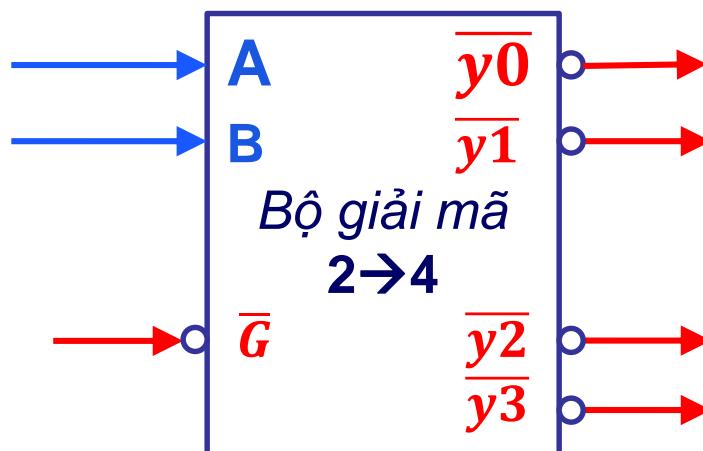
Giải:

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

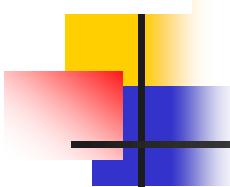
Tăng số lượng từ nhớ



Bộ giải mã 2→4



\bar{G}	Vào		Ra			
	B	A	\bar{y}_0	\bar{y}_1	\bar{y}_2	\bar{y}_3
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0
1	x	x	1	1	1	1



Thiết kế kết hợp

Ví dụ 3:

- Cho chip nhớ SRAM $4K \times 4$ bit
- Thiết kế mô-đun nhớ $8K \times 8$ bit

Phương pháp thực hiện:

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

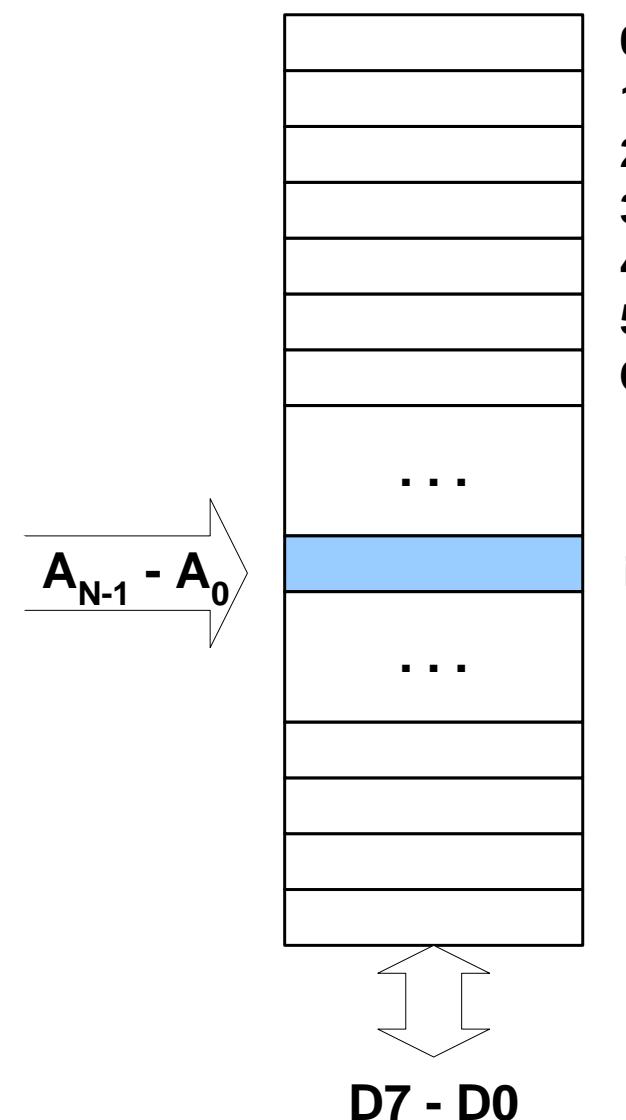
2. Các đặc trưng cơ bản của bộ nhớ chính

- 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 tùy thuộc vào hệ điều hành

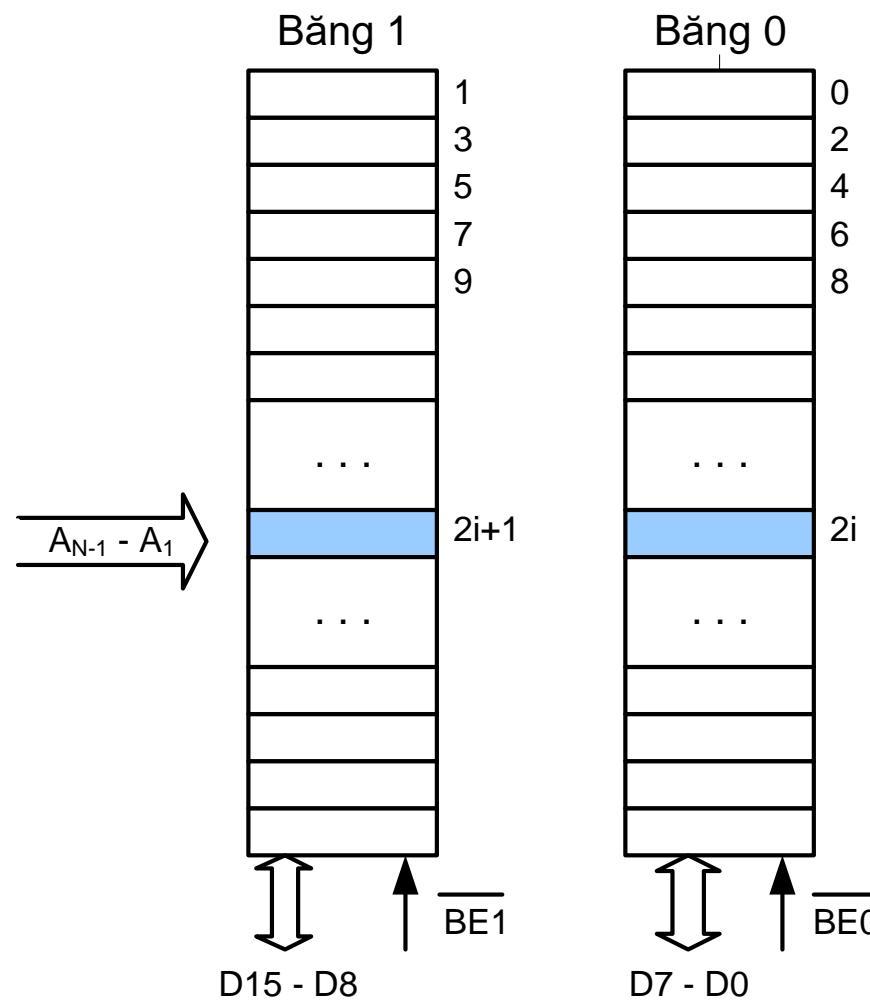
Tổ chức bộ nhớ đan xen (interleaved memory)

- Độ rộng của bus dữ liệu để trao đổi với bộ nhớ: $m = 8, 16, 32, 64, 128 \dots$ bit
- Các ngăn nhớ được tổ chức theo byte
→ tổ chức bộ nhớ vật lý khác nhau

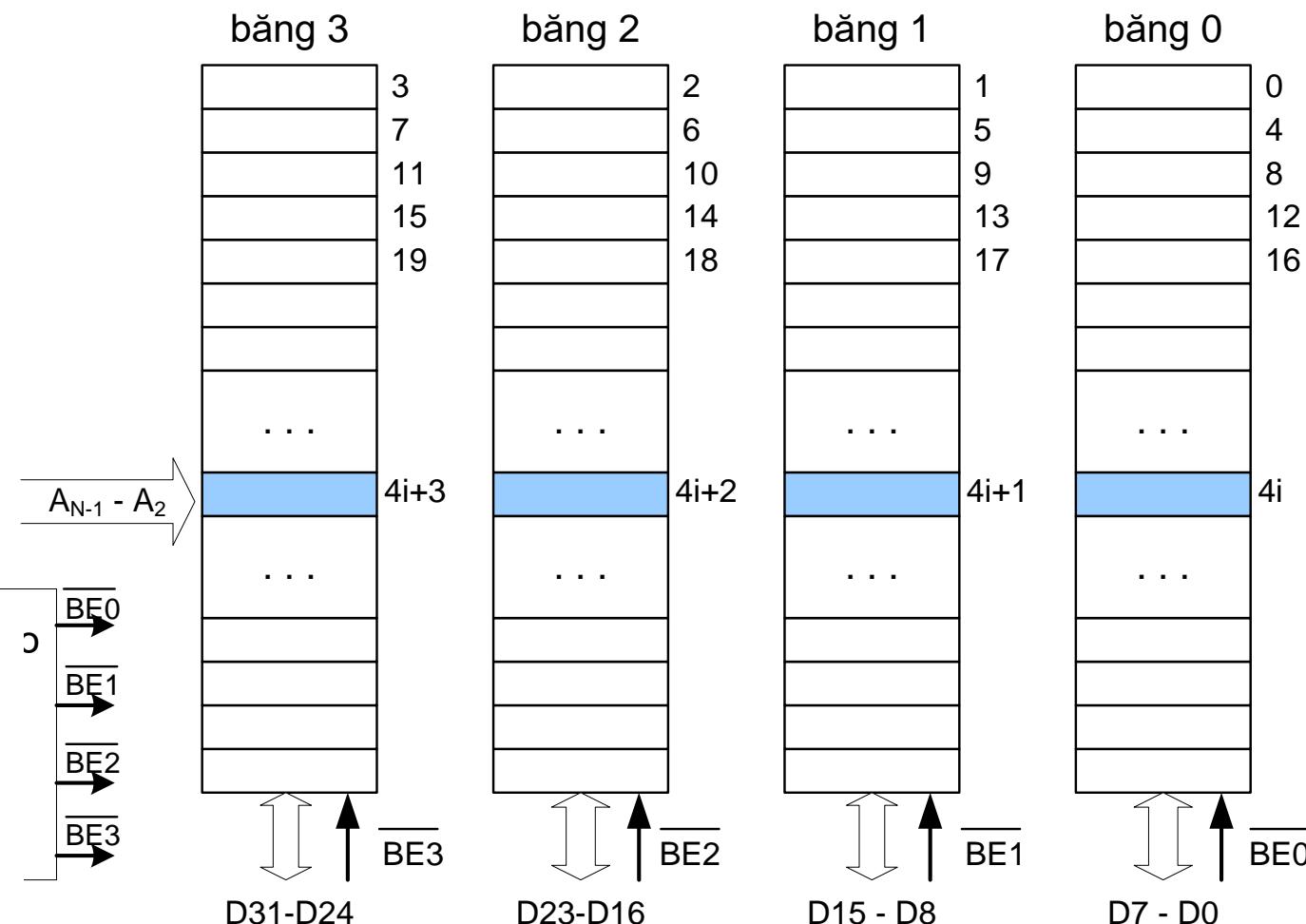
$m=8\text{bit} \rightarrow$ một băng nhớ tuyến tính



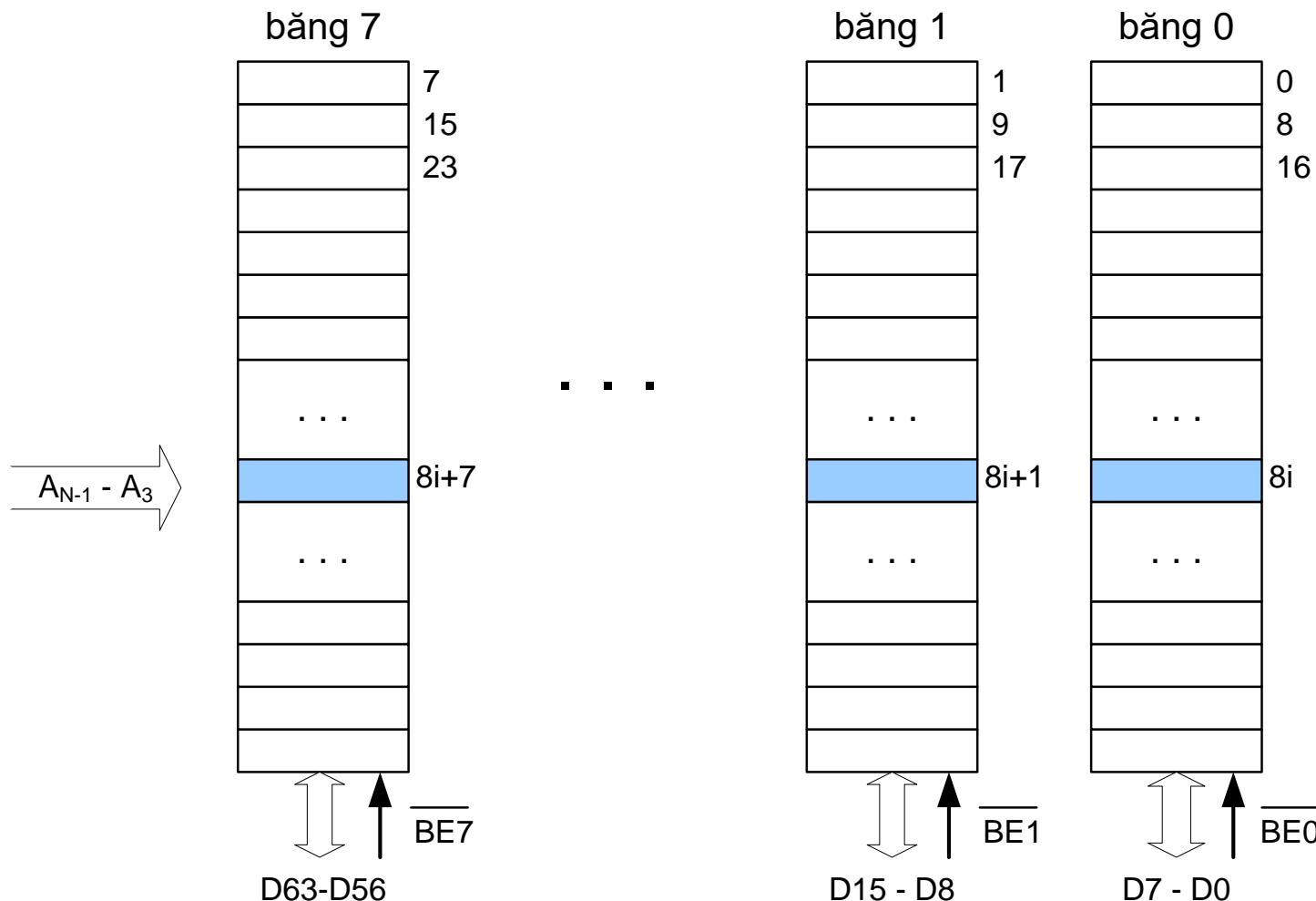
$m = 16\text{bit} \rightarrow \text{hai băng nhớ đan xen}$



$m = 32\text{bit} \rightarrow$ bốn băng nhớ đan xen



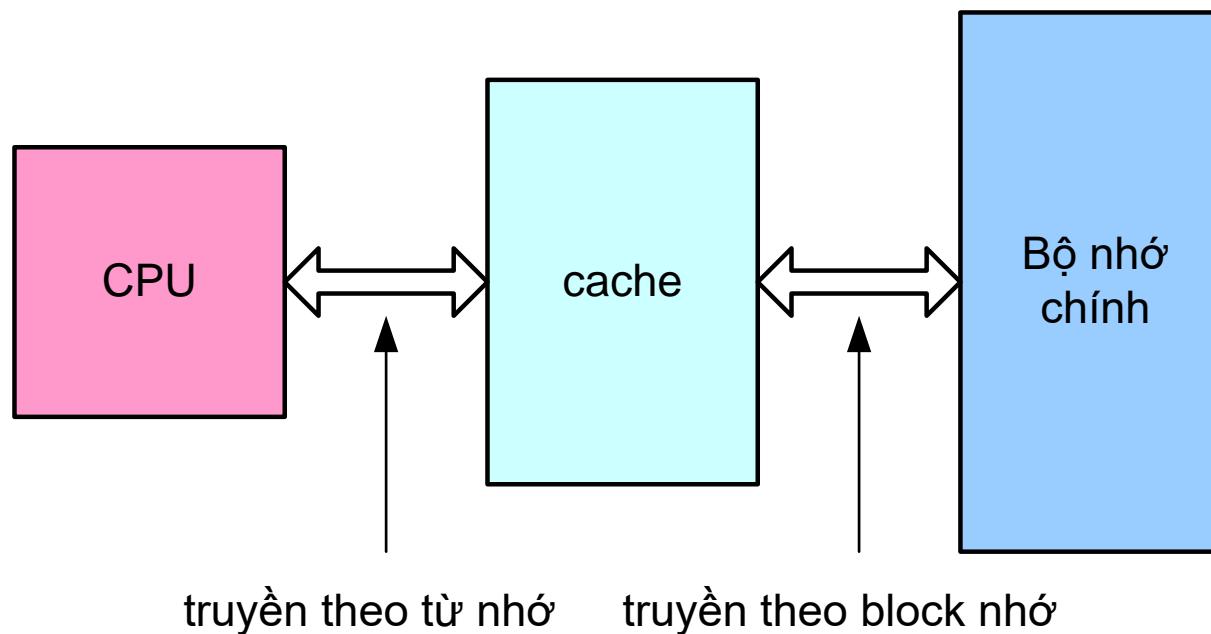
$m = 64\text{bit} \rightarrow$ tám bǎng nhó đan xen



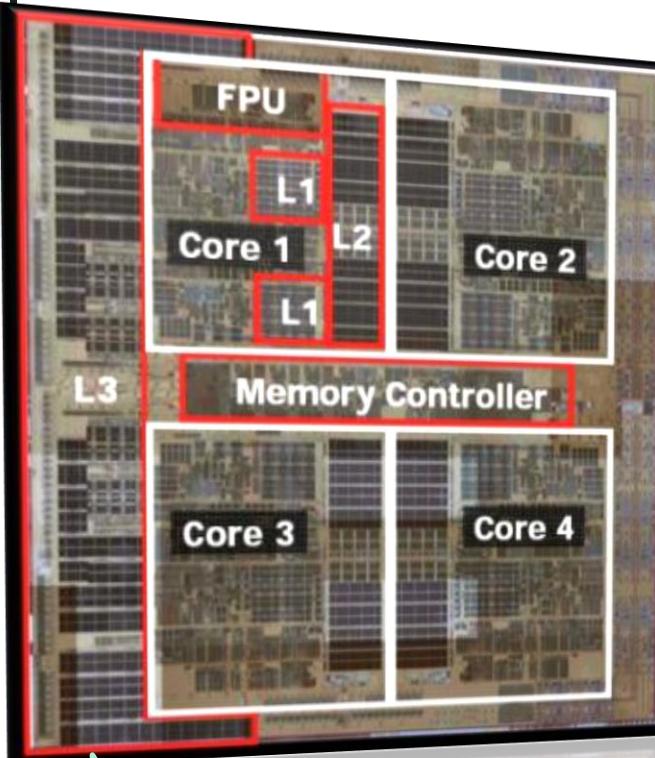
7.3. Bộ nhớ đệm nhanh (cache memory)

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



Lịch sử hình thành cache



AMD Athlon

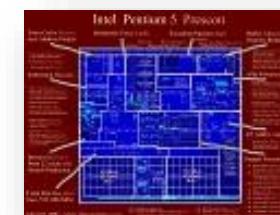
Cache chiếm
nhiều diện tích



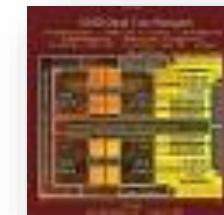
■ Intel Quad Core



■ Intel Core i7



■ Intel Pentium 5



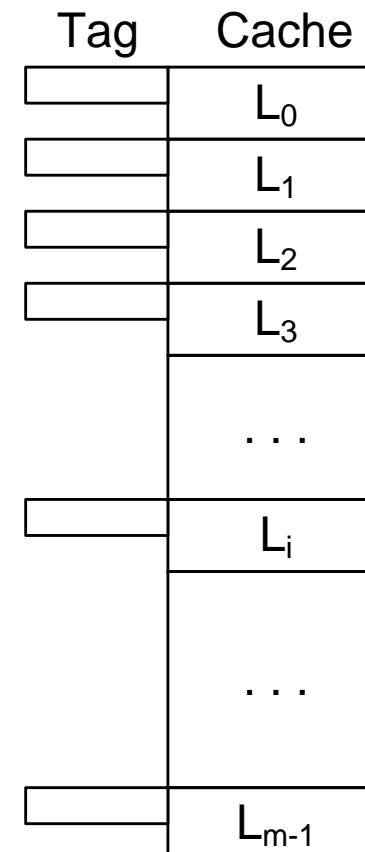
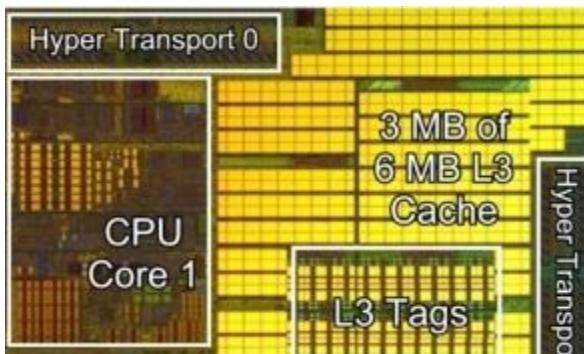
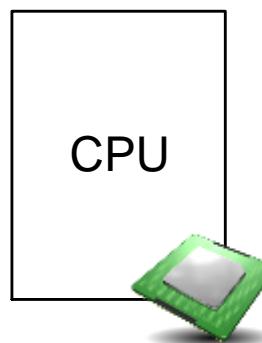
■ AMD Quad Core



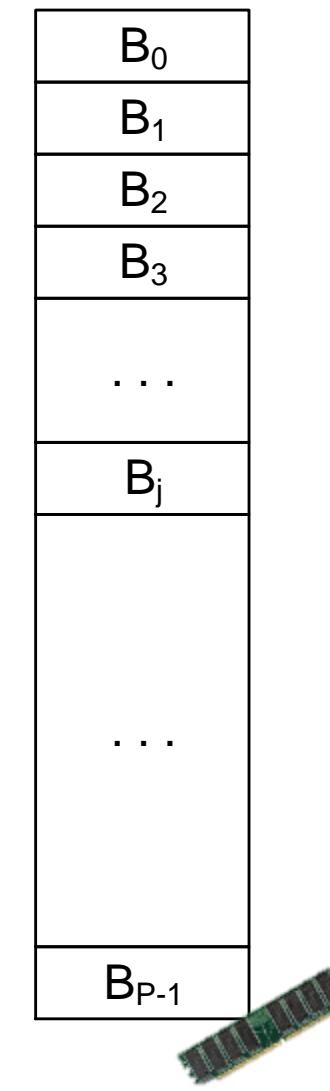
Ví dụ về thao tác của cache

- 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



Bộ nhớ chính



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

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

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

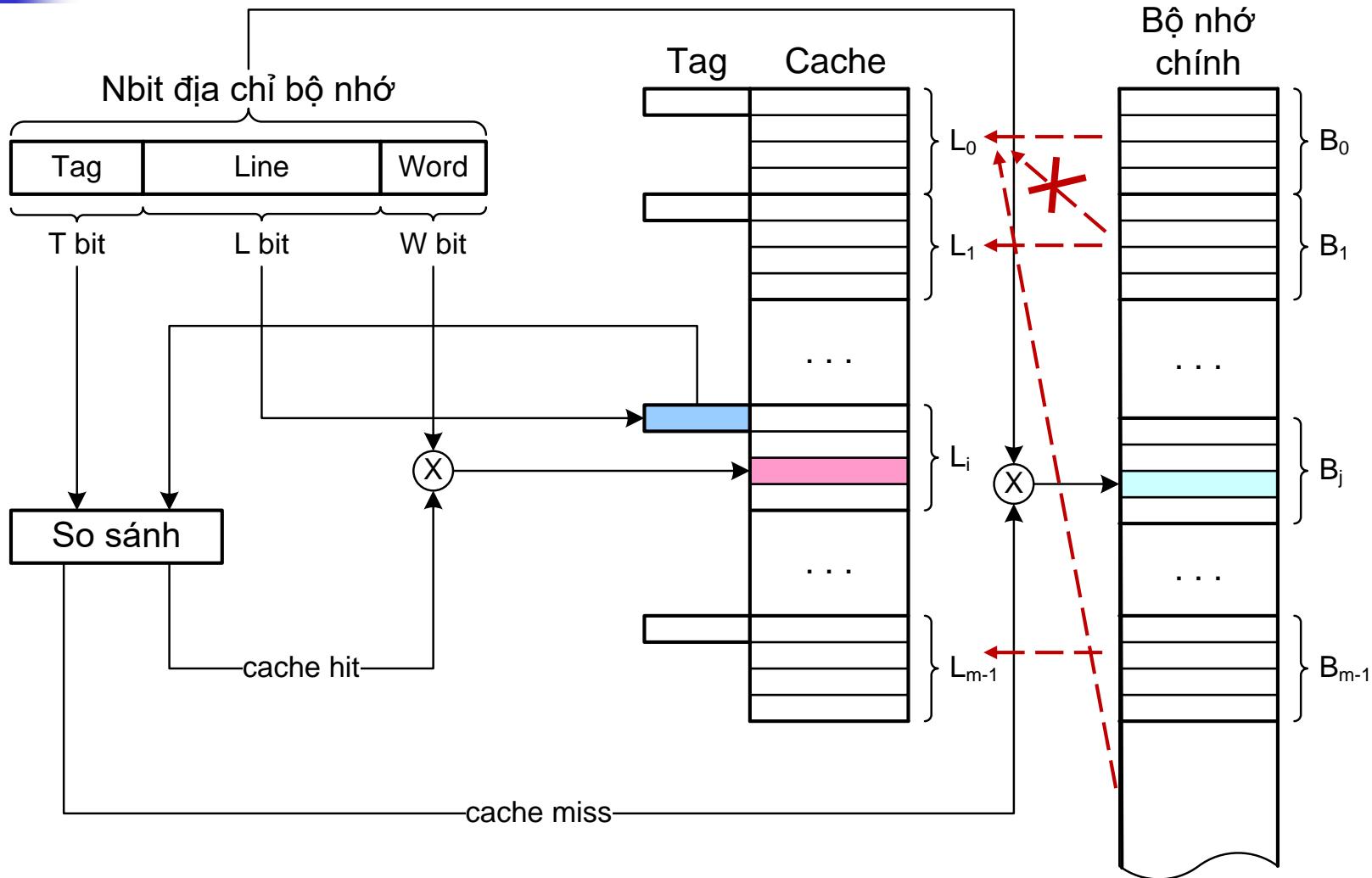
(Chính là các phương pháp tổ chức bộ nhớ cache)

- 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

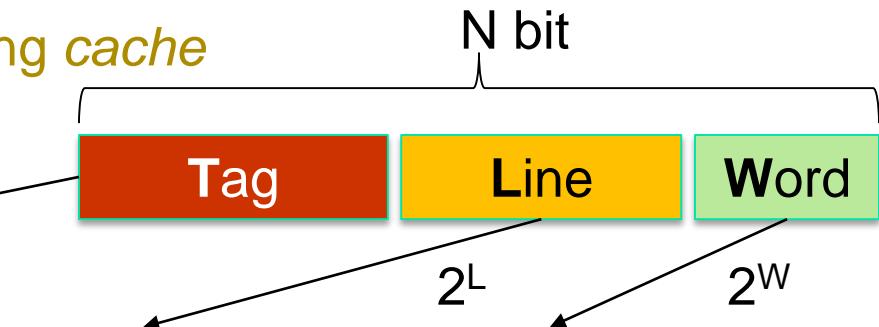
- Mỗi Block của bộ nhớ chính chỉ có thể được nạp vào một Line của cache:
 - $B_0 \rightarrow L_0$
 - $B_1 \rightarrow L_1$
 -
 - $B_{m-1} \rightarrow L_{m-1}$
 - $B_m \rightarrow L_0$
 - $B_{m+1} \rightarrow L_1$
 -
- Tổng quát
 - B_j chỉ có thể nạp vào $L_{j \bmod m}$
 - m là số Line của cache.

