



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 (SolICT)

Version: CA-2017

NKK-HUST

Contact Information

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

2017

Kiến trúc máy tính

NKK-HUST

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 nguyên tắc cơ bản trong thiết kế máy tính.
- Sau khi học xong học phần này, sinh viên có khả năng:
 - Tìm hiểu kiến trúc tập lệnh của các bộ xử lý cụ thể
 - Lập trình hợp ngữ
 - Đánh giá hiệu năng máy tính và cải thiện hiệu năng của chương trình
 - Khai thác và quản trị hiệu quả các hệ thống máy tính
 - Phân tích và thiết kế máy tính

NKK-HUST



Tài liệu học tập

- *Bài giảng Kiến trúc máy tính*
<ftp://dce.soict.hust.edu.vn/khanhnk/CA/>
- *Sách tham khảo:*
 - [1] William Stallings
Computer Organization and Architecture – 2013, 9th edition
 - [2] David A. Patterson, John L. Hennessy
Computer Organization and Design – 2012, Revised 4th edition
 - [3] David Money Harris, Sarah L. Harris
Digital Design and Computer Architecture – 2013, 2nd edition
 - [4] Andrew S. Tanenbaum
Structured Computer Organization – 2013, 6th edition
- *Phần mềm lập trình hợp ngữ và mô phỏng cho MIPS:
MARS (MIPS Assembler and Runtime Simulator)*
download tại: <http://courses.missouristate.edu/KenVollmar/MARS/>



Nội dung học phần

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

 NKK-HUST

Content

- Chapter 1. Introduction
- Chapter 2. The Basics of Digital Logic
- Chapter 3. Computer Systems
- Chapter 4. Computer Arithmetic
- Chapter 5. Instruction Set Architecture
- Chapter 6. The Processors
- Chapter 7. Computer Memory
- Chapter 8. Input-Output Systems
- Chapter 9. Parallel Architectures

2017

Kiến trúc máy tính

Nội dung của chương 1

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

NKK-HUST

1.1. Máy tính và phân loại máy tính

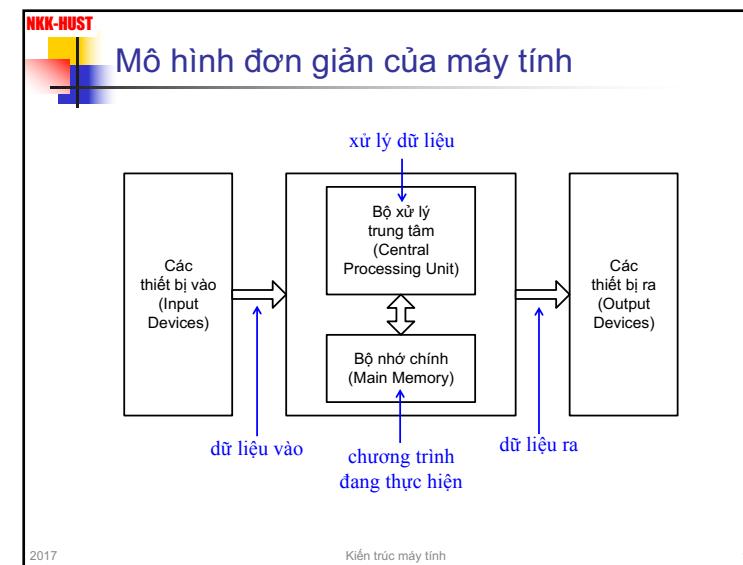
- **Máy tính (Computer)** là thiết bị điện tử thực hiện các công việc sau:
 - Nhận dữ liệu vào,
 - Xử lý dữ liệu theo dãy các lệnh được nhớ sẵn bên trong,
 - Đưa dữ liệu (thông tin) ra.
- Dãy các lệnh nằm trong bộ nhớ để yêu cầu máy tính thực hiện công việc cụ thể gọi là **chương trình (program)**.

→ Máy tính hoạt động theo chương trình

2017

Kiến trúc máy tính

9



2017

Kiến trúc máy tính

10

NKK-HUST

Phân loại máy tính kỹ nguyên PC

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

2017

Kiến trúc máy tính

11

NKK-HUST

Phân loại máy tính kỹ nguyên sau PC

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

2017

Kiến trúc máy tính

12

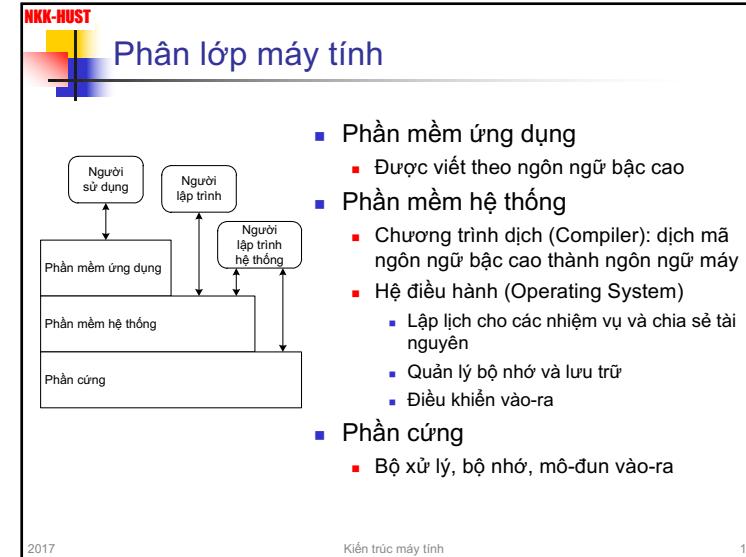
1.2. Khái niệm kiến trúc máy tính

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

2017

Kiến trúc máy tính

1



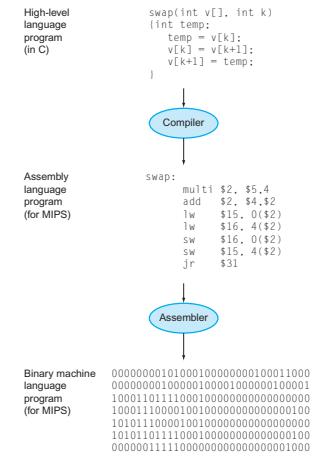
Phân lớp máy tính

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

14

Các mức của mã chương trình

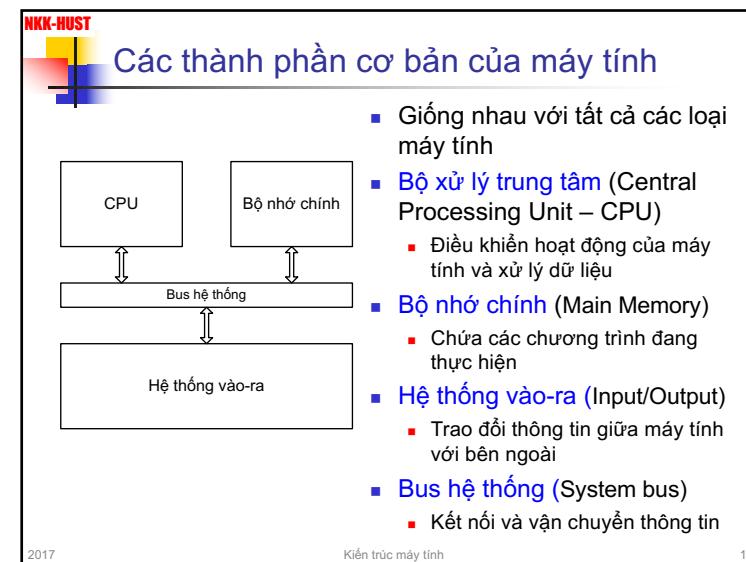
- Ngôn ngữ bậc cao
 - High-level language – HLL
 - Mức trừu tượng gần với vấn đề cần giải quyết
 - Hiệu quả và linh động
 - Hợp ngữ
 - Assembly language
 - Mô tả lệnh dưới dạng text
 - Ngôn ngữ máy
 - Machine language
 - Mô tả theo phần cứng
 - Các lệnh và dữ liệu được mã hóa theo nhị phân



2017

Kiến trúc máy tính

1



Các thành phần cơ bản của máy tính

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

16

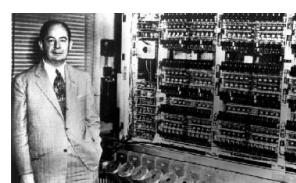
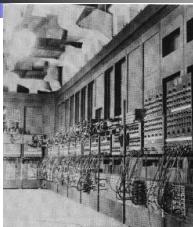
NKK-HUST

1.3. Sự tiến hóa của công nghệ máy tính

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

NKK-HUST

Máy tính đầu tiên: ENIAC và IAS



- Electronic Numerical Integator and Computer
- Dự án của Bộ Quốc phòng Mỹ
- Do John Mauchly ở đại học Pennsylvania thiết kế
- 30 tấn
- Xử lý theo số thập phân
- Thực hiện tại Princeton Institute for Advanced Studies
- Do John von Neumann thiết kế theo ý tưởng “stored program”
- Xử lý theo số nhị phân
- Trở thành mô hình cơ bản của máy tính



Một số loại vi mạch số điển hình

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

NKK-HUST

Sự phát triển của bộ vi xử lý

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



NKK-HUST

1.4. Hiệu năng máy tính

- Định nghĩa hiệu năng P (Performance):
$$\text{Hiệu năng} = 1/(\text{thời gian thực hiện})$$
hay là: $P = 1/t$

“Máy tính A nhanh hơn máy B k lần”

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

NKK-HUST

Tốc độ xung nhịp của CPU

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



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

NKK-HUST

Thời gian thực hiện của CPU

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

Thời gian thực hiện của CPU =

Số chu kỳ xung nhịp x Thời gian một chu kỳ

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

n: số chu kỳ xung nhịp

- Hiệu năng được tăng lên bằng cách:
 - Giảm số chu kỳ xung nhịp n
 - Tăng tốc độ xung nhịp f₀

2017

Kiến trúc máy tính

Ví dụ

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

2017

Kiến trúc máy tính

25

Ví dụ (tiếp)

Ta có: $t = \frac{n}{f}$

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

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

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

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

Tốc độ xung nhịp cần thiết cho máy B:

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

2017

Kiến trúc máy tính

26

Số lệnh và số chu kỳ trên một lệnh

Số chu kỳ xung nhịp của chương trình:

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

$$n = IC \times CPI$$

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

Vậy thời gian thực hiện của CPU:

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

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

2017

Kiến trúc máy tính

27

Ví dụ

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

2017

Kiến trúc máy tính

28

Ví dụ (tiếp)

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

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

$$IC_A = IC_B = IC$$

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

$$t_A = IC_A \times CPI_A \times T_A = IC \times 2.0 \times 250\text{ps} = IC \times 500\text{ps}$$

$$t_B = IC_B \times CPI_B \times T_B = IC \times 1.2 \times 500\text{ps} = IC \times 600\text{ps}$$

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

Kết luận: máy A nhanh hơn máy B 1.2 lần

2017

Kiến trúc máy tính

29

CPI trung bình

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

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

- CPI trung bình:

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

2017

Kiến trúc máy tính

30

Ví dụ

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

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

2017

Kiến trúc máy tính

31

Ví dụ

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

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

- Dãy lệnh 1: Số lệnh = 50
 - Số chu kỳ = $= 1 \times 20 + 2 \times 10 + 3 \times 20 = 100$
 - $CPI_{TB} = 100/50 = 2.0$
- Dãy lệnh 2: Số lệnh = 60
 - Số chu kỳ = $= 1 \times 40 + 2 \times 10 + 3 \times 10 = 90$
 - $CPI_{TB} = 90/60 = 1.5$

2017

Kiến trúc máy tính

32

NKK-HUST

Tóm tắt về Hiệu năng

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

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

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

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

NKK-HUST

MIPS như là thước đo hiệu năng

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

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

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

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

2017

Kiến trúc máy tính

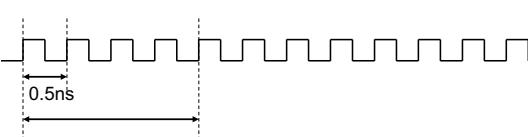
NKK-HUST

Ví dụ

Tính MIPS của bộ xử lý với:
clock rate = 2GHz và CPI = 4

Ví dụ

Tính MIPS của bộ xử lý với:
clock rate = 2GHz và CPI = 4



- Chu kỳ $T_0 = 1/(2 \times 10^9) = 0.5\text{ns}$
- CPI = 4 \rightarrow thời gian thực hiện 1 lệnh = $4 \times 0.5\text{ns} = 2\text{ns}$
- Số lệnh thực hiện trong 1s = $(10^9\text{ns})/(2\text{ns}) = 5 \times 10^8$ lệnh
- Vậy bộ xử lý thực hiện được 500 MIPS

NKK-HUST

Ví dụ

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

Ví dụ

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



Chu kỳ $T_0 = 1/10^9 = 1\text{ns}$

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

NKK-HUST

MFLOPS

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

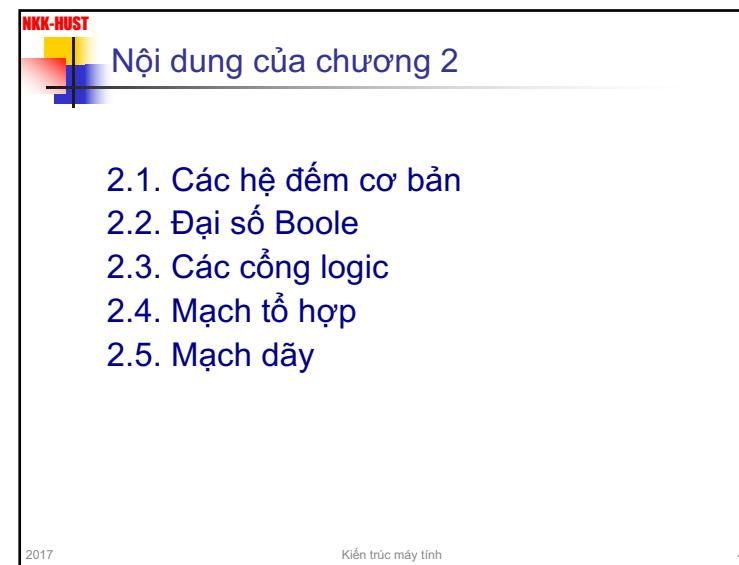
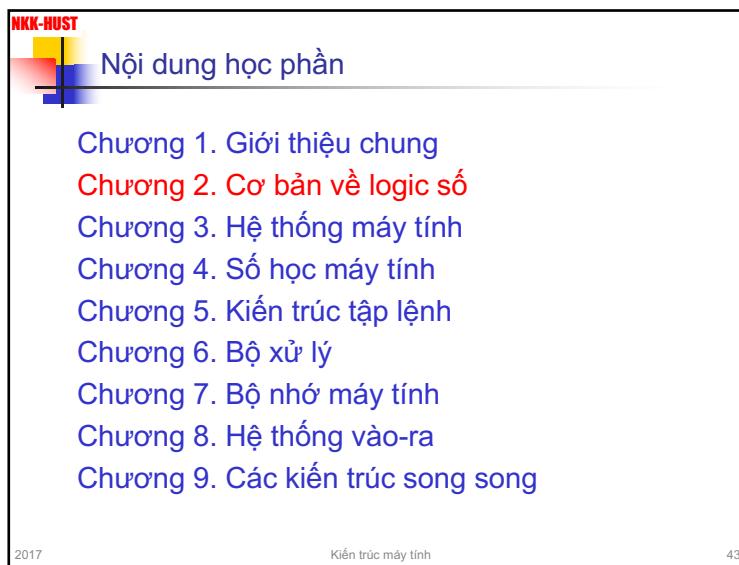
$$\text{MFLOPS} = \frac{\text{Executed floating point operations}}{\text{Execution time} \times 10^6}$$

GFLOPS (10^9)
TFLOPS (10^{12})
PFLOPS (10^{15})

Các ý tưởng tuyệt vời trong kiến trúc máy tính

1. Design for **Moore's Law**
Thiết kế theo luật Moore
2. Use **abstraction** to simplify design
Sử dụng trừu tượng hóa để đơn giản thiết kế
3. Make the **common case fast**
Làm cho các trường hợp phổ biến thực hiện nhanh
4. Performance via **parallelism**
Tăng hiệu năng qua xử lý song song
5. Performance via **pipelining**
Tăng hiệu năng qua kỹ thuật đường ống
6. Performance via **prediction**
Tăng hiệu năng thông qua dự đoán
7. **Hierarchy** of memories
Phân cấp bộ nhớ
8. **Dependability** via redundancy
Tăng độ tin cậy thông qua dự phòng





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

NKK-HUST

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:

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

NKK-HUST

Ví dụ số thập phân

472.38 = 4x10² + 7x10¹ + 2x10⁰ + 3x10⁻¹ + 8x10⁻²

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

NKK-HUST

2. Hệ nhị phân

- Cơ số 2
- 2 chữ số nhị phân: 0 và 1
- Chữ số nhị phân được gọi là **bit** (*binary digit*)
- **bit** là đơn vị thông tin nhỏ nhất
- Dùng n bit có thể biểu diễn được 2^n giá trị khác nhau:
 - $00\dots000 = 0$
 - $11\dots111 = 2^n - 1$
- Các lệnh của chương trình và dữ liệu trong máy tính đều được mã hóa bằng số nhị phân

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

NKK-HUST

Đơn vị dữ liệu và thông tin trong máy tính

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

NKK-HUST

Qui ước mới về ký hiệu đơn vị dữ liệu

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

2017

Kiến trúc máy tính

NKK-HUST

Phương pháp phân tích thành tổng của các 2ⁱ

- 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

NKK-HUST

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

biểu diễn
theo
chiều
mũi tên

NKK-HUST

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

- Ví dụ 2: chuyển đổi $0.81_{(10)}$

■	$0.81 \times 2 =$	1.62	$\text{phần nguyên} =$	1	↓
■	$0.62 \times 2 =$	1.24	$\text{phần nguyên} =$	1	
■	$0.24 \times 2 =$	0.48	$\text{phần nguyên} =$	0	
■	$0.48 \times 2 =$	0.96	$\text{phần nguyên} =$	0	
■	$0.96 \times 2 =$	1.92	$\text{phần nguyên} =$	1	
■	$0.92 \times 2 =$	1.84	$\text{phần nguyên} =$	1	
■	$0.84 \times 2 =$	1.68	$\text{phần nguyên} =$	1	
■	$0.81_{(10)} \approx 0.1100111_{(2)}$				

NKK-HUST

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

2017

Kiến trúc máy tính

6

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

NKK-HUST

2.2. Đại số Boolean

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

Phép toán đại số Boole với hai biến		
A	B	A AND B $A \bullet B$
0	0	0
0	1	0
1	0	0
1	1	1

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

NOT là phép toán 1 biến

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

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

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

NKK-HUST

Các đồng nhất thức của đại số Boole

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

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

$$1 \cdot A = A$$

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

$$0 \cdot A = 0$$

$$A \cdot A = A$$

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

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

$$A + B = B + A$$

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

$$0 + A = A$$

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

$$1 + A = 1$$

$$A + A = A$$

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

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

2017

Kiến trúc máy tính

NKK-HUST

2.3. Các cổng logic (Logic Gates)

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

2017

Kiến trúc máy tính

65

NKK-HUST

Ký hiệu các cổng logic

Name	Graphical Symbol	Algebraic Function	Truth Table															
AND		$F = A \bullet B$ 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		$F = \overline{A \bullet B}$	<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																
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																

2017

66

NKK-HUST

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}

2017

Kiến trúc máy tính

67

NKK-HUST

Sử dụng cổng NAND

The four diagrams show the implementation of:

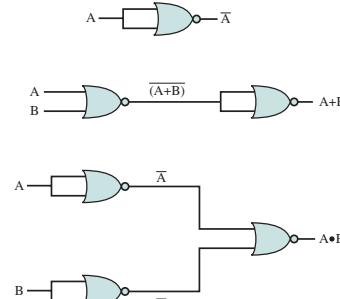
- NOT gate: A single NAND gate with both inputs connected together (A and A).
- AND gate: Two NAND gates in series. The first NAND gate has inputs A and B, and its output is connected to one input of the second NAND gate along with its own output (forming a feedback loop). The second NAND gate's output is the final output F.
- OR gate: Two NAND gates in series. The first NAND gate has inputs A and B, and its output is connected to one input of the second NAND gate along with its own output (forming a feedback loop). The second NAND gate's output is the final output F.
- XOR gate: Two NAND gates in series. The first NAND gate has inputs A and B, and its output is connected to one input of the second NAND gate along with its own output (forming a feedback loop). The second NAND gate's output is connected to one input of a third NAND gate, along with its own output (forming a feedback loop). The third NAND gate's output is the final output F.

2017

Kiến trúc máy tính

68

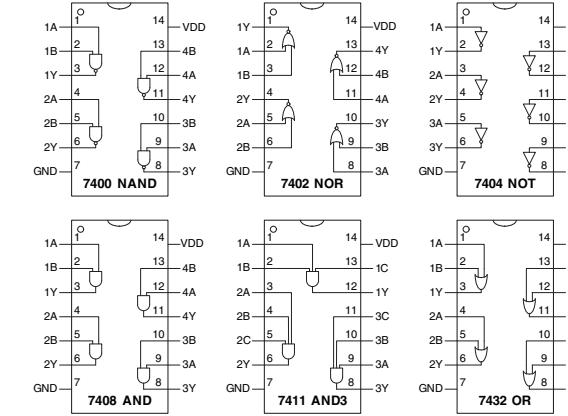
Sử dụng cổng NOR



2017

Kiến trúc máy tín

69



2017

Kiến trúc máy tính

70

2.4. Mạch tổ hợp

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

2017

Kiến trúc máy tín

71

Mạch tổ hợp

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

2017

Kiến trúc máy tính

72

Ví dụ

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

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

2017 Kiến trúc máy tính 73

Bộ chọn kênh (Multiplexer - MUX)

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

2017 Kiến trúc máy tính 74

Bộ chọn kênh 4 đầu vào

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

$$F = D0 \cdot \bar{S}2 \cdot \bar{S}1 + D1 \cdot \bar{S}2 \cdot S1 + D2 \cdot S2 \cdot \bar{S}1 + D3 \cdot S2 \cdot S1$$

2017 Kiến trúc máy tính 75

Bộ giải mã (Decoder)

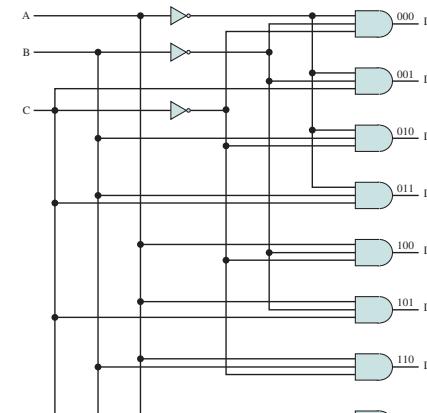
- N đầu vào, 2^N đầu ra
- Với một tổ hợp của N đầu vào, chỉ có một đầu ra tích cực (khác với các đầu ra còn lại)
- Ví dụ: Bộ giải mã 2 ra 4

2:4 Decoder					
A ₁	A ₀	Y ₃	Y ₂	Y ₁	Y ₀
11					
10					
01					
00					

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

2017 Kiến trúc máy tính 76

Thực hiện bộ giải mã 3 ra 8



201

Kiến trúc máy tí

77

Bộ công

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

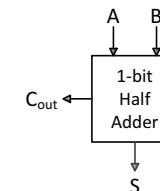
2017

Kiến trúc máy tính

78

Bộ cộng bán phần 1-bit

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



$$S = A \oplus A^\perp$$

$$C_{out} = A E$$

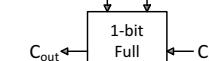
Đầu vào		Đầu ra	
A	B	S	C _{out}
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

