

The cover page features the logo of Hanoi University of Science and Technology (HUST) in the top left corner. The title "KIẾN TRÚC MÁY TÍNH" (Computer Architecture) is prominently displayed in large yellow and white letters. Below the title, the author's name "Nguyễn Kim Khánh" is written in red. Further down, the text "Bộ môn Kỹ thuật máy tính" (Department of Computer Engineering), "Viện Công nghệ thông tin và Truyền thông" (School of Information and Communication Technology), and "Version: Jan 2015" are included.

This page contains the "Contact Information" section with the NKK-HUST logo. It lists the address (502-B1), mobile number (091-358-5533), and two email addresses: khanhnk@soict.hust.edu.vn and khanh.nguyenkim@hust.edu.vn. The footer includes the date "Jan2015", the course name "Computer Architecture", and the page number "2".

The slide is titled "Mục tiêu học phần" (Learning Objectives). It contains two main bullet points: one about the basic knowledge of computer architecture and another detailing specific skills such as understanding instruction set architecture, assembly language, and program optimization.

The slide is titled "Tài liệu học tập" (Study Materials). It lists the course material ("Bài giảng Kiến trúc máy tính: CA-Jan2015") available at <ftp://dce.hust.edu.vn/khanhnk/CA/>, and four recommended books: William Stallings' "Computer Organization and Architecture", David A. Patterson and John L. Hennessy's "Computer Organization and Design", David Money Harris and Sarah L. Harris' "Digital Design and Computer Architecture", and Andrew S. Tanenbaum's "Structured Computer Organization". The footer includes the date "Jan2015", the course name "Computer Architecture", and the page number "4".

NKK-HUST

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

Jan2015 Computer Architecture 5

NKK-HUST

Content

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

Jan2015 Computer Architecture 6

NKK-HUST

Kiến trúc máy tính

Chương 1 GIỚI THIỆU CHUNG

Nguyễn Kim Khánh
Trường Đại học Bách khoa Hà Nội

Jan2015 Computer Architecture 7

NKK-HUST

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

Jan2015 Computer Architecture 8

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

Jan2015 Computer Architecture 9

NKK-HUST

Mô hình đơn giản của máy tính

```

graph LR
    A["Các thiết bị vào (Input Devices)"] -- dữ liệu vào --> B["Bộ xử lý trung tâm (Central Processing Unit)"]
    B -- dữ liệu ra --> C["Các thiết bị ra (Output Devices)"]
    B <-- dữ liệu ra --> D["Bộ nhớ chính (Main Memory)"]
    D -- dữ liệu vào --> B
    
```

The diagram illustrates a simple computer architecture. It consists of four main components: 'Các thiết bị vào (Input Devices)' (left), 'Bộ xử lý trung tâm (Central Processing Unit)' (top center), 'Bộ nhớ chính (Main Memory)' (bottom center), and 'Các thiết bị ra (Output Devices)' (right). Arrows indicate the flow of data: 'dữ liệu vào' (data in) from the input devices to the CPU, 'dữ liệu ra' (data out) from the CPU to the output devices, and bidirectional 'dữ liệu ra' (data out) between the CPU and memory. The memory also receives 'dữ liệu vào' (data in) from the CPU.

Jan2015 Computer Architecture 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 sẵn trong thiết bị khác
 - Được thiết kế chuyên dụng

Jan2015 Computer Architecture 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

Jan2015 Computer Architecture 12

NKK-HUST

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 **Vì 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

Jan2015 Computer Architecture 13

NKK-HUST

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

Jan2015 Computer Architecture 14

NKK-HUST

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

Jan2015 Computer Architecture 15

NKK-HUST

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

Jan2015 Computer Architecture 16

1.3. Sự tiến hóa của công nghệ máy tính

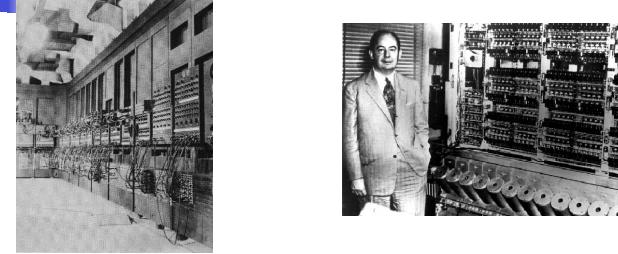
- Máy tính dùng đèn điện tử chân không (1950s)
 - Máy tính ENIAC: máy tính đầu tiên (1946)
 - Máy tính IAS: máy tính von Neumann (1952)
- Máy tính dùng transistors (1960s)
- Máy tính dùng vi mạch SSI, MSI và LSI (1970s)
 - SSI - Small Scale Integration
 - MSI - Medium Scale Integration
 - LSI - Large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