Minh họa ánh xạ trực tiếp



Đặc điểm của ánh xạ trực tiếp

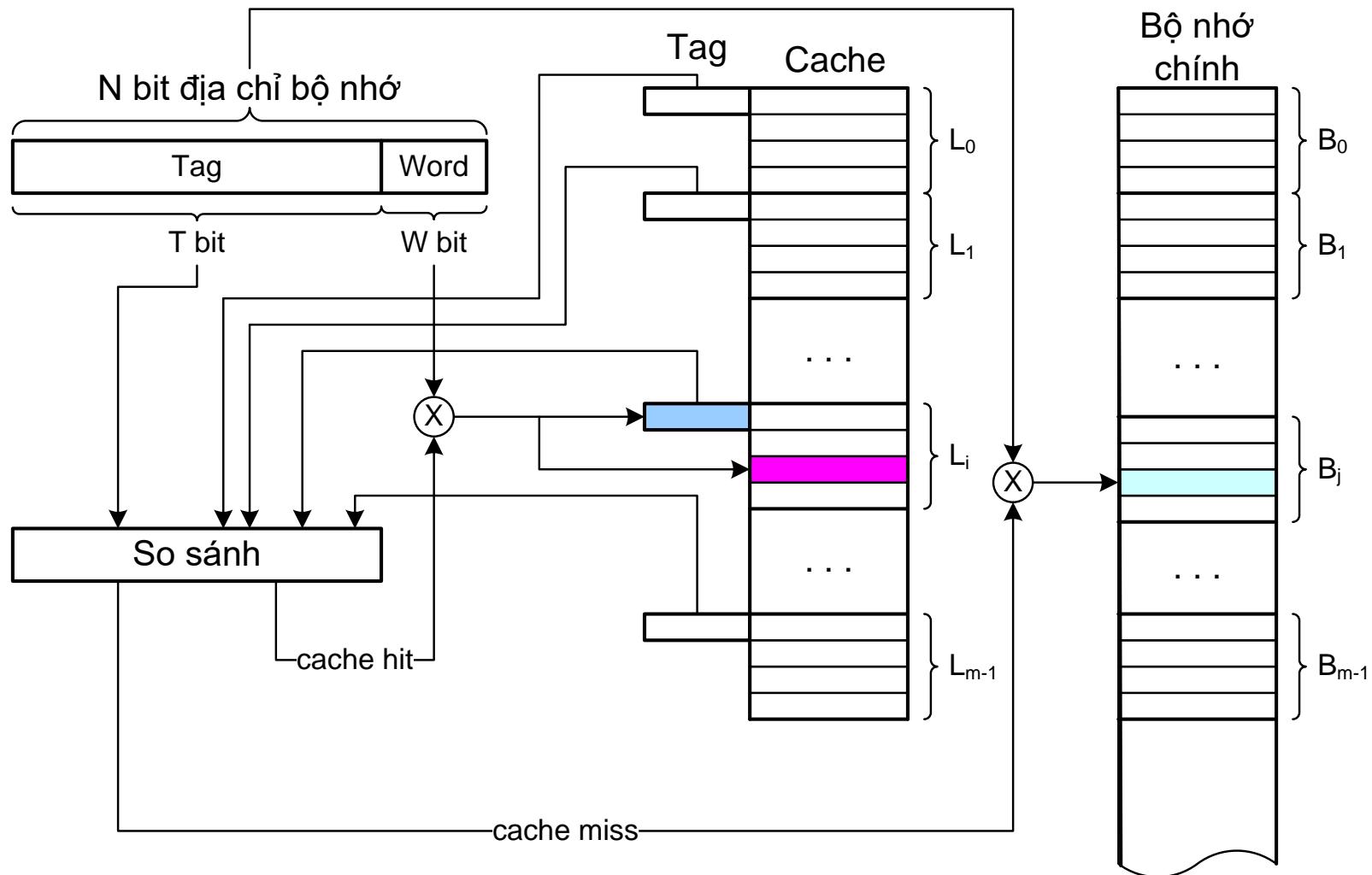
- Mỗi một địa chỉ N bit của bộ nhớ chính gồm ba trường:
 - Trường **Word** gồm W bit xác định một từ nhớ trong *Block* hay *Line*:
$$2^W = \text{kích thước của } Block \text{ hay } Line$$
 - Trường **Line** gồm L bit xác định một trong số các *Line* trong *cache*:
$$2^L = \text{số Line trong cache}$$
 - Trường **Tag** gồm T bit:
$$T = N - (W+L)$$
 - Mối quan hệ
$$\langle \text{Kích thước Cache} \rangle = \langle \text{Số line} \rangle * \langle \text{Kích thước line} \rangle$$
- Bộ so sánh đơn giản
- Xác suất *cache hit* thấp



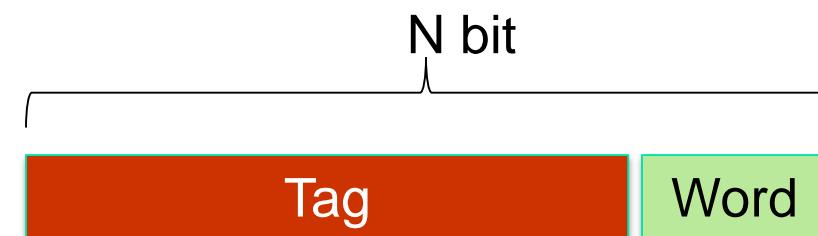
Ánh xạ liên kết toàn phần

- Mỗi *Block* có thể nạp vào bất kỳ *Line* nào của *cache*.
- Địa chỉ của bộ nhớ chính bao gồm hai trường:
 - Trường **Word** giống như trường hợp ở trên.
 - Trường **Tag** dùng để xác định *Block* của bộ nhớ chính.
- Tag xác định *Block* đang nằm ở *Line* đó

Minh họa ánh xạ liên kết toàn phần



Đặc điểm của ánh xạ liên kết toàn phần

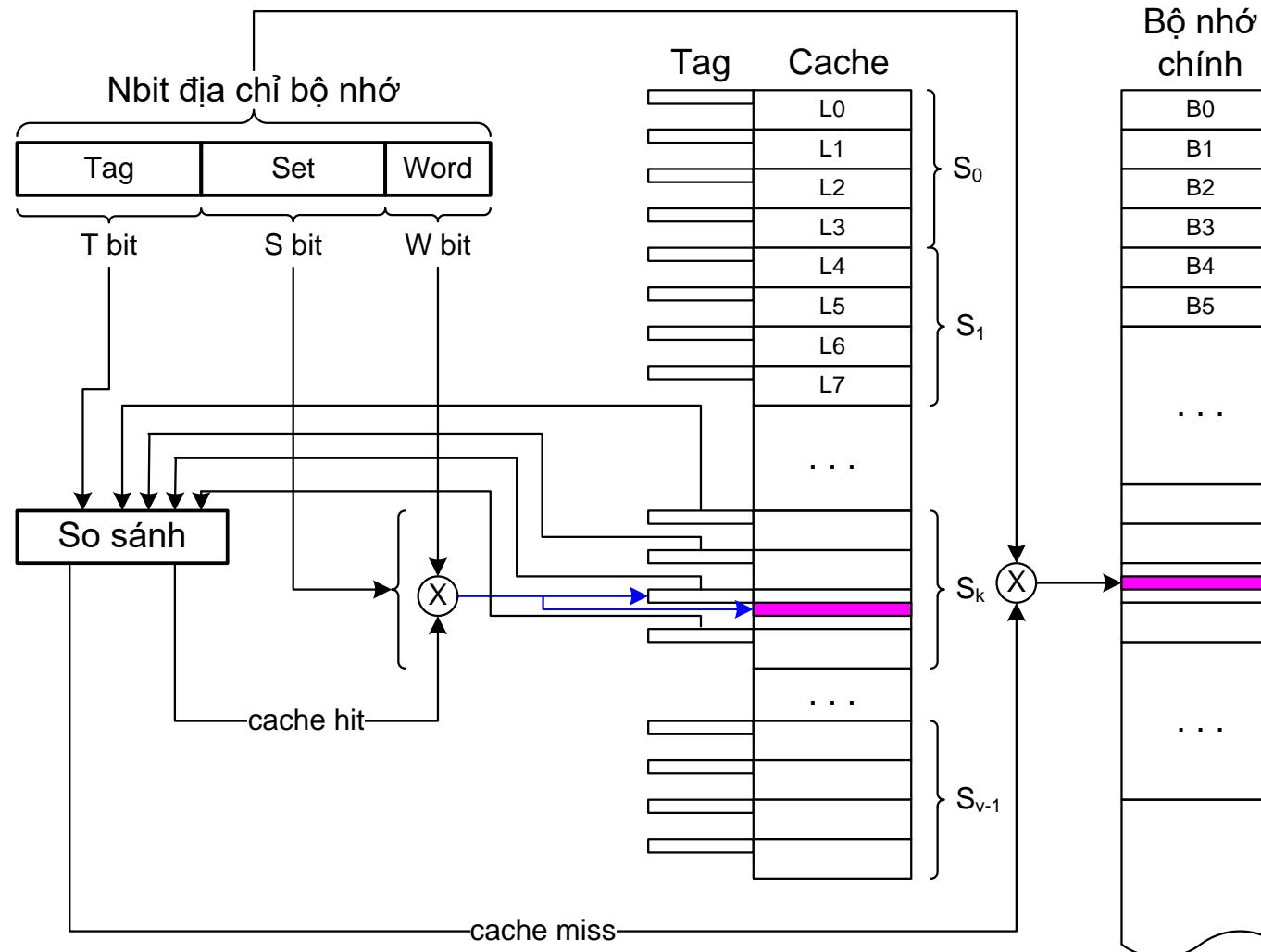


- Mọi quan hệ:
 $\text{<Kích thước Line>} = 2^{\text{<Số bit Word>}}$
- So sánh đồng thời với tất cả các Tag → mất nhiều thời gian
- Xác suất *cache hit* cao.
- Bộ so sánh phức tạp.

Ánh xạ liên kết tập hợp

- 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:
 - $B_0 \rightarrow S_0 \rightarrow$ line bất kì thuộc S_0
 - $B_1 \rightarrow S_1 \rightarrow$ line bất kì thuộc S_1
 - $B_2 \rightarrow S_2 \rightarrow$ line bất kì thuộc S_2
 - $B_3 \rightarrow S_3 \rightarrow$ line bất kì thuộc S_3
 - $B_4 \rightarrow S_0 \rightarrow$ line bất kì thuộc S_0

Minh họa ánh xạ liên kết tập hợp

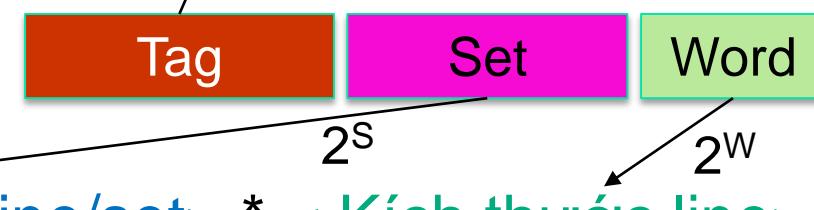


Đặc điểm của ánh xạ liên kết tập hợp

- Kích thước Block = 2^W Word
- Trường Set có S bit dùng để xác định một trong số $V = 2^S$ Set
- Trường Tag có T bit: $T = N - (W+S)$
- Mọi quan hệ

$\langle \text{Kích thước cache} \rangle =$

$$\langle \text{Số set} \rangle * \langle \text{Số line/set} \rangle * \langle \text{Kích thước line} \rangle$$
- Tổng quát cho cả hai phương pháp trên
- Thông thường 2,4,8,16 Lines/Set



Ví dụ về ánh xạ địa chỉ

- 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 = $4\text{GiB} = 2^{32}$ byte $\rightarrow N = 32$ bit
- $Line = 32$ byte $= 2^5$ byte $\rightarrow W = 5$ bit
- $Cache = 256 \text{ KiB} = 2^{18}$ byte.
- Số Line trong cache $= 2^{18} / 2^5 = 2^{13}$ Line
 $\rightarrow L = 13$ bit
- $T = N - (W + L) = 32 - (13 + 5) = 14$ bit

Tag 14 bit	Line 13 bit	Word 5 bit
---------------	----------------	---------------

Với ánh xạ liên kết toàn phần

- Bộ nhớ chính = $4\text{GiB} = 2^{32}$ byte $\rightarrow N = 32$ bit
- $Line = 32$ byte $= 2^5$ byte $\rightarrow W = 5$ bit
- Số bit của trường Tag sẽ là: $T = 32 - 5 = 27$ bit

Tag 27 bit	Word 5 bit
---------------	---------------

Với ánh xạ liên kết tập hợp 4 đường

- Bộ nhớ chính = $4\text{GiB} = 2^{32}$ byte $\rightarrow N = 32$ bit
- $Line = 32$ byte = 2^5 byte $\rightarrow W = 5$ bit
- Số $Line$ trong $cache = 2^{18}/ 2^5 = 2^{13}$ $Line$
- Một Set có 4 $Line$ = 2^2 $Line$
 \rightarrow số Set trong $cache = 2^{13}/ 2^2 = 2^{11}$ $Set \rightarrow S = 11$ bit
- Số bit của trường Tag sẽ là: $T = 32 - (11 + 5) = 16$ bit

Tag 16 bit	Set 11 bit	Word 5 bit
---------------	---------------	---------------

Bài tập

Giả thiết rằng máy tính có **128KiB cache** tổ chức theo kiểu ánh xạ **liên kết tập hợp 4-line**. Cache có tất cả là **1024 Set** từ S0 đến S1023. Địa chỉ bộ nhớ chính là **32-bit** và đánh địa chỉ cho từng byte.

- Tính số bit cho các trường địa chỉ khi truy nhập cache ?
- Xác định byte nhớ có địa chỉ $003D02AF_{(16)}$ được ánh xạ vào Set nào của cache ?

Đáp án

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

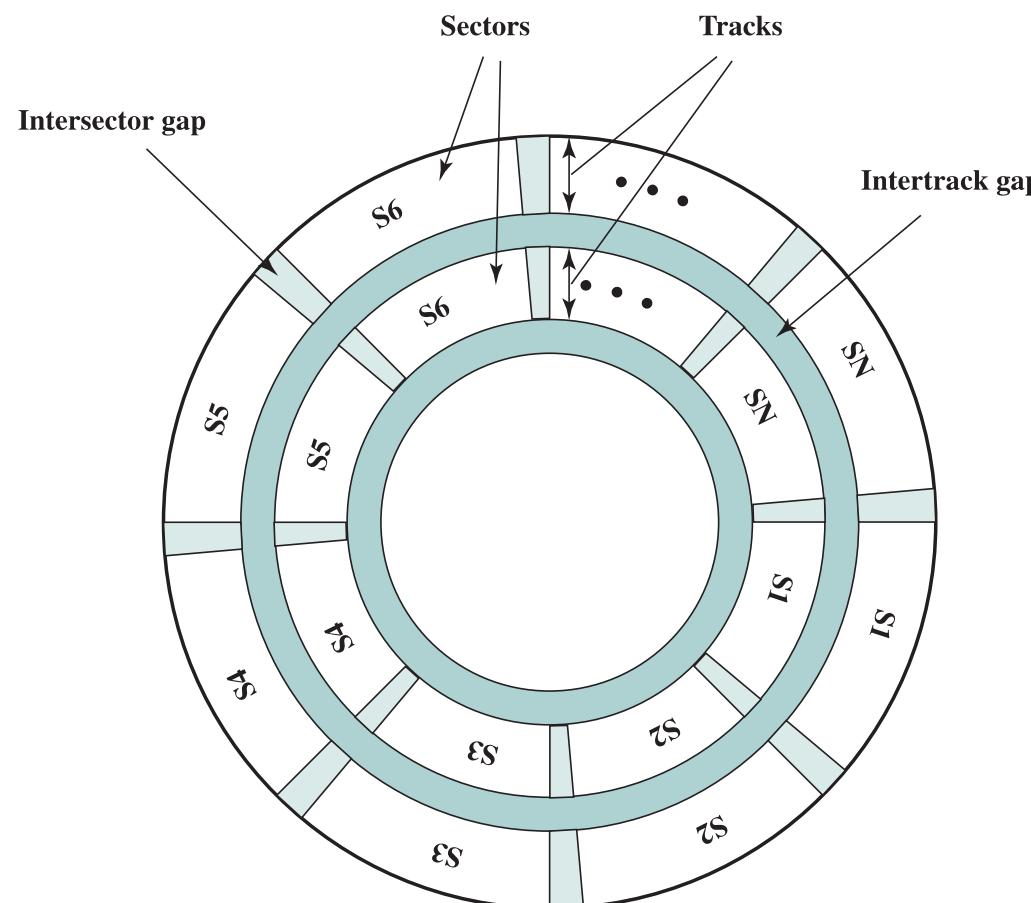
- Ghi xuyên qua (Write-through):
 - ghi cả cache và cả bộ nhớ chính
 - tốc độ chậm
- Ghi trả sau (Write-back):
 - chỉ ghi ra cache
 - tốc độ nhanh
 - khi Block trong cache bị thay thế cần phải ghi trả cả Block về bộ nhớ chính

7.4. Bộ nhớ ngoài

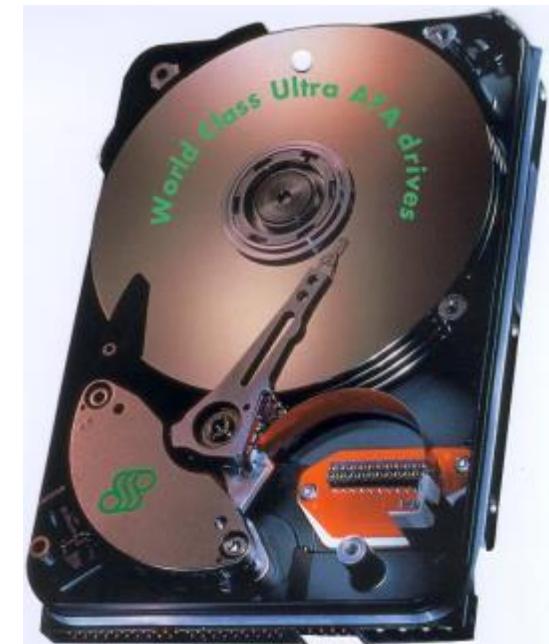
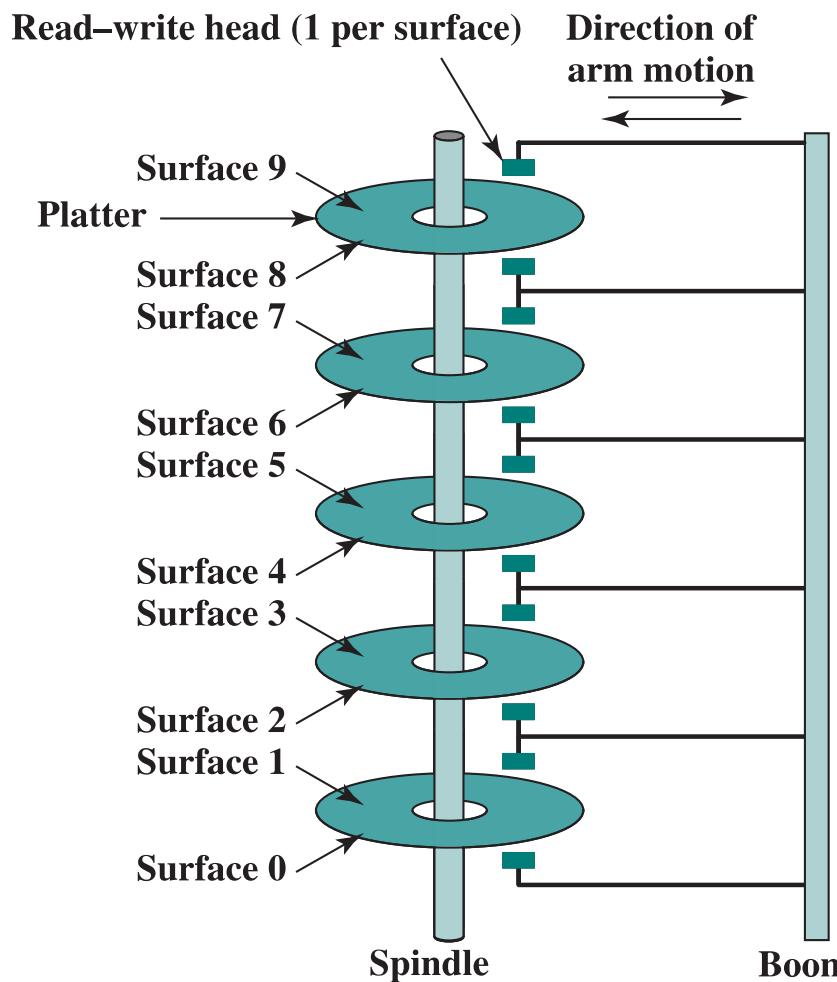
Các kiểu bộ nhớ ngoài

- Băng từ
- Đĩa từ
- Đĩa quang
- Bộ nhớ Flash

Đĩa tòn



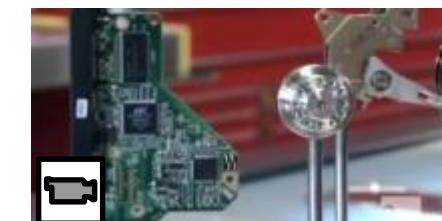
Nhiều đĩa



First Harddisk IBM

Ổ đĩa cứng

- Một hoặc nhiều đĩa
- Thông dụng
- Dung lượng tăng lên rất nhanh
 - 1993: ~ 200MiB
 - 2003: ~ 40GiB
 - 2013: ~ 500GiB - 1TiB
- Tốc độ đọc/ghi nhanh
- Rẻ tiền