201

Kiến trúc máy tính

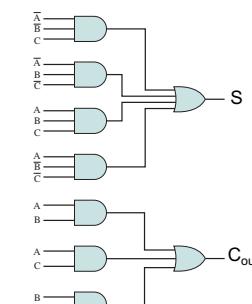
79

Bộ cộng toàn phần 1-bit



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

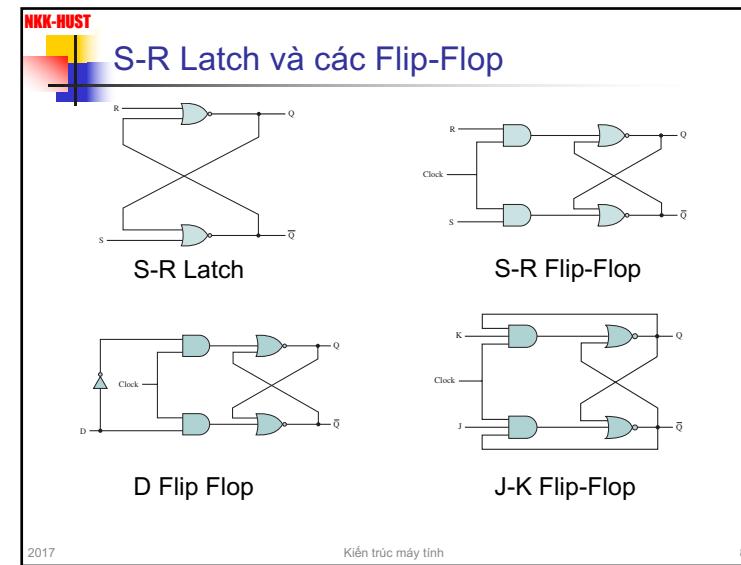
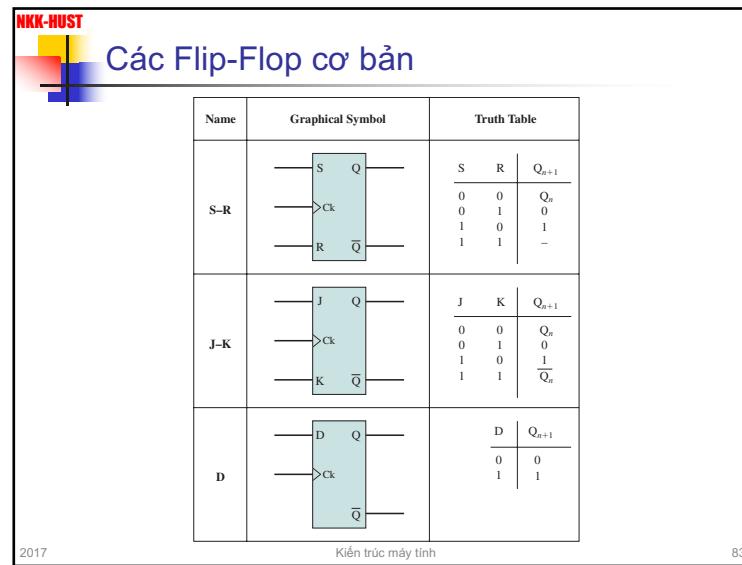
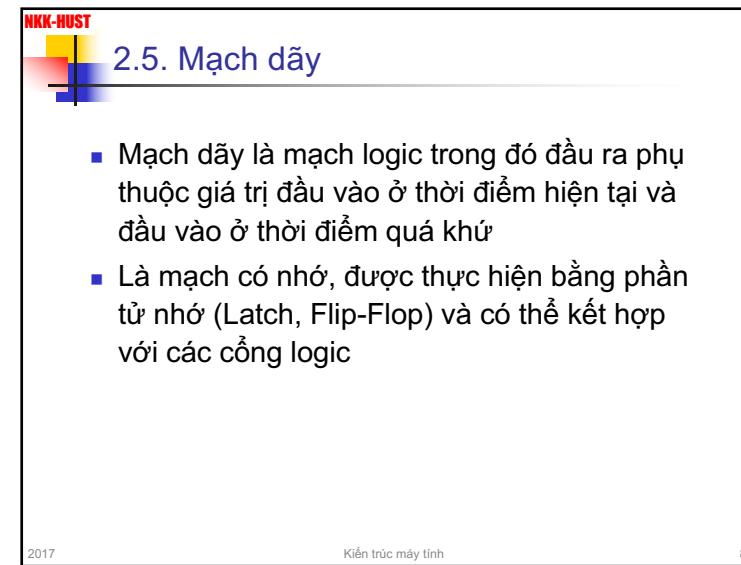
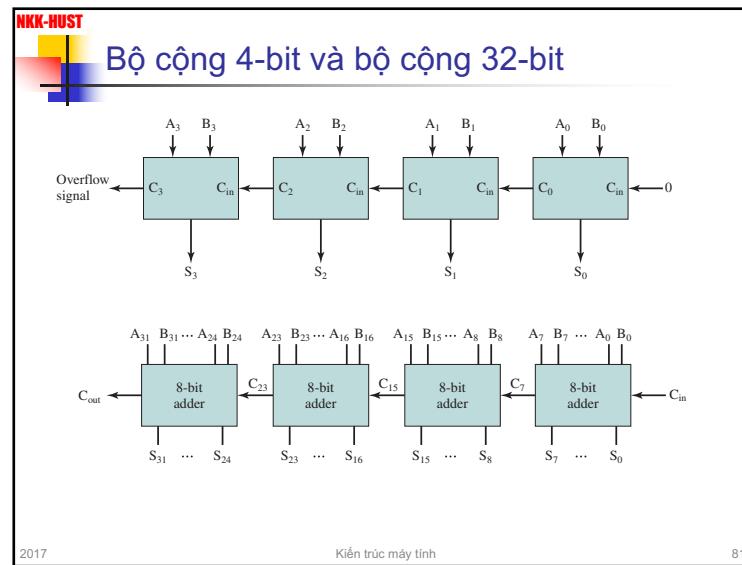
Đầu vào			Đầu ra	
C _{in}	A	B	S	C _{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

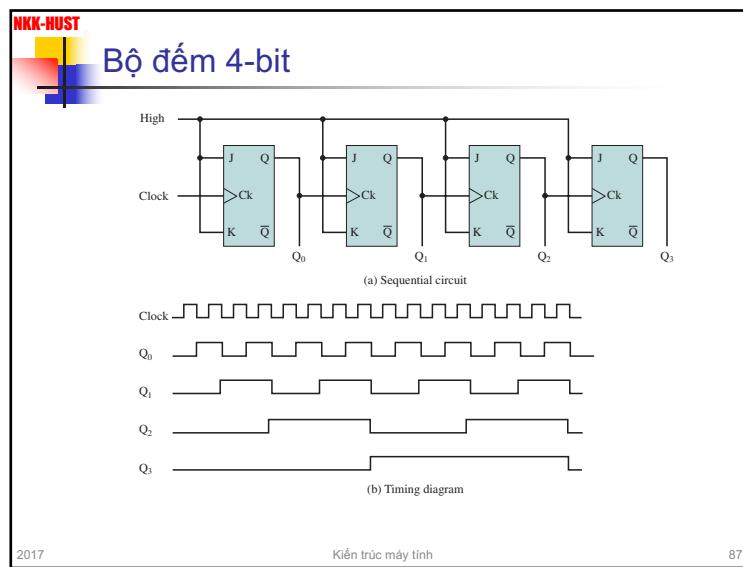
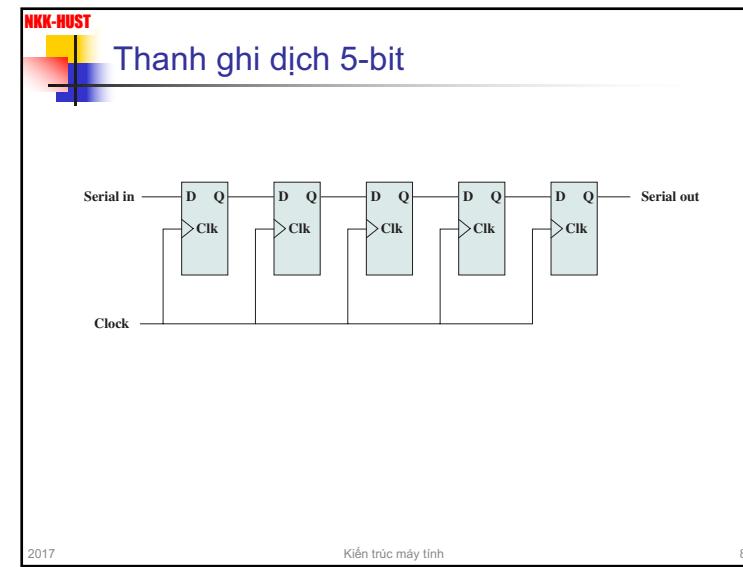
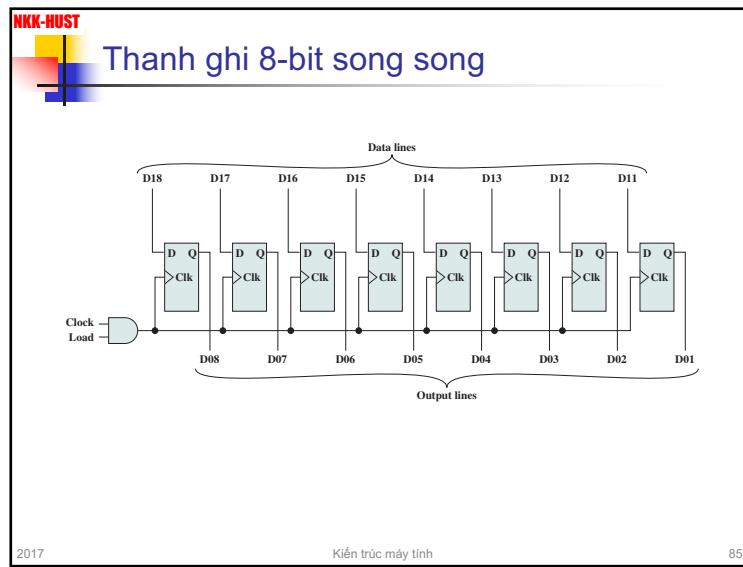


2017

Kiến trúc máy tính

81





Hết chương 2

NKK-HUST

Kiến trúc máy tính

Chương 3

HỆ THỐNG MÁY TÍNH



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

NKK-HUST

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

2017

Kiến trúc máy tính

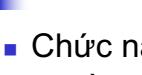
9

The diagram illustrates the basic components of a computer system and their interconnections:

- CPU** and **Bộ nhớ chính** (Main Memory) are connected to the **Bus hệ thống** (System Bus).
- The **Bus hệ thống** is also connected to the **Hệ thống vào-ra** (Input/Output System).
- Double-headed vertical arrows indicate bidirectional communication between each component and the bus.

3.1. Các thành phần cơ bản của máy tính

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



1. Bộ xử lý trung tâm (CPU)

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

NKK-HUST

Các thành phần cơ bản của CPU

The diagram illustrates the basic components of a CPU. It features three rectangular boxes arranged vertically. The top box is labeled 'Đơn vị điều khiển' (Control Unit). The middle box is labeled 'Đơn vị số học và logic' (Arithmetic and Logic Unit). The bottom box is labeled 'Tập thanh ghi' (Register File). A double-headed horizontal arrow between the middle and bottom boxes is labeled 'Bus hệ thống' (System Bus), indicating their connection.

■ Đơn vị điều khiển

- *Control Unit (CU)*
- Điều khiển hoạt động của máy tính theo chương trình đã định sẵn

■ Đơn vị số học và logic

- *Arithmetic and Logic Unit (ALU)*
- Thực hiện các phép toán số học và phép toán logic

■ Tập thanh ghi

- *Register File (RF)*
- Gồm các thanh ghi chứa các thông tin phục vụ cho hoạt động của CPU

2017

Kiến trúc máy tính

NKK-HUST

2. Bộ nhớ máy tính

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

```
graph LR; CPU[CPU] <--> Cache[Bộ nhớ đệm]; Cache <--> Main[Bộ nhớ chính]; Main <--> Storage[Các thiết bị lưu trữ]
```

2017

Kiến trúc máy tính

98

NKK-HUST

Bộ nhớ chính (Main memory)

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

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

Kiến trúc máy tính

2017

Bộ nhớ đệm (Cache memory)

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

201

Kiến trúc máy tí

97

Thiết bị lưu trữ (Storage Devices)

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

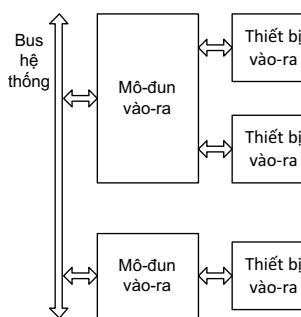
201

Kiến trúc máy tính

9

3. Hệ thống vào-rã

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



201

Kiến trúc máy tí

99

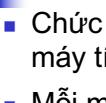
Các thiết bị vào-ra

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

201

Kiến trúc máy tính

10



Mô-đun vào-ra

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

3.2. Hoạt động cơ bản của máy tính

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

NKK-HUST

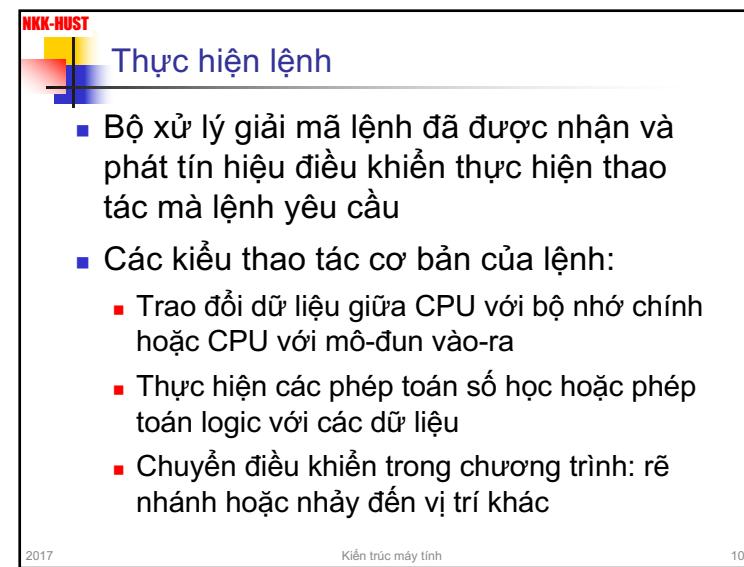
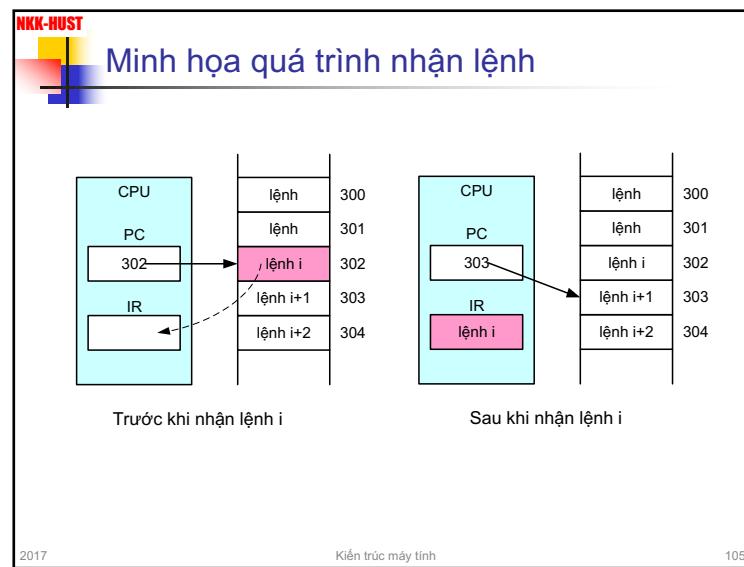
1. Thực hiện chương trình

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



Nhận lệnh

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



2. Ngắt (Interrupt)

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

Hoạt động với ngắt từ bên ngoài

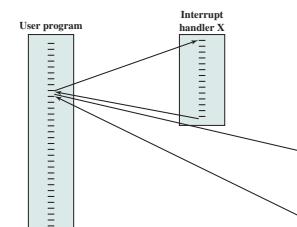
- Sau khi hoàn thành mỗi một lệnh, bộ xử lý kiểm tra tín hiệu ngắt
 - Nếu không có ngắt, bộ xử lý nhận lệnh tiếp theo của chương trình hiện tại
 - Nếu có tín hiệu ngắt:
 - Tạm dừng (suspend) chương trình đang thực hiện
 - Cắt ngắt cảnh (các thông tin liên quan đến chương trình bị ngắt)
 - Thiết lập bộ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

NKK-HUST

Hoạt động ngắt (tiếp)

Xử lý với nhiều tín hiệu yêu cầu ngắt

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



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

Xử lý ngắt ưu tiên

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

User program

Interrupt handler X

Interrupt handler Y

Kiến trúc máy tính



3. Hoạt động vào-ra

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

3.3. Bus máy tính

1. Luồng thông tin trong máy tính

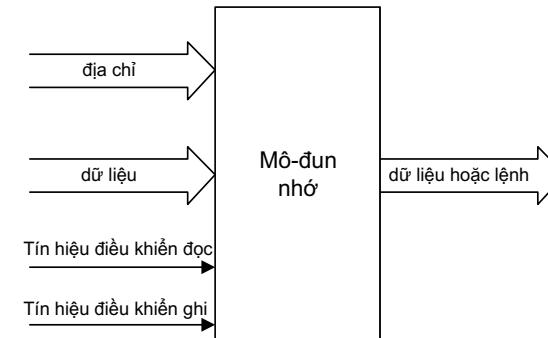
- Các mô-đun trong máy tính:
 - CPU
 - Mô-đun nhớ
 - Mô-đun vào-ra

2017

Kiến trúc máy tín

113

Kết nối mô-đun nhớ



201

Kiến trúc máy tính

114

Kết nối mô-đun nhớ (tiếp)

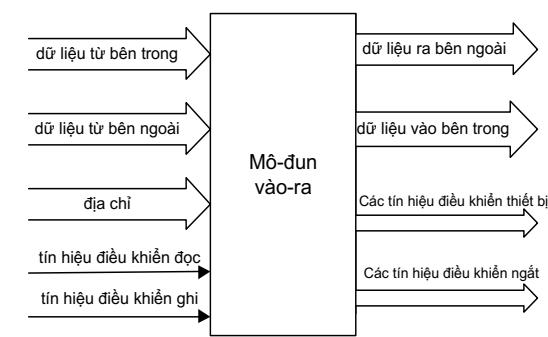
- Địa chỉ đưa đến để xác định ngăn nhớ
 - Dữ liệu được đưa đến khi ghi
 - Dữ liệu hoặc lệnh được đưa ra khi đọc
 - Bộ nhớ không phân biệt lệnh và dữ liệu
 - Nhận các tín hiệu điều khiển:
 - Điều khiển đọc (Read)
 - Điều khiển ghi (Write)

2017

Kiến trúc máy tín

115

Kết nối mô-đun vào-rã



201

Kiến trúc máy tính

110

Kết nối mô-đun vào-ra (tiếp)

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

Kết nối CPU

lệnh

dữ liệu

Các tín hiệu điều khiển ngắt

địa chỉ

dữ liệu

Các tín hiệu điều khiển bộ nhớ và vào-ra

CPU

Kết nối CPU (tiếp)

NKK-HUST

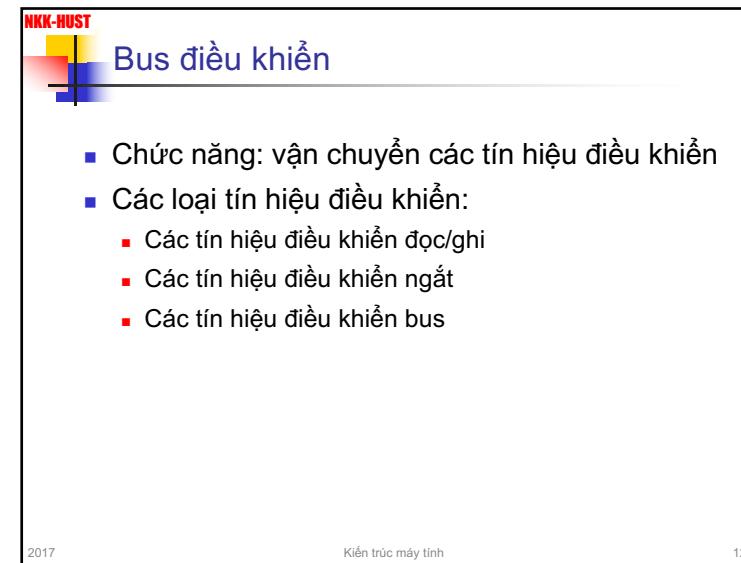
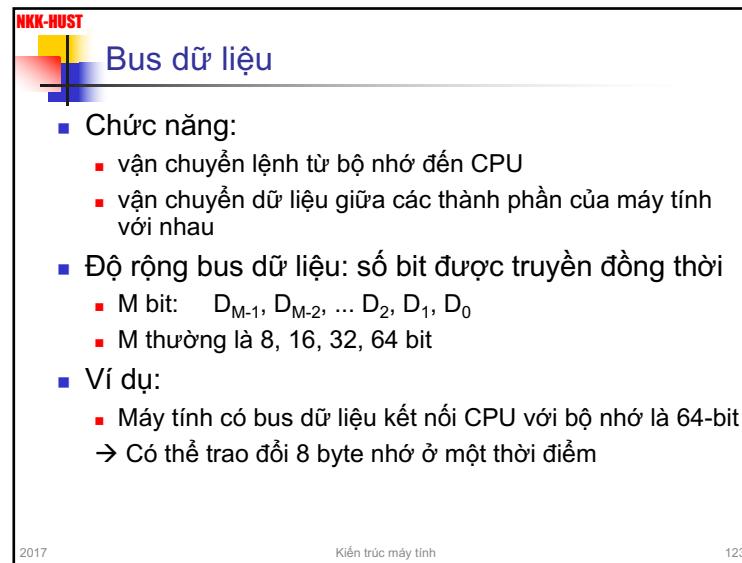
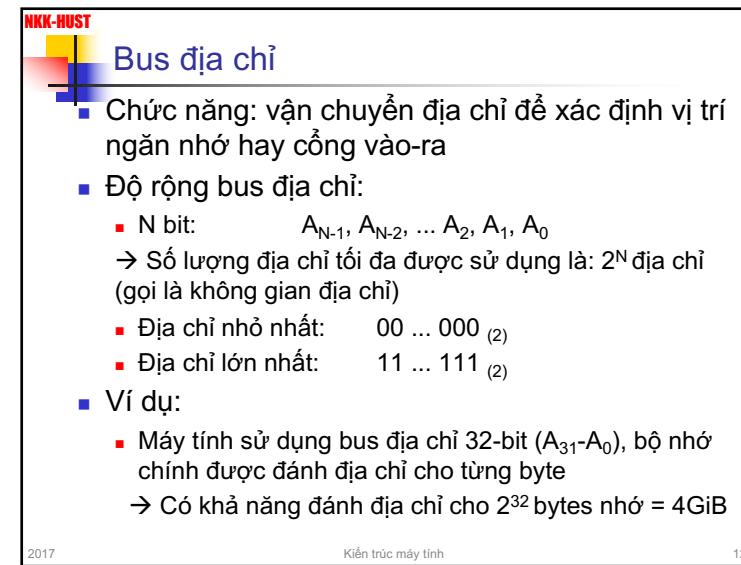
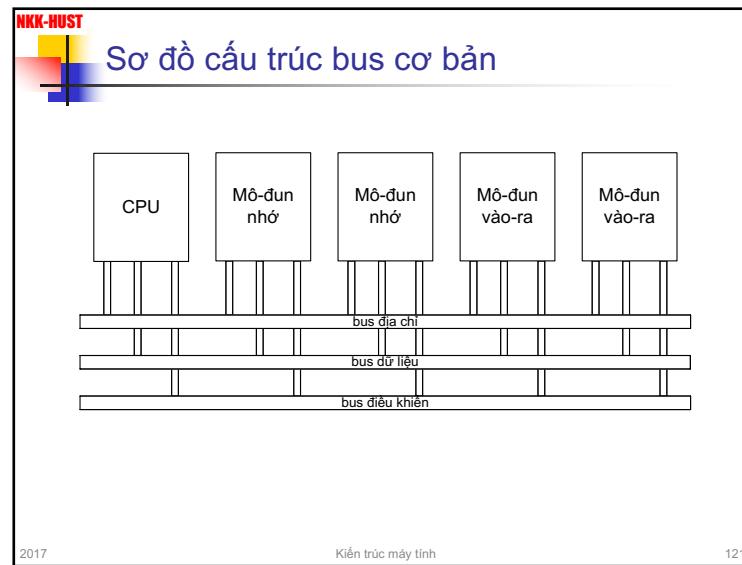
2. Cấu trúc bus cơ bản

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

2017

Kiến trúc máy tính

1



Một số tín hiệu điều khiển diễn hình

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

2017

Kiến trúc máy tín

125

Một số tín hiệu điều khiển điển hình (tiếp)

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

2017

Kiến trúc máy tính

126

Một số tín hiệu điều khiển điển hình (tiếp)

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

2017

Kiến trúc máy tín

127

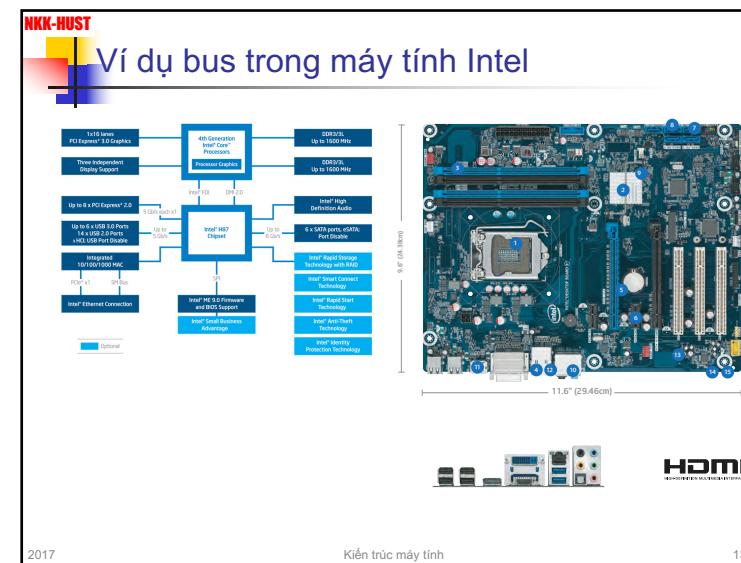
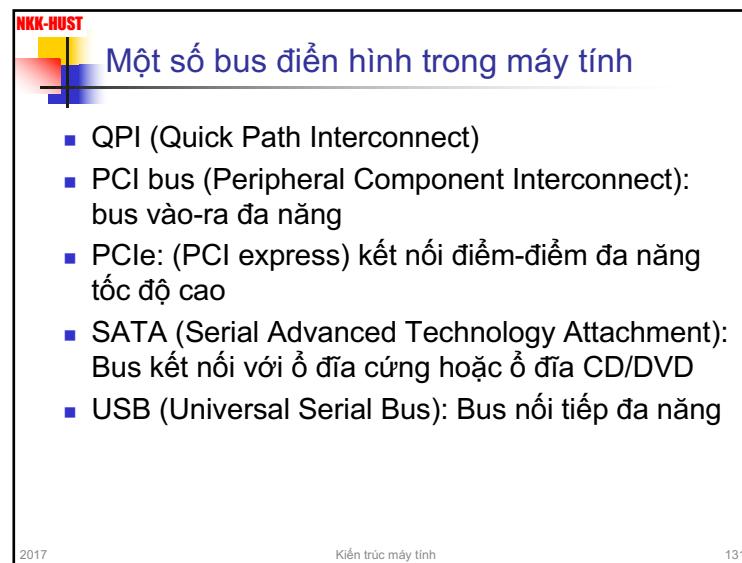
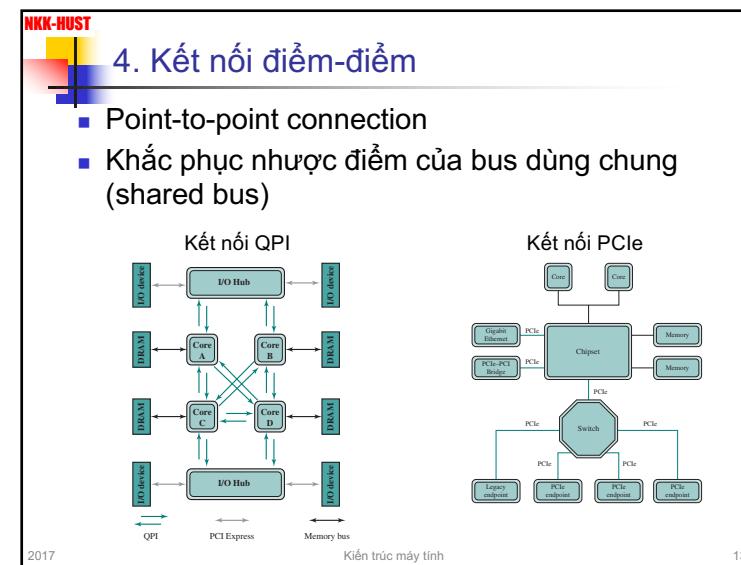
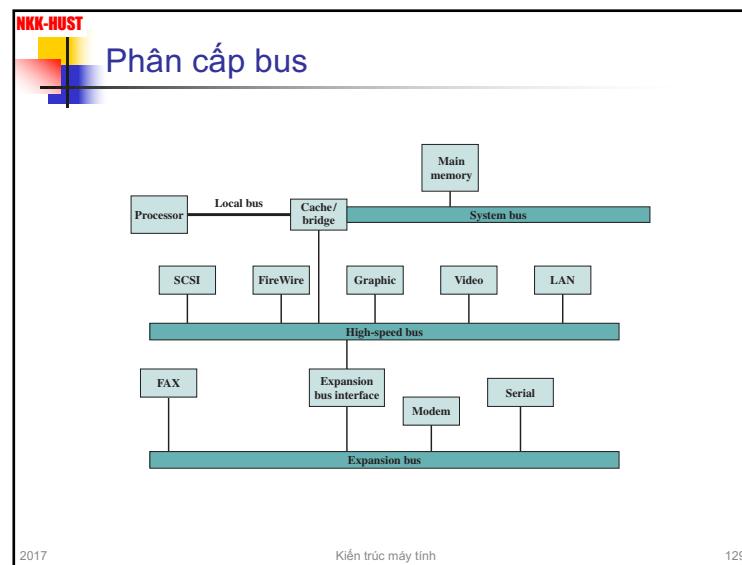
3. Phân cấp bus