Jan2015

Computer Architecture

17

Máy tính đầu tiên: ENIAC và IAS



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

Jan2015

Computer Architecture

18

Máy tính ngày nay



Jan2015

Computer Architecture

19

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

Jan2015

Computer Architecture

20

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



Jan2015 Computer Architecture 21

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”

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

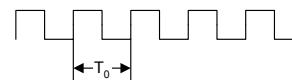
- Ví dụ: Thời gian chạy chương trình:
 - 10s trên máy A, 15s trên máy B
 - $t_B/t_A = 15s / 10s = 1.5$
 - Vậy máy A nhanh hơn máy B 1.5 lần

Jan2015 Computer Architecture 22

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 $f_0 = 4\text{GHz} = 4 \times 10^9\text{Hz}$
 $T_0 = 1/(4 \times 10^9) = 0.25 \times 10^{-9}\text{s} = 0.25\text{ns}$

Jan2015 Computer Architecture 23

NKK-HUST

Thời gian thực hiện của CPU

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

$\text{Thời gian thực hiện của CPU} = \text{Số chu kỳ xung nhịp} \times \text{Thời gian một chu kỳ}$

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

n : số chu kỳ xung nhịp

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

Jan2015 Computer Architecture 24

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)?

Jan2015

Computer Architecture

25

Ví dụ (tiếp)

$$\text{Ta có: } t = \frac{n}{f}$$

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:

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

Jan2015

Computer Architecture

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

Jan2015

Computer Architecture

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 ?

Jan2015

Computer Architecture

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

Jan2015

Computer Architecture

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:

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

Jan2015

Computer Architecture

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

Jan2015

Computer Architecture

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 = $= 1 \times 40 + 2 \times 10 + 3 \times 10 = 90$
 - $CPI_{TB} = 90/60 = 1.5$

Jan2015

Computer Architecture

32

Tóm tắt về Hiệu năng

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

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

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

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

Jan2015 Computer Architecture 33

MIPS như là thước đo hiệu năng

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

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

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

Jan2015 Computer Architecture 34

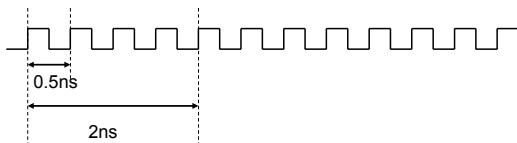
Ví dụ

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

Jan2015 Computer Architecture 35

Ví dụ

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



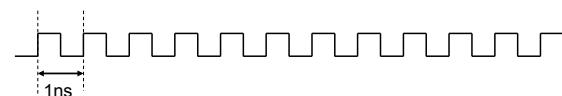
- Chu kỳ $T_0 = 1/(2 \times 10^9) = 0.5\text{ns}$
- CPI = 4 → thời gian thực hiện 1 lệnh = $4 \times 0.5\text{ns} = 2\text{ns}$
- Số lệnh thực hiện trong 1s = $(10^9\text{ns})/(2\text{ns}) = 5 \times 10^8$ lệnh
- Vậy bộ xử lý thực hiện được 500 MIPS

Jan2015 Computer Architecture 36

NKK-HUST

Ví dụ

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

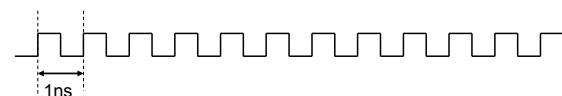


Jan2015 Computer Architecture 37

NKK-HUST

Ví dụ

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



- Chu kỳ $T_0 = 1/10^9 = 1\text{ns}$
- 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

Jan2015 Computer Architecture 38

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})

Jan2015 Computer Architecture 39

NKK-HUST

Các ý tưởng vĩ đại trong kiến trúc máy tính

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



Jan2015 Computer Architecture 40



Hết chương 1

Jan2015 Computer Architecture 41



Kiến trúc máy tính

Chương 2 CƠ BẢN VỀ LOGIC SỐ

Nguyễn Kim Khánh
Trường Đại học Bách khoa Hà Nội

Jan2015 Computer Architecture 42



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

Jan2015 Computer Architecture 43



Nội dung của chương 2

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

Jan2015 Computer Architecture 44

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

Jan2015

Computer Architecture

45

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 = $10^n - 1$

Jan2015

Computer Architecture

46

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:

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

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

Jan2015

Computer Architecture

47

Ví dụ số thập phân

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

Jan2015

Computer Architecture

48

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...000 = 0
 - 11...111 = $2^n - 1$
- Các lệnh của chương trình và dữ liệu trong máy tính đều được mã hóa bằng số nhị phân

Jan2015 Computer Architecture 49

NKK-HUST

Biểu diễn số nhị phân

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

Jan2015 Computer Architecture 50

NKK-HUST

Đơn vị dữ liệu trong máy tính

- Byte = 8-bit
- Qui ước mới về đơn vị dữ liệu trong máy tính:

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}

Jan2015 Computer Architecture 51

NKK-HUST

Dạng tổng quát của số nhị phân

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

Giá trị của A được tính như sau:

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

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

Jan2015 Computer Architecture 52

NKK-HUST

Ví dụ số nhị phân

$$1101001.1011_{(2)} =$$

$$\begin{array}{r} 6 \ 5 \ 4 \ 3 \ 2 \ 1 \ 0 \ -1 \ -2 \ -3 \ -4 \\ \hline 1 \ 1 \ 0 \ 1 \ 0 \ 0 \ 1 \ . \ 1 \ 0 \ 1 \ 1 \end{array}$$

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

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

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

Jan2015

Computer Architecture

53

NKK-HUST

Chuyển đổi số nguyên thập phân sang nhị phân

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

Jan2015

Computer Architecture

54

NKK-HUST

Phương pháp chia dần cho 2

- Ví dụ: chuyển đổi $105_{(10)}$

▪ $105 : 2 = 52$	dư 1	biểu diễn số dư theo chiều mũi tên
▪ $52 : 2 = 26$	dư 0	
▪ $26 : 2 = 13$	dư 0	
▪ $13 : 2 = 6$	dư 1	
▪ $6 : 2 = 3$	dư 0	
▪ $3 : 2 = 1$	dư 1	
▪ $1 : 2 = 0$	dư 1	
- Kết quả: $105_{(10)} = 1101001_{(2)}$

Jan2015

Computer Architecture

55

NKK-HUST

Phương pháp phân tích thành tổng của các 2^i

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

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

 - Kết quả: $105_{(10)} = 0110\ 1001_{(2)}$
- Ví dụ 2: $17000_{(10)} = 16384 + 512 + 64 + 32 + 8$

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

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

Jan2015

Computer Architecture

56

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

biểu diễn theo chiều mũi tên
- Kết quả : $0.6875_{(10)} = 0.1011_{(2)}$

Jan2015

Computer Architecture

57

NKK-HUST

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

- Ví dụ 2: chuyển đổi $0.81_{(10)}$
 - $0.81 \times 2 = 1.62$ phần nguyên = 1
 - $0.62 \times 2 = 1.24$ phần nguyên = 1
 - $0.24 \times 2 = 0.48$ phần nguyên = 0
 - $0.48 \times 2 = 0.96$ phần nguyên = 0
 - $0.96 \times 2 = 1.92$ phần nguyên = 1
 - $0.92 \times 2 = 1.84$ phần nguyên = 1
 - $0.84 \times 2 = 1.68$ phần nguyên = 1
- $0.81_{(10)} \approx 0.1100111_{(2)}$

Jan2015

Computer Architecture

58

NKK-HUST

3. Hệ mười sáu (Hexa)

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

Jan2015

Computer Architecture

59

NKK-HUST

Quan hệ giữa số nhị phân và số Hexa

Ví dụ:

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

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

Jan2015

Computer Architecture

60

2.2. Đại số Boole

- Đại số Boole sử dụng các biến logic và phép toán logic
- Biến logic có thể nhận giá trị 1 (TRUE) hoặc 0 (FALSE)
- Các phép toán logic cơ bản: AND, OR và NOT
 - A AND B : $A \cdot B$ 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: $A \oplus B = A \cdot \bar{B} + \bar{A} \cdot B$

Jan2015

Computer Architecture

61

Phép toán đại số Boole với hai biến

A	B	A AND B $A \cdot B$
0	0	0
0	1	0
1	0	0
1	1	1

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

A	NOT A \bar{A}
0	1
1	0

NOT là phép toán 1 biến

A	B	A NAND B $\overline{A \cdot B}$
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