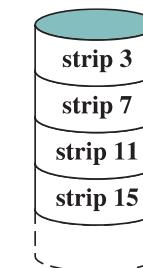
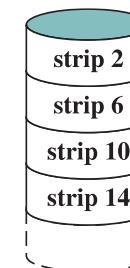
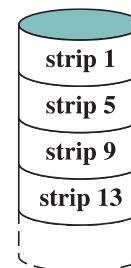
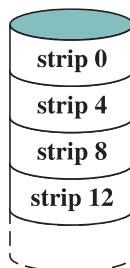


Inside a Hard Disc Drive

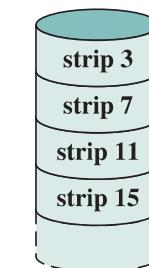
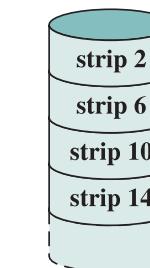
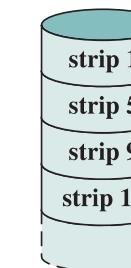
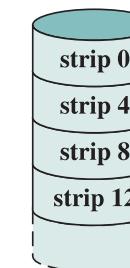
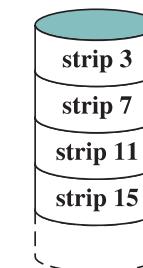
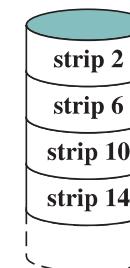
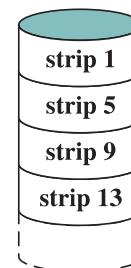
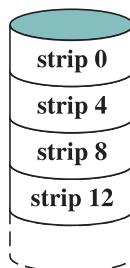
Hệ thống lưu trữ dung lượng lớn: RAID

- Redundant Array of Inexpensive Disks
- (Redundant Array of Independent Disks)
- Tập các 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
- 3 loại phổ biến RAID 0,1,5

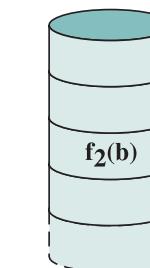
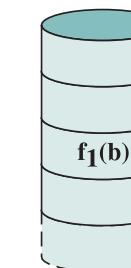
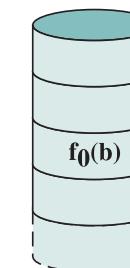
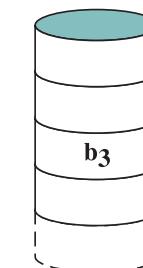
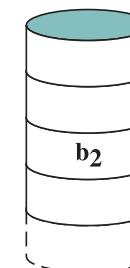
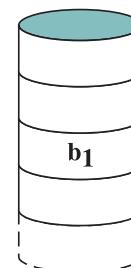
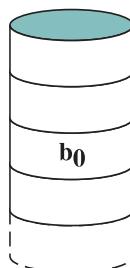
RAID 0, 1, 2



(a) RAID 0 (Nonredundant)

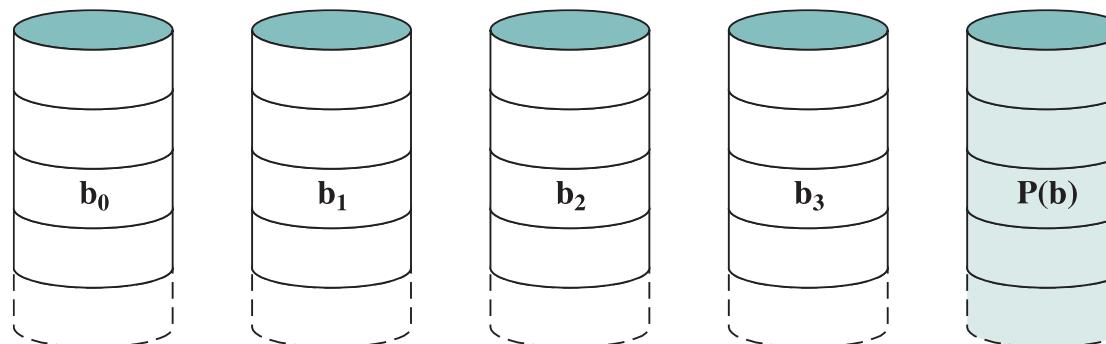


(b) RAID 1 (Mirrored)

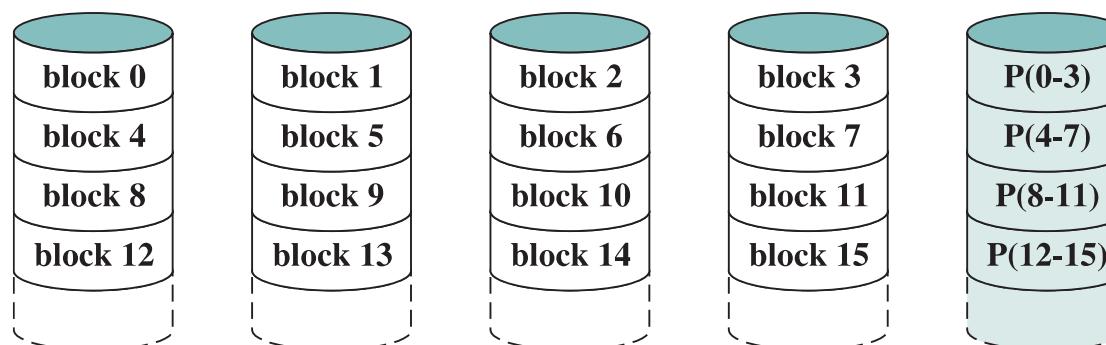


(c) RAID 2 (Redundancy through Hamming code)

RAID 3 & 4

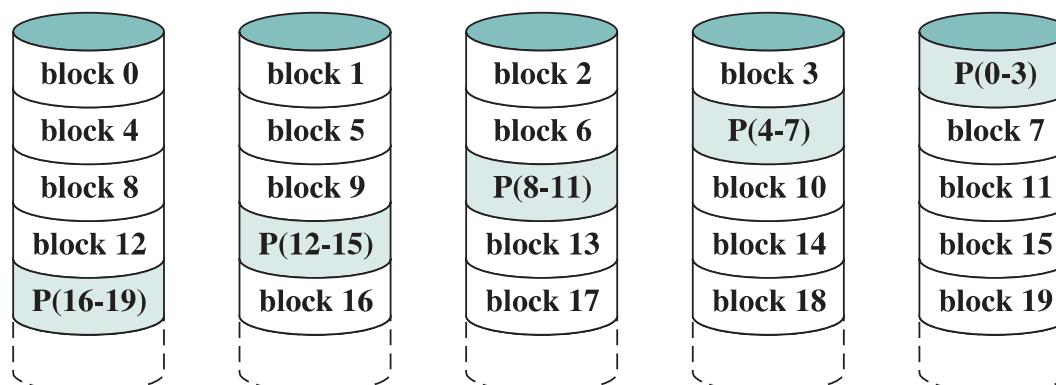


(d) RAID 3 (Bit-interleaved parity)

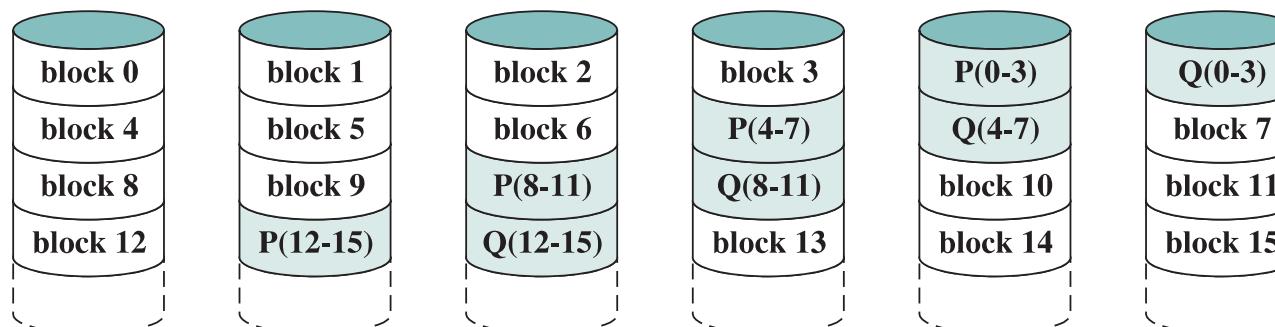


(e) RAID 4 (Block-level parity)

RAID 5 & 6

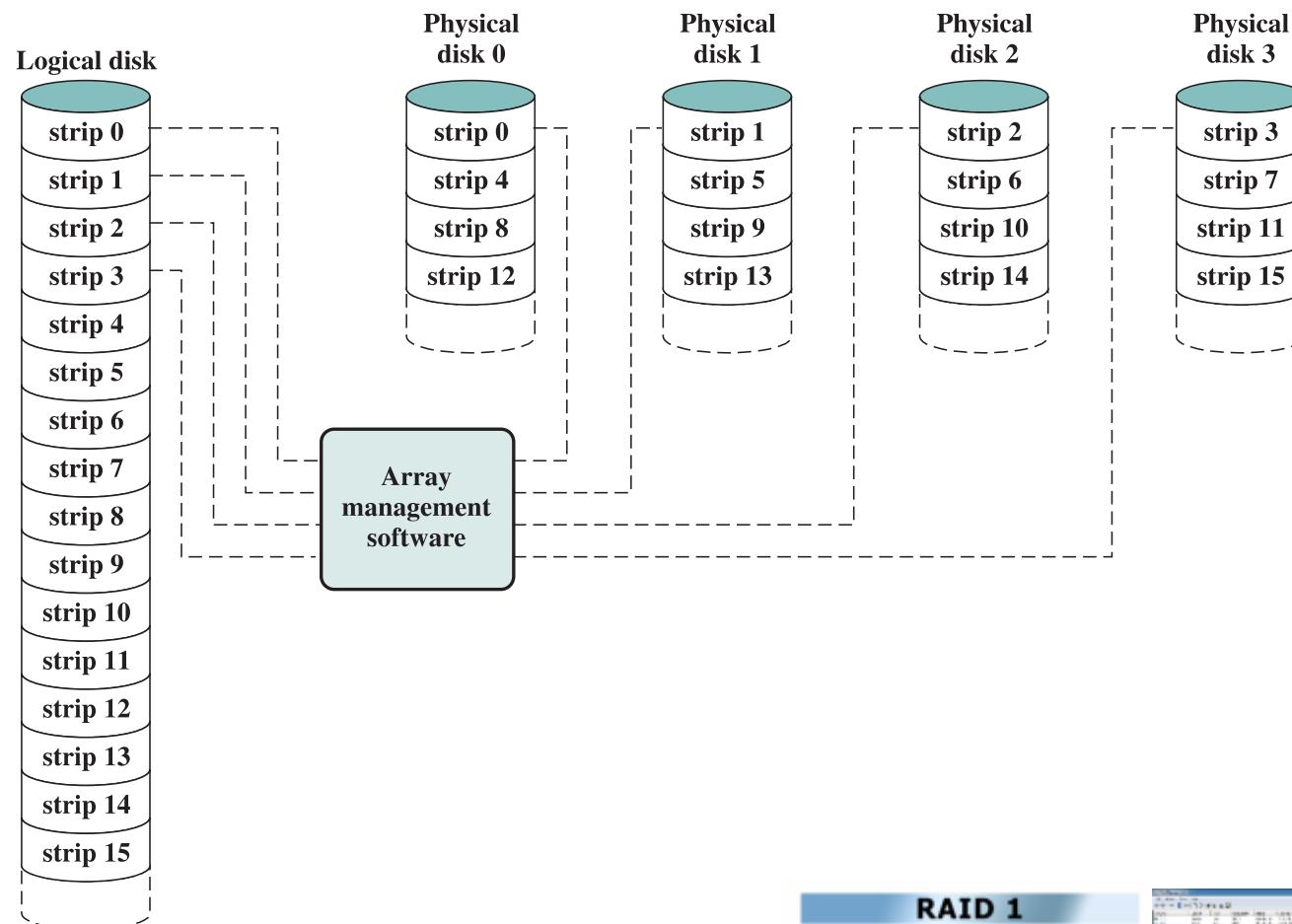


(f) RAID 5 (Block-level distributed parity)



(g) RAID 6 (Dual redundancy)

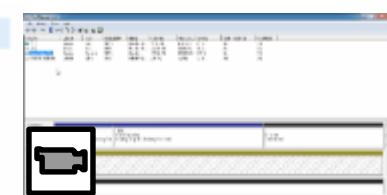
Ánh xạ dữ liệu của RAID 0



RAID 1



Explaining RAID



Dynamic Disks and Raid

Đĩa quang

- CD (Compact Disc)
 - Dung lượng thông dụng 650MiB
 - Tốc độ đọc cơ sở 150 KiB/s.
 - Tốc độ bội, ví dụ: 48x, 52x,...
- DVD
 - Digital Video Disc hoặc Digital Versatile Disk
 - Ghi một hoặc hai mặt
 - Một hoặc hai lớp trên một mặt
 - Thông dụng: 4,7GiB/lớp

Bộ nhớ flash

- Bộ nhớ bán dẫn
- Không khả biến
- Tốc độ nhanh
- Các dạng:
 - Ổ nhớ kết nối qua cổng USB
 - Thẻ nhớ
 - Ổ SSD (Solid State Drive): kết nối nhiều chip nhớ flash và cho phép truy cập song song



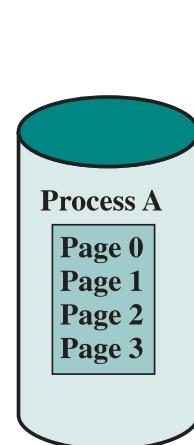
7.5. Bộ nhớ ảo (Virtual Memory)

- 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 = 4KiBytes
 - Kỹ thuật phân đoạn: Chia không gian nhớ thành các đoạn nhớ có kích thước thay đổi, các đoạn nhớ có thể gói 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
- HĐH 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

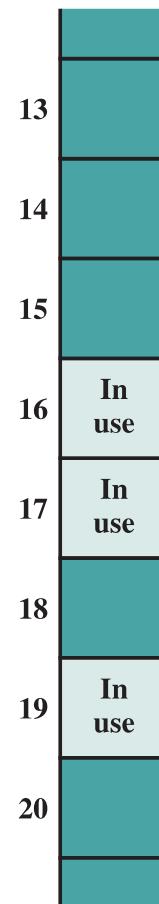
Cấp phát các khung trang



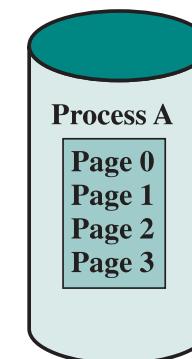
Free frame list

- 13
- 14
- 15
- 18
- 20

Main memory



(a) Before



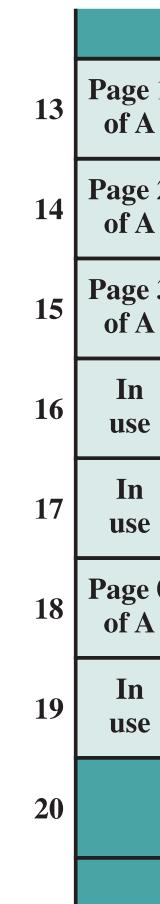
Free frame list

- 20

Process A page table

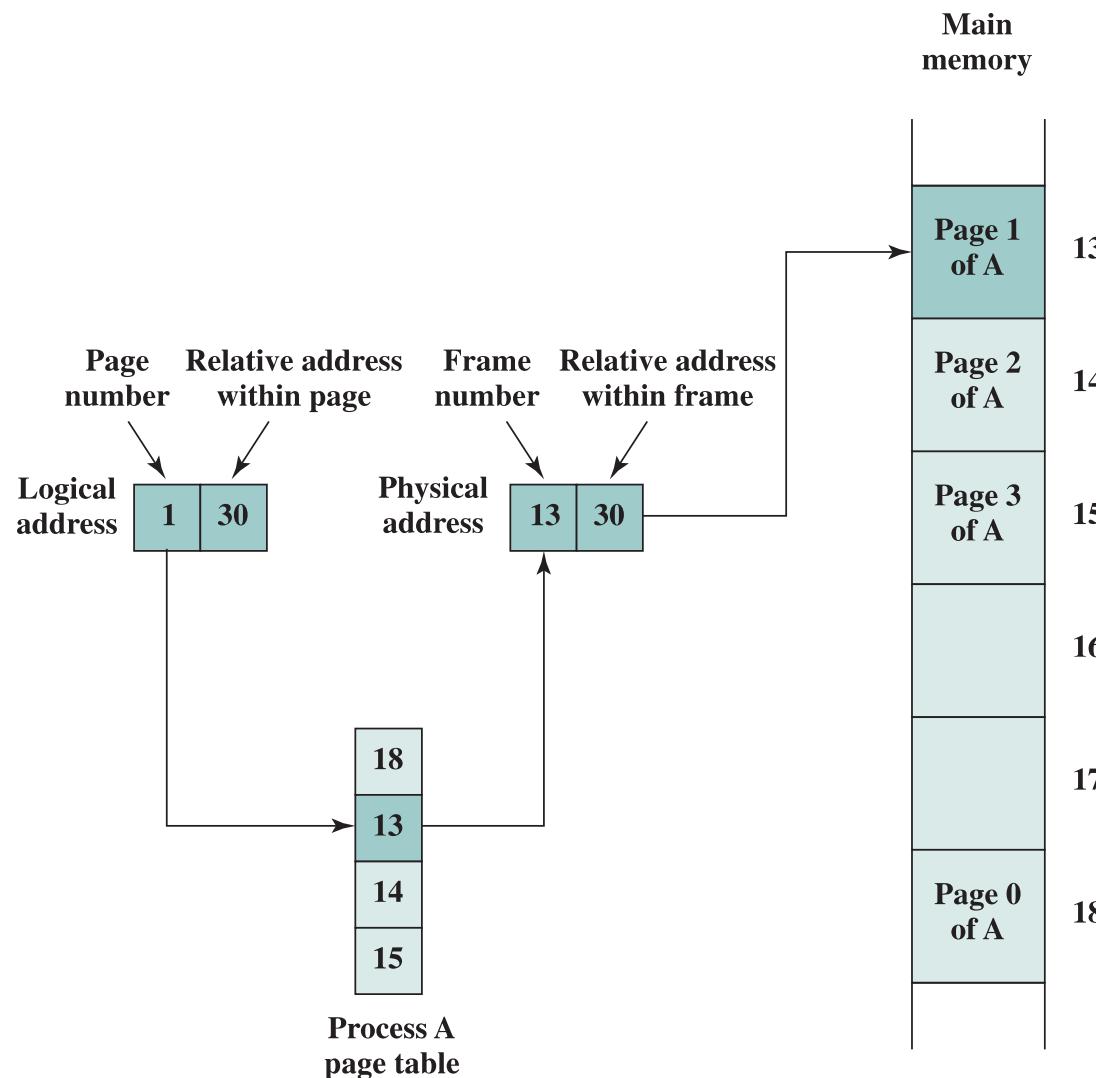
18
13
14
15

Main memory



(b) After

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

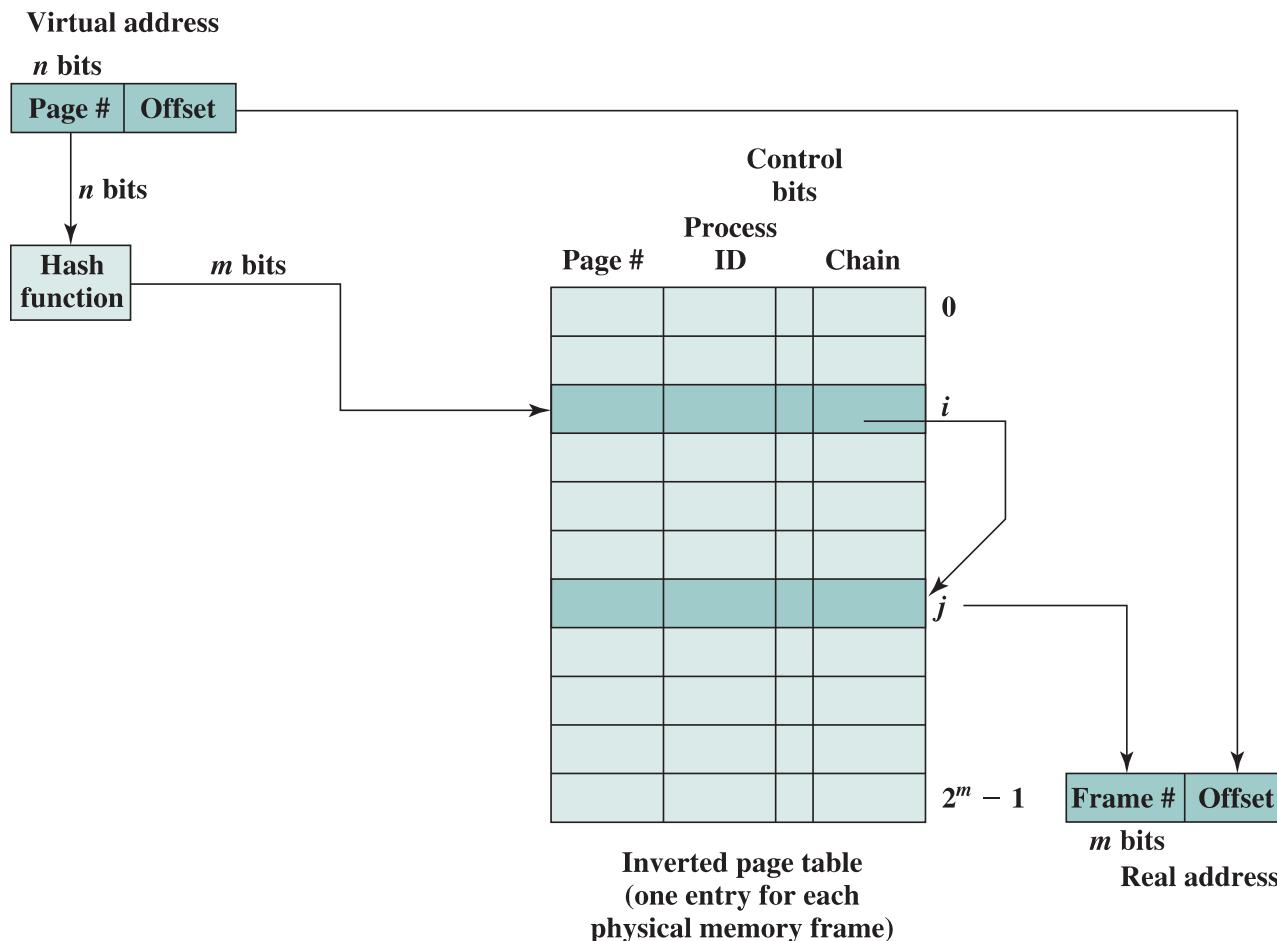
Thất bại

- Quá nhiều tiến trình trong bộ nhớ quá nhỏ
- HĐH 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