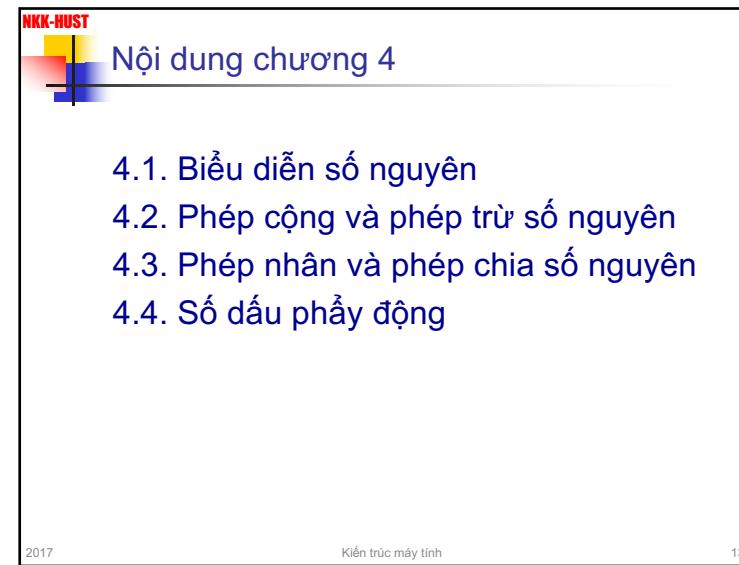
- **Đơn bus:** Tất cả các mô-đun kết nối vào bus chung
 - Bus chỉ phục vụ được một yêu cầu trao đổi dữ liệu tại một thời điểm → độ trễ lớn
 - Bus phải có tốc độ bằng tốc độ bus của mô-đun nhanh nhất trong hệ thống
 - **Đa bus:** Phân cấp thành nhiều bus cho các mô-đun khác nhau và có tốc độ khác nhau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

2017

Kiến trúc máy tính

128





NKK-HUST

4.1. Biểu diễn số nguyên

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

NKK-HUST

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: [0, $2^n - 1$]

NKK-HUST

Ví dụ 1

- Biểu diễn các số nguyên không dấu sau đây bằng 8-bit:
 $A = 41 ; B = 150$
Giải:
 $A = 41 = 32 + 8 + 1 = 2^5 + 2^3 + 2^0$
 $41 = 0010\ 1001$

 $B = 150 = 128 + 16 + 4 + 2 = 2^7 + 2^4 + 2^2 + 2^1$
 $150 = 1001\ 0110$

NKK-HUST

Ví dụ 2

- Cho các số nguyên không dấu M, N được biểu diễn bằng 8-bit như sau:
 - $M = 0001\ 0010 = 2^4 + 2^1 = 16 + 2 = 18$
 - $N = 1011\ 1001 = 2^7 + 2^5 + 2^4 + 2^3 + 2^0 = 128 + 32 + 16 + 8 + 1 = 185$

Xác định giá trị của chúng ?

Giải:

$$\begin{aligned} M &= 0001\ 0010 = 2^4 + 2^1 = 16 + 2 = 18 \\ N &= 1011\ 1001 = 2^7 + 2^5 + 2^4 + 2^3 + 2^0 \\ &\quad = 128 + 32 + 16 + 8 + 1 = 185 \end{aligned}$$

2017

Kiến trúc máy tính

1

NKK-HUST

Với n = 8 bit

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

Chú ý:

$$\begin{array}{r}
 1111\ 1111 \\
 +\ 0000\ 0001 \\
 \hline
 1\ 0000\ 0000
 \end{array}$$

có **nhớ ra ngoài** (Carry out)

255 + 1 = 0 ???

do vượt ra khỏi dải biểu diễn

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

2017

Kiến trúc máy tính

141

NKK-HUST

Trục số học với n = 8 bit

Trục số học:

Trục số học máy tính:

2017

Kiến trúc máy tính

142

NKK-HUST

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 $2^{64} - 1$

2017

Kiến trúc máy tính

143

NKK-HUST

2. Biểu diễn số nguyên có dấu

Số bù một và Số bù hai

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

2017

Kiến trúc máy tính

144

Ví dụ

Với $n = 8$ bit, cho $A = 0010\ 0101$

- Số bù một của A được tính như sau:

$$\begin{array}{r} 1111\ 1111 \quad (2^8 - 1) \\ - 0010\ 0101 \quad (A) \\ \hline 1101\ 1010 \end{array}$$

→ đảo các bit của A

- Số bù hai của A được tính như sau:

$$\begin{array}{r} 1\ 0000\ 0000 \quad (2^8) \\ - 0010\ 0101 \quad (A) \\ \hline 1101\ 1011 \end{array}$$

→ thực hiện khó khăn

2017

Kiến trúc máy tính

145

Quy tắc tìm Số bù một và Số bù hai

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

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

(bỏ qua bit nhớ ra ngoài)

→ Số bù hai của $A = -A$

2017

Kiến trúc máy tính

146

Biểu diễn số nguyên có dấu theo mã bù hai

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

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

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

2017

Kiến trúc máy tính

147

Ví dụ

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

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

Giải:

$$\begin{array}{rcl} A & = & +58 & = & 0011\ 1010 \\ B & = & -80 & = & 1011\ 0000 \end{array}$$

Ta có: $+80 = 0101\ 0000$

$$\begin{array}{rcl} \text{Số bù một} & = & 1010\ 1111 \\ & & + 1 \\ \text{Số bù hai} & = & 1011\ 0000 \end{array}$$

Vậy: $B = -80 = 1011\ 0000$

2017

Kiến trúc máy tính

148

Xác định giá trị của số dương

- Dạng tổng quát của số dương:
$$0a_{n-2} \dots a_2 a_1 a_0$$
- Giá trị của số dương:
$$A = \sum_{i=0}^{n-2} a_i 2^i$$
- Dải biểu diễn cho số dương: $[0, +(2^{n-1} - 1)]$

2017

Kiến trúc máy tính

149

Xác định giá trị của số âm

- Dạng tổng quát của số âm:
$$1a_{n-2} \dots a_2 a_1 a_0$$
- Giá trị của số âm:
$$A = -2^{n-1} + \sum_{i=0}^{n-2} a_i 2^i$$

Chứng minh?

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

2017

Kiến trúc máy tính

150

Công thức xác định giá trị số âm

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

$$\begin{aligned} -A &= 0 \overline{a_{n-2}} \overline{a_{n-3}} \dots \overline{a_2} \overline{a_1} \overline{a_0} + 1 \\ &= 11 \dots 111 - a_{n-2} a_{n-3} \dots a_2 a_1 a_0 + 1 \end{aligned}$$

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

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

2017

Kiến trúc máy tính

151

Công thức tổng quát cho số nguyên có dấu

- Dạng tổng quát của số nguyên có dấu A:
$$a_{n-1} a_{n-2} \dots a_2 a_1 a_0$$
- Giá trị của A được xác định như sau:
$$A = -a_{n-1} 2^{n-1} + \sum_{i=0}^{n-2} a_i 2^i$$
- Dải biểu diễn: $[-(2^{n-1}), +(2^{n-1}-1)]$

2017

Kiến trúc máy tính

152

NKK-HUST



Ví dụ

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

Giải:

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

NKK-HUST

Với n = 8 bit

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

Chú ý:

$$+127 + 1 = -128$$

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

có tràn xảy ra (Overflow)

(do vượt ra khỏi dài biểu diễn)

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

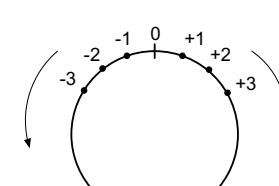
2017

Kiến trúc máy tính

1

NKK-HUST

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

- Trục số học:
- Trục số học máy tính:


NKK-HUST

Với n = 16 bit, 32 bit, 64 bit

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

2017

Kiến trúc máy tính

1

Mở rộng bit cho số nguyên

- Mở rộng theo số không dấu (Zero-extended): thêm các bit 0 vào bên trái
- Mở rộng theo số có dấu (Sign-extended):
 - Số dương:
 $+19 = \begin{array}{r} 0001\ 0011 \\ (8\text{bit}) \end{array}$
 - $+19 = \begin{array}{r} 0000\ 0000\ 0001\ 0011 \\ (16\text{bit}) \end{array}$
 - thêm các bit 0 vào bên trái
- Số âm:
 - $-19 = \begin{array}{r} 1110\ 1101 \\ (8\text{bit}) \end{array}$
 - $-19 = \begin{array}{r} 1111\ 1111\ 1110\ 1101 \\ (16\text{bit}) \end{array}$
 - thêm các bit 1 vào bên trái

2017

Kiến trúc máy tính

157

4.2. Thực hiện phép cộng/trừ với số nguyên

1. Phép cộng số nguyên không dấu

Bộ cộng n-bit

```

    graph LR
        Y((Y)) -- "n bit" --> Adder[ ]
        X((X)) -- "n bit" --> Adder
        Cin((C_in)) --> Adder
        Adder -- "n bit" --> S((S))
        Adder --> Cout((C_out))
    
```

2017

Kiến trúc máy tính

158

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 có *nhớ ra ngoài* (Carry Out)
- Hiện tượng *nhớ ra ngoài* xảy ra khi: $tổng > (2^n - 1)$

2017

Kiến trúc máy tính

159

Ví dụ cộng số nguyên không dấu

- $\begin{array}{r} 57 \\ + 34 \\ \hline 91 \end{array} = \begin{array}{r} 0011\ 1001 \\ + 0010\ 0010 \\ \hline 0101\ 1011 \end{array} = 64+16+8+2+1=91 \rightarrow \text{đúng}$
- $\begin{array}{r} 209 \\ + 73 \\ \hline 282 \end{array} = \begin{array}{r} 1101\ 0001 \\ + 0100\ 1001 \\ \hline 1\ 0001\ 1010 \end{array}$
kết quả = 0001 1010 = 16+8+2=26 → sai
do có *nhớ ra ngoài* ($C_{out}=1$)

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

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

2017

Kiến trúc máy tính

160

2. Phép đảo dấu

- Ta có:

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

- Lấy bù hai của số âm:

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

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

2017

Kiến trúc máy tính

16

3. Cộng số nguyên có dấu

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

2017

Kiến trúc máy tính

16

Ví dụ cộng số nguyên có dấu không tròn

- $$\begin{array}{r}
 (+ 70) \\
 + (+ 42) \\
 \hline
 + 112
 \end{array}
 = \begin{array}{r}
 0100\ 0110 \\
 \underline{0010\ 1010} \\
 0111\ 0000
 \end{array}
 = +112$$
 - $$\begin{array}{r}
 (+ 97) \\
 + (- 52) \\
 \hline
 + 45
 \end{array}
 = \begin{array}{r}
 0110\ 0001 \\
 \underline{1100\ 1100} \\
 1\ 0010\ 1101
 \end{array}
 = +45$$
(+52=0011 0100)
 - $$\begin{array}{r}
 (- 90) \\
 + (+ 36) \\
 \hline
 - 54
 \end{array}
 = \begin{array}{r}
 1010\ 0110 \\
 \underline{0010\ 0100} \\
 1100\ 1010
 \end{array}
 = -54$$
(+90=0101 1010)
 - $$\begin{array}{r}
 (- 74) \\
 +(- 30) \\
 \hline
 -104
 \end{array}
 = \begin{array}{r}
 1011\ 0110 \\
 \underline{1110\ 0010} \\
 1\ 1001\ 1000
 \end{array}
 = -104$$
(+74=0100 1010)
(+30=0001 1110)

2017

Kiến trúc máy tính

16

Ví dụ cộng số nguyên có dấu bị tràn

- $(+ 75) = 0100\ 1011$
 $+ (\underline{+ 82}) = \underline{0101\ 0010}$
 $+ 157 \quad \quad \quad 1001\ 1101$
 $= -128 + 16 + 8 + 4 + 1 = -99 \rightarrow sai$
 - $(- 104) = 1001\ 1000 \quad (+104=0110\ 1000)$
 $+ (- 43) = \underline{1101\ 0101} \quad (+43=0010\ 1011)$
 $- 147 \quad \quad \quad 1\ 0110\ 1101$
 $= 64 + 32 + 8 + 4 + 1 = +109 \rightarrow sai$
 - Cả hai ví dụ đều **tròn** vì tổng nằm ngoài dải biểu diễn $[-128, +127]$

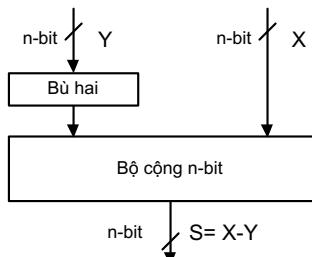
2017

Kiến trúc máy tính

16

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



201

Kiến trúc máy tí

16

Nhân số nguyên không dấu (tiếp)

- Các **tích riêng phần** được xác định như sau:
 - Nếu bit của số nhân bằng 0 \rightarrow tích riêng phần bằng 0
 - Nếu bit của số nhân bằng 1 \rightarrow 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)

201

Kiến trúc máy tí

16

4.3. Phép nhân và phép chia số nguyên

1. Nhân số nguyên không dấu

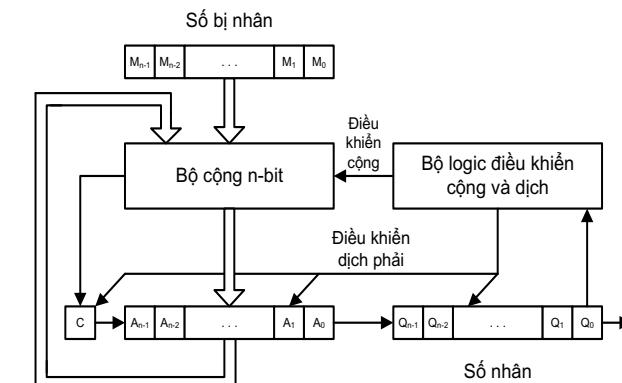
$$\begin{array}{r}
 1011 \\
 \times 1101 \\
 \hline
 1011 \\
 0000 \\
 1011 \\
 1011 \\
 \hline
 10001111
 \end{array}
 \quad \left. \begin{array}{l} \text{Số bị nhân (11)} \\ \text{Số nhân (13)} \\ \\ \text{Các tích riêng phần} \end{array} \right\} \quad \begin{array}{l} \text{Tích} \\ (143) \end{array}$$

2

Kiến trúc máy tính

16

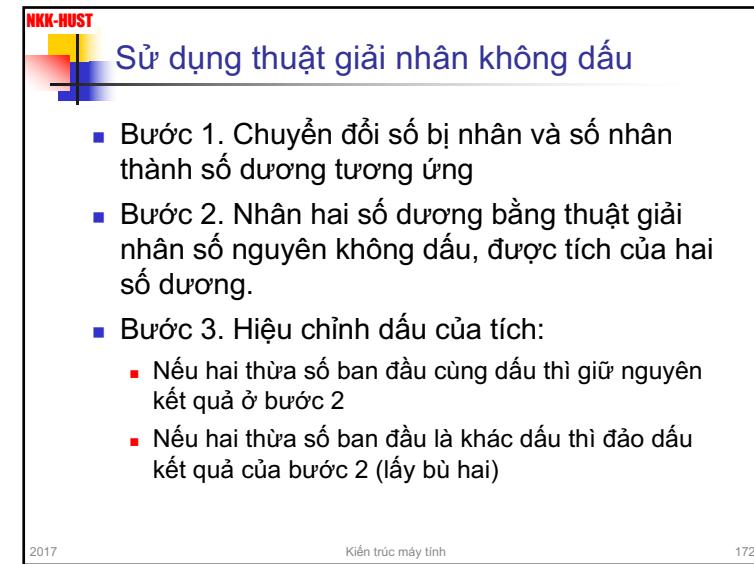
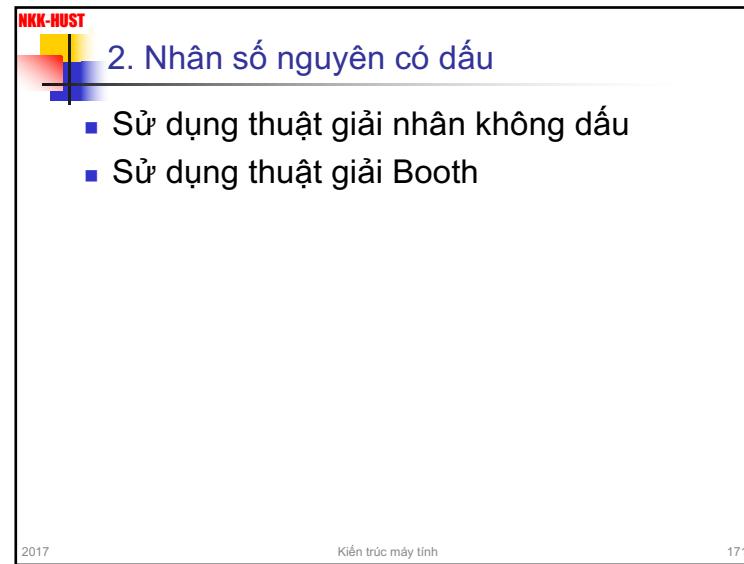
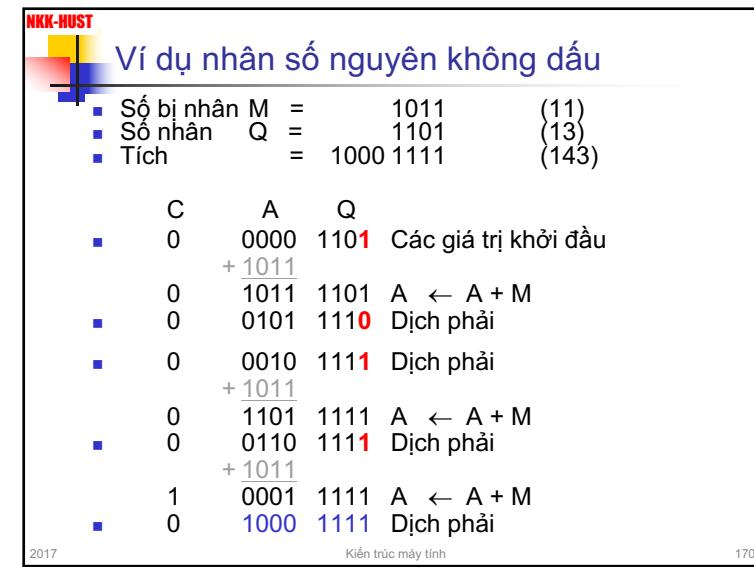
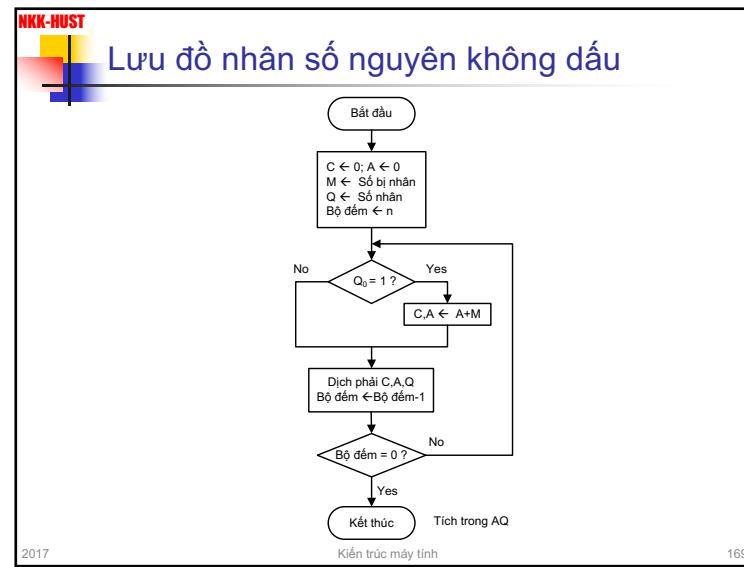
Bộ nhân số nguyên không dấu



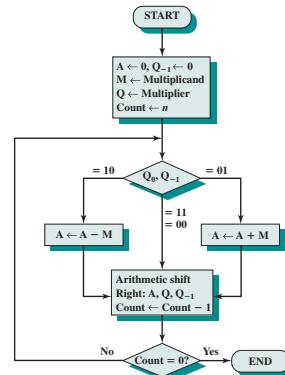
2

Kiến trúc máy tính

168



Thuật giải Booth (tham khảo sách COA)



2017

Kiến trúc máy tính

173

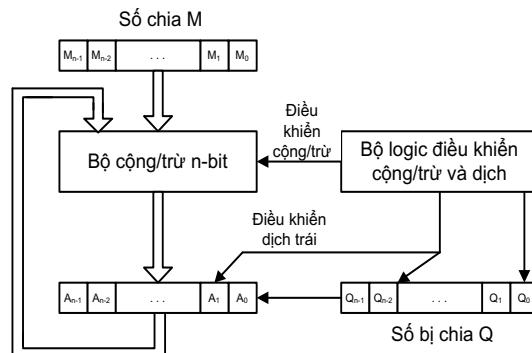
3. Chia số nguyên không dấu

Số bị chia	$\begin{array}{r} 10010011 \\ - \underline{1011} \\ \hline 00001101 \end{array}$	Số chia
	001110	Thương
	$\begin{array}{r} - \underline{1011} \\ \hline 001111 \end{array}$	
	$\begin{array}{r} - \underline{1011} \\ \hline 100 \end{array}$	
		Phần dư

Phần dù

174

Bộ chia số nguyên không dấu

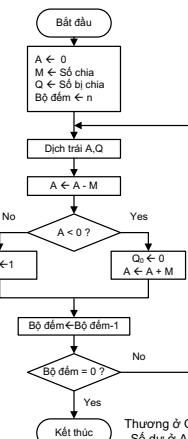


2017

Kiến trúc máy tính

175

Lưu đồ chia số nguyên không dấu



201

Kiến trúc máy tính

170

4. Chia số nguyên có dấu

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

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

2017

Kiến trúc máy tính

177

4.4. Số dấu phẩy động

1. Nguyên tắc chung

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

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

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

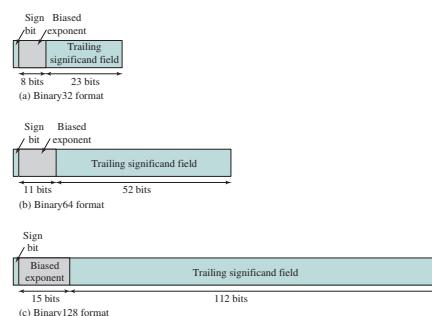
201

Kiến trúc máy tính

178

2. Chuẩn IEEE754-2008

- Cơ số R = 2
 - Các dạng:

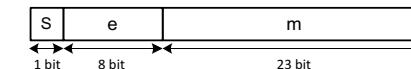


2017

Kiến trúc máy tính

179

Dang 32-bit



- **S** là bit dấu:
 - $S = 0 \rightarrow$ số dương
 - $S = 1 \rightarrow$ số âm
 - **e** (8 bit) là giá trị dịch chuyển của phần mũ E:
 - $e = E + 127 \rightarrow$ phần mũ $E = e - 127$
 - **m** (23 bit) là phần lẻ của phần định trị M:
 - $M = 1.m$

$$X = (-1)^{S_1} \cdot m_1 \cdot 2^{e-127}$$

201

Kiến trúc máy tính

180

Ví dụ 1

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

- **1100 0001 0101 0110 0000 0000 0000 0000**
 - S = 1 → số âm
 - e = $1000\ 0010_{(2)} = 130_{(10)}$ → E = $130 - 127 = 3$
- Vậy
 $X = -1.10101100_{(2)} \times 2^3 = -1101.011_{(2)} = -13.375_{(10)}$
- **0011 1111 1000 0000 0000 0000 0000 0000 = ?**