Jan2015

Computer Architecture

62

Các đồng nhất thức của đại số Boole

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

Jan2015

Computer Architecture

63

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

Jan2015

Computer Architecture

64

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		$F = \overline{AB}$	<table border="1"> <thead> <tr> <th>A</th> <th>B</th> <th>F</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>1</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> </tr> </tbody> </table>	A	B	F	0	0	1	0	1	1	1	0	1	1	1	0
A	B	F																
0	0	1																
0	1	1																
1	0	1																
1	1	0																
NOR		$F = \overline{A+B}$	<table border="1"> <thead> <tr> <th>A</th> <th>B</th> <th>F</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>1</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> </tr> </tbody> </table>	A	B	F	0	0	1	0	1	0	1	0	0	1	1	0
A	B	F																
0	0	1																
0	1	0																
1	0	0																
1	1	0																
XOR		$F = A \oplus B$	<table border="1"> <thead> <tr> <th>A</th> <th>B</th> <th>F</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> </tr> </tbody> </table>	A	B	F	0	0	0	0	1	1	1	0	1	1	1	0
A	B	F																
0	0	0																
0	1	1																
1	0	1																
1	1	0																

Jan2015

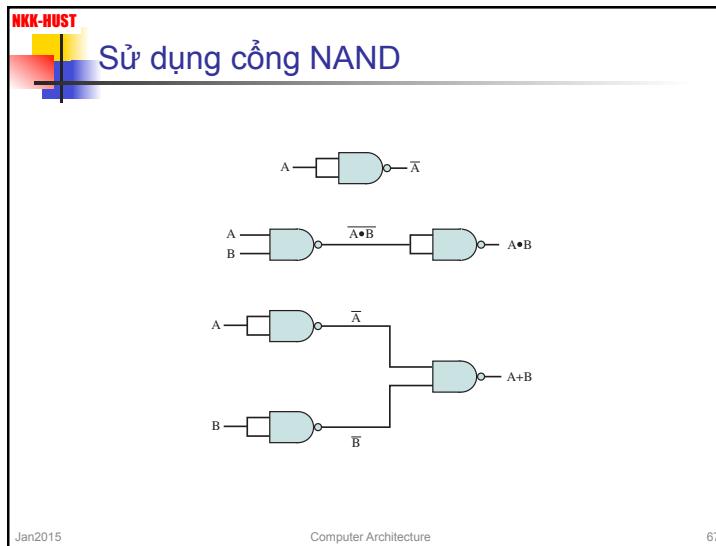
65

- 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}

Jan2015

Computer Architecture

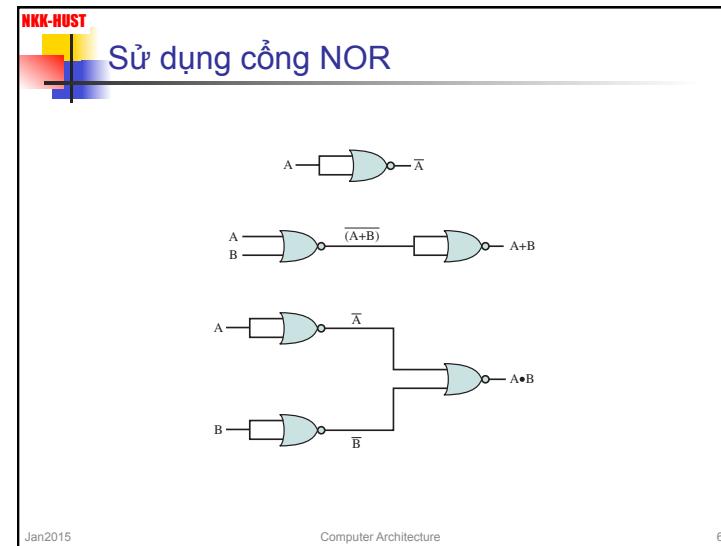
66



Jan2015

Computer Architecture

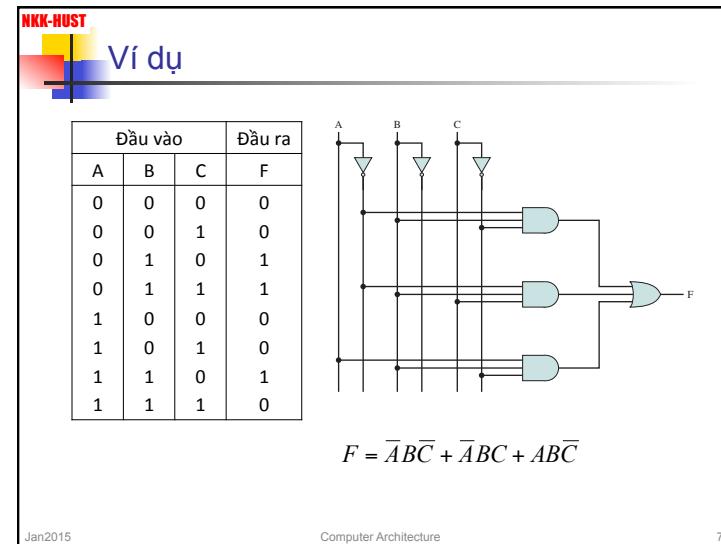
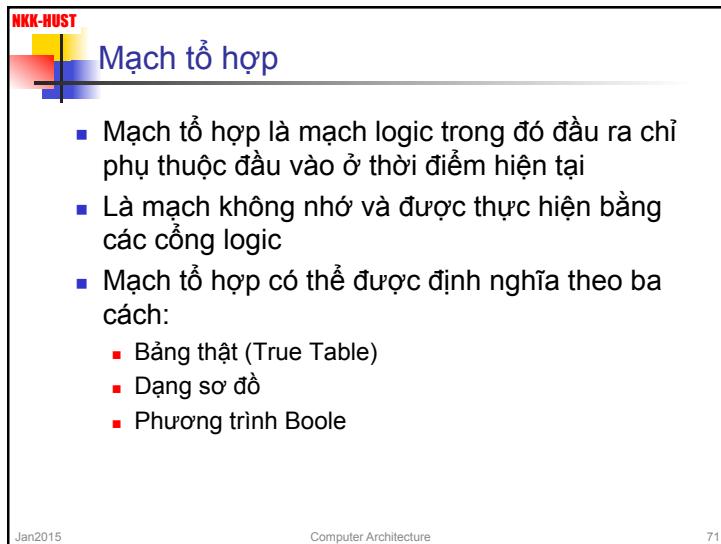
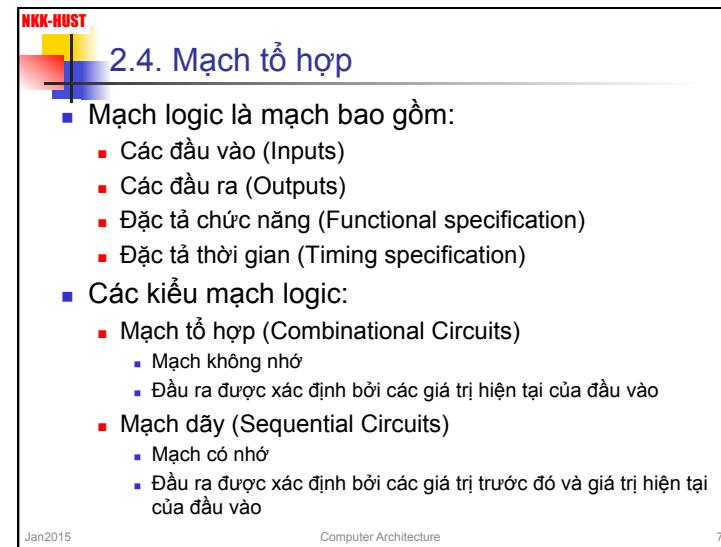
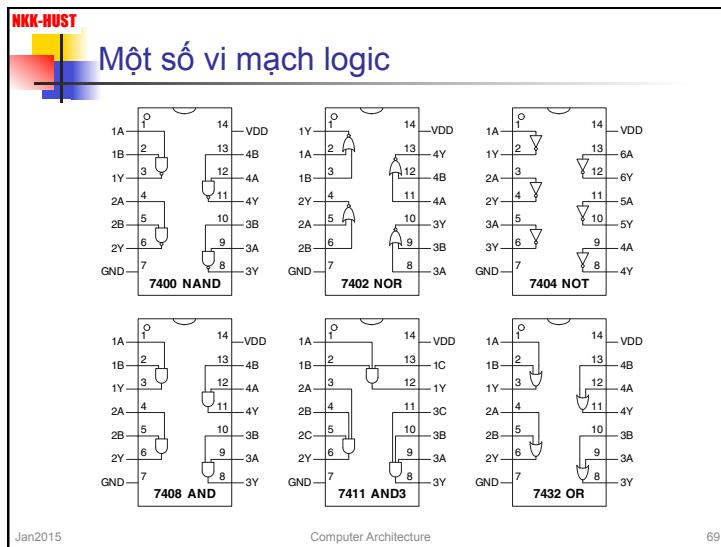
67