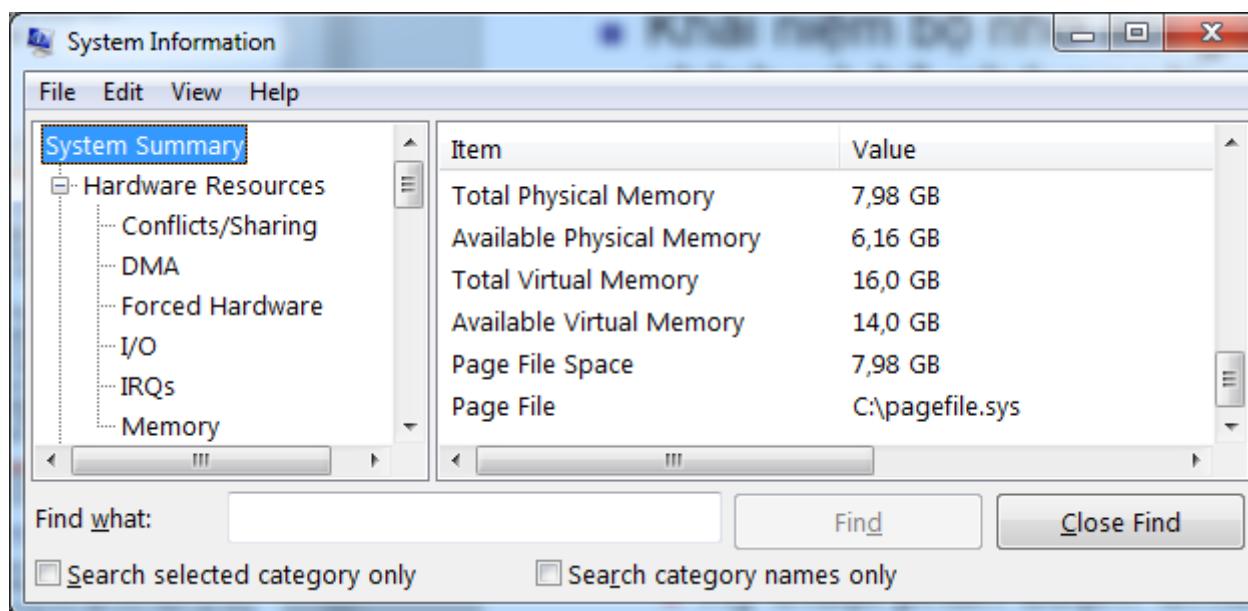
Lợi ích

- 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 ảo trong Windows OS

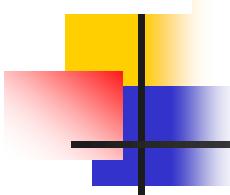


Bộ nhớ trên máy tính PC

- 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

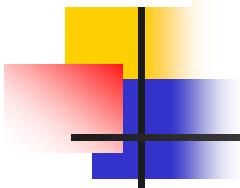


Hết chương 7

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

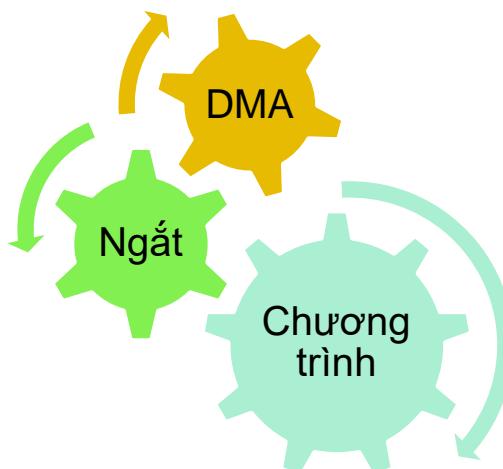
Chương 7. Bộ nhớ máy tính

Chương 8. Hệ thống vào-ra

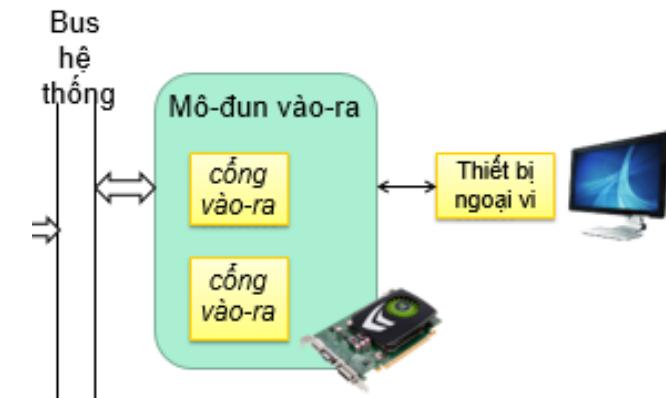
Chương 9. Các kiến trúc song song

Nội dung của chương 8

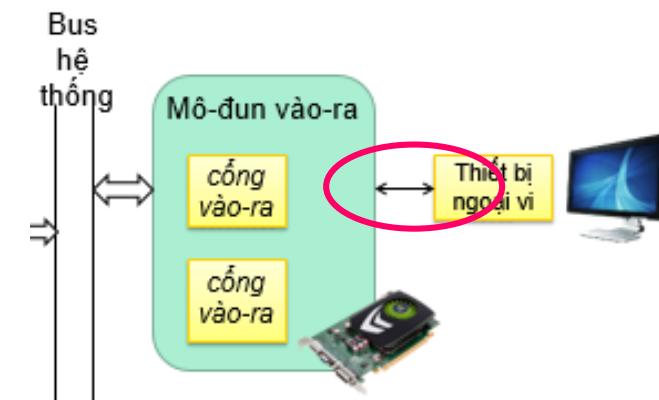
8.1. Tổng quan về hệ thống vào-ra



8.3. Nối ghép thiết bị ngoại vi



8.2. Các phương pháp điều khiển vào-ra

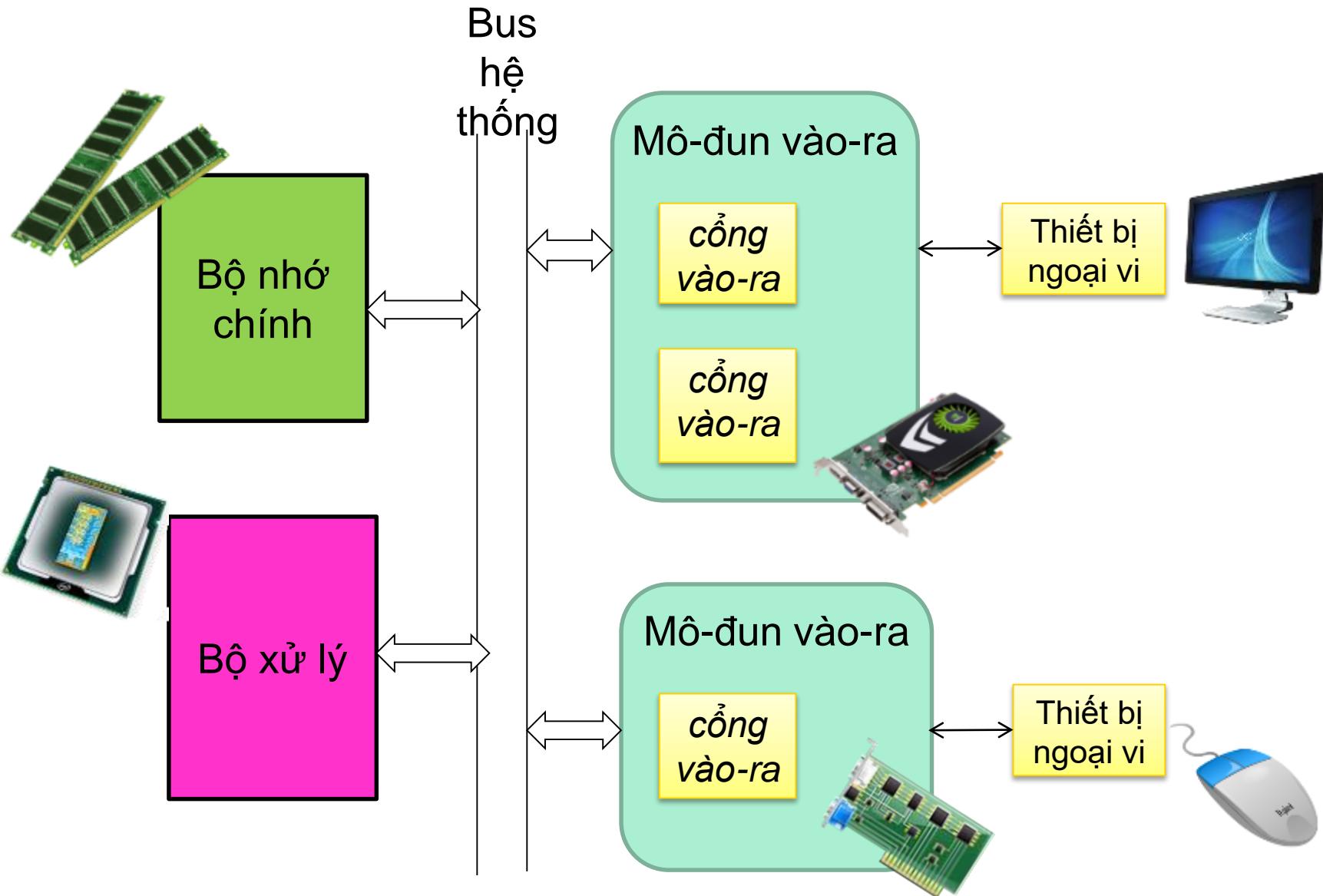


8.1. Tổng quan về hệ thống vào-ra

1. Giới thiệu chung

- Chức năng của hệ thống vào-ra: 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 ngoại vi
 - Các mô-đun vào-ra

Cấu trúc cơ bản của hệ thống vào-ra



Đặc điểm của hệ thống vào-ra

- Tồn tại đa dạng các thiết bị ngoại vi khác nhau về:
 - Nguyên tắc hoạt động
 - Tốc độ
 - Khuôn dạng dữ liệu
- Tất cả các thiết bị ngoại vi đều chậm hơn CPU và RAM
→ Cần có các mô-đun vào-ra để nối ghép các thiết bị ngoại vi với CPU và bộ nhớ chính

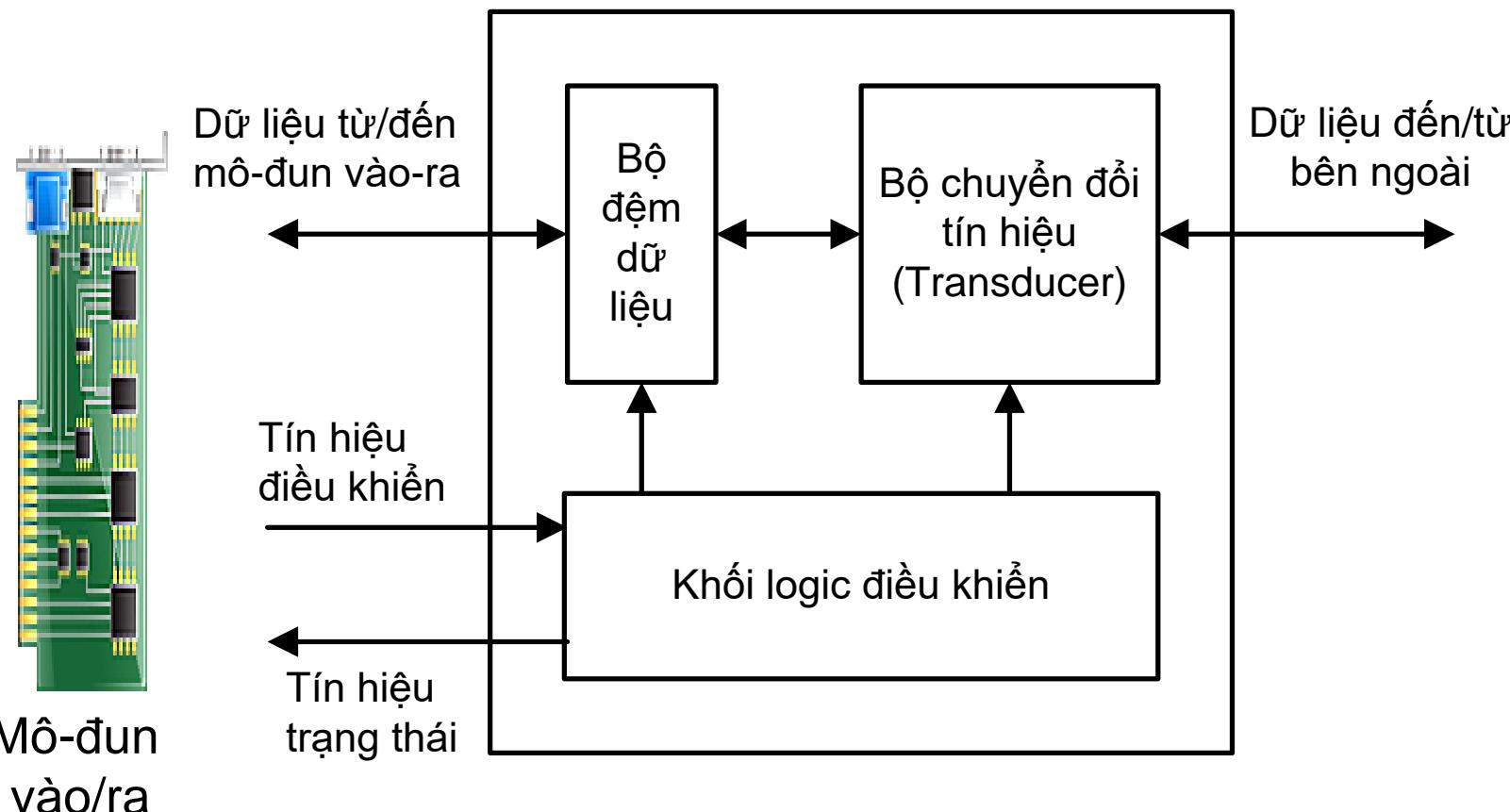
2. Các thiết bị ngoại vi

- Chức năng: chuyển đổi dữ liệu giữa bên trong và bên ngoài máy tính
- Phân loại:
 - Thiết bị ngoại vi giao tiếp người-máy: Bàn phím, Màn hình, Máy in,...
 - Thiết bị ngoại vi giao tiếp máy-máy: gồm các thiết bị theo dõi và kiểm tra
 - Thiết bị ngoại vi truyền thông: Modem, Network Interface Card (NIC)

Một số thiết bị ngoại vi

Device	Behavior	Partner	Data rate (Mbit/sec)
Keyboard	Input	Human	0.0001
Mouse	Input	Human	0.0038
Voice input	Input	Human	0.2640
Sound input	Input	Machine	3.0000
Scanner	Input	Human	3.2000
Voice output	Output	Human	0.2640
Sound output	Output	Human	8.0000
Laser printer	Output	Human	3.2000
Graphics display	Output	Human	800.0000–8000.0000
Cable modem	Input or output	Machine	0.1280–6.0000
Network/LAN	Input or output	Machine	100.0000–10000.0000
Network/wireless LAN	Input or output	Machine	11.0000–54.0000
Optical disk	Storage	Machine	80.0000–220.0000
Magnetic tape	Storage	Machine	5.0000–120.0000
Flash memory	Storage	Machine	32.0000–200.0000
Magnetic disk	Storage	Machine	800.0000–3000.0000

Cấu trúc chung của thiết bị ngoại vi



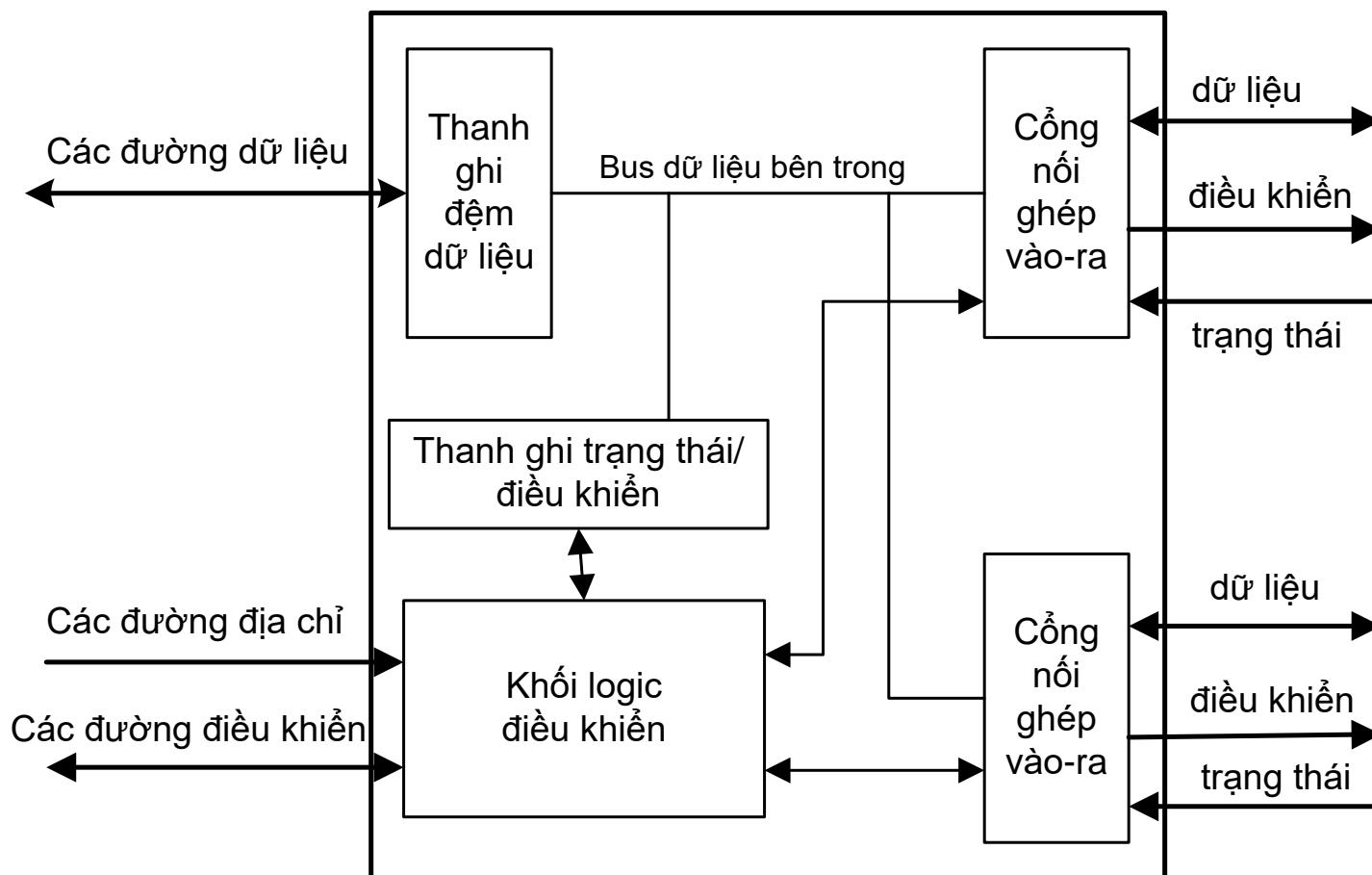
Các thành phần của thiết bị ngoại vi

- Bộ chuyển đổi tín hiệu: chuyển đổi dữ liệu giữa bên ngoài và bên trong máy tính
- Bộ đệm dữ liệu: đệm dữ liệu khi truyền giữa mô-đun vào-ra và thiết bị ngoại vi
- Khối logic điều khiển: điều khiển hoạt động của thiết bị ngoại vi đáp ứng theo yêu cầu từ mô-đun vào-ra

3. Mô-đun vào-ra

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

Cấu trúc chung của mô-đun vào-ra

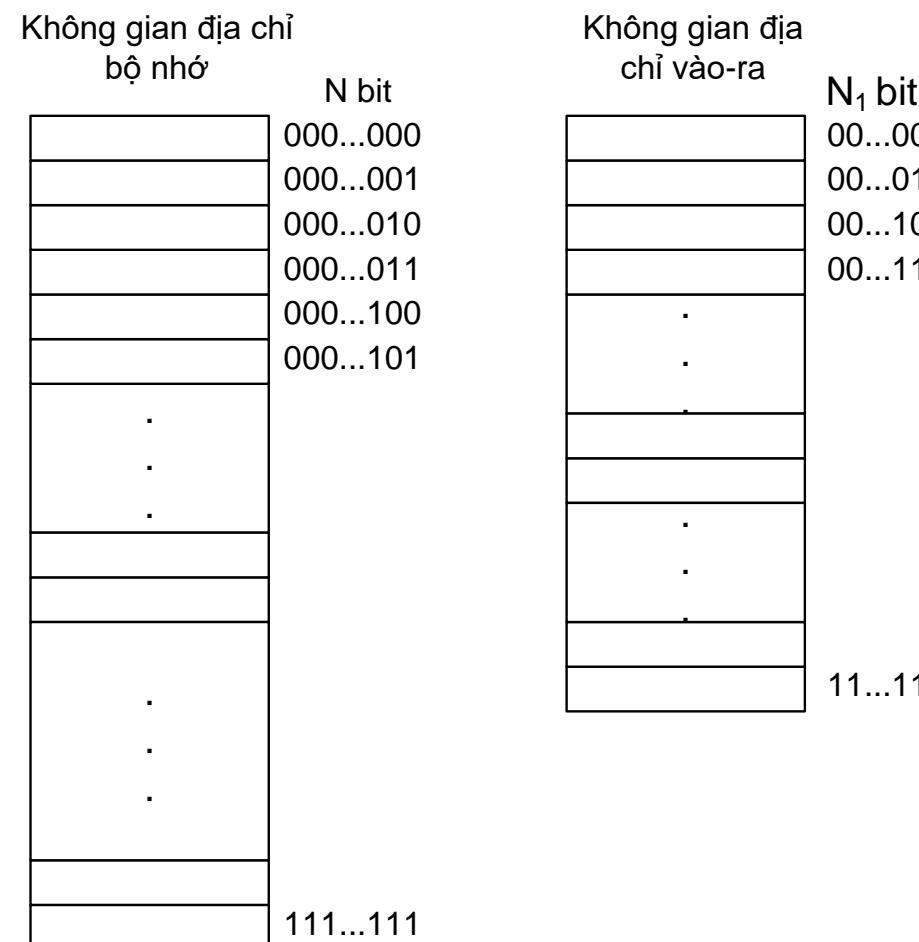


Các thành phần của mô-đun vào-ra

- Thanh ghi đệm dữ liệu: đệm dữ liệu trong quá trình trao đổi
- Các cổng vào-ra (I/O Port): kết nối với thiết bị ngoại vi, mỗi cổng có một địa chỉ xác định
- Thanh ghi trạng thái/điều khiển: lưu giữ thông tin trạng thái/điều khiển cho các cổng vào-ra
- Khối logic điều khiển: điều khiển mô-đun vào-ra

4. Địa chỉ hóa cổng vào-ra

Không gian địa chỉ của bộ xử lý

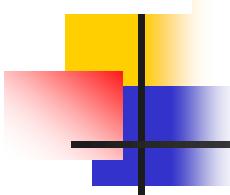


Không gian địa chỉ của bộ xử lý (tiếp)

- Một số bộ xử lý chỉ quản lý duy nhất một không gian địa chỉ:
 - không gian địa chỉ bộ nhớ: 2^N địa chỉ
- Ví dụ:
 - Các bộ xử lý 680x0 (Motorola)
 - Các bộ xử lý theo kiến trúc RISC: MIPS, ...