2017

Kiến trúc máy tính

181

Ví dụ 2

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

- Giải:
- $X = 83.75_{(10)} = 1010011.11_{(2)} = 1.0100111 \times 2^6$
 - Ta có:
 - S = 0 vì đây là số dương
 - E = e - 127 = 6 → e = 127 + 6 = $133_{(10)} = 1000\ 0101_{(2)}$
 - Vậy:
 $X = 0100\ 0010\ 1010\ 0111\ 1000\ 0000\ 0000$

2017

Kiến trúc máy tính

182

Các qui ước đặc biệt

- Các bit của e bằng 0, các bit của m bằng 0, thì $X = \pm 0$
 $x000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000 \rightarrow X = \pm 0$
- Các bit của e bằng 1, các bit của m bằng 0, thì $X = \pm \infty$
 $x111\ 1111\ 1000\ 0000\ 0000\ 0000\ 0000 \rightarrow X = \pm \infty$
- 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)

2017

Kiến trúc máy tính

183

Dải giá trị biểu diễn

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



2017

Kiến trúc máy tính

184

Dạng 64-bit

- S là bit dấu
- e (11 bit) là giá trị dịch chuyển của phần mũ E:
 - $e = E + 1023 \rightarrow$ phần mũ 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}

2017

Kiến trúc máy tính

185

Dạng 128-bit

- S là bit dấu
- e (15 bit) là giá trị dịch chuyển của phần mũ E:
 - $e = E + 16383 \rightarrow$ phần mũ 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}

2017

Kiến trúc máy tính

186

3. Thực hiện phép toán số dấu phẩy động

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

2017

Kiến trúc máy tính

187

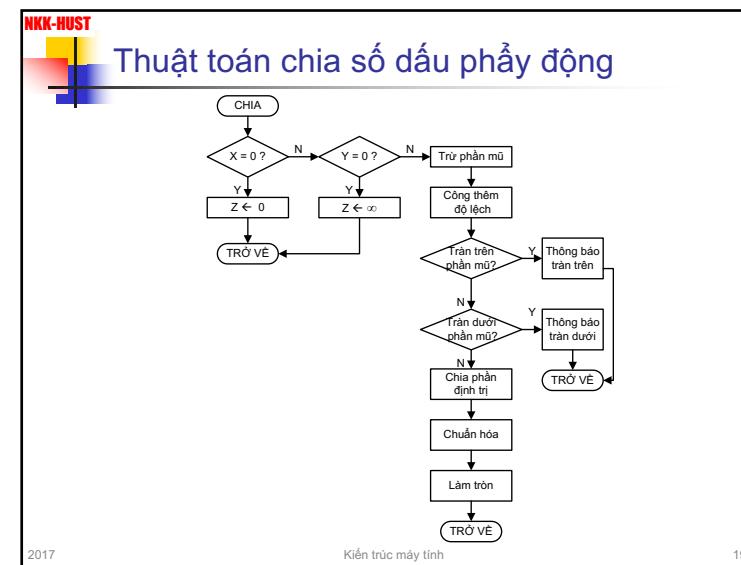
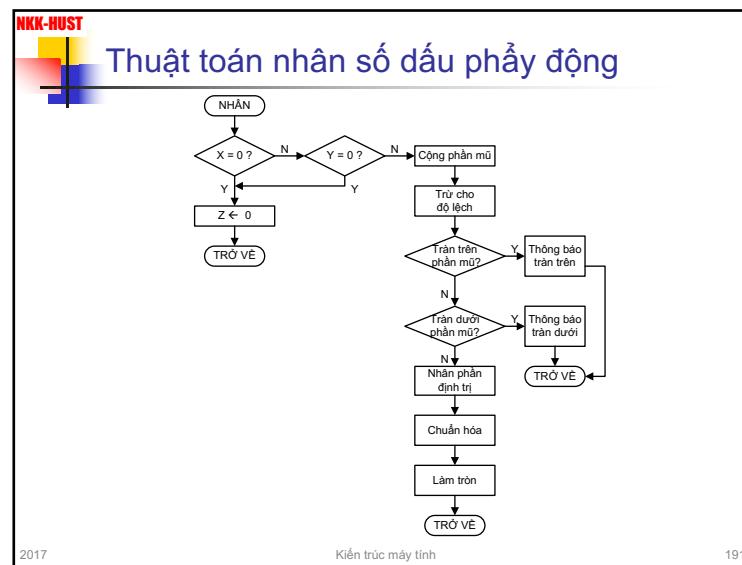
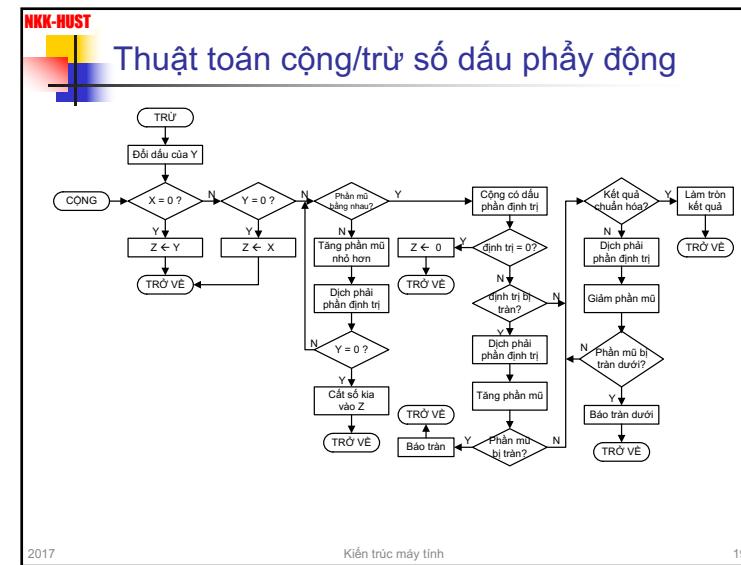
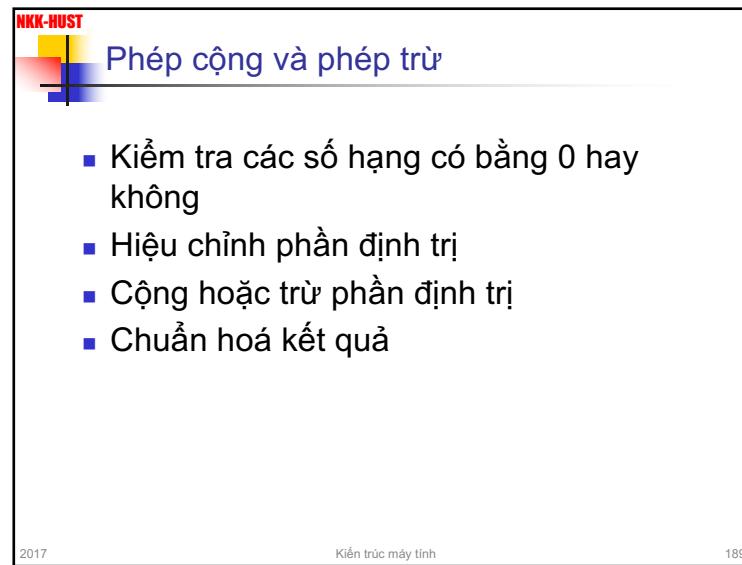
Các khả năng tràn số

- 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 nhó 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

2017

Kiến trúc máy tính

188



NKK-HUST

Hết chương 4

2017

Kiến trúc máy tính

193

NKK-HUST



Kiến trúc máy tính

Chương 5

KIẾN TRÚC TẬP LỆNH

Nguyễn Kim Khánh
Trường Đại học Bách khoa Hà Nội

2017

Kiến trúc máy tính

1



Nội dung học phần

NKK-HUST

Nội dung của chương 5

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

2017

Kiến trúc máy tính

1

5.1. Giới thiệu chung về kiến trúc tập lệnh

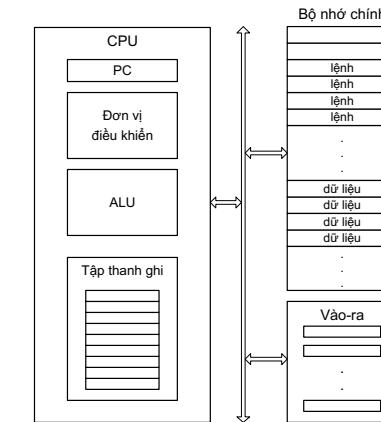
- **Kiến trúc tập lệnh** (Instruction Set Architecture): cách nhìn máy tính bởi người lập trình
 - **Vị kiến trúc** (Microarchitecture): cách thực hiện kiến trúc tập lệnh bằng phần cứng
 - Ngôn ngữ trong máy tính:
 - **Hợp ngữ (assembly language)**:
 - dạng lệnh có thể đọc được bởi con người
 - biểu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

201

Kiến trúc máy tính

19

Mô hình lập trình của máy tính

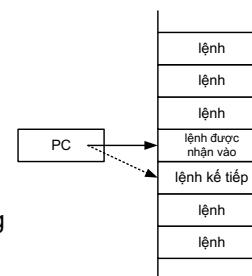


Kiến trúc máy tính

19

CPU nhận lệnh từ bộ nhớ

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



201

Kiến trúc máy tính

19

Giải mã và thực hiện lệnh

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

201

Kiến trúc máy tính

201

CPU đọc/ghi dữ liệu bộ nhớ

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

2017

Kiến trúc máy tính

201

Hàng số địa chỉ

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

2017

Kiến trúc máy tính

202

Sử dụng thanh ghi con trỏ

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

2017

Kiến trúc máy tính

203

Sử dụng địa chỉ cơ sở và dịch chuyển

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

2017

Kiến trúc máy tính

204

NKK-HUST

Ngăn xếp (Stack)

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

2017

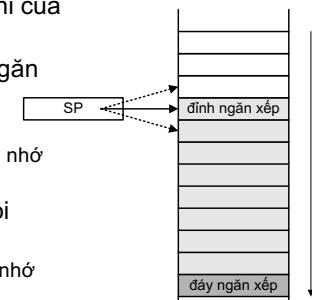
Kiến trúc máy tính

20

NKK-HUST

Con trỏ ngăn xếp SP (Stack Pointer)

- SP là thanh ghi chứa địa chỉ của ngăn nhớ đinh ngăn xếp
- Khi cất một thông tin vào ngăn xếp:
 - Giảm nội dung của SP
 - 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
 - Tăng nội dung của SP
- Khi ngăn xếp rỗng, SP trở vào đáy



chiều
địa
chi
tăng
dần

NKK-HUST

Thứ tự lưu trữ các byte trong bộ nhớ chính

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

NKK-HUST

Ví dụ lưu trữ dữ liệu 32-bit

Số nhị phân

0001	1010	0010	1011	0011	1100	0100	1101
------	------	------	------	------	------	------	------

Số Hexa

1A	2B	3C	4D
----	----	----	----

4D	4000
3C	4001
2B	4002
1A	4003

little-endian

1A	4000
2B	4001
3C	4002
4D	4003

big-endian

2017

Kiến trúc máy tính

20

NKK-HUST

Tập lệnh

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

2017

Kiến trúc máy tính

209

NKK-HUST

Dạng lệnh hợp ngữ

- Mã C:
 $a = b + c;$
- Ví dụ lệnh hợp ngữ:
add a, b, c # $a = b + c$
- trong đó:
 - **add**: ký hiệu gọi nhớ chỉ ra thao tác (phép toán) cần thực hiện.
 - Chú ý: mỗi lệnh chỉ thực hiện một thao tác
 - **b, c**: các toán hạng nguồn cho thao tác
 - **a**: toán hạng đích (nơi ghi kết quả)
 - phần sau dấu **#** là lời giải thích (chỉ có tác dụng đến hết dòng)

Mã thao tác	Địa chỉ toán hạng
<ul style="list-style-type: none">■ Mã thao tác (operation code hay opcode): mã hóa cho thao tác mà bộ xử lý phải thực hiện<ul style="list-style-type: none">■ Các thao tác chuyển dữ liệu■ Các phép toán số học■ Các phép toán logic■ Các thao tác chuyển điều khiển (rẽ nhánh, nhảy)■ Địa chỉ toán hạng: chỉ ra nơi chứa các toán hạng mà thao tác sẽ tác động<ul style="list-style-type: none">■ Toán hạng có thể là:<ul style="list-style-type: none">■ Hằng số nằm ngay trong lệnh■ Nội dung của thanh ghi■ Nội dung của ngăn nhớ (hoặc cổng vào-ra)	



Số lượng địa chỉ toán hạng trong lệnh

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

Các kiến trúc tập lệnh CISC và RISC

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

2017

Kiến trúc máy tính

213

Các đặc trưng của kiến trúc RISC

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

2017

Kiến trúc máy tính

214

Kiến trúc tập lệnh MIPS

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

2017

Kiến trúc máy tính

215

5.2. Lệnh hợp ngữ và các toán hạng

- Thực hiện phép cộng: 3 toán hạng
 - Là phép toán phổ biến nhất
 - Hai toán hạng nguồn và một toán hạng đích
$$\text{add } a, b, c \ # a = b + c$$
- Hầu hết các lệnh số học/logic có dạng trên
- Các lệnh số học sử dụng toán hạng thanh ghi hoặc hằng số

2017

Kiến trúc máy tính

216

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”

2017

Kiến trúc máy tính

217

Tập thanh ghi của MIPS

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

2017

Kiến trúc máy tính

218

Toán hạng thanh ghi

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


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

 - giả thiết: f, g, h, i, j nằm ở \$s0, \$s1, \$s2, \$s3, \$s4
- Được dịch thành mã hợp ngữ MIPS:


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

2017

Kiến trúc máy tính

219

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*)

2017

Kiến trúc máy tính

220

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 FFFF
byte	0xFFFF FFFC	word	0xFFFF FFF8
byte	0xFFFF FFFD	word	0xFFFF FFFC
byte	0xFFFF FFFE		
byte	0xFFFF FFFF		

2^{30} words

NKK-HUST

Lệnh load và lệnh store

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

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

- rs: thanh ghi chứa địa chỉ cơ sở (base address)
- imm (immediate): hằng số (offset)
→ địa chỉ của word dữ liệu cần đọc = địa chỉ cơ sở + hằng số
- rt: thanh ghi đích, chứa word dữ liệu được đọc vào

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

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

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

2017

Kiến trúc máy tính

2

NKK-HUST

Ví dụ toán hạng bộ nhớ

Mã C:

```
// A là mảng các phần tử 32-bit
g = h + A[8];


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

```

Ví dụ toán hạng bộ nhớ

- Mã C:


```
// A là mảng các phần tử 32-bit
g = h + A[8] ;
```

 - Cho g ở \$s1, h ở \$s2
 - \$s3 chứa địa chỉ cơ sở của mảng A
- Mã hợp ngữ MIPS:


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

A[0]
A[1]
A[2]
A[3]
A[4]
A[5]
A[6]
A[7]
A[8]
A[9]
A[10]
A[11]
A[12]

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

NKK-HUST

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

A[0]
A[1]
A[2]
A[3]
A[4]
A[5]
A[6]
A[7]
A[8]
A[9]
A[10]
A[11]
A[12]

Ví dụ toán hạng bộ nhớ (tiếp)

- Mã C:
$$A[12] = h + A[8];$$

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

- Mã hợp ngữ MIPS:

```

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

The diagram illustrates the memory layout for array A. On the left, an arrow points from the offset value 48 in the MIPS assembly code to the 13th element of the array. The array is represented as a vertical stack of 13 memory locations, indexed from A[0] at the top to A[12] at the bottom. The label "Địa chỉ cơ sở" (Base Address) is placed above the first element A[0], and the label "Offset = 48" is placed next to the arrow pointing to the 13th element A[12].



Thanh ghi với Bộ nhớ

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



Toán hạng tức thì (immediate)

- Dữ liệu hằng số được xác định ngay trong lệnh
 - `addi $s3, $s3, 4 # $s3 = $s3+4`
- Không có lệnh trừ (subi) với giá trị hằng số
 - Sử dụng hằng số âm trong lệnh addi để thực hiện phép trừ
 - `addi $s2, $s1, -1 # $s2 = $s1-1`

NKK-HUST

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** 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 hằng số nguyên trong hợp ngữ MIPS:
 - số thập phân: 12; 3456; -18
 - số Hexa (bắt đầu bằng **0x**): 0x12 ; 0x3456; 0x1AB6

NKK-HUST

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`

2017

Kiến trúc máy tính

2

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

NKK-HUST		Các kiểu lệnh máy của MIPS												
Lệnh kiểu R	<table border="1"> <tr> <td>op</td> <td>rs</td> <td>rt</td> <td>rd</td> <td>shamt</td> <td>funct</td> </tr> <tr> <td>6 bits</td> <td>5 bits</td> <td>5 bits</td> <td>5 bits</td> <td>5 bits</td> <td>6 bits</td> </tr> </table>	op	rs	rt	rd	shamt	funct	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	
op	rs	rt	rd	shamt	funct									
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits									
Lệnh kiểu I	<table border="1"> <tr> <td>op</td> <td>rs</td> <td>rt</td> <td>imm</td> </tr> <tr> <td>6 bits</td> <td>5 bits</td> <td>5 bits</td> <td>16 bits</td> </tr> </table>	op	rs	rt	imm	6 bits	5 bits	5 bits	16 bits					
op	rs	rt	imm											
6 bits	5 bits	5 bits	16 bits											
Lệnh kiểu J	<table border="1"> <tr> <td>op</td> <td>address</td> </tr> <tr> <td>6 bits</td> <td>26 bits</td> </tr> </table>	op	address	6 bits	26 bits									
op	address													
6 bits	26 bits													

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

Lệnh kiểu R (Registers)

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

NKK-HUST

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)

2017

Kiến trúc máy tính

2

Lệnh kiểu I (Immediate)

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

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

```

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

```

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

NKK-HUST

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 =	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>0</td><td>0000</td><td>0000</td><td>0000</td><td>0101</td></tr></table>	0	0000	0000	0000	0101	16-bit			
0	0000	0000	0000	0101						
+5 =	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>0</td><td>0000</td><td>0000</td><td>0000</td><td>0000</td><td>0000</td><td>0000</td><td>0101</td></tr></table>	0	0000	0000	0000	0000	0000	0000	0101	32-bit
0	0000	0000	0000	0000	0000	0000	0101			
-12 =	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>1</td><td>1111</td><td>1111</td><td>1111</td><td>0100</td></tr></table>	1	1111	1111	1111	0100	16-bit			
1	1111	1111	1111	0100						
-12 =	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>1</td><td>1111</td><td>1111</td><td>1111</td><td>1111</td><td>1111</td><td>1111</td><td>0100</td></tr></table>	1	1111	1111	1111	1111	1111	1111	0100	32-bit
1	1111	1111	1111	1111	1111	1111	0100			

2017

Kiến trúc máy tính

2

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)

2017

Kiến trúc máy tính

237

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)

23

Lệnh kiểu J (Jump)

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

op	address
6 bits	26 bits

2017

Kiến trúc máy tính

239

5.4. Cơ bản về lập trình hợp ngữ

1. Các lệnh logic
 2. Nạp hàng 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

201

Kiến trúc máy tính

24

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	0011	0100	
ori \$s3,\$s1,0xFA34	\$s3	0000	0000	0000	0000	1111	1010	1111	1111
xori \$s4,\$s1,0xFA34	\$s4	0000	0000	0000	0000	1111	1010	1100	1011



Ý nghĩa của các phép toán logic

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

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 diền 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 diền các bit 0 vào bên trái
 - Dịch phải i bits là chia cho 2^i (chỉ với số nguyên không dấu)

NKK-HUST

Ví dụ lệnh dịch trái sll

Lệnh hợp ngữ:

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

Mã máy:

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

(0x000105100)

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

2017

Kiến trúc máy tính

2

NKK-HUST

Ví dụ lệnh dịch phải srl

Lệnh hợp ngữ:

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

Mã máy:

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

(0x00119082)

Ví dụ kết quả thực hiện lệnh:

\$s1	0000	0000	0000	0000	0000	0101	0110
------	------	------	------	------	------	------	------

= 86

\$s2	0000	0000	0000	0000	0000	0000	0001	0101
------	------	------	------	------	------	------	------	------

= 21
[86/4]

NKK-HUST

2. Nạp hằng số vào thanh ghi

- Trường hợp hằng số 16-bit → sử dụng lệnh addi:
 - Ví dụ: nạp hằng số 0x4F3C vào thanh ghi \$s0:
`addi $s0, $0, 0x4F3C # $s0 = 0x4F3C`
- Trong trường hợp hằng số 32-bit → sử dụng lệnh lui và lệnh ori:
 - **lui rt, constant_hi16bit**
 - Copy 16 bit cao của hằng số 32-bit vào 16 bit trái của rt
 - Xóa 16 bits bên phải của rt về 0
 - **ori rt,rt,constant_low16bit**
 - Đưa 16 bit thấp của hằng số 32-bit vào thanh ghi rt

2017

Kiến trúc máy tính

2

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

NKK-HUST

Ví dụ khởi tạo thanh ghi 32-bit

- Nạp vào thanh ghi \$s0 giá trị 32-bit sau:
0010 0001 1010 0000 0100 0000 0011 1011 =0x21A0 403B

```

lui $s0,0x21A0      # nạp 0x21A0 vào nửa cao
                     # của thanh ghi $s0

ori $s0,$s0,0x403B  # nạp 0x403B vào nửa thấp
                     # của thanh ghi $s0

```

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

\$s0	0010	0001	1010	0000	0000	0000	0000	0000
	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
------	------	------	------	------	------	------	------	------

NKK-HUST



3. Tạo các cấu trúc điều khiển

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

NKK-HUST

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

2017

Kiến trúc máy tính

2

NKK-HUST

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

```

graph TD
    Start(( )) --> Decision{i == j?}
    Decision -- Yes --> Add[f = g + h]
    Add --> Subtract[f = f - i]
    Subtract --> End(( ))
    Decision -- No --> End
  
```

NKK-HUST

Dịch câu lệnh if

- Mã C:

```

if (i==j)
    f = g+h;
    f = f-i;
■ f, g, h, i, j ở $s0, $s1, $s2, $s3, $s4

```

- Mã hợp ngữ MIPS:

```

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

```

```

graph TD
    A{i == j ?} -- Yes --> B[f = g + h]
    B --> C[f = f - i]
    C --> D(( ))
    D --> E(( ))
    E -- No --> F(( ))

```

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

2017

Kiến trúc máy tính

2

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

```

graph TD
    Start(( )) --> Cond{i == j ?}
    Cond -- i=j --> F1[f = g + h]
    Cond -- i ≠ j --> F2[f = g - h]
    F1 --> Join(( ))
    F2 --> Join
    Join --> End(( ))
  
```

2017 Kiến trúc máy tính 257

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

```

graph TD
    Start(( )) --> Cond{i == j ?}
    Cond -- i=j --> F1[f = g + h]
    Cond -- i ≠ j --> F2[f = g - h]
    F1 --> Join(( ))
    F2 --> Join
    Join --> End(( ))
  
```

2017 Kiến trúc máy tính 258

Mã hợp ngữ MIPS:

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

Dịch câu lệnh switch/case

Mã C:

```
switch (amount) {
    case 20: fee = 2; break;
    case 50: fee = 3; break;
    case 100: fee = 5; break;
    default: fee = 0;
}

// tương đương với sử dụng các câu lệnh if/else
if(amount == 20) fee = 2;
else if (amount == 50) fee = 3;
else if (amount == 100) fee = 5;
else fee = 0;
```

2017 Kiến trúc máy tính 259

Dịch câu lệnh switch/case

Mã hợp ngữ MIPS

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

2017 Kiến trúc máy tính 260

NKK-HUST

Dịch câu lệnh vòng lặp while

Mã C:

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

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

```

graph TD
    Start(( )) --> Decision{A[i] == k?}
    Decision -- No --> Exit(( ))
    Decision -- Yes --> Increment[i = i + 1]
    Increment --> Decision
  
```

Dịch câu lệnh vòng lặp while

```

graph TD
    Start(( )) --> Decision{A[i] == k?}
    Decision -- Yes --> Increment[i = i + 1]
    Increment --> Decision
    Decision -- No --> Exit(( ))
  
```

Mã C:

```

while (A[i] == k) i += 1;
  • i ở $s3, k ở $s5
  • địa chỉ cơ sở của mảng A ở $s6
  
```

Mã hợp ngữ MIPS:

```

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

Exit: ...
  
```

NKK-HUST

Dịch câu lệnh vòng lặp for

Mã C:

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

NKK-HUST

Dịch câu lệnh vòng lặp for

Mã C:

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

■ Mã hợp ngữ MIPS:

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

2017

Kiến trúc máy tính

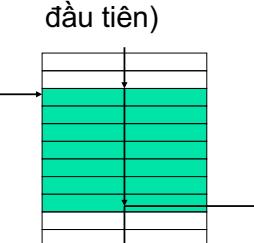
20

NKK-HUST

Khối lệnh cơ sở (basic block)

Khối lệnh cơ sở là dãy các lệnh với

- Không có lệnh rẽ nhánh nhúng trong đó (ngoại trừ ở cuối)
- Không có đích rẽ nhánh tối (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

NKK-HUST

Thêm các lệnh thao tác điều kiện

- Lệnh slt (set on less than)

slt rd, rs, rt

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

- Lệnh slti

slti rt, rs, constant

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

- Sử dụng kết hợp với các lệnh beq, bne

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

...

L1:

2017

Kiến trúc máy tính

20

NKK-HUST

So sánh số có dấu và không dấu

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

NKK-HUST

Ví dụ sử dụng lệnh slt

■ Mã C

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

2017

Kiến trúc máy tính

20

Ví dụ sử dụng lệnh slt

- Mã hợp ngữ MIPS

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

2017

Kiến trúc máy tính

269

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

2017

Kiến trúc máy tính

270

Ví dụ về mảng

- Mảng 5-phần tử, mỗi phần tử có độ dài 32-bit, chiếm 4 byte trong bộ nhớ

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

- 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

2017

Kiến trúc máy tính

271

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
```