Jan2015

Computer Architecture

68



NKK-HUST

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)

Jan2015 Computer Architecture 73

NKK-HUST

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

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

Jan2015 Computer Architecture 74

NKK-HUST

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

Jan2015 Computer Architecture 75

NKK-HUST

Thực hiện bộ giải mã 3 ra 8

Jan2015 Computer Architecture 76

NKK-HUST

Bộ cộng

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

Jan2015

Computer Architecture

77

NKK-HUST

Bộ cộng bán phần 1-bit

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

$$S = A \oplus B$$

$$C_{out} = AB$$

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

Jan2015

Computer Architecture

78

NKK-HUST

Bộ cộng toàn phần 1-bit

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

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

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

Jan2015

Computer Architecture

79

NKK-HUST

Bộ cộng 4-bit và bộ cộng 32-bit

Jan2015

Computer Architecture

80

2.5. Mạch dãy

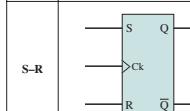
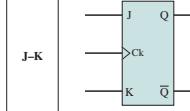
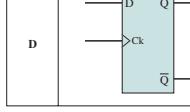
- Mạch dãy là mạch logic trong đó đầu ra phụ thuộc giá trị đầu vào ở thời điểm hiện tại và đầu vào ở thời điểm quá khứ
- Là mạch có nhớ, được thực hiện bằng phần tử nhớ (Latch, Flip-Flop) và có thể kết hợp với các cổng logic

Jan2015

Computer Architecture

81

Các Flip-Flop cơ bản

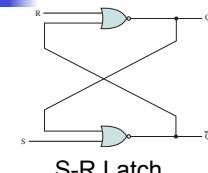
Name	Graphical Symbol	Truth Table															
S-R		<table border="1"> <thead> <tr> <th>S</th><th>R</th><th>Q_{n+1}</th></tr> </thead> <tbody> <tr> <td>0</td><td>0</td><td>Q_n</td></tr> <tr> <td>0</td><td>1</td><td>0</td></tr> <tr> <td>1</td><td>0</td><td>1</td></tr> <tr> <td>1</td><td>1</td><td>-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