Không gian địa chỉ của bộ xử lý (tiếp)

- Một số bộ xử lý quản lý hai không gian địa chỉ tách biệt:
 - Không gian địa chỉ bộ nhớ: 2^N địa chỉ
 - Không gian địa chỉ vào-ra: 2^{N_1} địa chỉ
 - Có tín hiệu điều khiển phân biệt truy nhập không gian địa chỉ
 - Tập lệnh có các lệnh vào-ra chuyên dụng
- Ví dụ: Pentium (Intel)
 - không gian địa chỉ bộ nhớ = 2^{32} byte = 4GiB
 - không gian địa chỉ vào-ra = 2^{16} byte = 64KiB
 - Tín hiệu điều khiển M/I/O
 - Lệnh vào-ra chuyên dụng: IN, OUT



Các phương pháp địa chỉ hóa cỗng 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 dữ 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: 0xFFFFFFFF4
 - Cổng 2: 0xFFFFFFFF8
- Ghi giá trị 0x41 ra cổng 1

```
addi $t0, $0, 0x41      # đưa giá trị 0x41  
sw $t0, 0xFFFF4($0)    # ra cổng 1
```

Chú ý: giá trị 16-bit 0xFFFF4 được sign-extended thành 32-bit 0xFFFFFFFF
- Đọc dữ liệu từ cổng 2 đưa vào \$t3

```
lw $t3, 0xFFFF8($0) # đọc dữ liệu cổng 2 đưa vào $t3
```

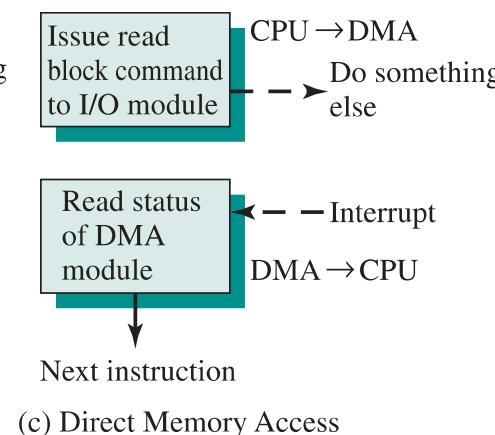
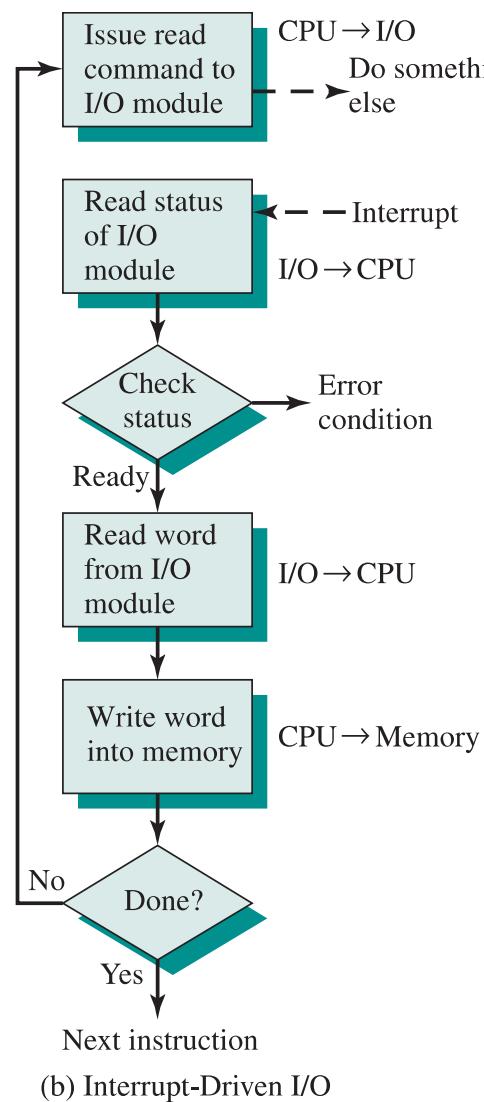
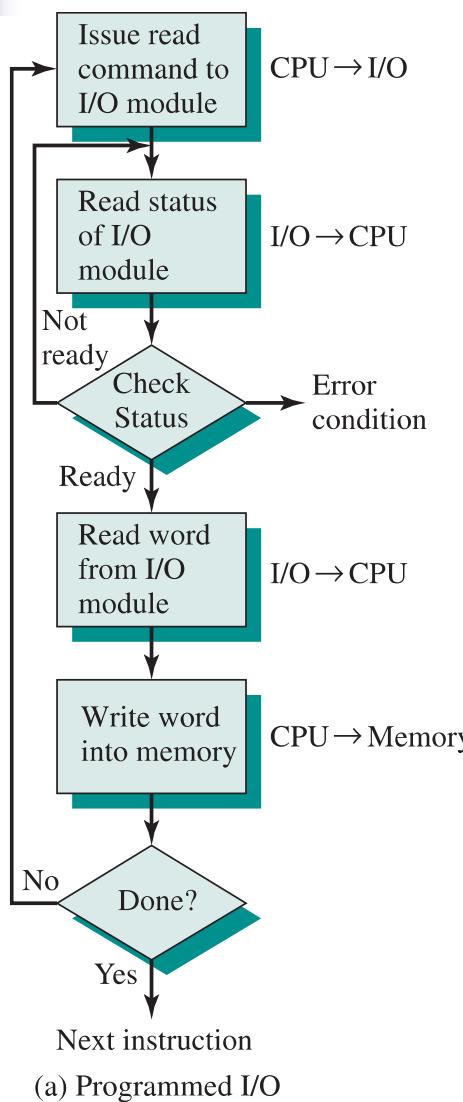
Vào-ra riêng biệt

- 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)

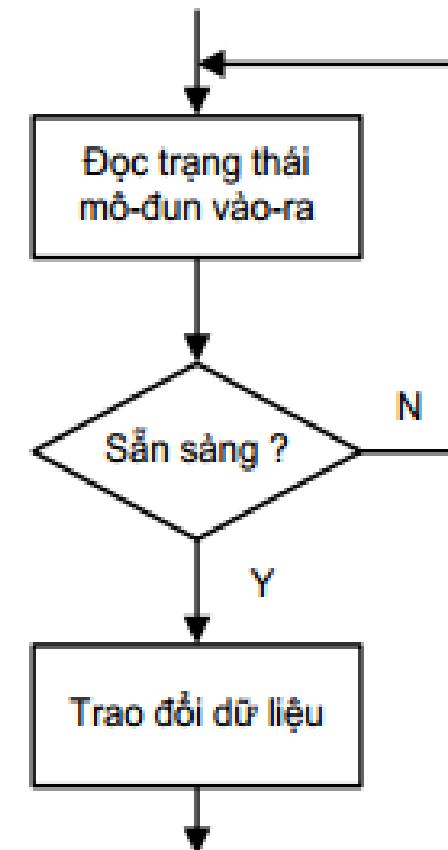
Ba kỹ thuật thực hiện vào một khối dữ liệu



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

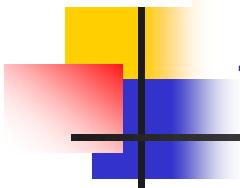


Các tín hiệu điều khiển vào-ra

- 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 ngoại vi
- Tín hiệu điều khiển **đọc** (*Read*): yêu cầu mô-đun vào-ra nhận dữ liệu từ thiết bị ngoại vi và đưa vào thanh ghi 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)



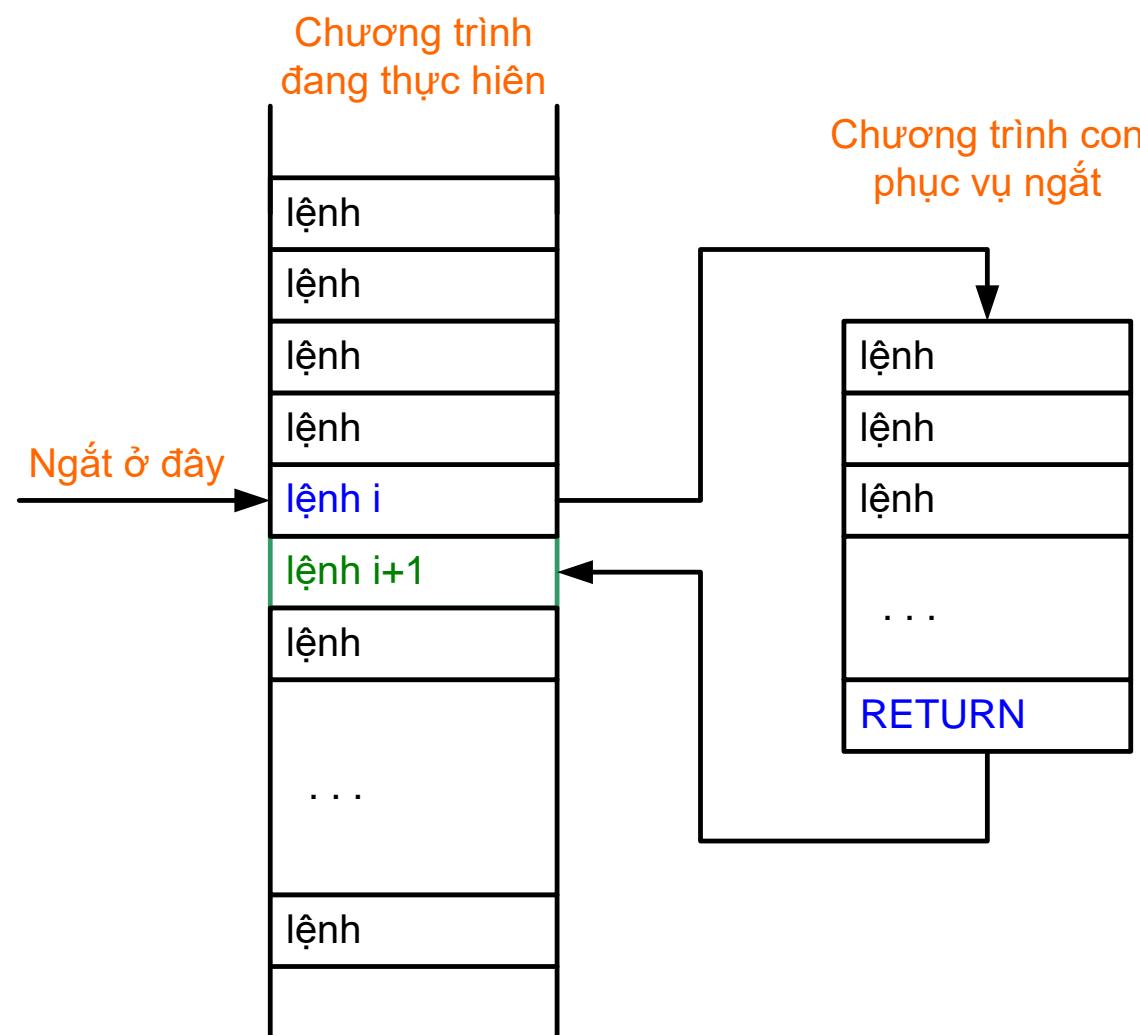
Đặc điểm

- Vào-ra do ý muốn của người lập trình
- CPU trực tiếp điều khiển trao đổi dữ liệu giữa CPU với module vào-ra
- CPU đợi mô-đun vào-ra → 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 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 ngoại vi, 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 ngắt
- Nếu bị ngắt:
 - Cắt ngũ cảnh (nội dung các thanh ghi)
 - Thực hiện chương trình con ngắt để vào dữ liệu
 - Khôi phục ngũ cảnh của chương trình đang thực hiện

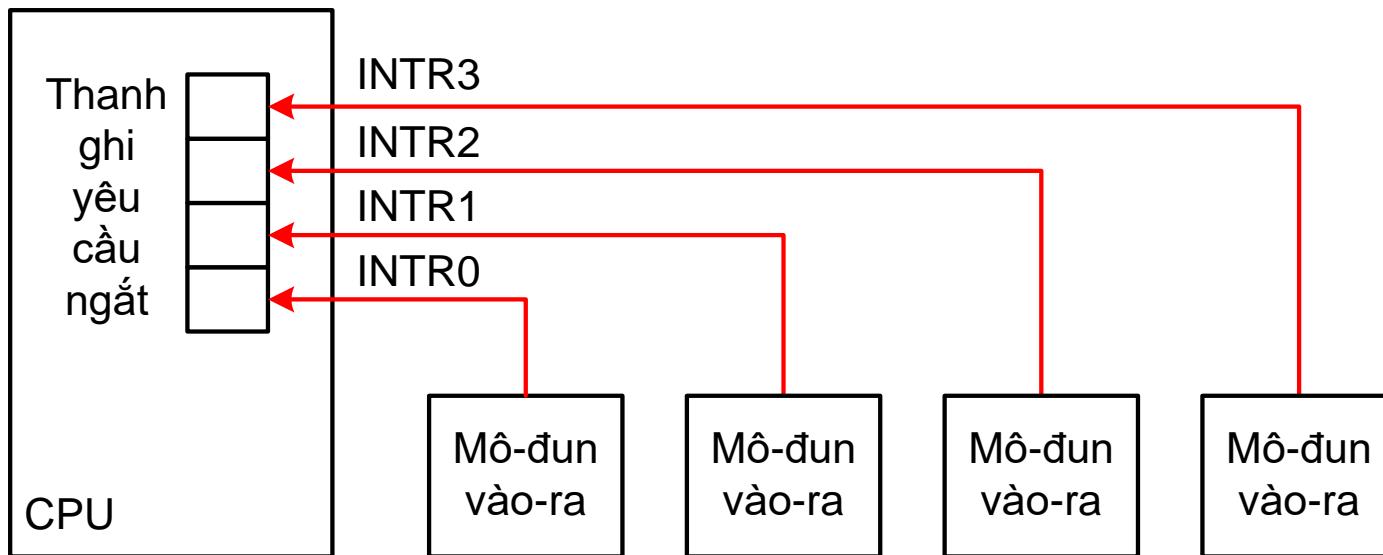
Các vấn đề nảy sinh khi thiết kế

- 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 xảy ra ?

Các phương pháp nối ghép ngắt

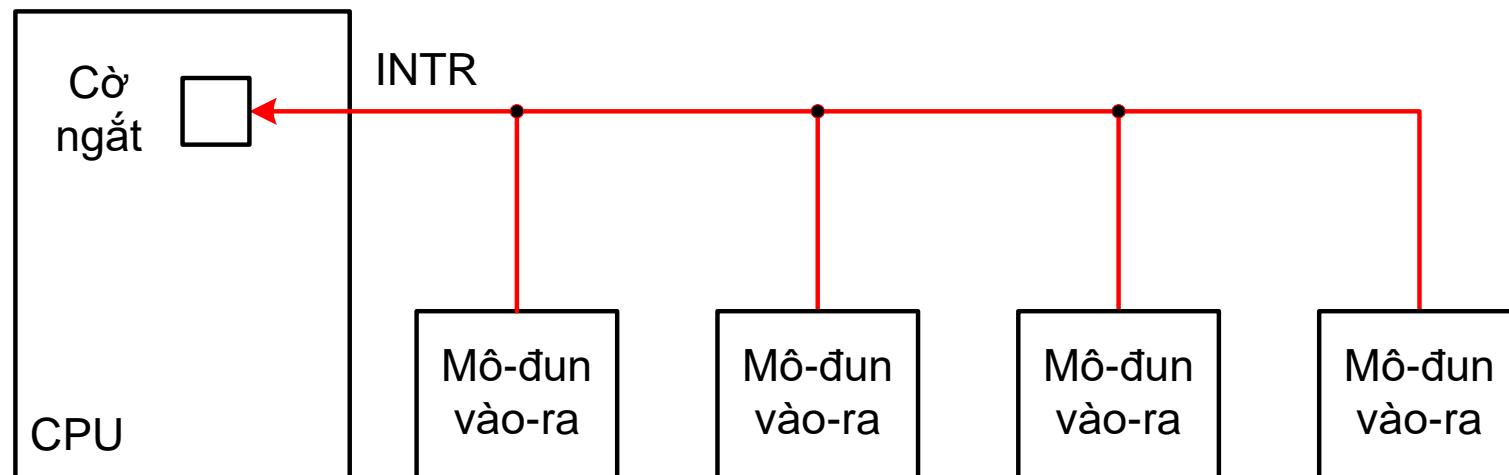
- Sử dụng nhiều đường yêu cầu ngắt
- Hỏi vòng bằng phần mềm (Software Poll)
- Hỏi vòng bằng phần cứng (Daisy Chain or Hardware Poll)
- Sử dụng bộ điều khiển ngắt (PIC)

Nhiều đường yêu cầu ngắt



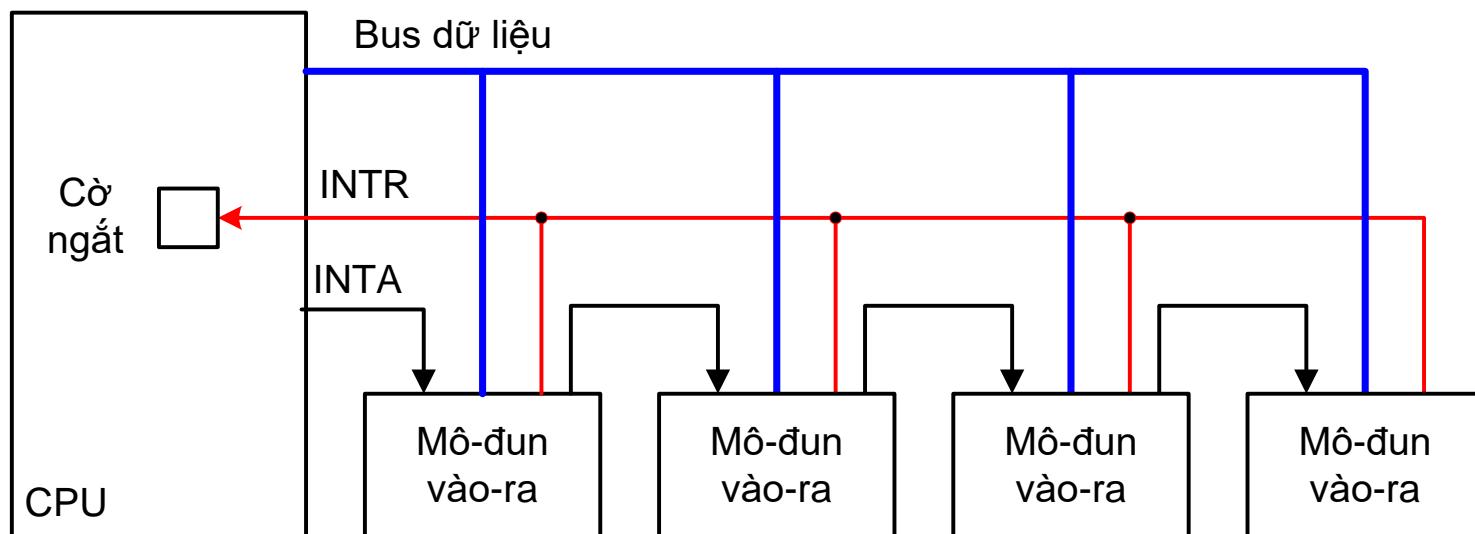
- Mỗi mô-đun vào-ra được nối với một đường yêu cầu ngắt
- CPU phải có nhiều đường tín hiệu yêu cầu ngắt
- Hạn chế số lượng mô-đun vào-ra
- Các đường ngắt được qui định mức ưu tiên

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

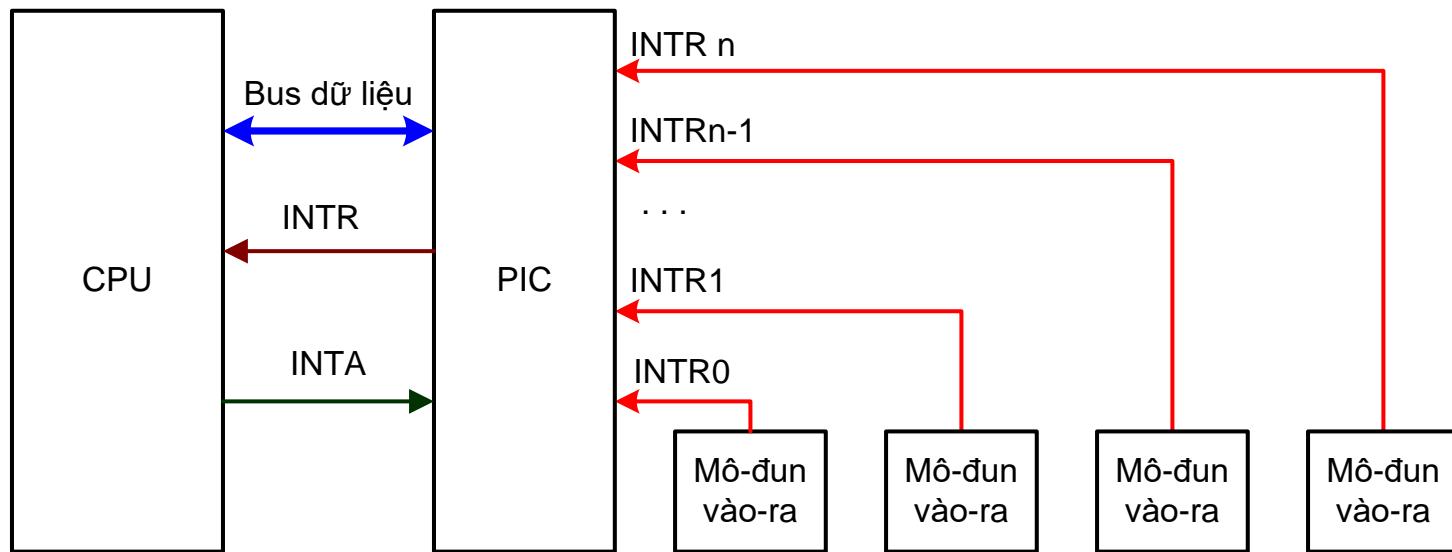
Hỏi vòng bằng phần cứng



Kiểm tra vòng bằng phần cứng (tiếp)

- CPU phát tín hiệu chấp nhận ngắn (INTA) đến mô-đun vào-ra đầu tiên
- Nếu mô-đun vào-ra đó không gây ra ngắn thì nó gửi tín hiệu đến mô-đun kế tiếp cho đến khi xác định được mô-đun gây ngắn
- 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
- CPU trực tiếp điều khiển vào-ra
- CPU không phải đợi mô-đun vào-ra → hiệu quả sử dụng CPU tốt hơn

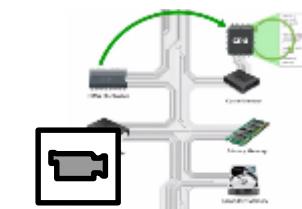
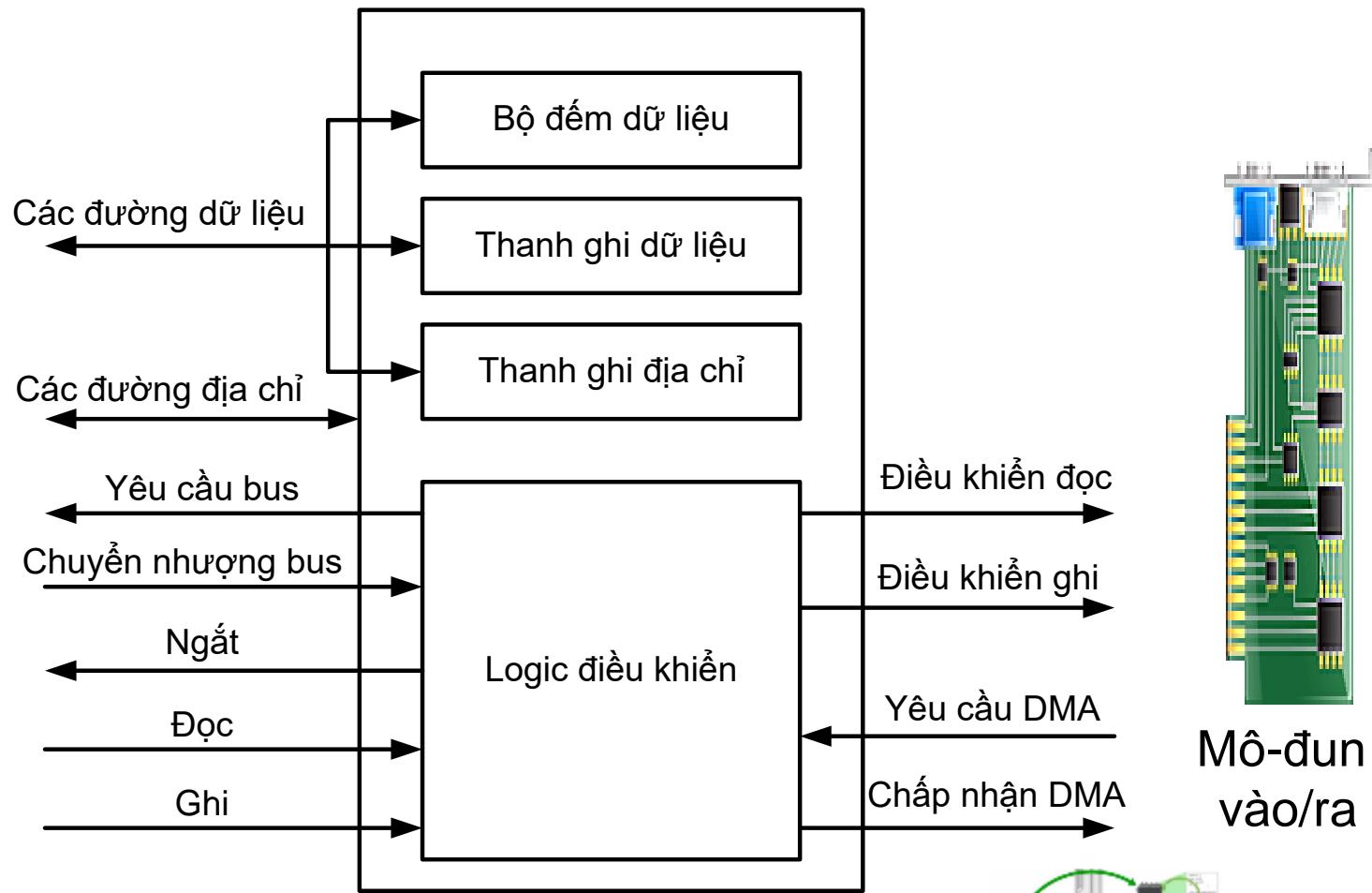
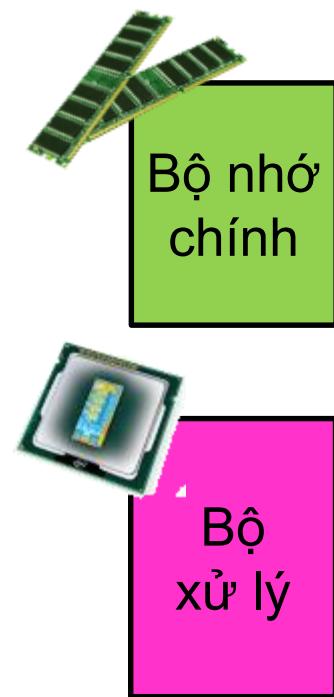


Polling vs. Interrupt

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 DMA
 - Thêm mô-đun phần cứng trên bus → DMAC (Controller)
 - DMAC điều khiển trao đổi dữ liệu giữa mô-đun vào-ra với bộ nhớ chính

Sơ đồ cấu trúc của DMA



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

Quản lý nói xe ôm

- Lấy hay cát hàng
- Địa chỉ khách hàng, phố nào?
- Vị trí đầu của lô hàng trong kho
- Tổng số lô hàng cần chuyển.

Quản lý làm việc khác

Xe ôm giao hàng

Chuyển được 1 lô thì:

- Lô hàng kế tiếp
- Giảm số lô còn lại

Khi số lô còn lại bằng 0, xe ôm báo quản lý.