2017

Kiến trúc máy tính

272

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
```

2017

Kiến trúc máy tính

273

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

Mã hợp ngữ MIPS

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

2017

Kiến trúc máy tính

274

5. Chương trình con - thủ tục

- Các bước yêu cầu:

- Đặt các tham số vào các thanh ghi
- Chuyển điều khiển đến thủ tục
- Thực hiện các thao tác của thủ tục
- Đặt kết quả vào thanh ghi cho chương trình đã gọi thủ tục
- Trở về vị trí đã gọi

2017

Kiến trúc máy tính

275

Sử dụng các thanh ghi

- \$a0 – \$a3: các tham số vào (các thanh ghi 4 – 7)
- \$v0, \$v1: các kết quả ra (các thanh ghi 2 và 3)
- \$t0 – \$t9: các giá trị tạm thời
 - Có thể được ghi lại bởi thủ tục được gọi
- \$s0 – \$s7: cất giữ các biến
 - Cần phải cắt/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)

2017

Kiến trúc máy tính

276

NKK-HUST

Các lệnh gọi thủ tục

- Gọi thủ tục: jump and link
 - jal ProcedureAddress**
 - Địa chỉ của lệnh kế tiếp (địa chỉ trả 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 trả về) trả lại cho bộ đếm chương trình PC

The diagram illustrates the call/return mechanism between two procedures: `main` and `proc`.

- Procedure `main`:** Represented by a large rectangle containing dashed horizontal lines. An arrow labeled `PC` points to the start of the `jal proc` instruction.
- Procedure `proc`:** Represented by a smaller rectangle containing dashed horizontal lines. It contains the `jr $ra` instruction at its end.
- Call Phase:** Indicated by a bracket labeled **Prepare to call**. This phase shows the state before the call, with the `jal proc` instruction being executed.
- Return Phase:** Indicated by a bracket labeled **Prepare to continue**. This phase shows the state after the call, where the control has transferred to the `proc` procedure.
- Save Phase:** Indicated by a bracket labeled **Save, etc.**. This phase shows the state during the return, where the `proc` procedure is executing its code.
- Restore Phase:** Indicated by a bracket labeled **Restore**. This phase shows the state after the return, where the control has returned to the `main` procedure.

The diagram illustrates the call mechanism between three procedures: `main`, `Procedure abc`, and `Procedure xyz`.

- main:** Contains the instruction `jal abc`. The stack frame for `main` is shown with the PC pointing to the `jal` instruction.
- Procedure abc:** Contains the instruction `jal xyz`. The stack frame for `abc` is shown with the label `Procedure abc` above it.
- Procedure xyz:** Contains the instruction `jr $ra`. The stack frame for `xyz` is shown with the label `Procedure xyz` above it.

The flow of control is as follows:

- Prepare to call:** The `jal abc` instruction in `main` pushes the return address (`$ra`) onto the stack.
- Call:** The `jal abc` instruction in `main` branches to the start of `Procedure abc`.
- Prepare to continue:** The `jal xyz` instruction in `Procedure abc` pushes the return address (`$ra`) onto the stack.
- Call:** The `jal xyz` instruction in `Procedure abc` branches to the start of `Procedure xyz`.
- Save:** The `jr $ra` instruction in `Procedure xyz` restores the previous return address (`xyz`) from the stack.
- Restore:** The `jr $ra` instruction in `Procedure xyz` restores the original return address (`abc`) from the stack.
- Return:** The `jr $ra` instruction in `Procedure xyz` returns control to the `jal xyz` instruction in `Procedure abc`.

NKK-HUST

Ví dụ Thủ tục lá

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

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

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

Kiến trúc máy tính

2017

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

2017

Kiến trúc máy tính

281

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	

2017

Kiến trúc máy tính

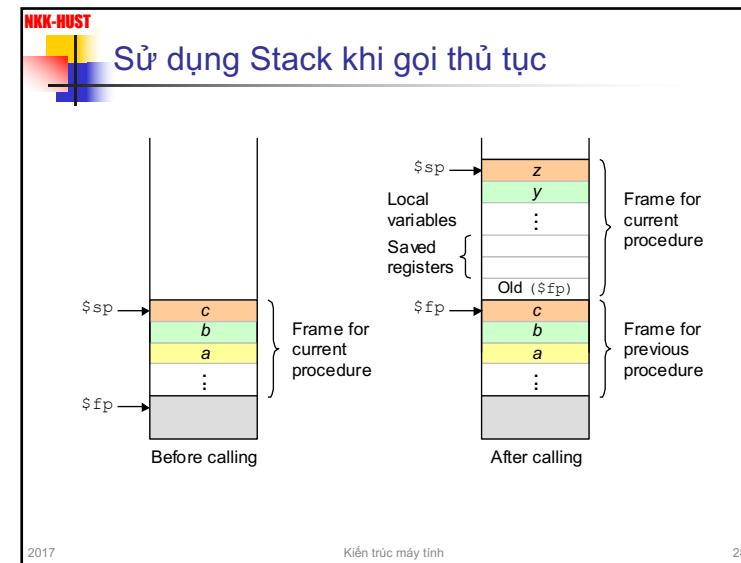
282

Mã hợp ngữ 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 quy
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

2017

Kiến trúc máy tính

283



2017

Kiến trúc máy tính

284

6. Dữ liệu ký tự

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

2017

Kiến trúc máy tí

285

Các thao tác với Byte/Halfword

- Có thể sử dụng các phép toán logic
 - Nạp/Lưu byte/halfword trong MIPS
 - **lb rt, offset(rs) lh rt, offset(rs)**
 - Mở rộng theo số có dấu thành 32 bits trong rt
 - **lbu rt, offset(rs) lhu rt, offset(rs)**
 - Mở rộng theo số không dấu thành 32 bits trong rt
 - **sb rt, offset(rs) sh rt, offset(rs)**
 - Chỉ lưu byte/halfword bên phải

201

Kiến trúc máy tính

28

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

2017

Kiến trúc máy tí

287

Ví dụ Copy String

- ## ■ Mã hợp ngữ MIPS

```

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

```

201

Kiến trúc máy tính

28

7. Các lệnh nhân và chia số nguyên

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

201

Kiến trúc máy tính

289

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
 - `lwcl, ldc1, swcl, sdc1`
 - Ví dụ: `ldc1 $f8, 32($s2)`

201

Kiến trúc máy tính

29

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`

201

Kiến trúc máy tính

29

5.5. Các phương pháp định địa chỉ của MIPS

- Các lệnh **Branch** chỉ ra:
 - Mã thao tác, hai thanh ghi, hằng số
 - Hầu hết các đích rẽ nhánh là rẽ nhánh gần
 - Rẽ xuôi hoặc rẽ ngược
 - Định địa chỉ tương đối với PC
 - PC-relative addressing
 - **Địa chỉ đích = PC + hằng số imm × 4**
 - Chú ý: trước đó PC đã được tăng lên
 - Hằng số imm 16-bit có giá trị trong dải $[-2^{15}, +2^{15} - 1]$

201

Kiến trúc máy tính

29

NKK-HUST

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

2017

Kiến trúc máy tính

29

NKK-HUST

Địa chỉ hóa cho lệnh Jump

- Đích của lệnh **Jump (j và jal)** có thể là bất kỳ chỗ nào trong chương trình
 - Cần mã hóa đầy đủ địa chỉ trong lệnh
- **Định địa chỉ nhảy (giả) trực tiếp (Pseudo)Direct jump addressing**
 - Địa chỉ đích = $PC_{31\dots 28} : (address \times 4)$

op	address
6 bits	26 bits

NKK-HUST

Ví dụ mã hóa lệnh

```

Loop: sll $t1, $s3, 2      0x8000
      add $t1, $t1, $s6    0x8004
      lw   $t0, 0($t1)      0x8008
      bne $t0, $s5, Exit    0x800C
      addi $s3, $s3, 1       0x8010
      j    Loop                 0x8014
Exit: ...

```

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	
0x8018					

2017

Kiến trúc máy tính

2

NKK-HUST

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:**

NKK-HUST

Tóm tắt về các phương pháp định địa chỉ

1. Định địa chỉ tức thi

op	rs	rt	Immediate
----	----	----	-----------
2. Định địa chỉ thanh ghi

op	rs	rt	rd	...	funct
----	----	----	----	-----	-------
3. Định địa chỉ cơ sở

op	rs	rt	Address
----	----	----	---------
4. PC-relative addressing

op	rs	rt	Address
----	----	----	---------
5. Định địa chỉ giả trực tiếp

op	Address
----	---------



NKK-HUST

5.6. Dịch và chạy chương trình hợp ngữ

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

```

graph TD
    HLC[High Level Code] --> C[Compiler]
    C --> AC[Assembly Code]
    AC --> A[Assembler]
    A --> OF[Object File]
    OF --> L[Linker]
    L --> E[Executable]
    E --> Ld[Loader]
    Ld --> M[Memory]
    OF --> L
    Lf[Object Files  
Library Files] --> L
  
```

The diagram illustrates the sequential steps in the application development process:

- High Level Code is processed by the Compiler.
- The Compiler generates Assembly Code.
- The Assembly Code is processed by the Assembler.
- The Assembler generates an Object File.
- The Object File is processed by the Linker.
- The Linker generates an Executable.
- The Executable is processed by the Loader.
- The Loader stores the application in the Memory.

Object Files and Library Files are input to the Linker step.

Chương trình trong bộ nhớ

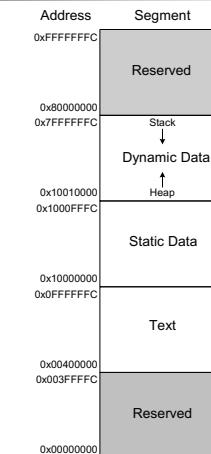
- Các lệnh (instructions)
 - Dữ liệu
 - Toàn cục/tĩnh: được cấp phát trước khi chương trình bắt đầu thực hiện
 - Động: được cấp phát trong khi chương trình thực hiện
 - Bộ nhớ:
 - 2^{32} bytes = 4 GiB
 - Địa chỉ từ 0x00000000 đến 0xFFFFFFFF

2017

Kiến trúc máy tính

301

Bản đồ bộ nhớ của MIPS



201

30

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);  
}
```

2017

Kiến trúc máy tính

303

Ví dụ chương trình hợp ngữ MIPS

```

.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

```

201

Kiến trúc máy tính

Bảng ký hiệu

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

2017

Kiến trúc máy tính

305

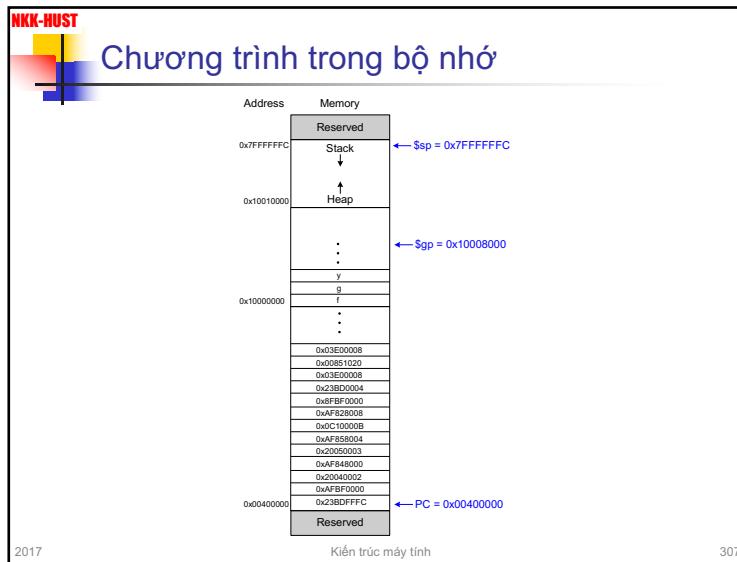
Chương trình thực thi

Executable file header	Text Size	Data Size
	0x34 (52 bytes)	0xC (12 bytes)
Text segment	Address	Instruction
	0x00400000	0x23BDFFFC
	0x00400004	sw \$ra, 0(\$sp)
	0x00400008	addi \$a0, \$0, 2
	0x0040000C	sw \$a0, 0x8000(\$gp)
	0x00400010	addi \$a1, \$0, 3
	0x00400014	sw \$a1, 0x8004(\$sp)
	0x00400018	jal 0x0040002C
	0x0040001C	sw \$v0, 0x8008(\$gp)
	0x00400020	lw \$ra, 0(\$sp)
	0x00400024	addi \$sp, \$sp, -4
	0x00400028	jr \$ra
	0x0040002C	add \$v0, \$a0, \$a1
	0x00400030	jr \$ra
Data segment	Address	Data
	0x10000000	f
	0x10000004	g
	0x10000008	y

2017

Kiến trúc máy tính

306



2017

Kiến trúc máy tính

307

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

2017

Kiến trúc máy tính

308

NKK-HUST

Hết chương 5

2017

Kiến trúc máy tính

309

NKK-HUST



Kiến trúc máy tính

Chương 6

BỘ XỬ LÝ

Nguyễn Kim Khánh
Trường Đại học Bách khoa Hà Nội

2017

Kiến trúc máy tính

31

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

NKK-HUST

Nội dung của chương 6

- 6.1. Tổ chức của CPU
- 6.2. Thiết kế đơn vị điều khiển
- 6.3. Kỹ thuật đường ống lệnh
- 6.4. Ví dụ thiết kế bộ xử lý theo kiến trúc MIPS (*)

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

2017

Kiến trúc máy tính

31

6.1. Tổ chức của CPU

1. Cấu trúc cơ bản của CPU

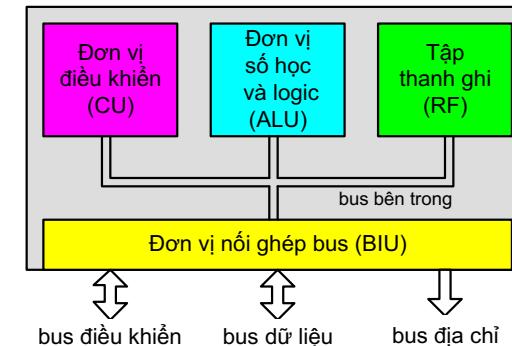
- Nhiệm vụ của CPU:
 - Nhận lệnh (Fetch Instruction): CPU đọc lệnh từ bộ nhớ
 - Giải mã lệnh (Decode Instruction): xác định thao tác mà lệnh yêu cầu
 - Nhận dữ liệu (Fetch Data): nhận dữ liệu từ bộ nhớ hoặc các cổng vào-ra
 - Xử lý dữ liệu (Process Data): thực hiện phép toán số học hay phép toán logic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

2017

Kiến trúc máy tính

31

Sơ đồ cấu trúc cơ bản của CPU



201

Kiến trúc máy tính

314

2. Đơn vị số học và logic

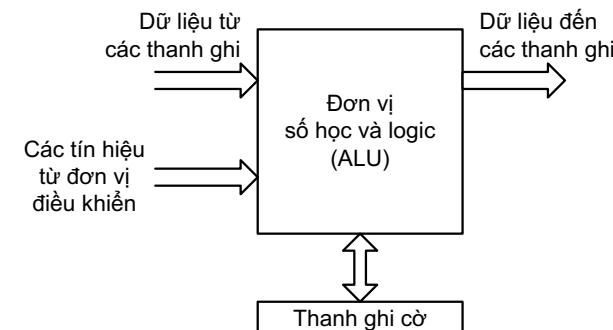
- **Chức năng:** Thực hiện các phép toán số học và phép toán logic:
 - Số học: cộng, trừ, nhân, chia, đảo dấu
 - Logic: AND, OR, XOR, NOT, phép dịch bit

2017

Kiến trúc máy tính

31

Mô hình kết nối ALU



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

201

Kiến trúc máy tính

316

3. Đơn vị điều khiển

■ Chức năng

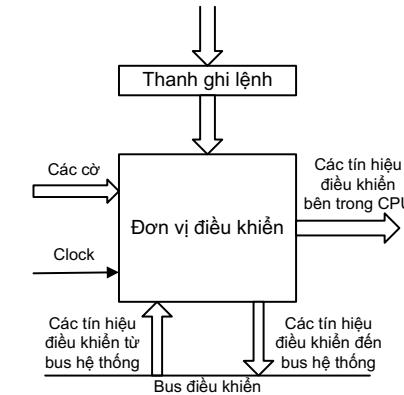
- Điều khiển nhận lệnh từ bộ nhớ đưa vào CPU
 - Tăng nội dung của PC để trả sang lệnh kế tiếp
 - Giải mã lệnh đã được nhận để xác định thao tác mà lệnh yêu cầu
 - Phát ra các tín hiệu điều khiển thực hiện lệnh
 - Nhận các tín hiệu yêu cầu từ bus hệ thống và đáp ứng với các yêu cầu đó.

2017

Kiến trúc máy tín

317

Mô hình kết nối đơn vị điều khiển



201

Kiến trúc máy tính

318

Các tín hiệu đưa đến đơn vị điều khiển

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

2017

Kiến trúc máy tín

319

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

201

Kiến trúc máy tính

320

4. Hoạt động của chu trình lệnh

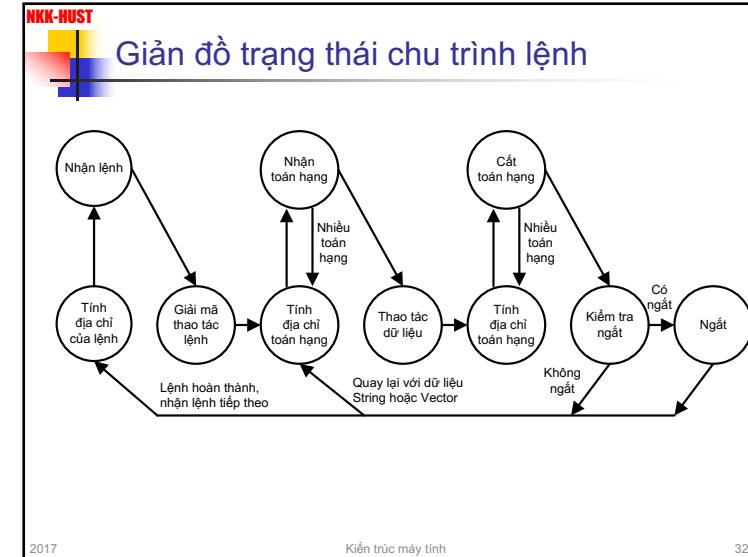
Chu trình lệnh

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

2017

Kiến trúc máy tính

32



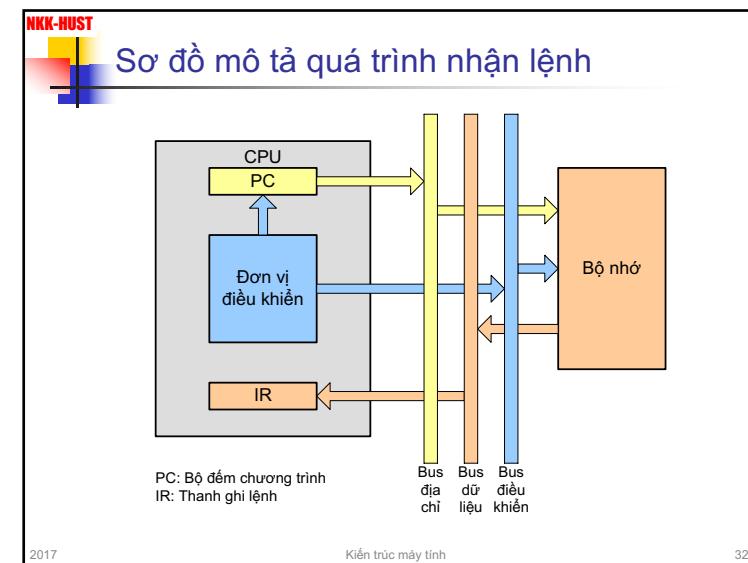
Nhận lệnh

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

2017

Kiến trúc máy tính

32



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

2017

Kiến trúc máy tính

325

Nhân dũ liêu từ bô nhớ

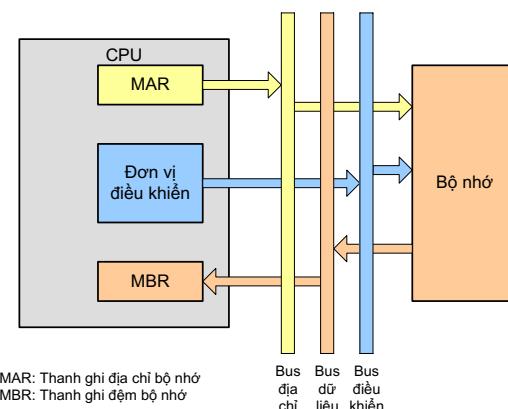
- CPU đưa địa chỉ của toán hạng ra bus địa chỉ
 - CPU phát tín hiệu điều khiển đọc
 - Toán hạng được đọc vào CPU
 - Tương tự như nhận lệnh

201

Kiến trúc máy tính

326

Sơ đồ mô tả nhân dữ liệu từ bộ nhớ



2017

Kiến trúc máy tính

327

Thực hiện lệnh

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

201

Kiến trúc máy tính

328

NKK-HUST

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 dữ bộ nhớ

Bus địa chỉ
Bus dữ liệu
Bus điều khiển

Ngắt

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

The diagram illustrates a microprogrammed control unit architecture. It features a central CPU block containing the SP (Stack Pointer), PC (Program Counter), and MBR (Memory Buffer Register). The PC feeds into the MAR (Memory Address Register) and the control unit's address bus. The MBR is connected to the data bus. The control unit consists of a microprogrammable controller (Đơn vị điều khiển) and a microinstruction memory (MBR). The controller outputs control signals to the data bus, address bus, and the microinstruction memory. The MBR outputs data to the data bus. The data bus connects to a vertical stack of three buses: 地址 (Address), 数据 (Data), and 控制 (Control). These three buses connect to a large orange block labeled "Bộ nhớ" (Memory).

MAR: Thanh ghi địa chỉ bộ nhớ

MBR: Thanh ghi đệm bộ nhớ

PC: Bộ đếm chương trình

SP: Con trỏ ngăn xếp

Bus

- Bus địa chỉ
- Bus dữ liệu
- Bus điều khiển

Bộ nhớ

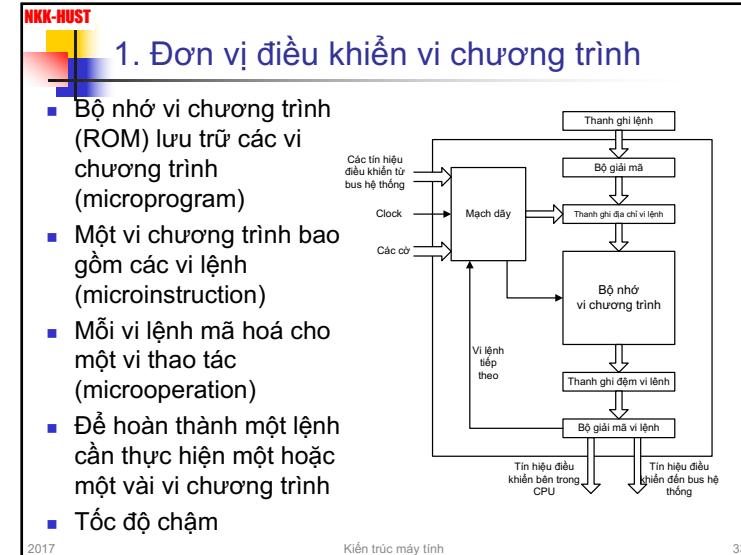
6.2. Các phương pháp thiết kế đơn vị điều khiển