Jan2015

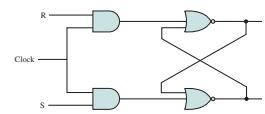
Computer Architecture

82

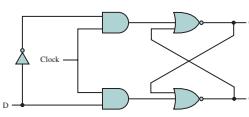
S-R Latch và các Flip-Flop



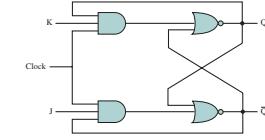
S-R Latch



S-R Flip-Flop



D Flip Flop



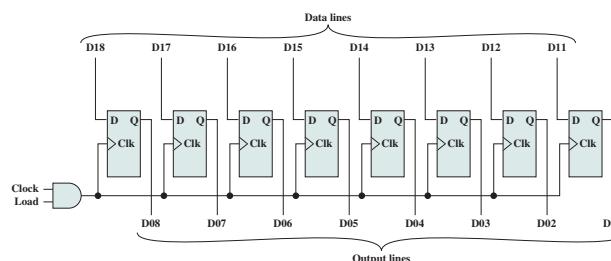
J-K Flip-Flop

Jan2015

Computer Architecture

83

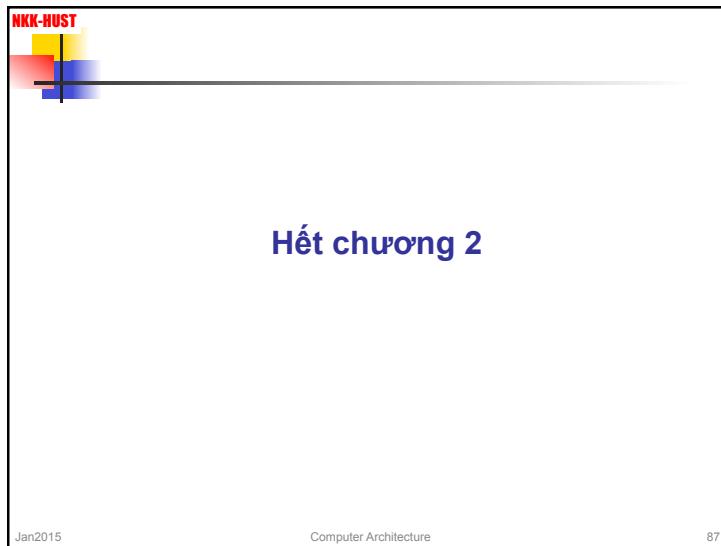
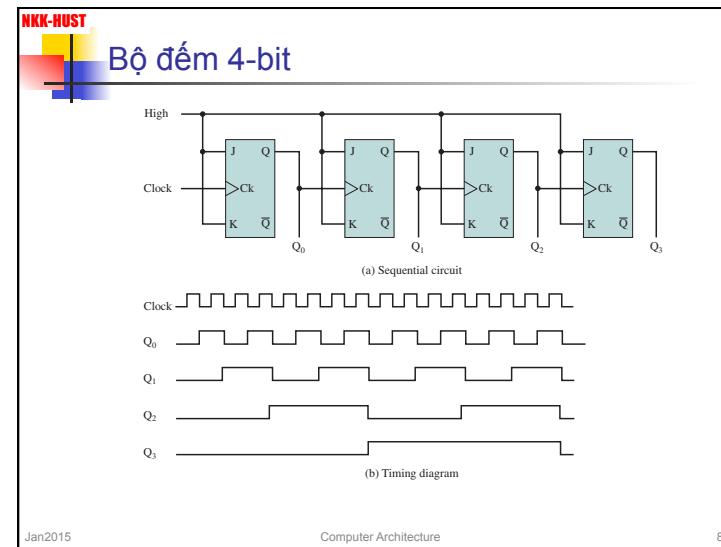
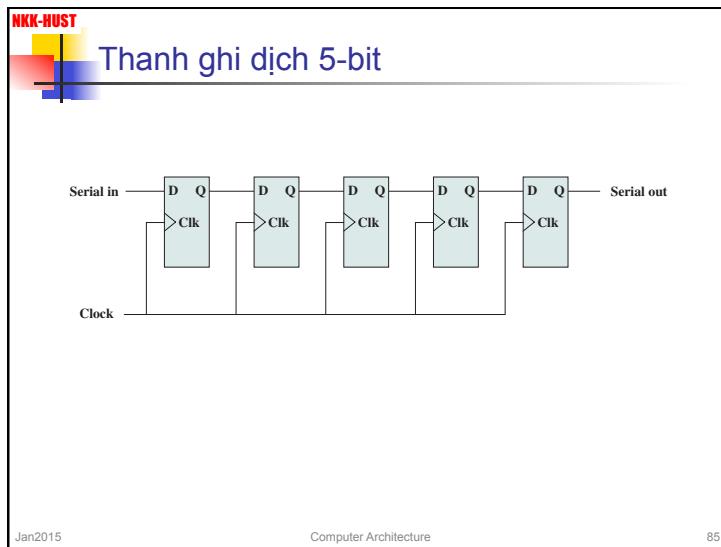
Thanh ghi 8-bit song song



Jan2015

Computer Architecture

84



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

Jan2015 Computer Architecture 89

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

Jan2015 Computer Architecture 90

3.1. Các thành phần cơ bản của máy tính

```

graph TD
    CPU[CPU] <--> Bus[Bus hệ thống]
    MainMemory[Bộ nhớ chính] <--> Bus
    IO[Input/Output] <--> Bus
    subgraph Bus [Bus hệ thống]
        direction TB
        CPU
        MainMemory
        IO
    end
    
```

- **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

Jan2015 Computer Architecture 91

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**

Jan2015 Computer Architecture 92

NKK-HUST

Các thành phần cơ bản của CPU

Jan2015 Computer Architecture 93

- **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

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)**

Jan2015 Computer Architecture 94

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

Jan2015 Computer Architecture 95

NKK-HUST

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

Jan2015 Computer Architecture 96

NKK-HUST

Thiết bị lưu trữ (Storage Devices)

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

Jan2015 Computer Architecture 97

NKK-HUST

3. Hệ thống vào-ra

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

Jan2015 Computer Architecture 98

NKK-HUST

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 Devives)