Các kiểu thực hiện DMA

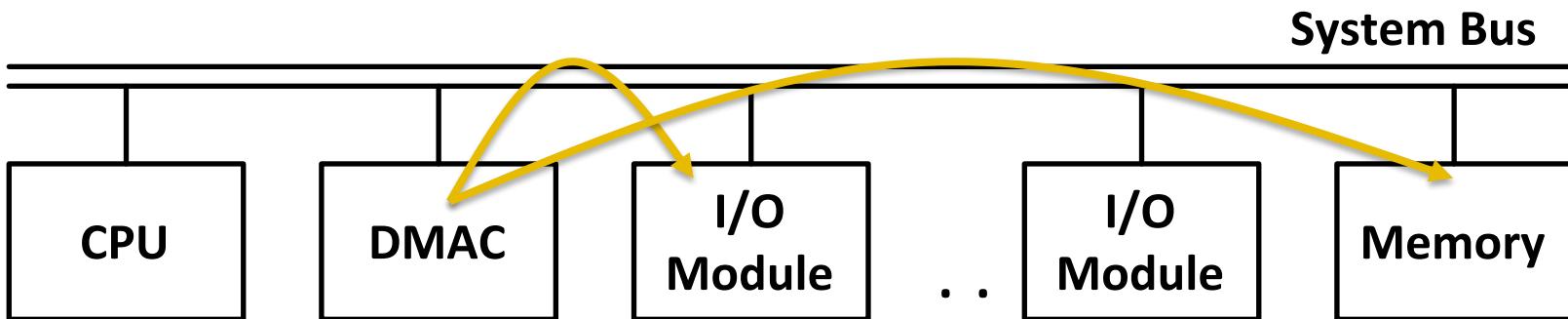
- DMA truyền theo khối (Block-transfer DMA): DMAC sử dụng bus để truyền xong cả khối dữ liệu
- DMA lấy chu kỳ (Cycle Stealing DMA): DMAC cưỡng bức CPU treo tạm thời từng chu kỳ bus, DMAC chiếm bus thực hiện truyền một từ dữ liệu.
- DMA trong suốt (Transparent DMA): DMAC nhận biết những chu kỳ nào CPU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

Giao hàng bằng xe tải: mỗi lần chở là một khối gồm nhiều lô hàng.

Xe ôm định kỳ 2 tiếng gặp quản lý để lấy xe máy đi giao hàng, kể cả khi không có hàng.

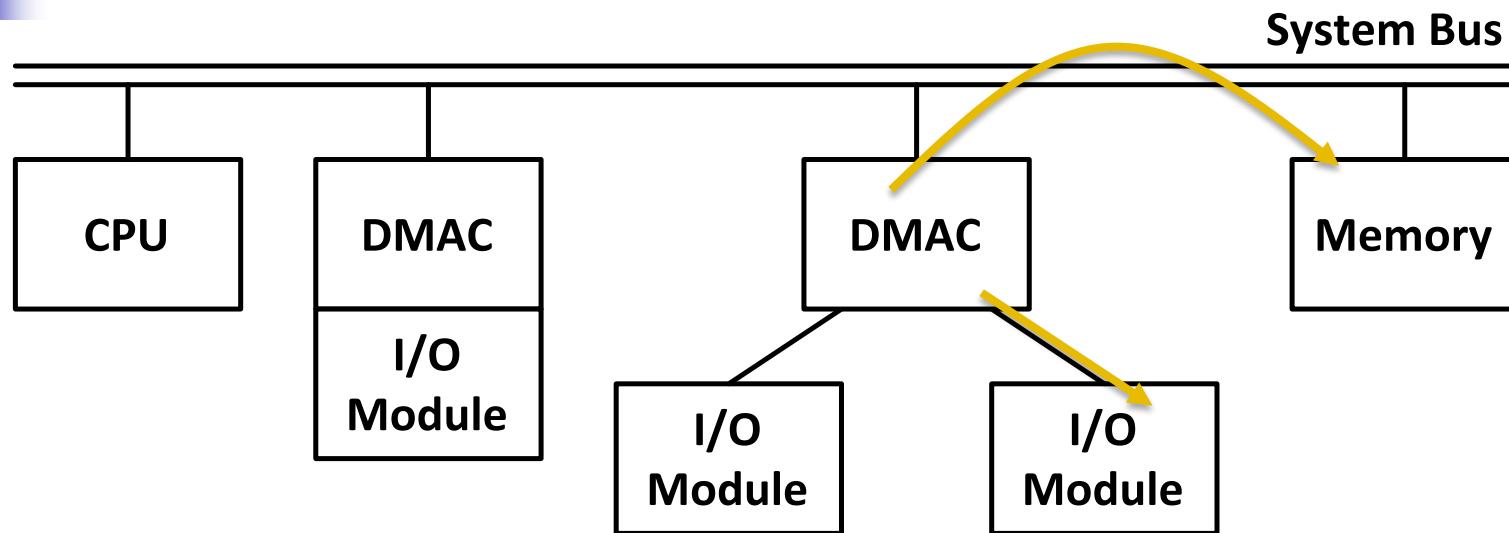
Xe ôm nhận biết khi nào Quản lý không dùng xe máy thì mới chiếm xe máy để giao hàng

Cấu hình DMA (1)



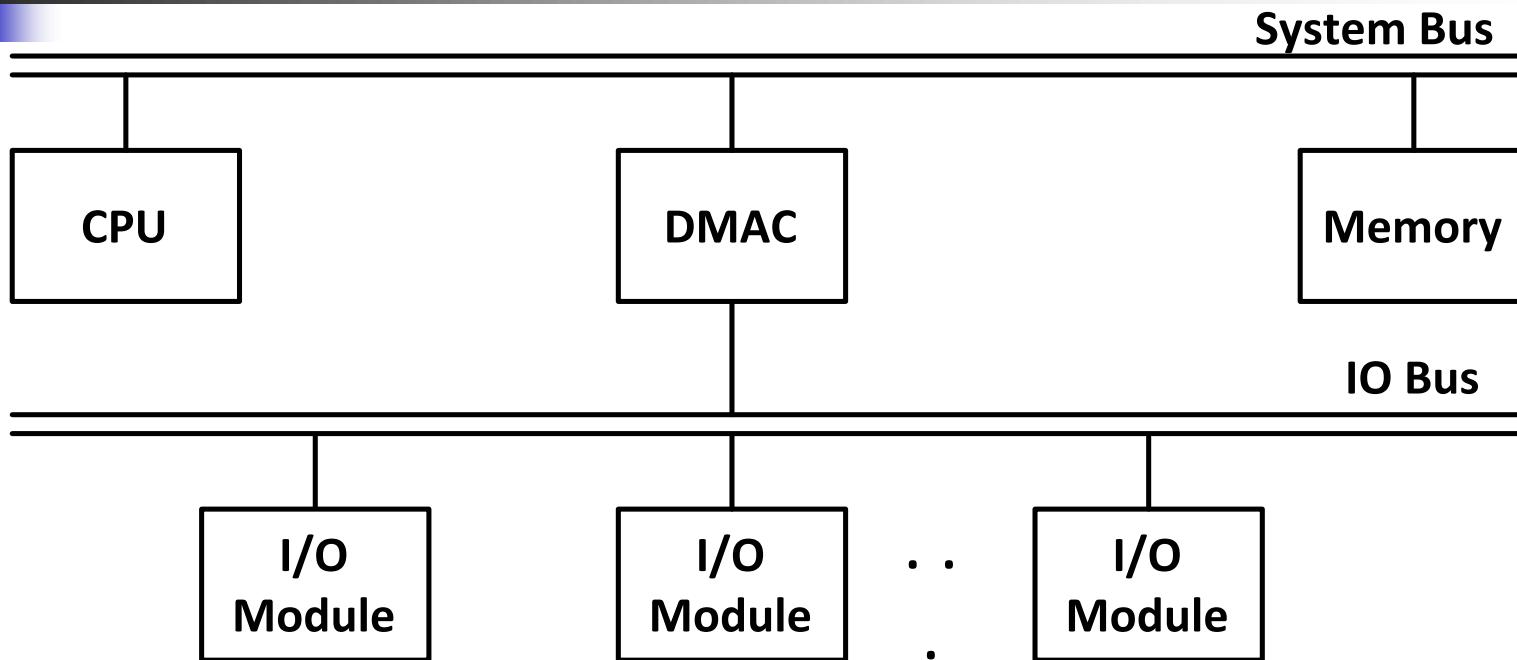
- Mỗi lần trao đổi một dữ liệu, DMAC sử dụng bus hai lần
 - Giữa mô-đun vào-ra với DMAC
 - Giữa DMAC với bộ nhớ

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

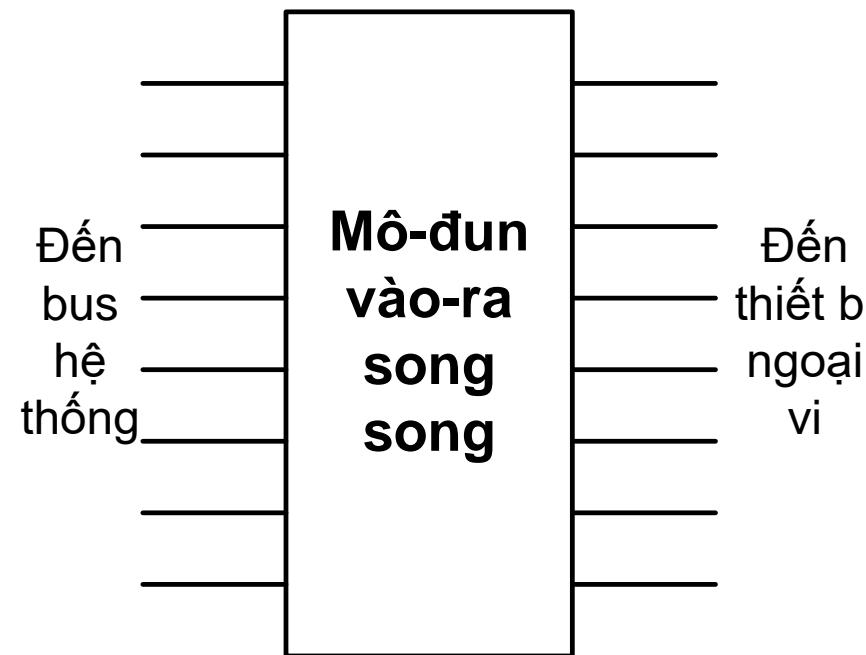
- Việc điều khiển vào-ra được thực hiện bởi một bộ xử lý vào-ra chuyên dụng
- Bộ xử lý vào-ra hoạt động theo chương trình của riêng nó
- Chương trình của bộ xử lý vào-ra có thể nằm trong bộ nhớ chính hoặc nằm trong một bộ nhớ riêng
- Hoạt động theo kiến trúc đa xử lý

8.3. Nối ghép thiết bị ngoại vi

1. Các kiểu nối ghép vào-ra

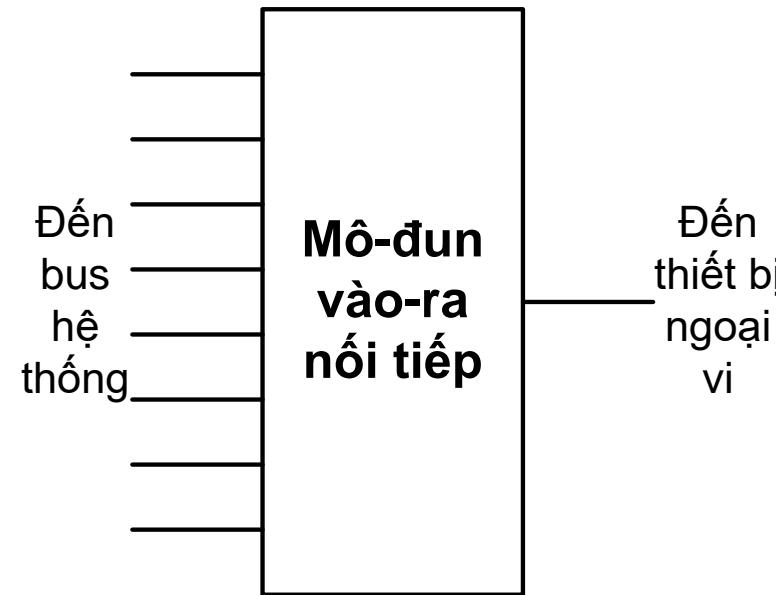
- Nối ghép song song
- Nối ghép nối tiếp

Nối ghép song song



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

Nối ghép nối tiếp

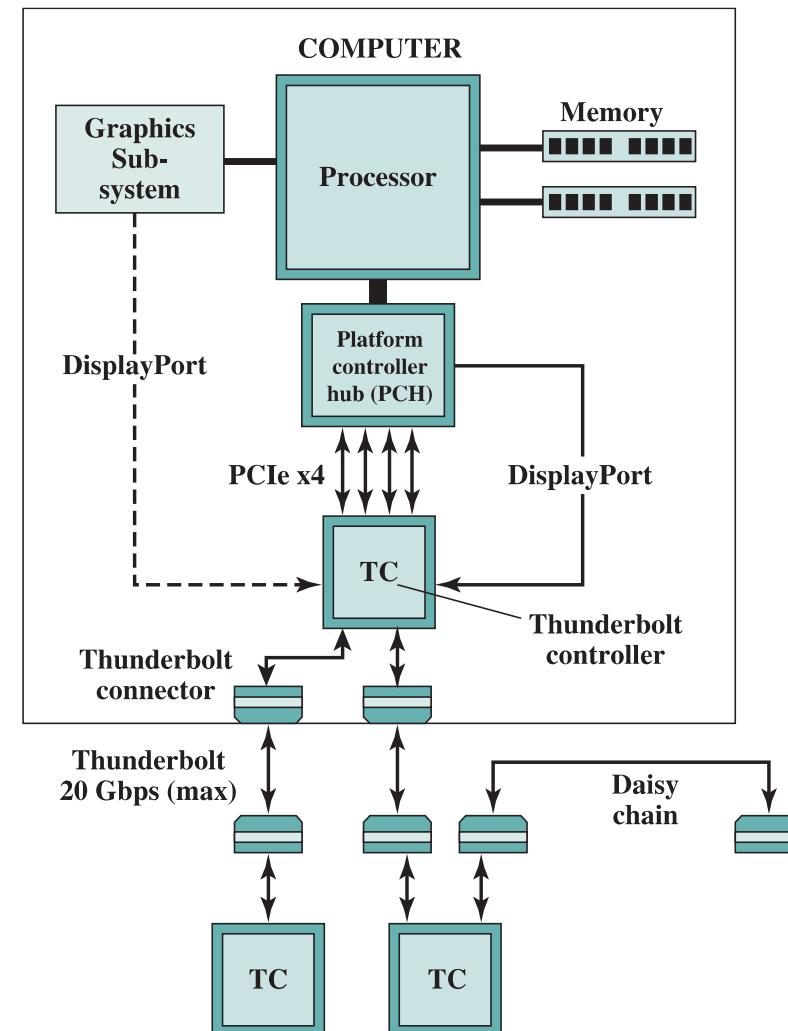


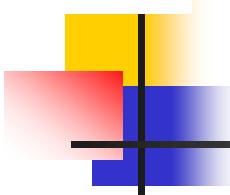
- Truyền lần lượt từng bit
- Cần có bộ chuyển đổi từ dữ liệu song song sang nối tiếp hoặc/và ngược lại
- Tốc độ chậm hơn
- Cần ít đường truyền dữ liệu

2. Các cấu hình nối ghép

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

Thunderbolt



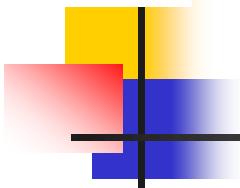


Hết chương 8

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

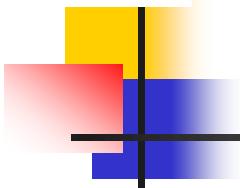
Chương 5. Kiến trúc tập lệnh

Chương 6. Bộ xử lý

Chương 7. Bộ nhớ máy tính

Chương 8. Hệ thống vào-ra

Chương 9. Các kiến trúc song song



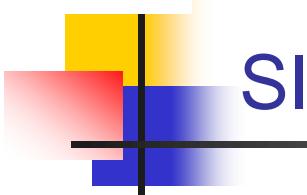
Nội dung của chương 9

- 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

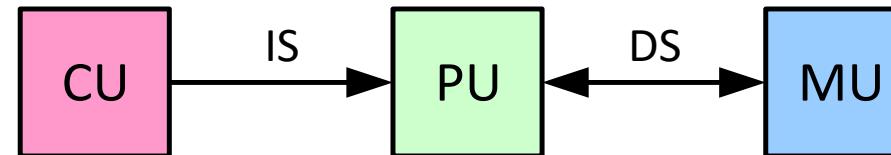
9.1. Phân loại kiến trúc máy tính

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

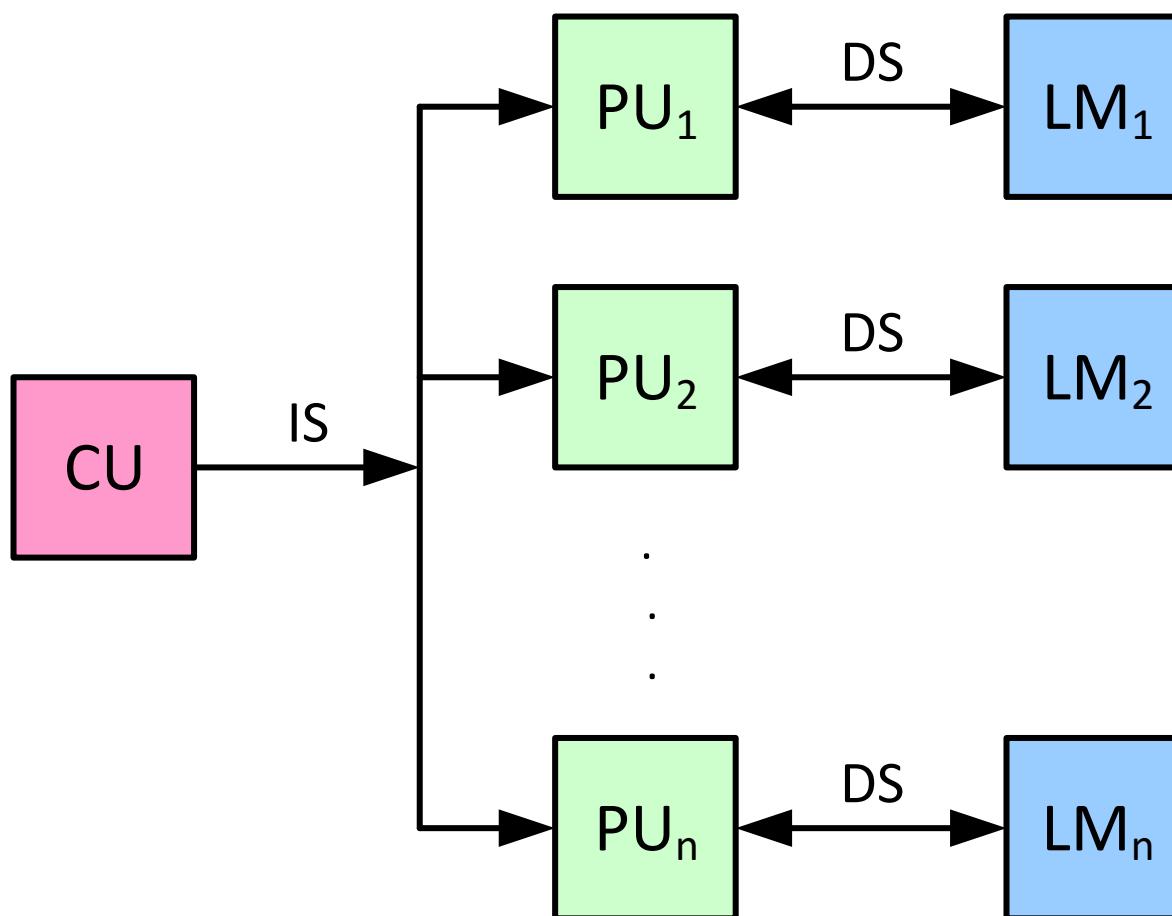
- SISD - Single Instruction Stream, Single Data Stream
- SIMD - Single Instruction Stream, Multiple Data Stream
- MISD - Multiple Instruction Stream, Single Data Stream
- MIMD - Multiple Instruction Stream, Multiple Data Stream



SISD



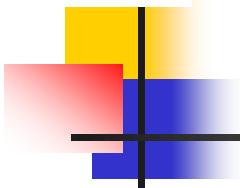
- CU: Control Unit
- PU: Processing Unit
- MU: Memory Unit
- Một bộ xử lý
- Đơn dòng lệnh
- Dữ liệu được lưu trữ trong một bộ nhớ
- Chính là Kiến trúc von Neumann (tuần tự)



SIMD (tiếp)

- Đơn dòng lệnh điều khiển đồng thời các đơn vị xử lý PUs
- Mỗi phần tử 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

- 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

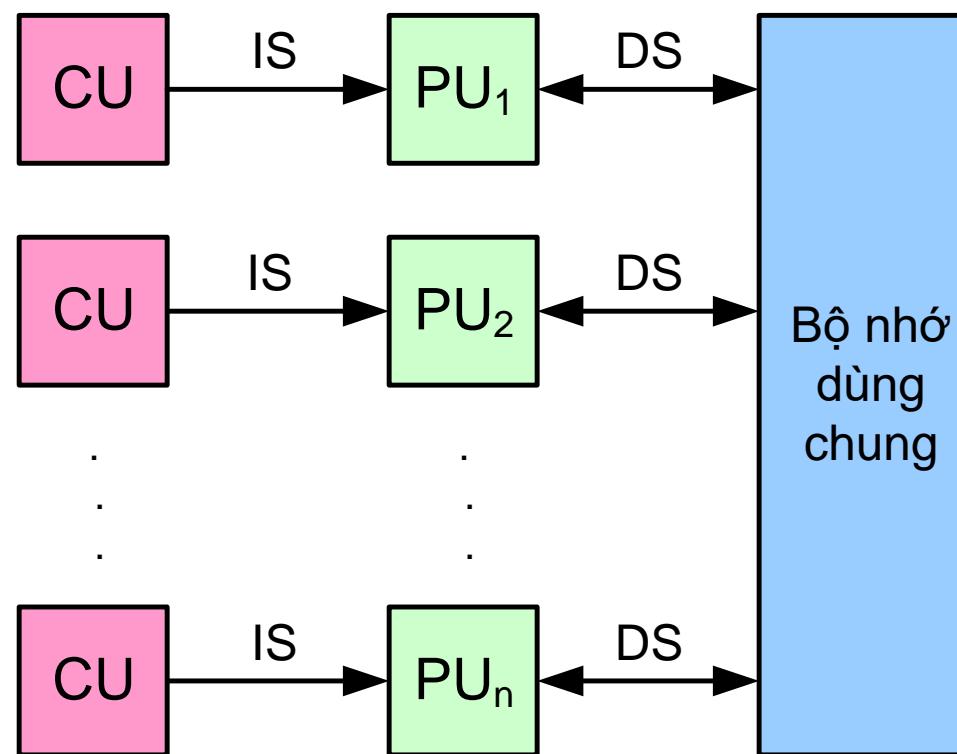


MIMD

- 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)