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

2017

Kiến trúc máy tính

33



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

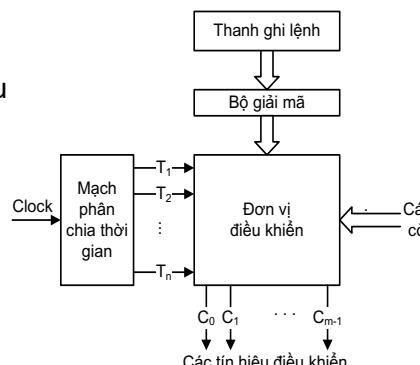
2

Kiến trúc máy tính

32

2. Đơn vị điều khiển nối kết cung

- 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



2017

Kiến trúc máy tính

33

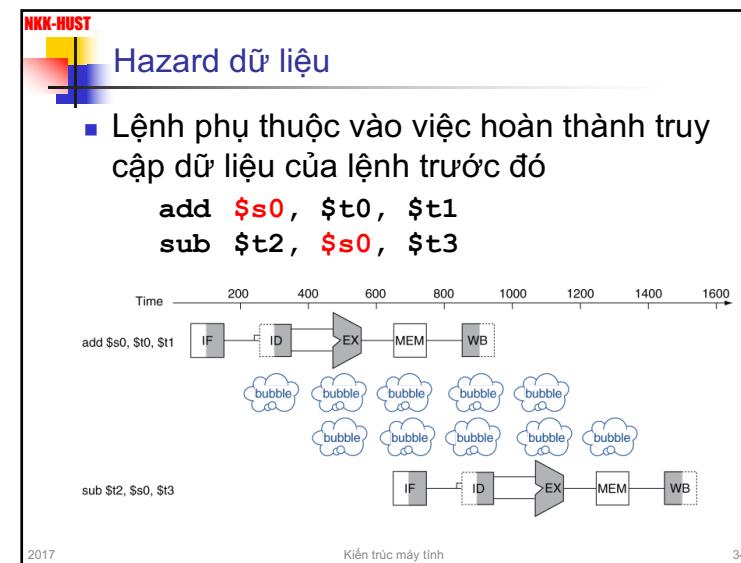
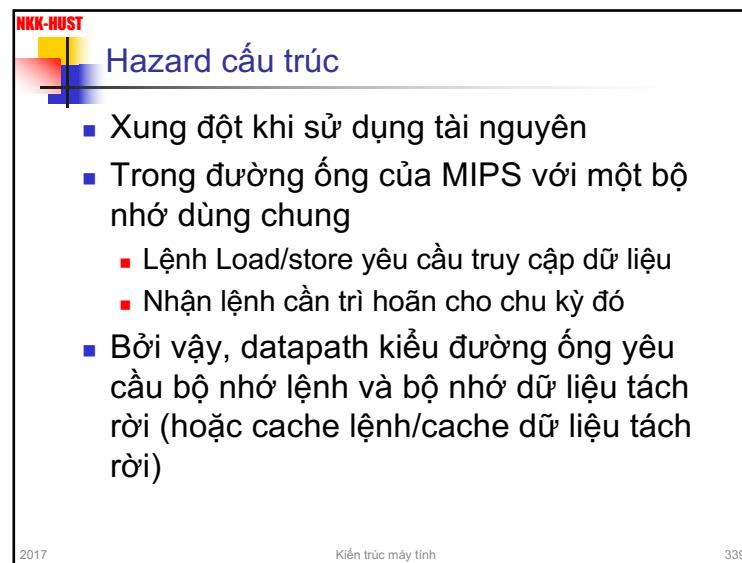
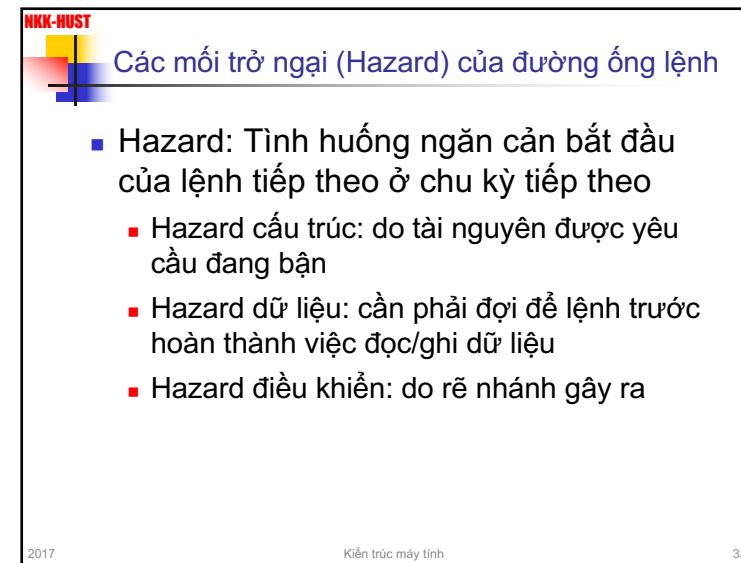
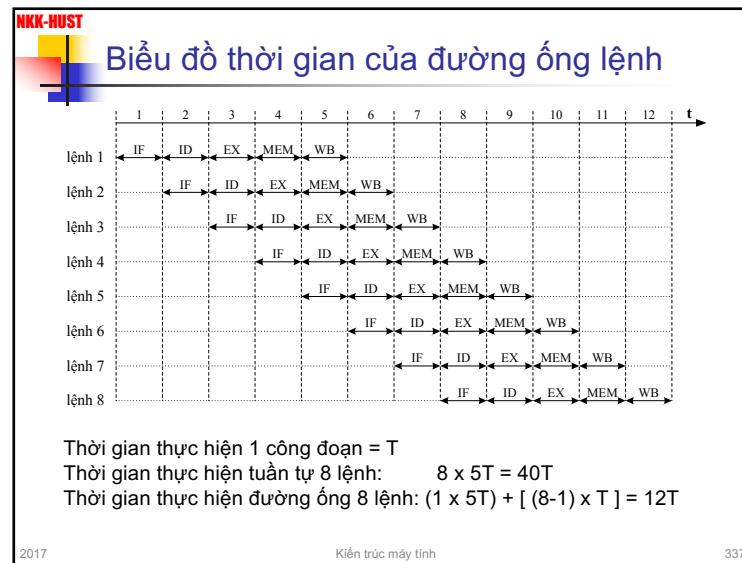
6.3. Kỹ thuật đường ống lệnh

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

2

Kiến trúc máy tính

3



Forwarding (gửi vượt trước)

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

Program execution order (in instructions): add \$s0, \$t0, \$t1; sub \$t2, \$s0, \$t3

Time: 200, 400, 600, 800, 1000

Kiến trúc máy tính

341

Hazard dữ liệu với lệnh load

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

Program execution order (in instructions): lw \$s0, 20(\$t1); sub \$t2, \$s0, \$t3

Time: 200, 400, 600, 800, 1000, 1200, 1400

Kiến trúc máy tính

342

Lập lịch mã để tránh trì hoãn

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

$$\begin{aligned} a &= b + e; \\ c &= b + f; \end{aligned}$$

stall → lw \$t1, 0(\$t0)
stall → lw \$t2, 4(\$t0)
add \$t3, \$t1, \$t2
sw \$t3, 12(\$t0)
lw \$t4, 8(\$t0)
add \$t5, \$t1, \$t4
sw \$t5, 16(\$t0)

13 cycles

11 cycles

Kiến trúc máy tính

343

Hazard điều khiển

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

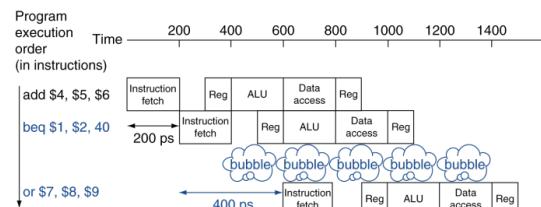
2017

Kiến trúc máy tính

344

Trì hoãn khi rẽ nhánh

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



2017

Kiến trúc máy tín

345

Dự đoán rẽ nhánh

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

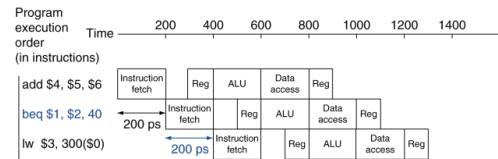
2017

Kiến trúc máy tính

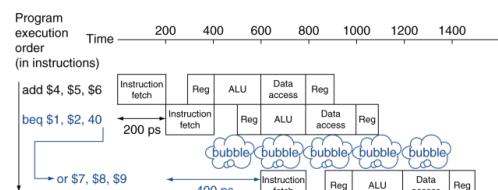
346

MIPS với dự đoán rẽ nhánh không xảy ra

Prediction correct



Prediction



2017

Kiến trúc máy tín

347

Đặc điểm của đường ống

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

2017

Kiến trúc máy tính

348

The diagram illustrates the execution of six instructions (Lệnh 1 to Lệnh 6) on two different processor architectures. The columns represent time steps, and the rows represent instructions.

Processor Architecture 1 (Left):

Lệnh	T1	T2	T3	T4	T5
Lệnh 1	Pink	Orange	Yellow	Green	Blue
Lệnh 2		Orange	Yellow	Green	Blue
Lệnh 3		Orange	Yellow	Green	Blue
Lệnh 4	Pink	Orange	Yellow	Green	Blue
Lệnh 5		Orange	Yellow	Green	Blue
Lệnh 6	Pink	Orange	Yellow	Green	Blue

Processor Architecture 2 (Right):

Lệnh	T1	T2	T3	T4	T5
Lệnh 1	Pink	Orange	Yellow	Green	Blue
Lệnh 2	Pink	Orange	Yellow	Green	Blue
Lệnh 3	Pink	Orange	Yellow	Green	Blue
Lệnh 4		Orange	Yellow	Green	Blue
Lệnh 5		Orange	Yellow	Green	Blue
Lệnh 6	Pink	Orange	Yellow	Green	Blue

Legend:

- Pink: Instruction 1
- Orange: Instruction 2
- Yellow: Instruction 3
- Green: Instruction 4
- Blue: Instruction 5

NKK-HUST

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

Dành riêng cho Chương trình Tài năng và Chất lượng cao
MIPS.pptx

NKK-HUST

Kiến trúc máy tính

Chương 7

BỘ NHỚ MÁY TÍNH

Nội dung học phần

NKK-HUST

Nội dung của chương 7

- 7.1. Tổng quan hệ thống nhớ
- 7.2. Bộ nhớ chính
- 7.3. Bộ nhớ đệm (cache)
- 7.4. Bộ nhớ ngoài
- 7.5. Bộ nhớ ảo

2017

Kiến trúc máy tính

3

NKK-HUST

7.1. Tổng quan hệ thống nhớ

1. Các đặc trưng của bộ nhớ

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



Các đặc trưng của bộ nhớ (tiếp)

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

Các đặc trưng của bộ nhớ (tiếp)

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

2017

Kiến trúc máy tín

357

Các đặc trưng của bộ nhớ (tiếp)

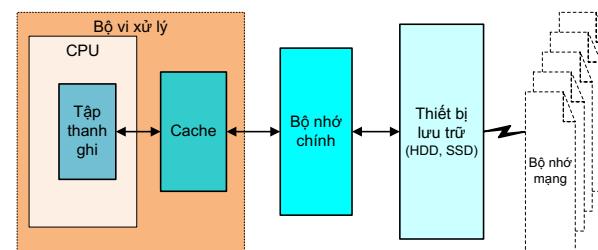
- Các đặc tính vật lý
 - Khả biến / Không khả biến (volatile / nonvolatile)
 - Xoá được / không xoá được
 - Tổ chức

2017

Kiến trúc máy tính

358

2. Phân cấp bộ nhớ



Tù trái sang phải

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

2017

Kiến trúc máy tín

359

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

2017

Kiến trúc máy tính

360

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

2017

Kiến trúc máy tính

361

7.2. Bộ nhớ chính

1. Bộ nhớ bán dẫn

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

2017

Kiến trúc máy tính

362

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
 - Ví chương trình

2017

Kiến trúc máy tính

363

Các kiểu ROM

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

2017

Kiến trúc máy tính

364

NKK-HUST

Bộ nhớ Flash

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

2017

Kiến trúc máy tính

365

NKK-HUST

RAM (Random Access Memory)

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



SRAM (Static) – RAM tĩnh

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

DRAM (Dynamic) – RAM động

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

Một số DRAM tiên tiến thông dụng

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

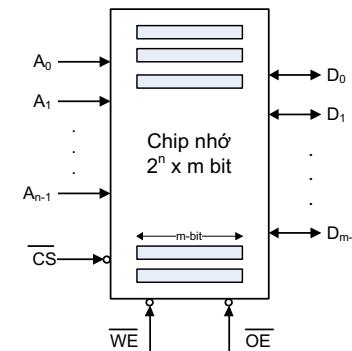
2017

Kiến trúc máy tính

369

Tổ chức của chip nhớ

Sơ đồ cơ bản của chip nhớ



Hình ảnh module nhó

201

Kiến trúc máy tính

370

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)

2017

Kiến trúc máy tính

371

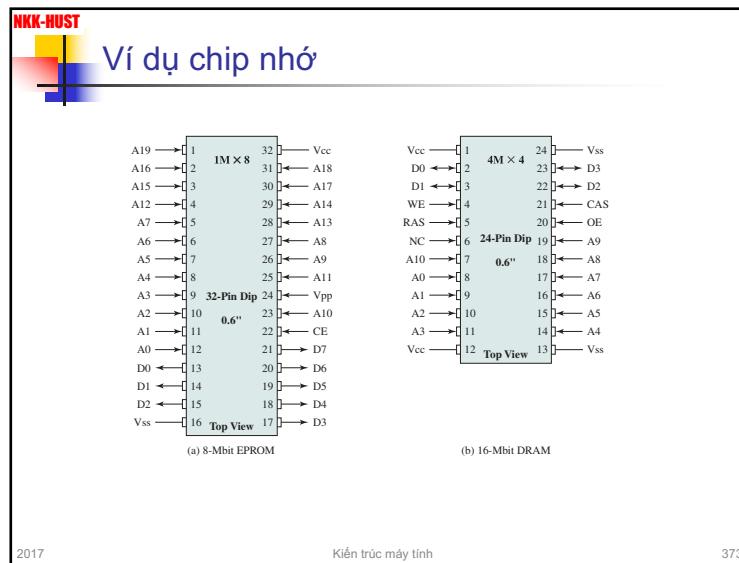
Tổ chức của DRAM

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

201

Kiến trúc máy tính

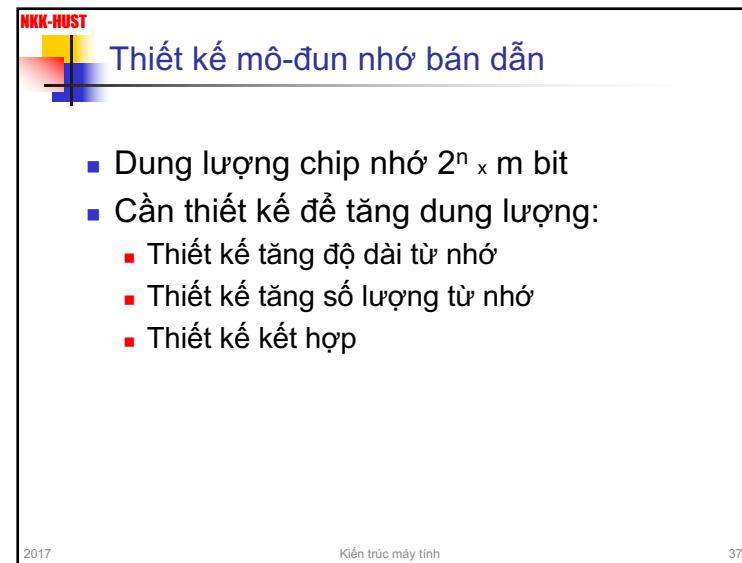
372



2017

Kiến trúc máy tính

373



2017

Kiến trúc máy tính

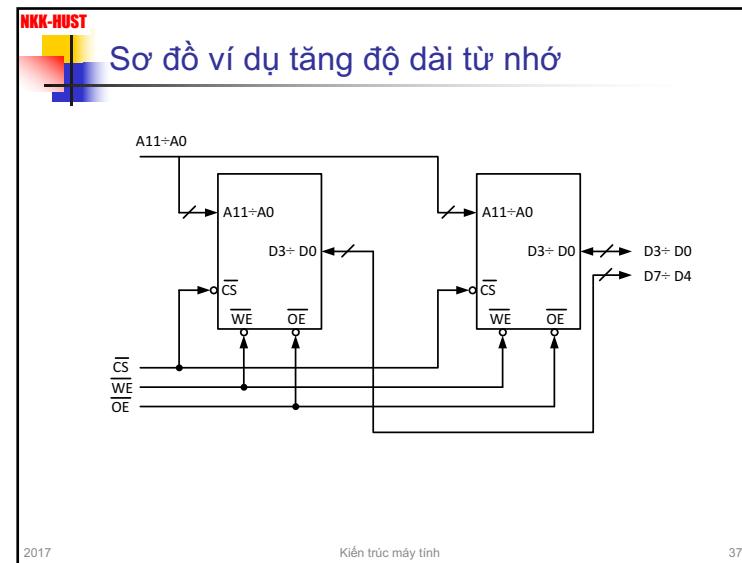
374



2017

Kiến trúc máy tính

375



2017

Kiến trúc máy tính

376

Bài toán tăng độ dài từ nhớ tổng quát

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

2017

Kiến trúc máy tín

377

Tăng số lượng từ nhớ

VD2:

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

Giải

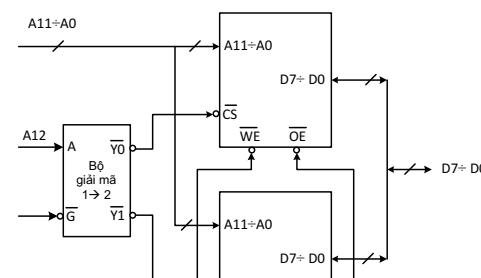
- Dung lượng chip nhớ = $2^{12} \times 8$ bit
 - chip nhớ có:
 - 12 chân địa chỉ
 - 8 chân dữ liệu
 - Dung lượng mô-đun nhớ = $2^{13} \times 8$ bit
 - 13 chân địa chỉ
 - 8 chân dữ liệu

2017

Kiến trúc máy tính

378

Sơ đồ ví dụ tăng số lượng từ nhó



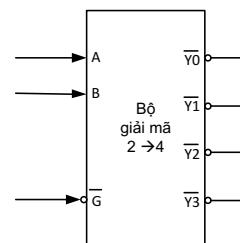
\bar{G}	A	\bar{Y}_0	\bar{Y}_1
0	0	0	1
0	1	1	0
1	x	1	1

2017

Kiến trúc máy tín

379

Bộ giải mã 2→4



\bar{G}	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

2017

Kiến trúc máy tính

380

Thiết kế kết hợp

Ví du 3:

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

Phương pháp thực hiện:

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

2017

Kiến trúc máy tính

38

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

201

Kiến trúc máy tính

382

Tổ chức bộ nhớ đan xen (interleaved memory)

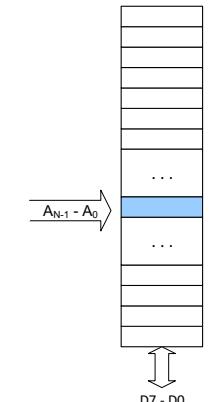
- Độ rộng của bus dữ liệu để trao đổi với bộ nhớ: $m = 8, 16, 32, 64, 128 \dots$ bit
 - Các ngăn nhớ được tổ chức theo byte
→ tổ chức bộ nhớ vật lý khác nhau

2017

Kiến trúc máy tính

38

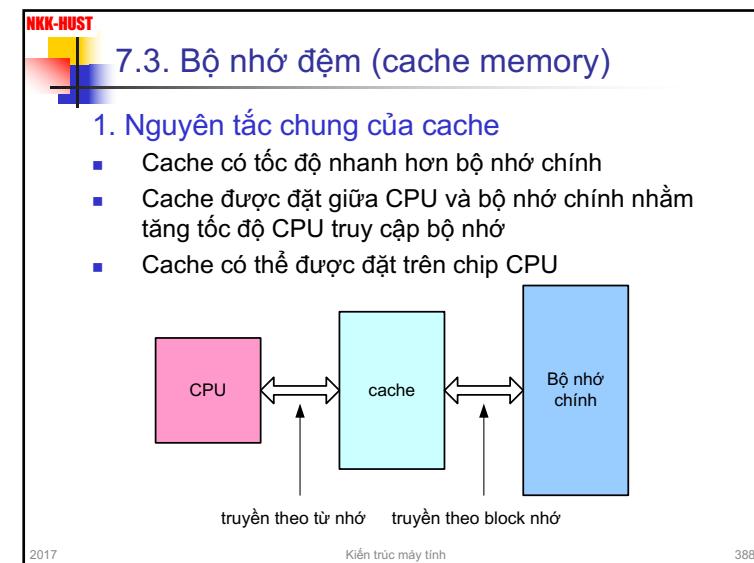
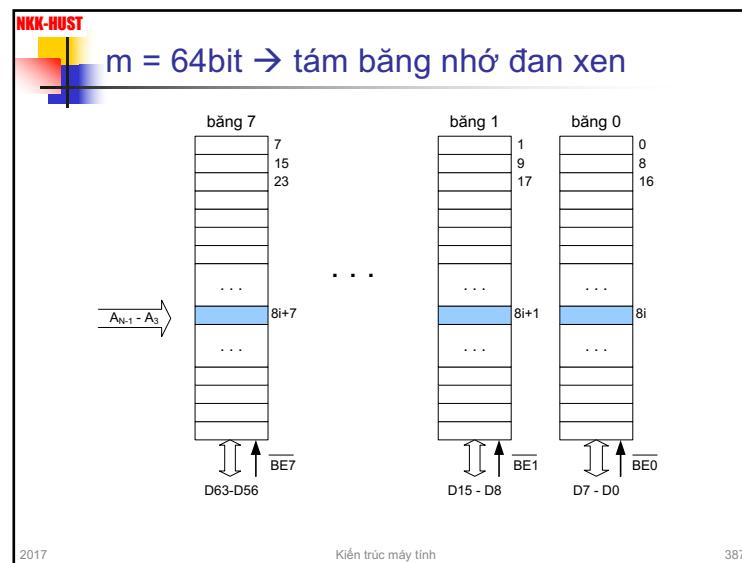
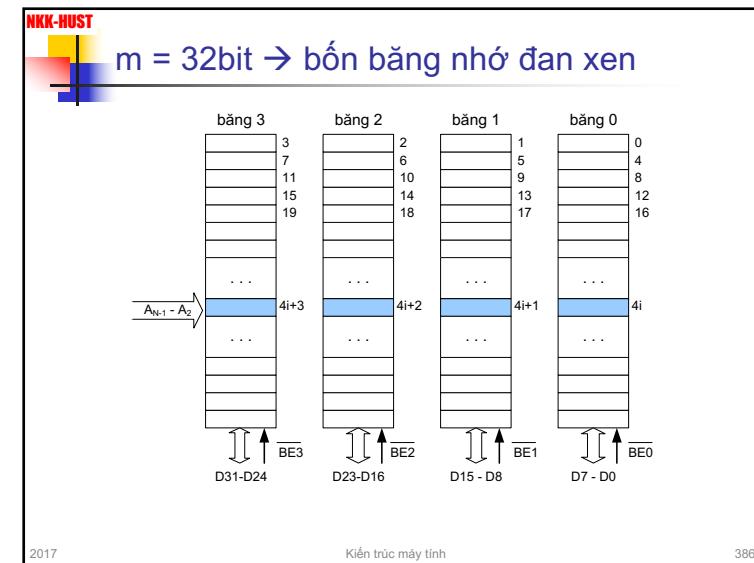
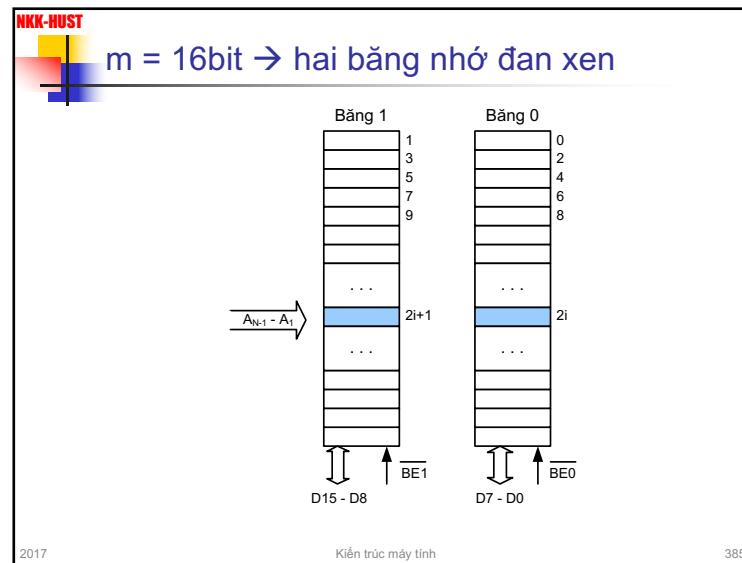
$m=8\text{bit} \rightarrow$ một băng nhớ tuyến tính



201

Kiến trúc máy tính

384



Ví dụ về thao tác của cache

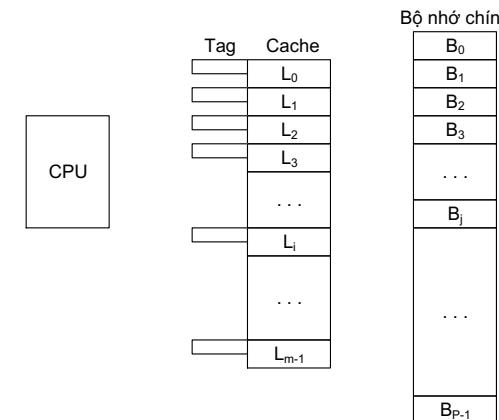
- CPU yêu cầu nội dung của ngăn nhớ
 - CPU kiểm tra trên cache với dữ liệu này
 - Nếu có, CPU nhận dữ liệu từ cache (nhanh)
 - Nếu không có, đọc Block nhớ chứa dữ liệu từ bộ nhớ chính vào cache
 - Tiếp đó chuyển dữ liệu từ cache vào CPU

2017

Kiến trúc máy tính

389

Cấu trúc chung của cache / bộ nhớ chính



39

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

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

2017

Kiến trúc máy tính

391

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).

201

Kiến trúc máy tính

392

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)

2017

Kiến trúc máy tính

393

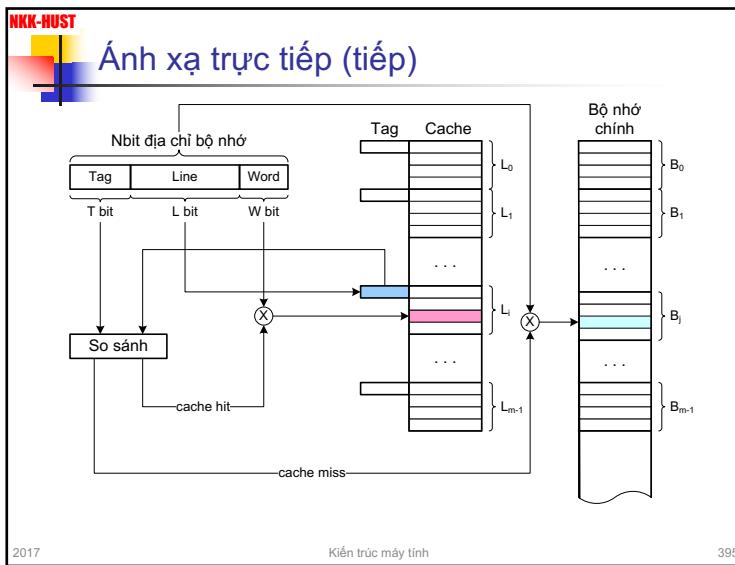
Ánh xạ trực tiếp

- 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.

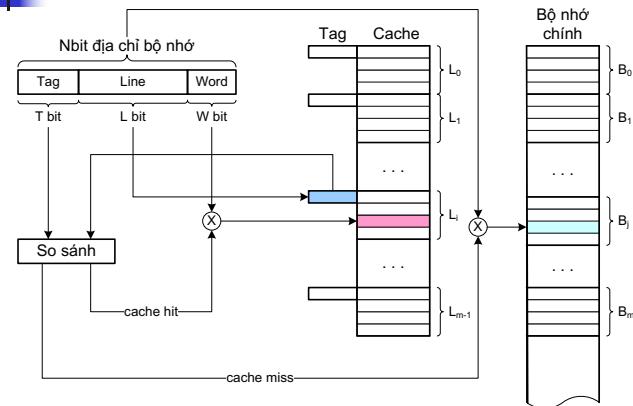
2017

Kiến trúc máy tính

394



Ánh xạ trực tiếp (tiếp)



2017

Kiến trúc máy tính

395

Ánh xạ trực tiếp (tiếp)

- Địa chỉ N bit của bộ nhớ chính chia thành ba trường:
 - Trường Word gồm W bit xác định một từ nhớ trong Block hay Line:
 2^W = kích thước của Block hay Line
 - Trường Line gồm L bit xác định một trong số các Line trong cache:
 2^L = số Line trong cache = m
 - Trường Tag gồm T bit:
 $T = N - (W+L)$

2017

Kiến trúc máy tính

396

Ánh xạ trực tiếp (tiếp)

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

2017

Kiến trúc máy tính

39

Ánh xạ liên kết toàn phần

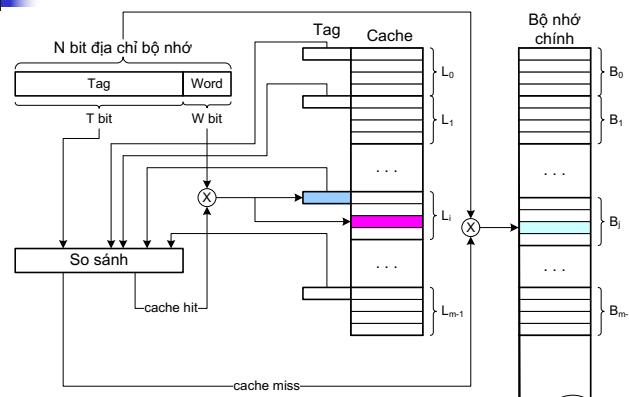
- Mỗi *Block* có thể nạp vào bất kỳ *Line* nào của *cache*
 - Địa chỉ của bộ nhớ chính chia thành hai trường:
 - Trường Word
 - Trường Tag dùng để xác định *Block* của bộ nhớ chính
 - Tag xác định *Block* đang nằm ở *Line* đó

2017

Kiến trúc máy tính

39

Ánh xạ liên kết toàn phần (tiếp)



2017

Kiến trúc máy tính

39

Ánh xạ liên kết toàn phần (tiếp)

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

2017

Kiến trúc máy tính

40

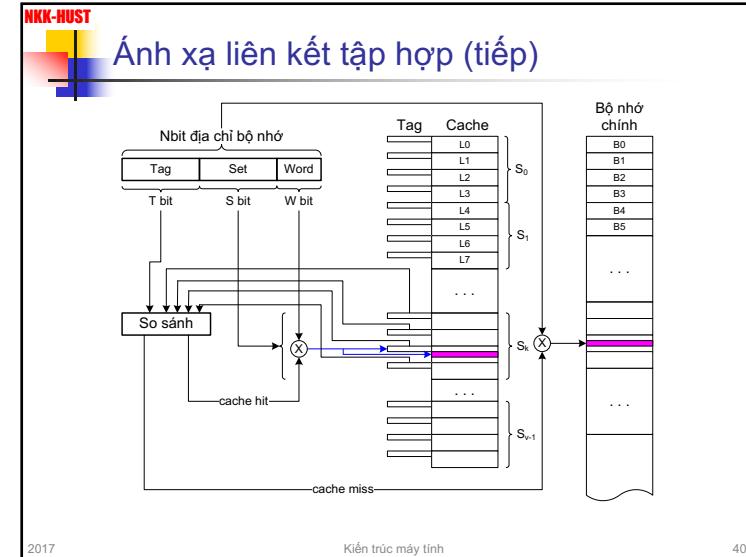
Ánh xạ liên kết tập hợp

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

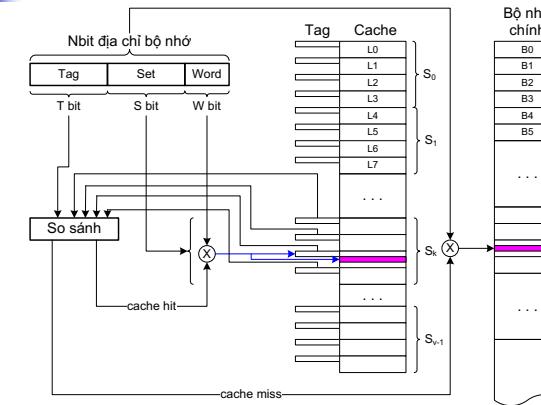
2017

Kiến trúc máy tính

40



Ánh xạ liên kết tập hợp (tiếp)



40

Ánh xạ liên kết tập hợp (tiếp)

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

2017

Kiến trúc máy tính

40

Ví dụ về ánh xạ địa chỉ

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

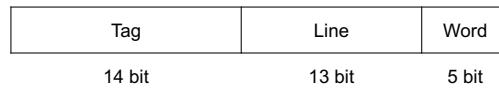
201

Kiến trúc máy tính

40

Với ánh xạ trực tiếp

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



2017

Kiến trúc máy tín

405

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



Kiến trúc máy tính

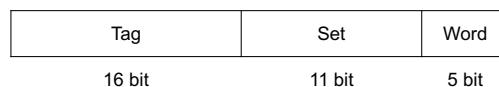
406

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



2017

Kiến trúc máy tín

407

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

Kiến trúc máy tính

408

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**

201

Kiến trúc máy tính

40

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

201

Kiến trúc máy tính

41

7.4. Bộ nhớ ngoài

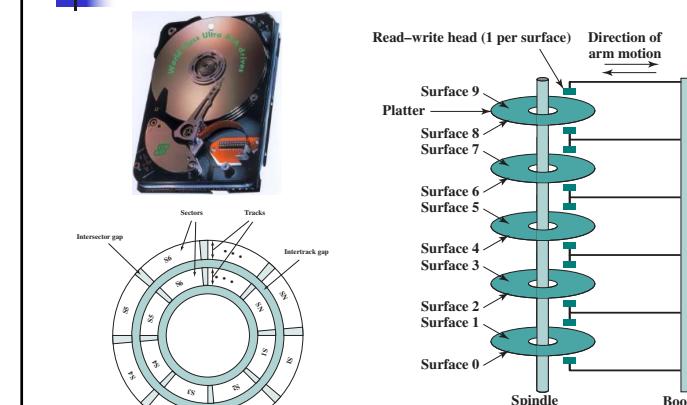
- Tồn tại dưới dạng các thiết bị lưu trữ
 - Các kiểu bộ nhớ ngoài
 - Băng từ: ít sử dụng
 - Đĩa từ: Ổ đĩa cứng HDD (Hard Disk Drive)
 - Đĩa quang: CD, DVD
 - Bộ nhớ Flash:
 - Ổ nhớ thẻ rắn SSD (Solid State Drive)
 - USB flash
 - Thẻ nhớ

201

Kiến trúc máy tí

41

Ổ đĩa cứng (HDD –Hard Disk Drive)



201

Kiến trúc máy tính

412

NKK-HUST

HDD

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

2017

Kiến trúc máy tính

413

NKK-HUST

Ổ SSD (Solid State Drive)

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

A photograph of a Toshiba 128GB Solid State Drive (SSD). The drive is rectangular with a silver and black plastic case. On the top cover, there is a blue label with the word 'TOSHIBA' at the top, followed by 'Solid State Drive' and '128 GB' below it. The bottom part of the drive shows the internal circuit board and heat spreader.

2017

Kiến trúc máy tính

41



NKK-HUST

Đĩa quang

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

NKK-HUST

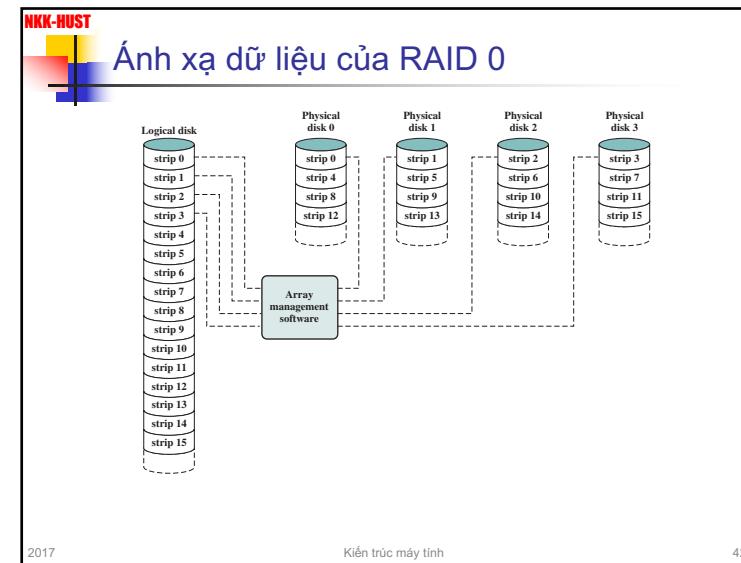
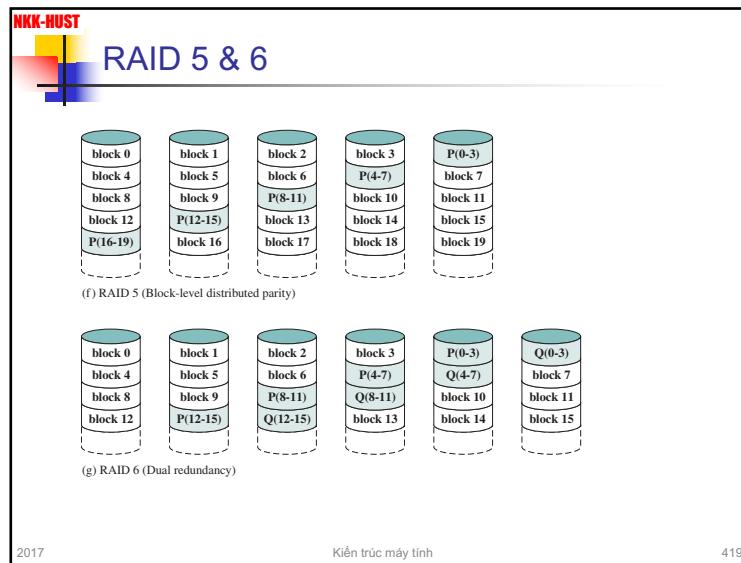
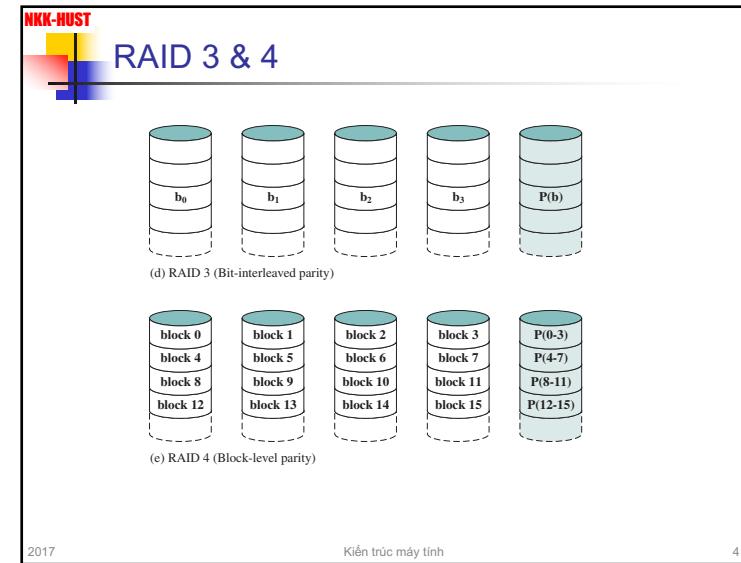
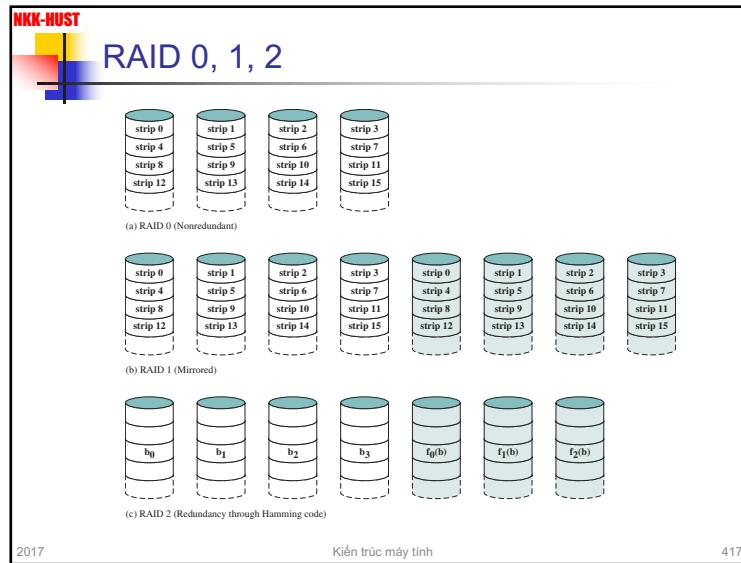
Hệ thống lưu trữ dung lượng lớn: RAID

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

2017

Kiến trúc máy tính

41



7.5. Bộ nhớ ảo (Virtual Memory)

- Khái niệm bộ nhớ ảo: gồm bộ nhớ chính và bộ nhớ ngoài mà được CPU coi như là một bộ nhớ duy nhất (bộ nhớ chính).
 - Các kỹ thuật thực hiện bộ nhớ ảo:
 - Kỹ thuật phân trang: Chia không gian địa chỉ bộ nhớ thành các trang nhớ có kích thước bằng nhau và nằm liền kề nhau
Thông dụng: kích thước trang = 4KiB
 - Kỹ thuật phân đoạn: Chia không gian nhớ thành các đoạn nhớ có kích thước thay đổi, các đoạn nhớ có thể gói lên nhau.

2017

Kiến trúc máy tí

42

Phân trang

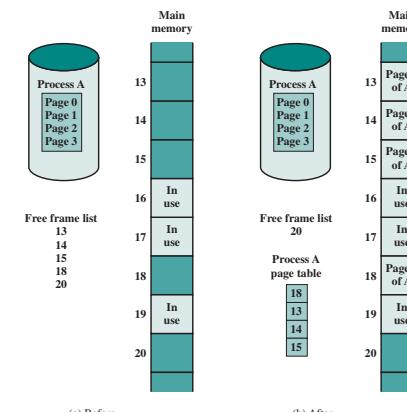
- Phân chia bộ nhớ thành các phần có kích thước bằng nhau gọi là các khung trang
 - Chia chương trình (tiến trình) thành các trang
 - Cấp phát số hiệu khung trang yêu cầu cho tiến trình
 - OS duy trì danh sách các khung trang nhớ trống
 - Tiến trình không yêu cầu các khung trang liên tiếp
 - Sử dụng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

2017

Kiến trúc máy tính

42

Cấp phát các khung trang

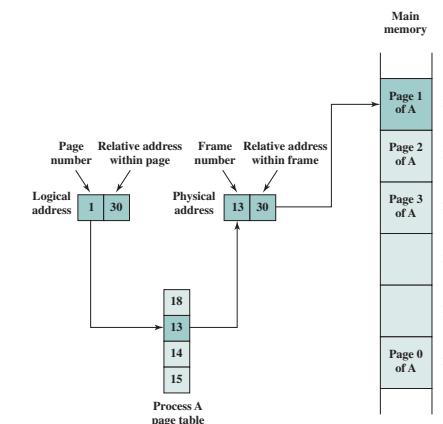


2017

Kiến trúc máy tí

42

Địa chỉ logic và địa chỉ vật lý của phân trang



2017

Kiến trúc máy tính

424

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

2017

Kiến trúc máy tính

425

Thất bại

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

201

Kiến trúc máy tính

42

Lợi ích

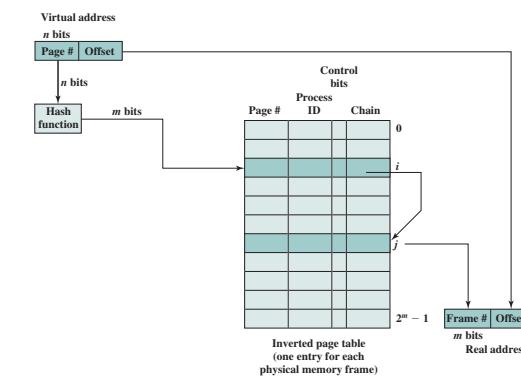
- Không cần toàn bộ tiến trình nằm trong bộ nhớ để chạy
 - Có thể hoán đổi trang được yêu cầu
 - Như vậy có thể chạy những tiến trình lớn hơn tổng bộ nhớ sẵn dùng
 - Bộ nhớ chính được gọi là bộ nhớ thực
 - Người dùng cảm giác bộ nhớ lớn hơn bộ nhớ thực

2017

Kiến trúc máy tính

427

Cấu trúc bảng trang



201

Kiến trúc máy tính

428

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

2017

Kiến trúc máy tính

429

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

201

Kiến trúc máy tính

430

Hết chương 7

2017

Kiến trúc máy tính

431

Chương 8

HỆ THỐNG VÀO-RA

Nguyễn Kim Khánh
Trường Đại học Bách khoa Hà Nội

201

Kiến trúc máy tính

432

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

NKK-HUST

Nội dung của chương 8

- 8.1. Tổng quan về hệ thống vào-ra
- 8.2. Các phương pháp điều khiển vào-ra
- 8.3. Nối ghép thiết bị vào-ra

2017

Kiến trúc máy tính

4

NKK-HUST

8.1. Tổng quan về hệ thống vào-ra

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

```
graph TD; B[Bus hệ thống] <--> M1[Mô-đun vào-ra]; B <--> M2[Mô-đун vào-ра]; M1 <--> T1[Thiết bị vào-ra]; M2 <--> T2[Thiết bị vào-ra]
```



Đặc điểm của hệ thống vào-ra

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

NKK-HUST

Thiết bị vào-ra

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

Cấu trúc chung của thiết bị vào-ra

```

graph LR
    A[Dữ liệu từ/dến mô-đun vào-ra] <--> B[Bộ đệm dữ liệu]
    B <--> C[Bộ chuyển đổi tín hiệu]
    C <--> D[Tín hiệu điều khiển]
    D --> E[Khối logic điều khiển]
    E --> C
    E --> F[Tín hiệu trạng thái]
    F --> B
  
```

NKK-HUST

Mô-đun vào-ra

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

The diagram illustrates the internal structure of an input-output module. It features two main functional blocks: a 'Bộ đếm dữ liệu' (Data Counter) and a 'Khối logic điều khiển' (Control Logic Block). The 'Bộ đếm dữ liệu' receives 'Các đường dữ liệu' (Data paths) from the 'Bus' and sends 'dữ liệu' (data) to the 'Công vào ra' (Input/Output Port). It also receives 'Tín hiệu điều khiển' (Control signals) and 'Tín hiệu trạng thái' (Status signals) from the 'Công vào ra'. The 'Khối logic điều khiển' receives 'Các đường địa chỉ' (Address paths) from the 'Bus' and sends 'dữ liệu' (data) to the 'Công vào ra'. It also receives 'Tín hiệu điều khiển' (Control signals) and 'Tín hiệu trạng thái' (Status signals) from the 'Công vào ra'. Both blocks have bidirectional connections with the 'Công vào ra'.

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

NKK-HUST

Không gian địa chỉ tách biệt

Không gian địa chỉ bộ nhớ	N bit	Không gian địa chi vào-ra	N ₁ bit
	000...000		00...00
	000...001		00...01
	000...010		00...10
	000...011		00...11
	000...100	.	
	000...101	.	
.			
		.	
		.	
		.	
.			
			11...11
	111...111		