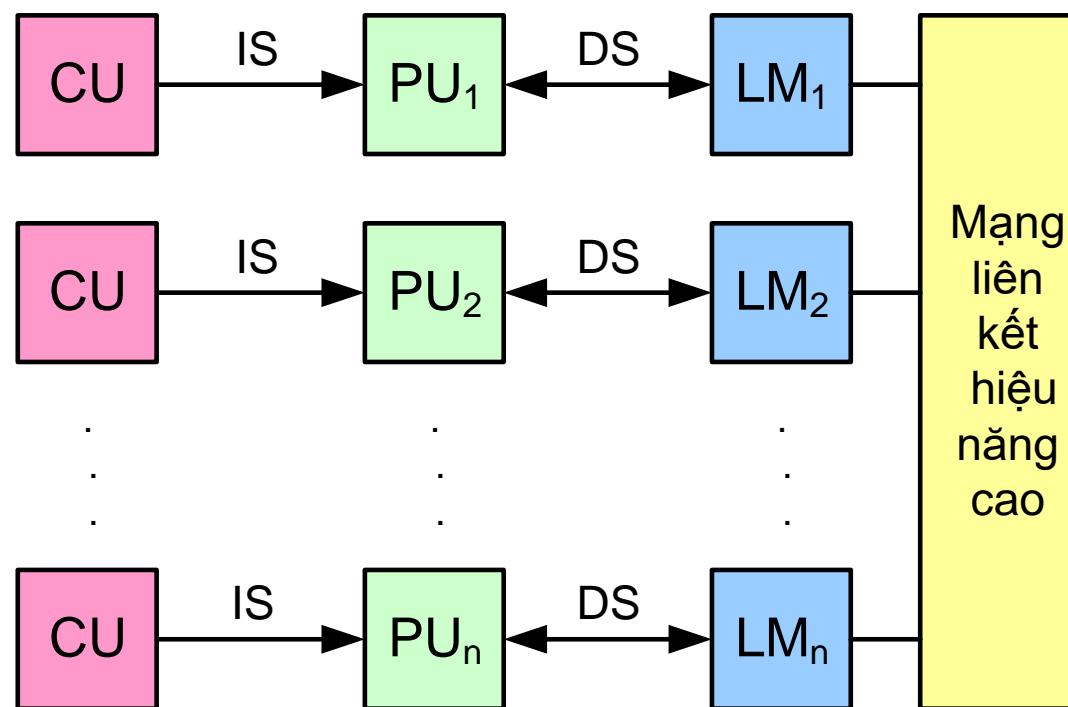
MIMD - Shared Memory

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



MIMD - Distributed Memory

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



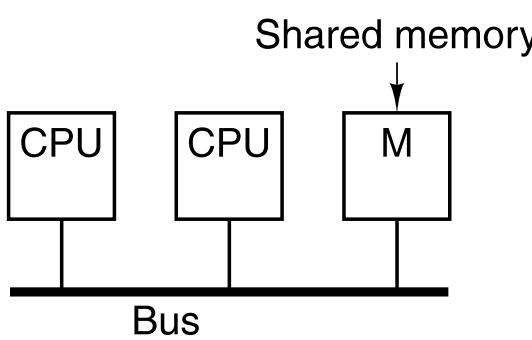
Phân loại các kỹ thuật song song

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

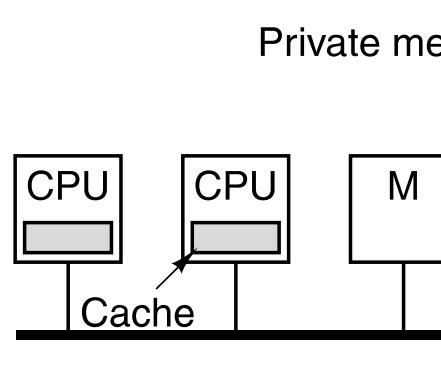
9.2. Đa xử lý bộ nhớ dùng chung

- 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)

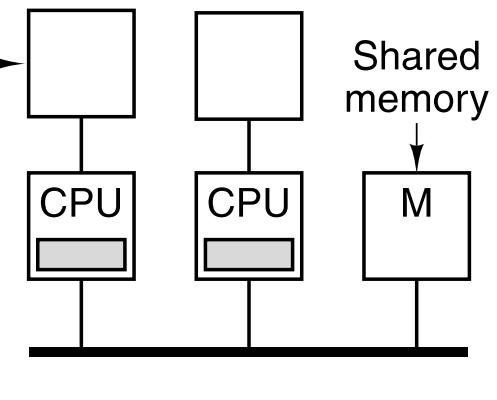
SMP hay UMA (Uniform Memory Access)



(a)



(b)

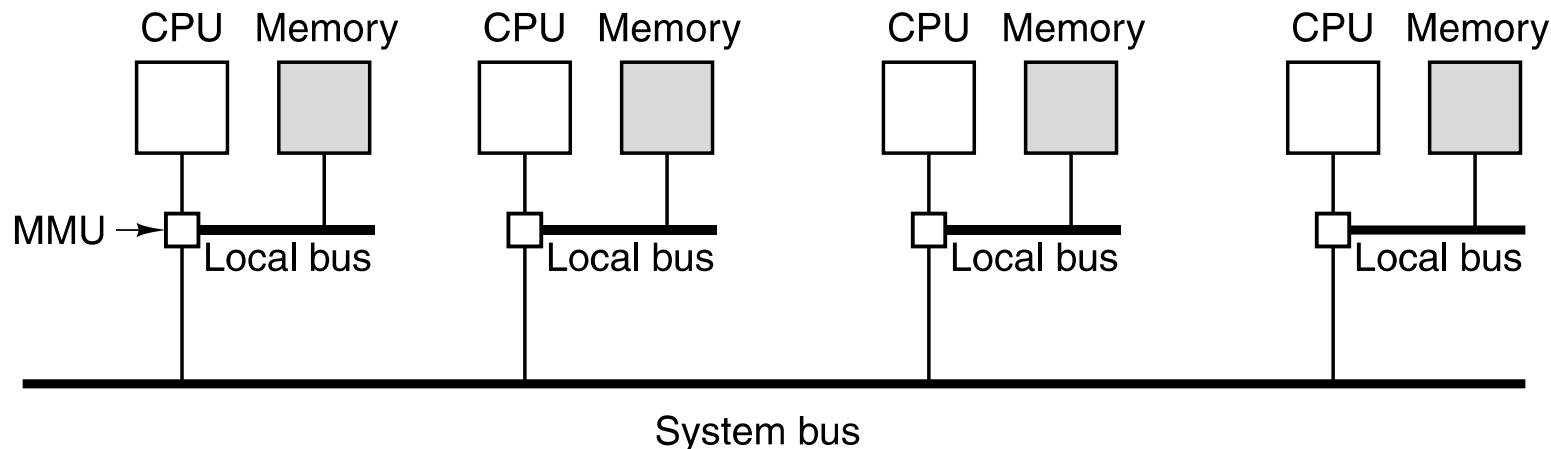


(c)

SMP (tiếp)

- Một máy tính có $n \geq 2$ 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)

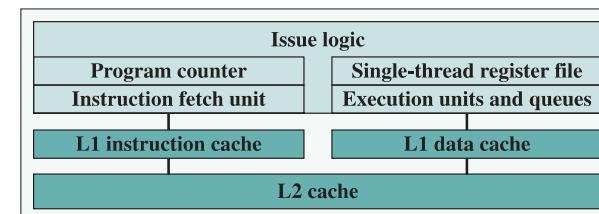


- Có một không gian địa chỉ chung cho tất cả CPU
- Mỗi CPU có thể truy cập từ xa sang bộ nhớ của CPU khác
- Truy nhập bộ nhớ từ xa chậm hơn truy nhập bộ nhớ cục bộ

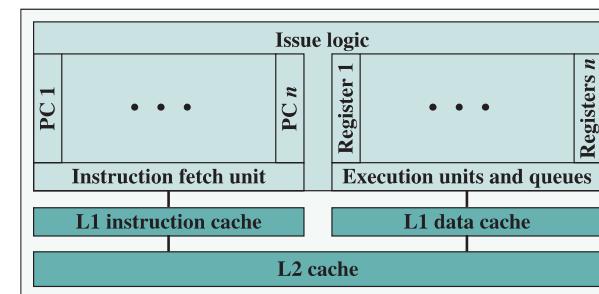
Bộ xử lý đa lõi (multicores)

- Thay đổi của bộ xử lý:

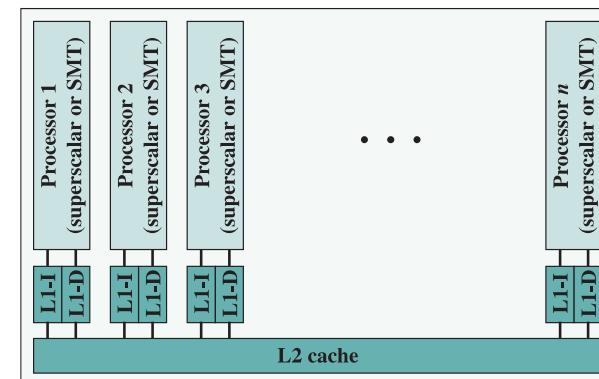
- Tuần tự
- Pipeline
- Siêu vô hướng
- Đa luồng
- Đa lõi: nhiều CPU trên một chip



(a) Superscalar

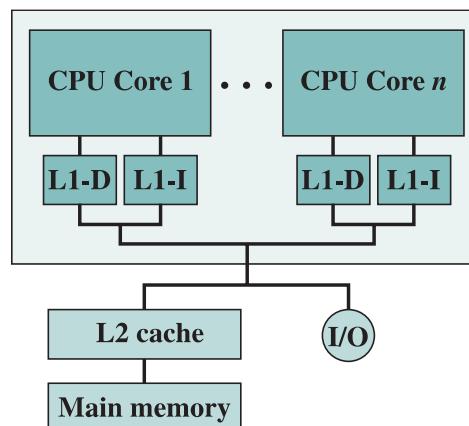


(b) Simultaneous multithreading

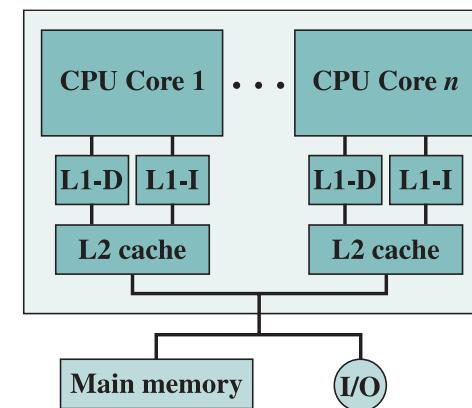


(c) Multicore

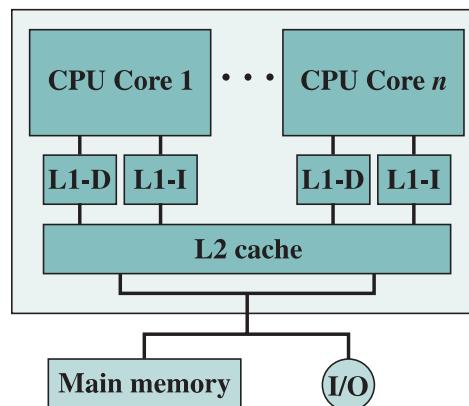
Các dạng tổ chức bộ xử lý đa lõi



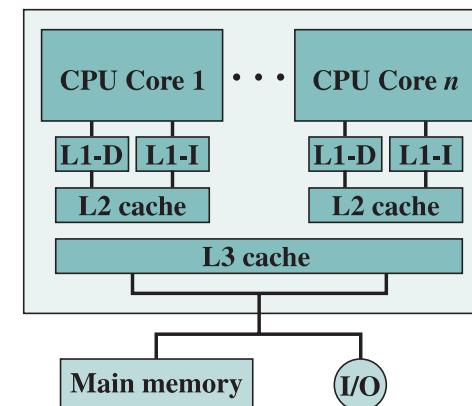
(a) Dedicated L1 cache



(b) Dedicated L2 cache



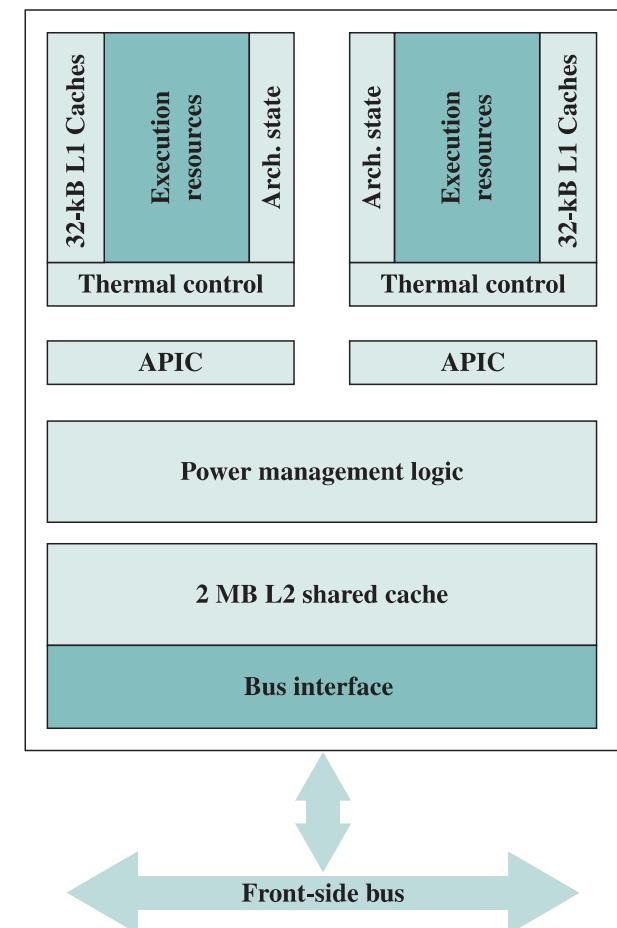
(c) Shared L2 cache



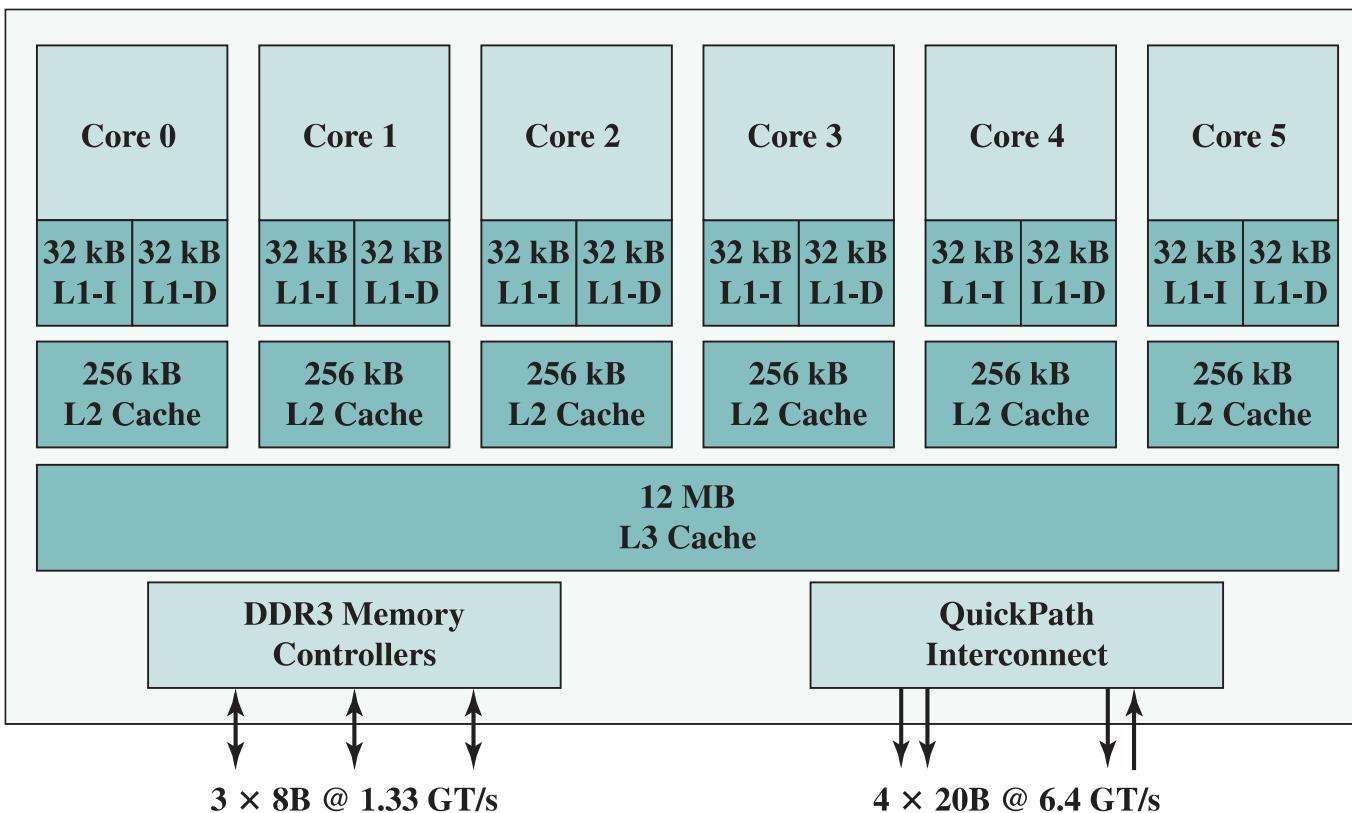
(d) Shared L3 cache

Intel - Core Duo

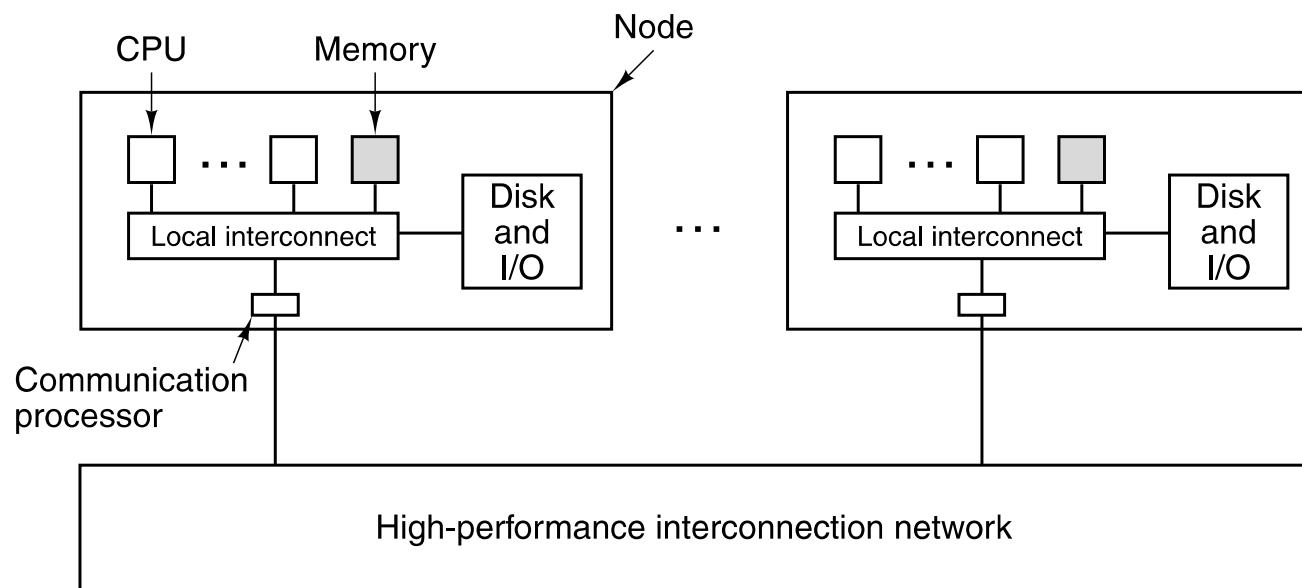
- 2006
- Two x86 superscalar, shared L2 cache
- Dedicated L1 cache per core
 - 32KiB instruction and 32KiB data
- 2MiB shared L2 cache



Intel Core i7-990X

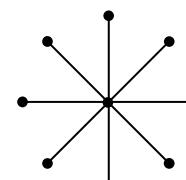


9.3. Đa xử lý bộ nhớ phân tán

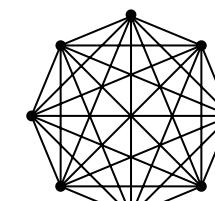


- Máy tính qui mô lớn (Warehouse Scale Computers or Massively Parallel Processors – MPP)
- Máy tính cụm (clusters)

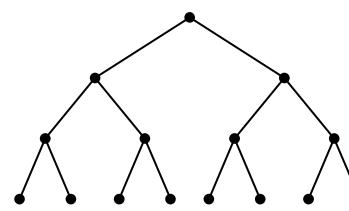
Mạng liên kết



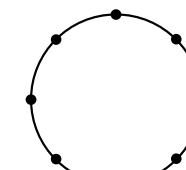
(a)



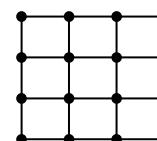
(b)



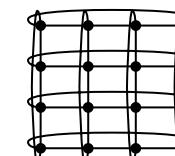
(c)



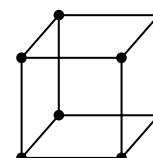
(d)



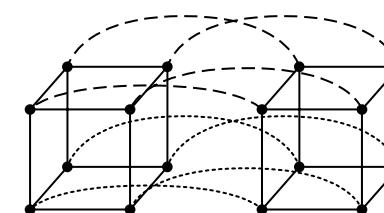
(e)



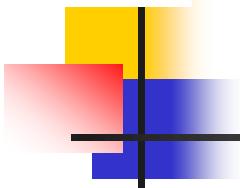
(f)



(g)



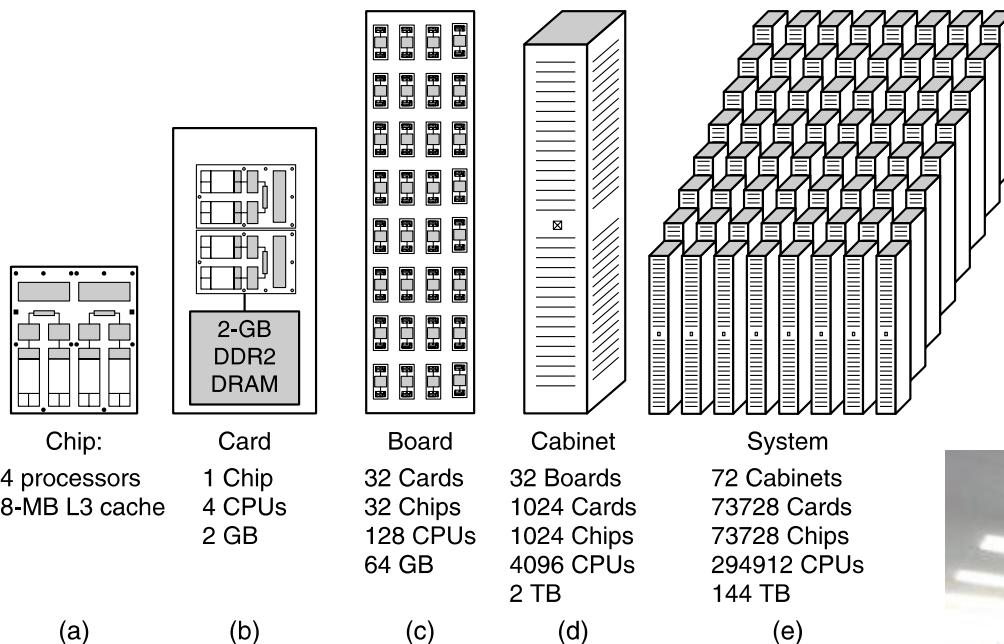
(h)



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

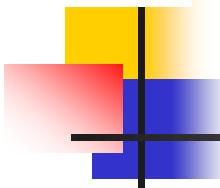


IBM Blue GeneL supercomputer



Installation of an IBM BlueGene/Q supercomputer at CSCS for EPFL BlueBrain project