2017

Kiến trúc máy tính

4

Các phương pháp địa chỉ hóa cổng vào-ra

NKK-HUST

Vào-ra theo bản đồ bộ nhớ

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

2017

Kiến trúc máy tính

4

NKK-HUST

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, 0xFFFF($0)    # ra cổng 1
```

Chú ý, giá trị 16-bit 0xFFFF được sign-extended thành 32-bit 0xFFFFFFFF4
- Đọc dữ liệu từ cổng 2 đưa vào \$t3


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

2017

Kiến trúc máy tính

445

NKK-HUST

Vào-ra riêng biệt (Isolated IO)

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

2017

Kiến trúc máy tính

446

NKK-HUST

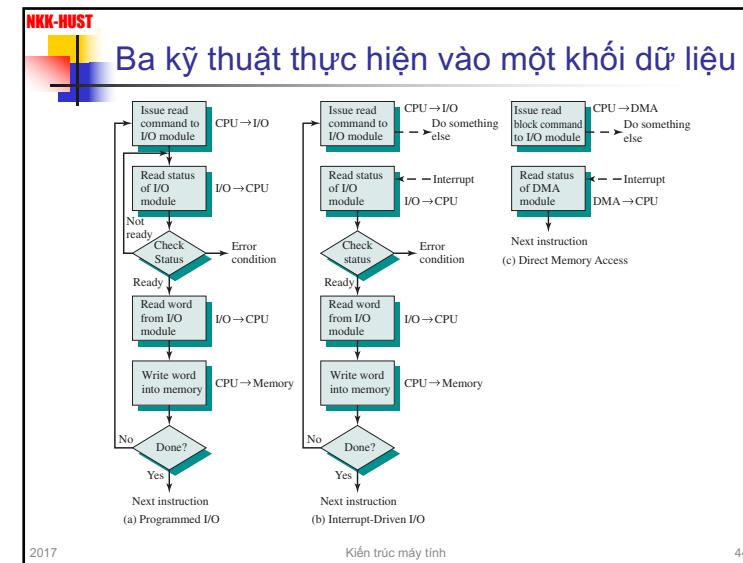
8.2. Các phương pháp điều khiển vào-ra

- 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)

2017

Kiến trúc máy tính

447



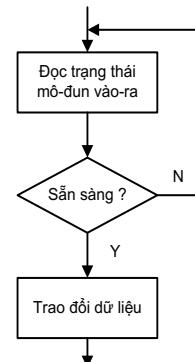
2017

Kiến trúc máy tính

448

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



201

Kiến trúc máy tí

44

Các tín hiệu điều khiển vào-ra

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

2017

Kiến trúc máy tính

45

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)

201

Kiến trúc máy tí

45

Đặc điểm

- Vào-ra do ý muốn của người lập trình
 - CPU trực tiếp điều khiển trao đổi dữ liệu giữa CPU với mô-đun vào-ra
 - CPU đợi mô-đun vào-ra → tiêu tốn nhiều thời gian của CPU

2017

Kiến trúc máy tính

452

2. Vào-ra điều khiển bằng ngắt

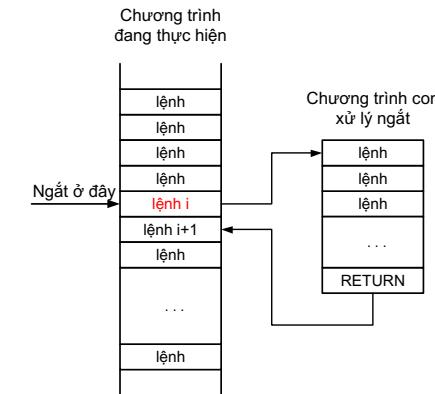
- Nguyên tắc chung:
 - CPU không phải đợi trạng thái sẵn sàng của mô-đun vào-ra, CPU thực hiện một chương trình nào đó
 - Khi mô-đun vào-ra sẵn sàng thì nó phát tín hiệu ngắt CPU
 - CPU thực hiện chương trình con xử lý ngắt vào-ra tương ứng để trao đổi dữ liệu
 - CPU trở lại tiếp tục thực hiện chương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

2017

Kiến trúc máy tín

453

Chuyển điều khiển đến chương trình con ngắn



2017

Kiến trúc máy tính

454

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

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

2017

Kiến trúc máy tín

455

Hoạt động vào dữ liệu: nhìn từ CPU

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

2017

Kiến trúc máy tính

456

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 ?

2017

Kiến trúc máy tính

457

Các phương pháp nối ghép ngắn

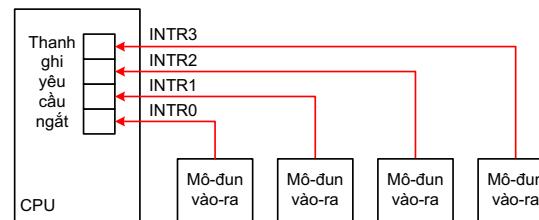
- Sử dụng nhiều đường yêu cầu ngắt
 - Hỏi vòng bằng phần mềm (Software Poll)
 - Hỏi vòng bằng phần cứng (Daisy Chain or Hardware Poll)
 - Sử dụng bộ điều khiển ngắt (PIC)

2017

Kiến trúc máy tính

458

Nhiều đường yêu cầu ngắt



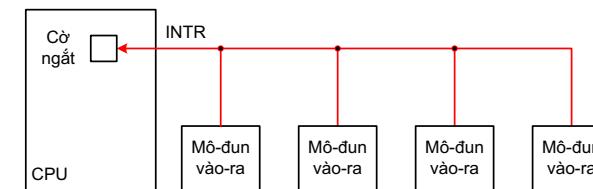
- Mỗi mô-đun vào-ra được nối với một đường yêu cầu ngắt
 - CPU phải có nhiều đường tín hiệu yêu cầu ngắt
 - Hạn chế số lượng mô-đun vào-ra
 - Các đường ngắt được qui định mức ưu tiên

2017

Kiến trúc máy tính

459

Hỏi vòng bằng phần mềm



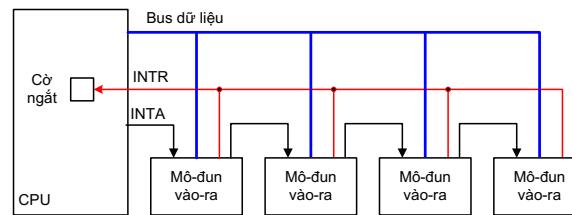
- CPU thực hiện phần mềm hỏi lần lượt từng mô-đun vào-ra
 - Chậm
 - Thứ tự các mô-đun được hỏi vòng chính là thứ tự ưu tiên

2017

Kiến trúc máy tính

460

Hỏi vòng bằng phần cứng



201

Kiến trúc máy tính

461

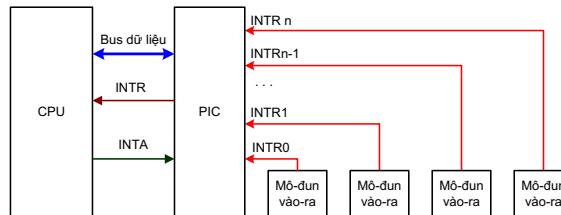
- CPU phát tín hiệu chấp nhận ngắn (INTA) đến mô-đun vào-ra đầu tiên
 - Nếu mô-đun vào-ra đó không gây ra ngắn thì nó gửi tín hiệu đến mô-đun kế tiếp cho đến khi xác định được mô-đun gây ngắn
 - Thứ tự các mô-đun vào-ra kết nối trong chuỗi xác định thứ tự ưu tiên

2017

Kiến trúc máy tính

46

Bộ điều khiển ngắt lập trình được



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

201

Kiến trúc máy tính

463

Đặc điểm của vào-ra điều khiển bằng ngắt

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

2017

Kiến trúc máy tính

46

3. DMA (Direct Memory Access)

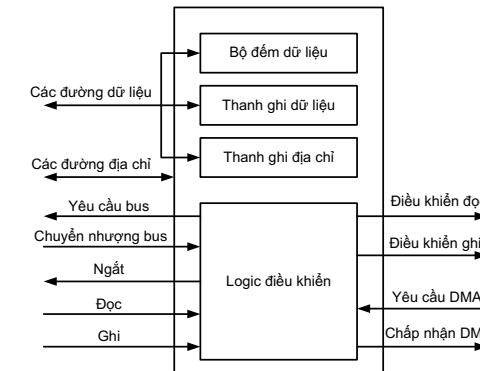
- Vào-ra bằng chương trình và bằng ngắt do CPU trực tiếp điều khiển:
 - Chiếm thời gian của CPU
 - Để khắc phục dùng kỹ thuật DMA
 - Sử dụng mô-đun điều khiển vào-ra chuyên dụng, gọi là DMAC (Controller), điều khiển trao đổi dữ liệu giữa mô-đun vào-ra với bộ nhớ chính

2017

Kiến trúc máy tính

465

Sơ đồ cấu trúc của DMA



201

Kiến trúc máy tính

46

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

2017

Kiến trúc máy tính

467

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

201

Kiến trúc máy tính

46

Các kiểu thực hiện DMA

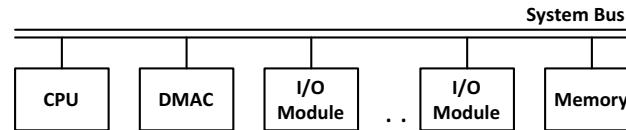
- DMA truyền theo khối (Block-transfer DMA): DMAC sử dụng bus để truyền xong cả khối dữ liệu
 - DMA lấy chu kỳ (Cycle Stealing DMA): DMAC cưỡng bức CPU treo tạm thời từng chu kỳ bus, DMAC chiếm bus thực hiện truyền một từ dữ liệu.
 - DMA trong suốt (Transparent DMA): DMAC nhận biết những chu kỳ nào CPU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

2017

Kiến trúc máy tín

469

Cấu hình DMA (1)



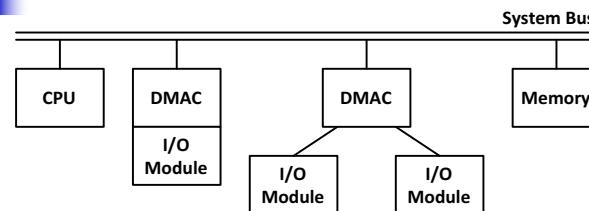
- Mỗi lần trao đổi một dữ liệu, DMAC sử dụng bus hai lần
 - Giữa mô-đun vào-ra với DMAC
 - Giữa DMAC với bộ nhớ

201

Kiến trúc máy tính

470

Cấu hình DMA (2)



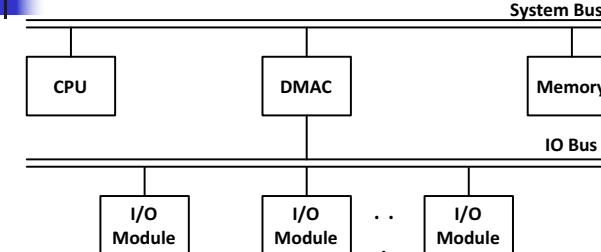
- DMAC điều khiển một hoặc vài mô-đun vào-ra
 - Mỗi lần trao đổi một dữ liệu, DMAC sử dụng bus một lần
 - Giữa DMAC với bộ nhớ

2017

Kiến trúc máy tín

471

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

201

Kiến trúc máy tính

472

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

2017

Kiến trúc máy tín

473

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

2017

Kiến trúc máy tính

474

8.3. Nối ghép thiết bị vào-ra

1. Các kiểu nối ghép vào-ra

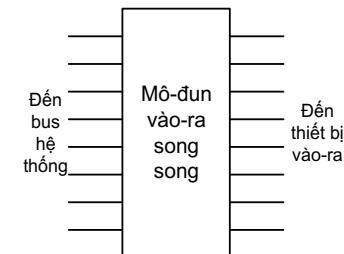
- Nối ghép song song
 - Nối ghép nối tiếp

2017

Kiến trúc máy tín

475

Nối ghép song song



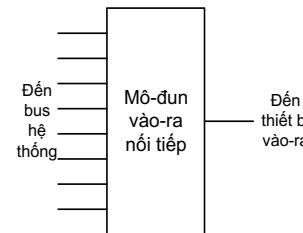
- Truyền nhiều bit song song
 - Tốc độ nhanh
 - Cần nhiều đường truyền dữ liệu

2017

Kiến trúc máy tính

476

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

2017

Kiến trúc máy tính

477

2. Các cấu hình nối ghép

■ Điểm tới điểm (Point to Point)

- Thông qua một cổng vào-ra nối ghép với một thiết bị

- Điểm tới đa điểm (Point to Multipoint)

- Thông qua một cổng vào-ra cho phép nối ghép được với nhiều thiết bị

- Ví dụ

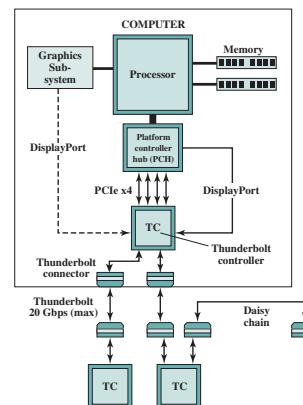
- USB (Universal Serial Bus): 127 thiết bị
 - IEEE 1394 (FireWire): 63 thiết bị
 - Thunderbolt

2017

Kiến trúc máy tính

478

Thunderbolt



2017

Kiến trúc máy tính

479

Hết chương 8

2017

Kiến trúc máy tính

480



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

NKK-HUST

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

2017

Kiến trúc máy tính

483

NKK-HUST

9.1. Phân loại kiến trúc máy tính

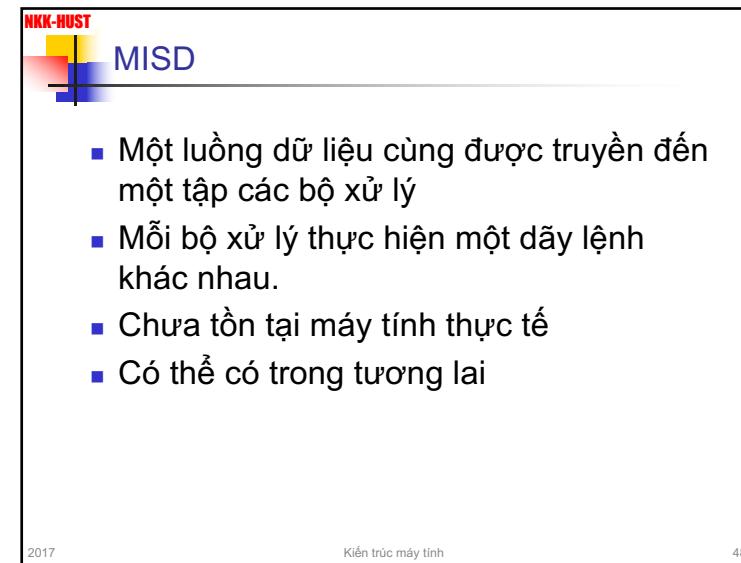
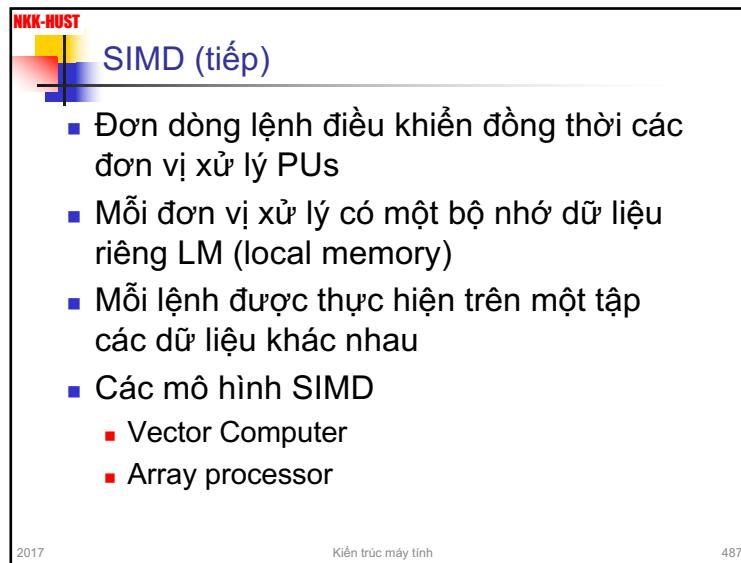
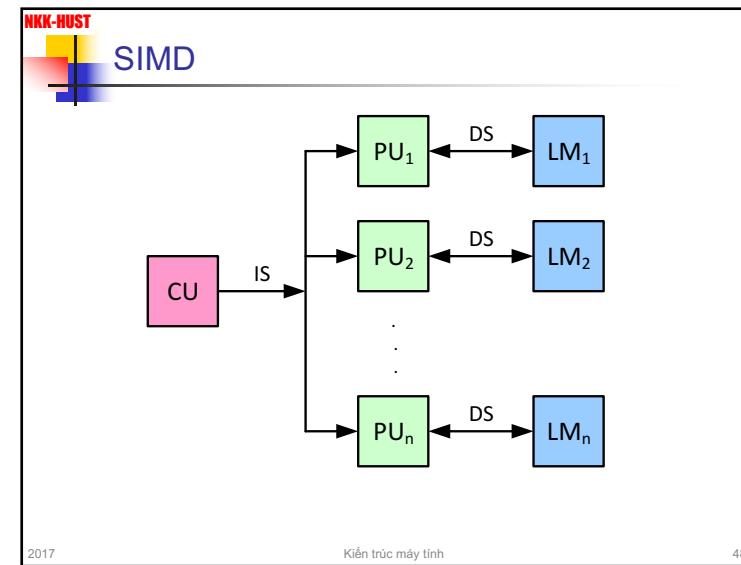
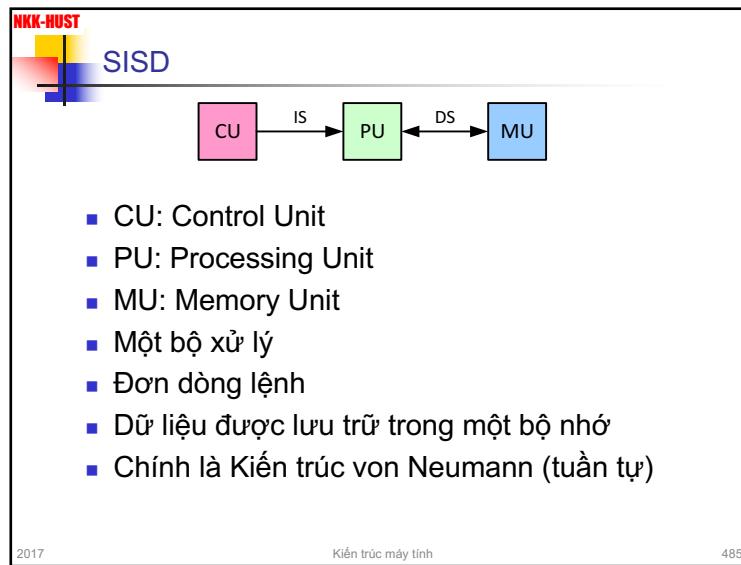
Phân loại kiến trúc máy tính (Michael Flynn -1966)

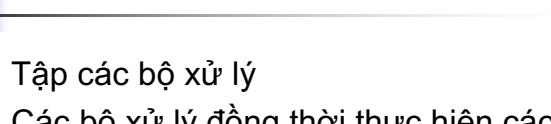
- SISD - Single Instruction Stream, Single Data Stream
- SIMD - Single Instruction Stream, Multiple Data Stream
- MISD - Multiple Instruction Stream, Single Data Stream
- MIMD - Multiple Instruction Stream, Multiple Data Stream

2017

Kiến trúc máy tính

4





- 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)

```

graph TD
    CU1[CU1] -- IS --> PU1[PU1]
    CU2[CU2] -- IS --> PU2[PU2]
    CUn[CUn] -- IS --> PUn[PUn]
    PU1 <-- DS --> SM[Bộ nhớ dùng chung]
    PU2 <-- DS --> SM
    PUn <-- DS --> SM
    style CU fill:#f08080,stroke:#000
    style PU fill:#90EE90,stroke:#000
    style SM fill:#ADD8E6,stroke:#000
  
```

MIMD - Distributed Memory



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

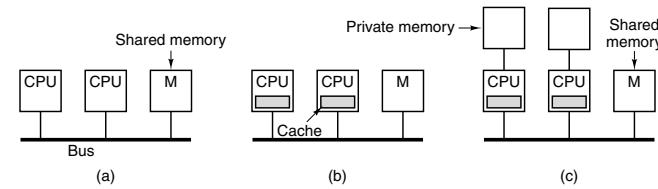
- Hệ thống đa xử lý đối xứng (SMP - Symmetric Multiprocessors)
 - Hệ thống đa xử lý không đối xứng (NUMA – Non-Uniform Memory Access)
 - Bộ xử lý đa lõi (Multicore Processors)

2017

Kiến trúc máy tính

493

SMP hay UMA (Uniform Memory Access)



2017

Kiến trúc máy tính

49

SMP (tiếp)

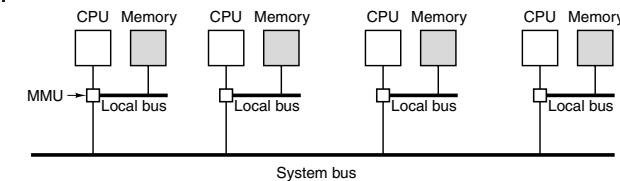
- Một máy tính có n \geq 2 bộ xử lý giống nhau
 - Các bộ xử lý dùng chung bộ nhớ và hệ thống vào-ra
 - Thời gian truy cập bộ nhớ là bằng nhau với các bộ xử lý
 - Các bộ xử lý có thể thực hiện chức năng giống nhau
 - Hệ thống được điều khiển bởi một hệ điều hành phân tán
 - Hiệu năng: Các công việc có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

2017

Kiến trúc máy tính

495

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

2017

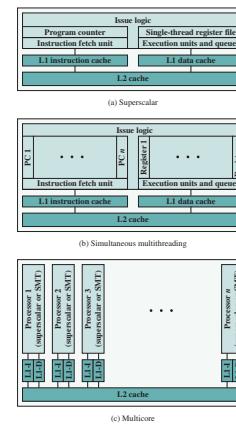
Kiến trúc máy tính

49

Bộ xử lý đa lõi (multicores)

- Thay đổi của bộ xử lý:

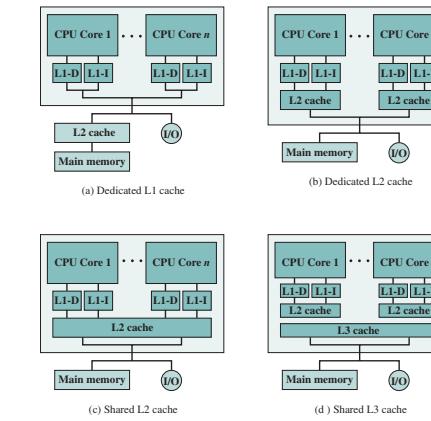
- Tuần tự
 - Pipeline
 - Siêu vô hướng
 - Đa luồng
 - Đa lõi: nhiều CPU
trên một chip



201

Kiến trúc máy tí

49



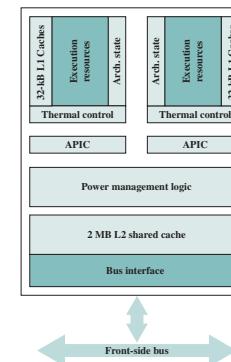
2

Kiến trúc máy tính

49

Intel - Core Duct

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

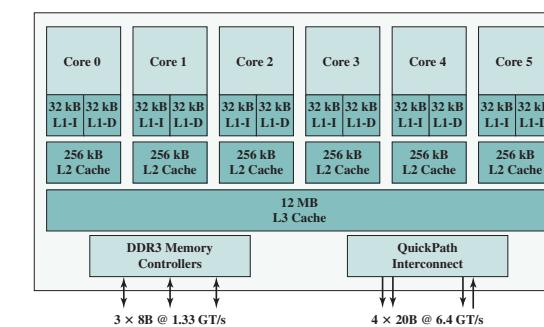


201

Kiến trúc máy tí

49

Intel Core i7-990X

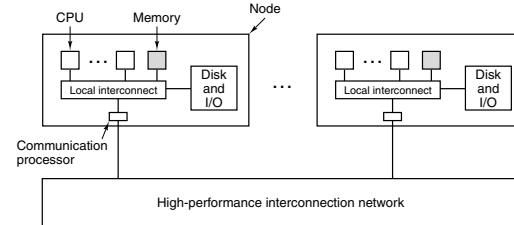


2

Kiến trúc máy tính

50

9.3. Đa xử lý bộ nhớ phân tán



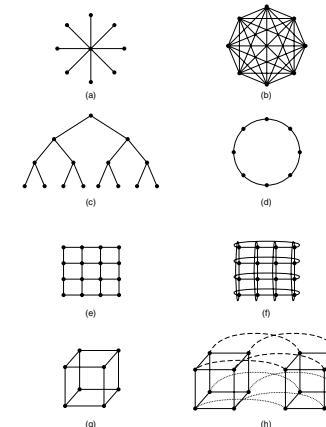
- Máy tính qui mô lớn (Warehouse Scale Computers or Massively Parallel Processors – MPP)
 - Máy tính cụm (clusters)

2017

Kiến trúc máy tín

501

Mạng liên kết



2017

Kiến trúc máy tính

502

Massively Parallel Processors

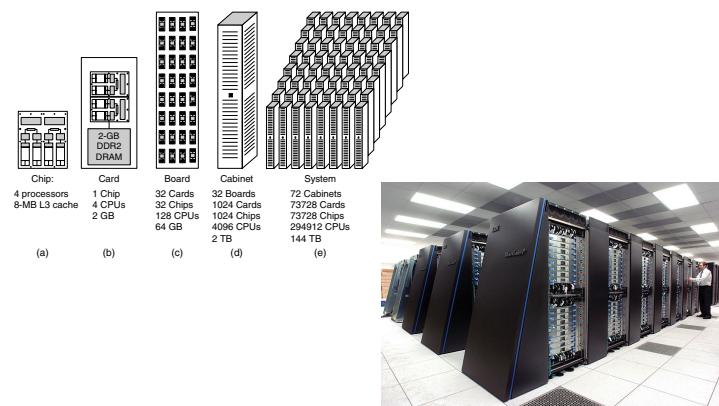
- Hệ thống qui mô lớn
 - Đắt tiền: nhiều triệu USD
 - Dùng cho tính toán khoa học và các bài toán có số phép toán và dữ liệu rất lớn
 - Siêu máy tính

2017

Kiến trúc máy tín

503

IBM Blue Gene/P



2017

Kiến trúc máy tính

504

Cluster

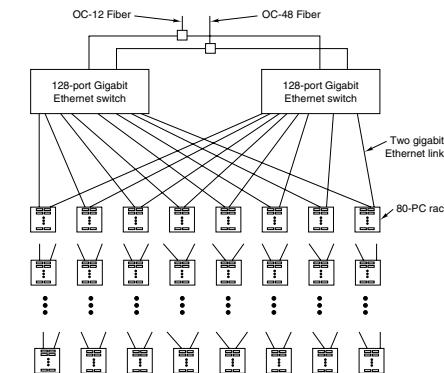
- Nhiều máy tính được kết nối với nhau bằng mạng liên kết tốc độ cao (~ Gbps)
 - Mỗi máy tính có thể làm việc độc lập (PC hoặc SMP)
 - Mỗi máy tính được gọi là một node
 - Các máy tính có thể được quản lý làm việc song song theo nhóm (cluster)
 - Toàn bộ hệ thống có thể coi như là một máy tính song song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

2017

Kiến trúc máy tính

505

PC Cluster của Google



201

Kiến trúc máy tính

50

9.4. Bộ xử lý đồ họa đa dụng

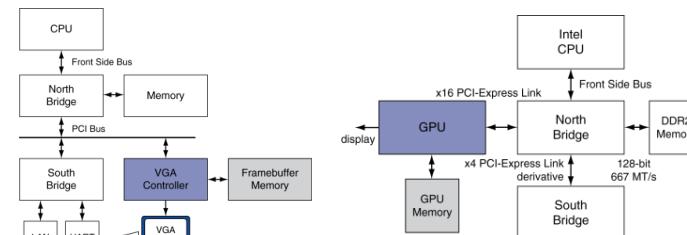
- Kiến trúc SIMD
 - Xuất phát từ bộ xử lý đồ họa GPU (Graphic Processing Unit) hỗ trợ xử lý đồ họa 2D và 3D: xử lý dữ liệu song song
 - GPGPU – General purpose Graphic Processing Unit
 - Hệ thống lai CPU/GPGPU
 - CPU là host: thực hiện theo tuần tự
 - GPGPU: tính toán song song

2017

Kiến trúc máy tính

507

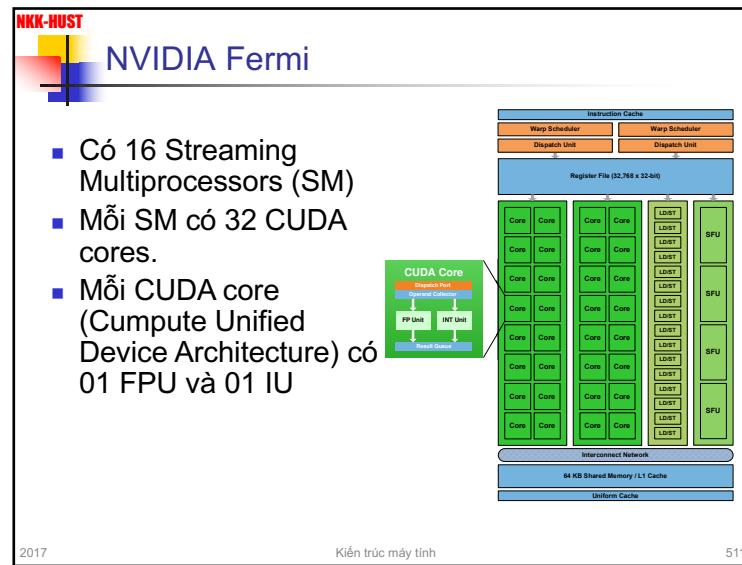
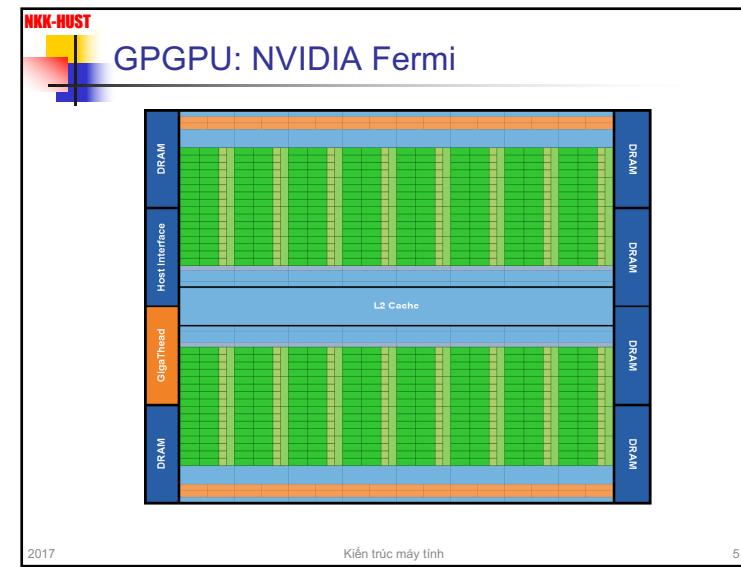
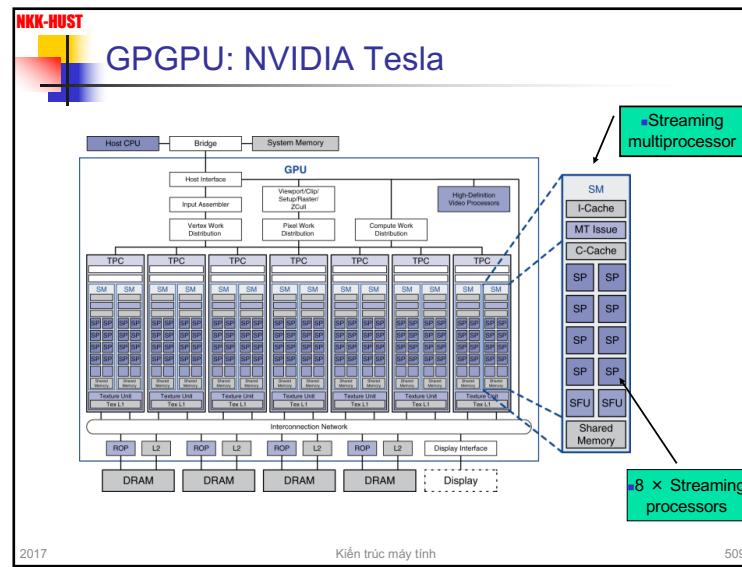
Bộ xử lý đồ họa trong máy tính



201

Kiến trúc máy tính

50



Hết