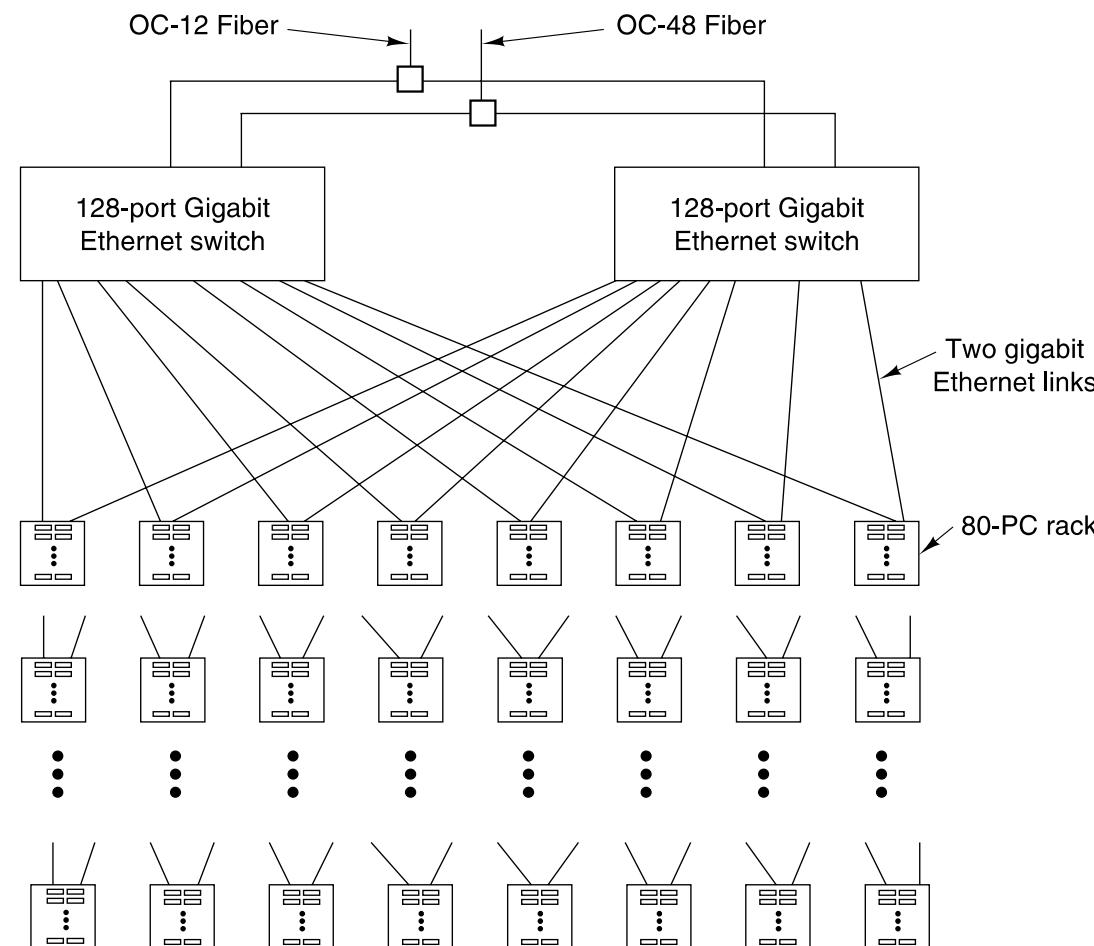




Cluster

- 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

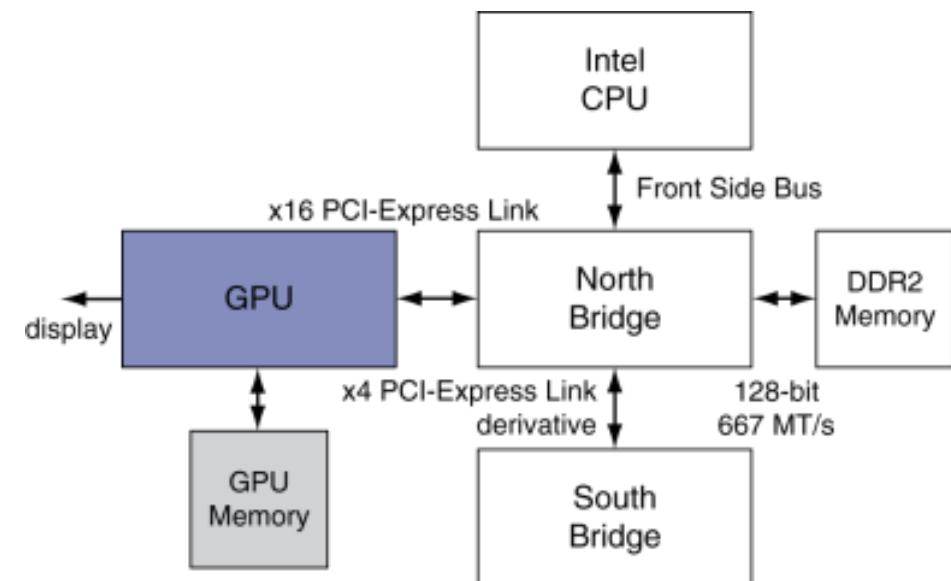
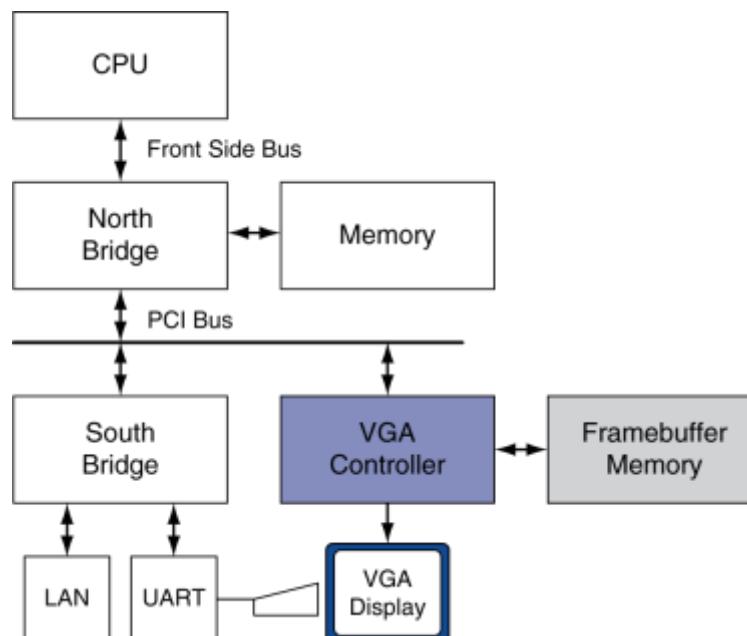
PC Cluster của Google



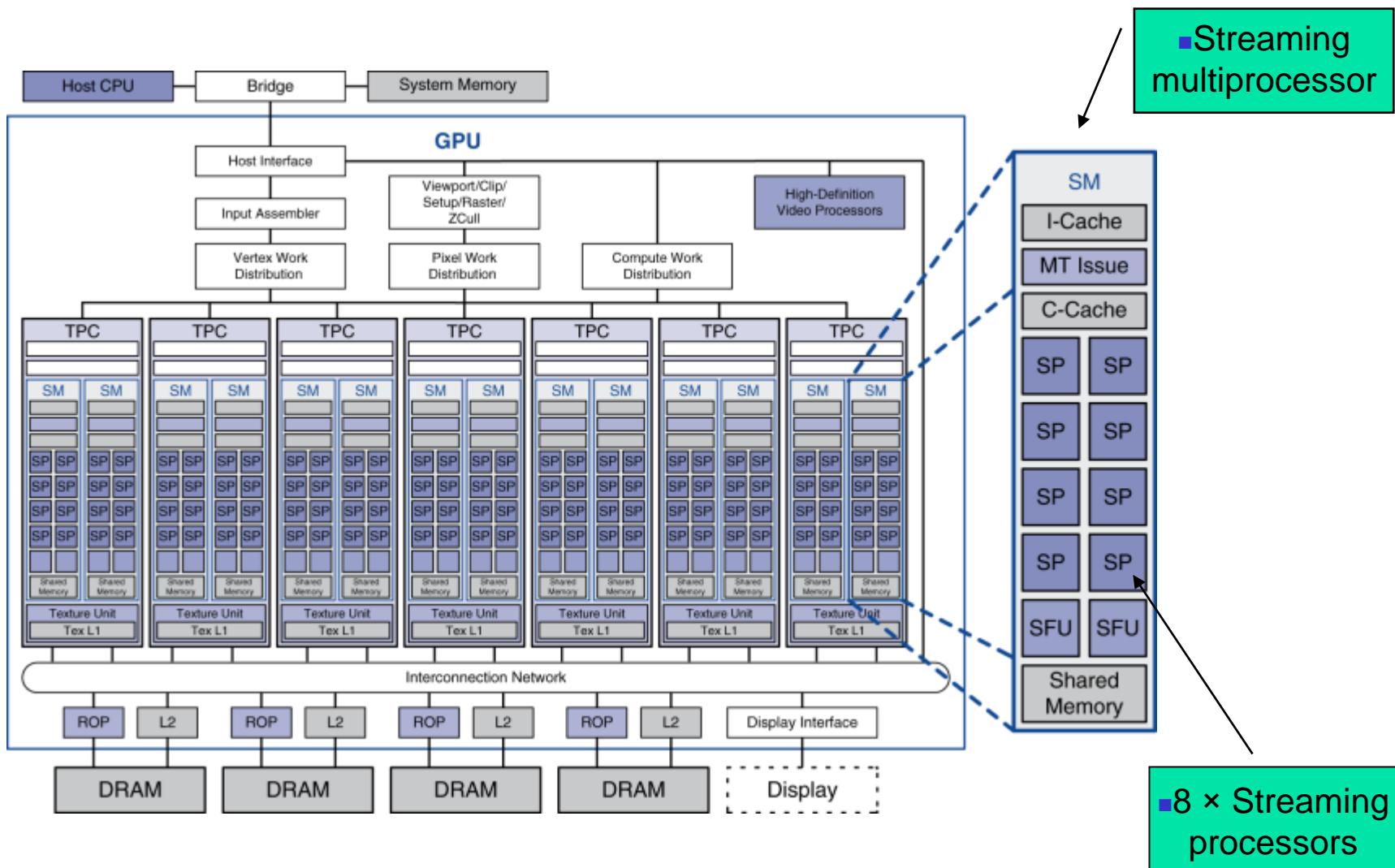
9.4. Bộ xử lý đồ họa tính toán đa năng

- 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

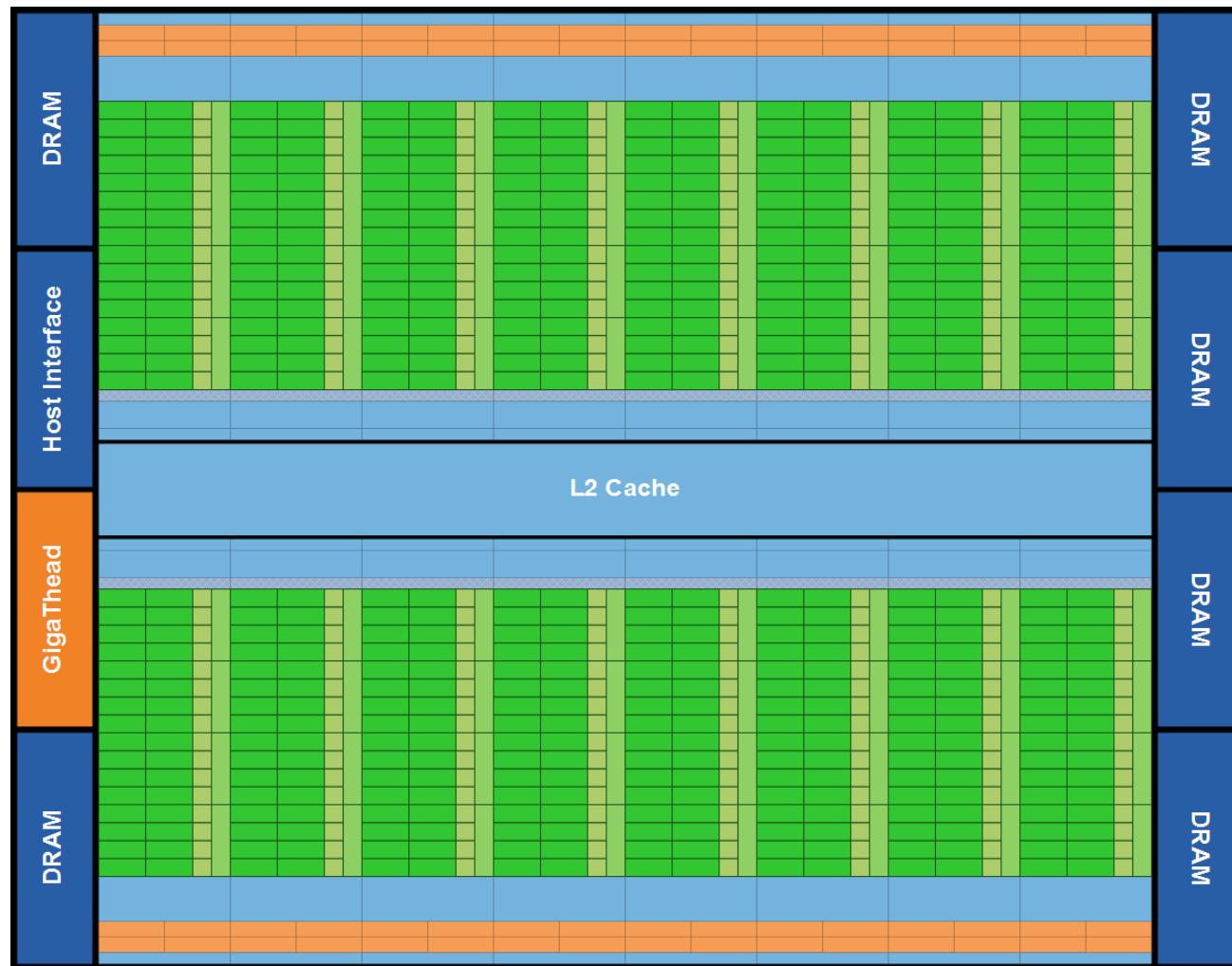
Bộ xử lý đồ họa trong máy tính



GPGPU: NVIDIA Tesla

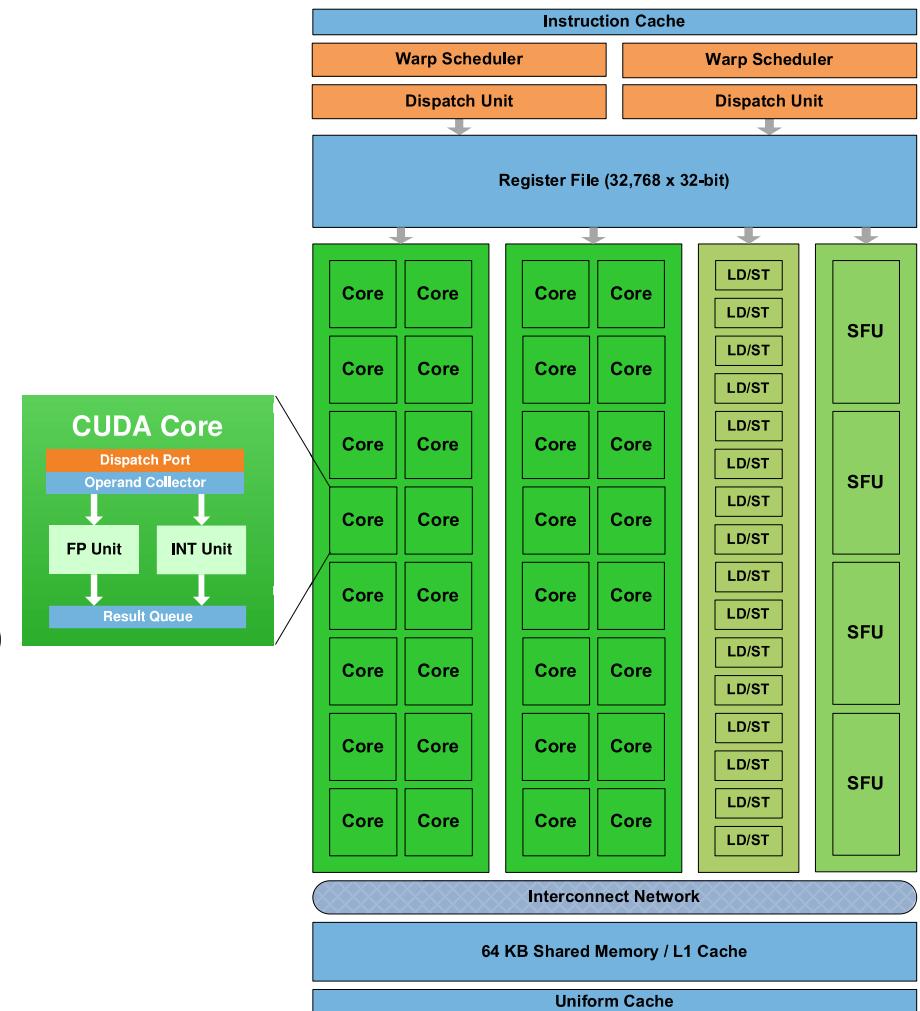


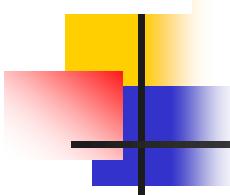
GPGPU: NVIDIA Fermi



NVIDIA Fermi

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





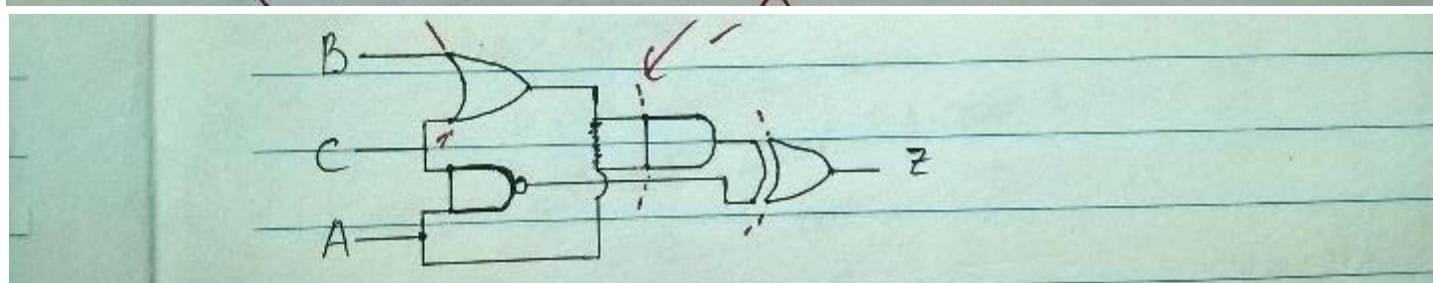
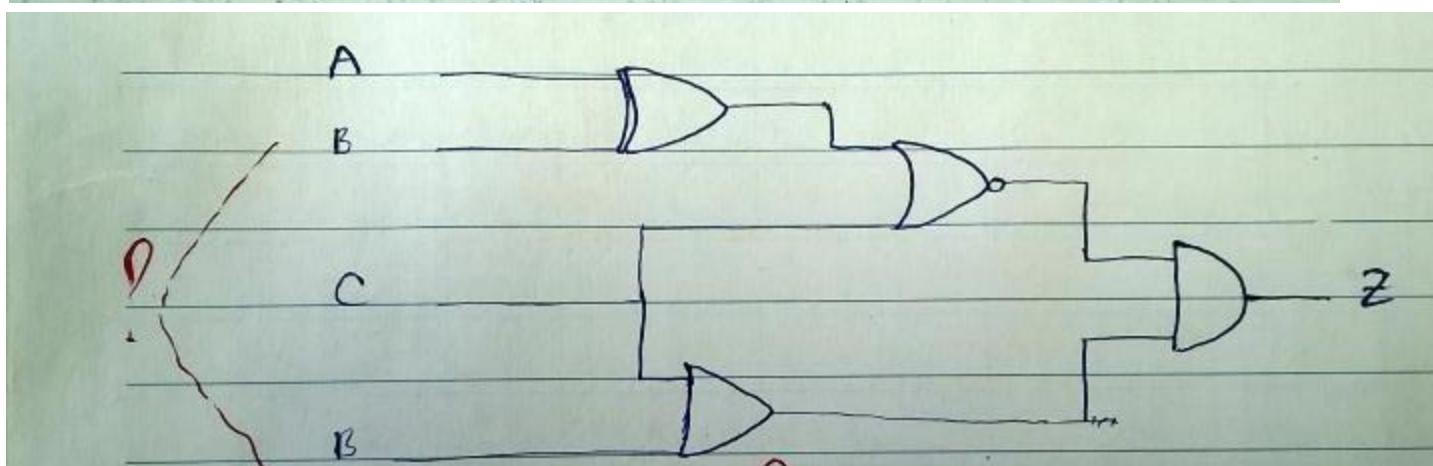
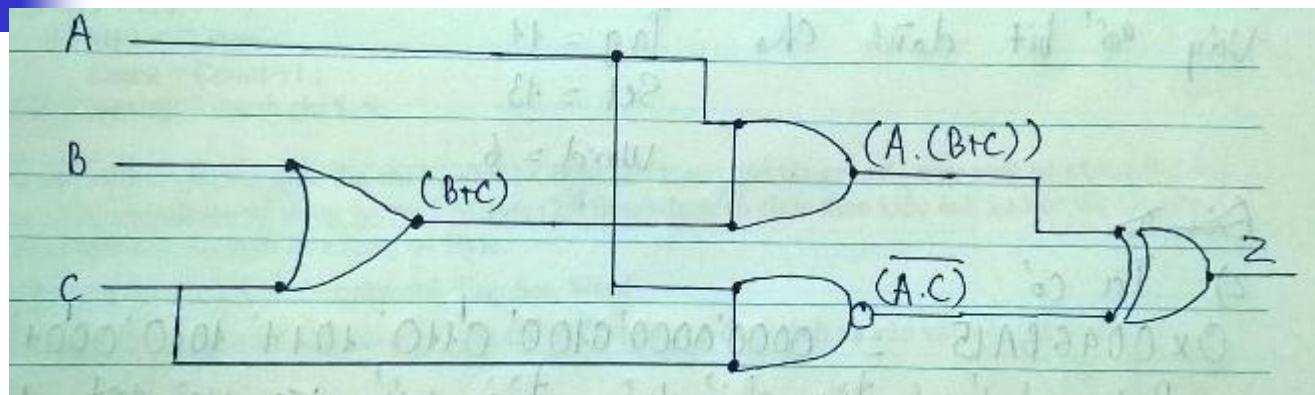
Hết

Issue 01

1) Vì lệnh JR cách lệnh SLL 3 lệnh mà mỗi lệnh là 32 bit
vì vậy với 4 byte \Rightarrow địa chỉ của lệnh JR là $0x40000014_{(16)} + 3 \cdot 4_{(16)}$
 $= 0x40000014_{(16)} + C_{(16)} = 0x40000021$

$$\text{số bit trống tag } b = N - (W + S) = 30 - (6 + 13) = 12$$

Issue 02



Issue 03

$$\text{IC} = \frac{t_{f0}}{\text{CPI}} = \frac{3 \cdot 1,5 \cdot 10^9}{3,6} = 1,25 \cdot 10^9 \text{ (lens)} \quad \checkmark$$
$$\rightarrow \text{IC} = 1,25 \cdot 10^9 \text{ lens} \quad \times$$

Issue 04:

Đó : $\lceil \text{Kích thước mỗi byte} = 32 \rceil = 2^5 \Rightarrow \text{word} = 5$

Bandwidth

■ Transfers per second:

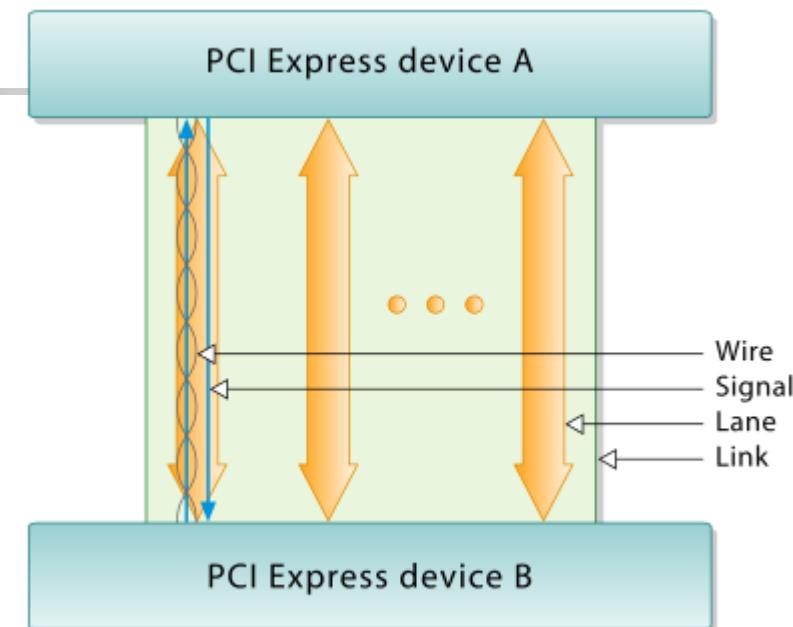
- Đơn vị: T/s, MT/s, GT/s
- số lượng các thao tác truyền dữ liệu trong một giây, xảy ra trên một kênh truyền dữ liệu vật lý, hay còn được gọi là tốc độ lấy mẫu (sample rate)

■ Quan hệ

- $\langle \text{bit rate} \rangle = \langle \text{channel width} \rangle * \langle \text{transfer/second} \rangle$
- Ví dụ, bus dữ liệu 64 bit với transfer/rate = 1GT/s thì bit rate của đường truyền đó = 64 Gb/s

PCIe

- Point to Point
- Mỗi lane gồm
 - 1 đường Tx
 - 1 đường Rx
 - Tín hiệu vi sai



PCIe version	Line code encoding	Transfer Rate	Bandwidth	
			Per lane	x16
1.0 (2003)	<u>8b/10b</u>	2.5 GiT/s (250 MiB/s)	2 Gbit/s (250 MiB/s)	32 Gbit/s (4 GiB/s)
2.0 (1/2007)	8b/10b	5 GiT/s	4 Gbit/s (500 MiB/s)	64 Gbit/s (8 GiB/s)
3.0 (11/2010)	<u>128b/130b</u>	8 GiT/s	7.877 Gbit/s (984.6 MiB/s)	126.032 Gbit/s (15.754 GiB/s)
4.0 (11/2011)	128b/130b	16 GiT/s	15.754 Gbit/s (1969.2 MiB/s)	252.064 Gbit/s (31.508 GiB/s)

Thermal Design Power - TDP

- Năng suất tỏa nhiệt biểu kiến, là công suất điện “tối đa” mà một CPU chuyển hóa thành nhiệt năng.
 - Các hệ thống làm mát phải thỏa mãn TDP
- TDP được tính trong điều kiện làm việc bình thường
- Khi bị nhiễm power virus, lượng nhiệt thực sự có thể lớn hơn TDP