Jan2015 Computer Architecture 99

NKK-HUST

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

Jan2015 Computer Architecture 100

NKK-HUST

3.2. Hoạt động cơ bản của máy tính

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

Jan2015 Computer Architecture 101

NKK-HUST

1. Thực hiện chương trình

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

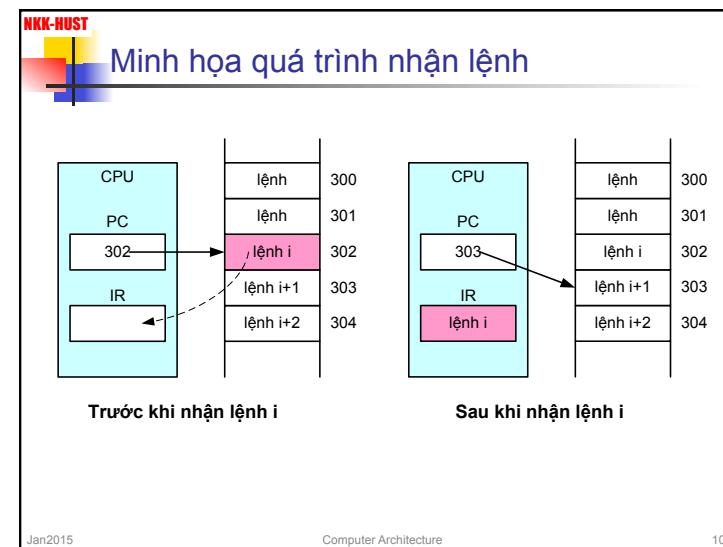
Jan2015 Computer Architecture 102

NKK-HUST

Nhận lệnh

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

Jan2015 Computer Architecture 103



Thực hiện lệnh

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

Jan2015

Computer Architecture

105

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 (qua mô-đun vào-ra) gửi tín hiệu ngắt đến CPU để yêu cầu trao đổi dữ liệu

Jan2015

Computer Architecture

106

Hoạt động với ngắt từ bên ngoài

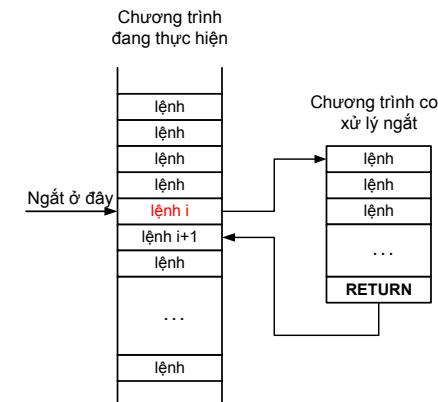
- Sau khi hoàn thành mỗi lệnh, bộ xử lý kiểm tra tín hiệu ngắt
- Nếu không có ngắt, bộ xử lý nhận lệnh tiếp theo của chương trình hiện tại
- Nếu có tín hiệu ngắt:
 - Tạm dừng (suspend) chương trình đang thực hiện
 - Cắt ngữ cảnh (các thông tin liên quan đến chương trình bị ngắt)
 - Thiết lập bộ đếm chương 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

Jan2015

Computer Architecture

107

Hoạt động ngắt (tiếp)



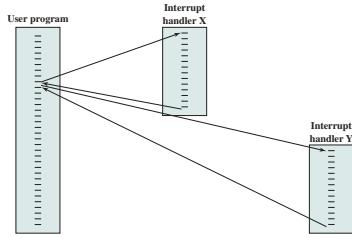
Jan2015

Computer Architecture

108

Xử lý với nhiều tín hiệu yêu cầu ngắt

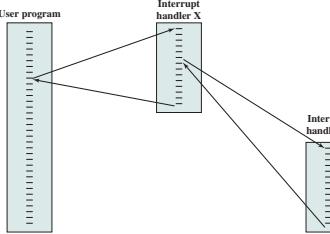
- Xử lý ngắt tuần tự
 - Khi một ngắt đang được thực hiện, các ngắt khác bị cấm (disabled interrupt)
 - Bộ xử lý sẽ bỏ qua các yêu cầu ngắt tiếp theo
 - Các yêu cầu ngắt tiếp theo vẫn đang đợi và được kiểm tra sau khi ngắt hiện tại được xử lý xong
 - Các ngắt được thực hiện tuần tự



Jan2015 Computer Architecture 109

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



Jan2015 Computer Architecture 110

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).

Jan2015 Computer Architecture 111

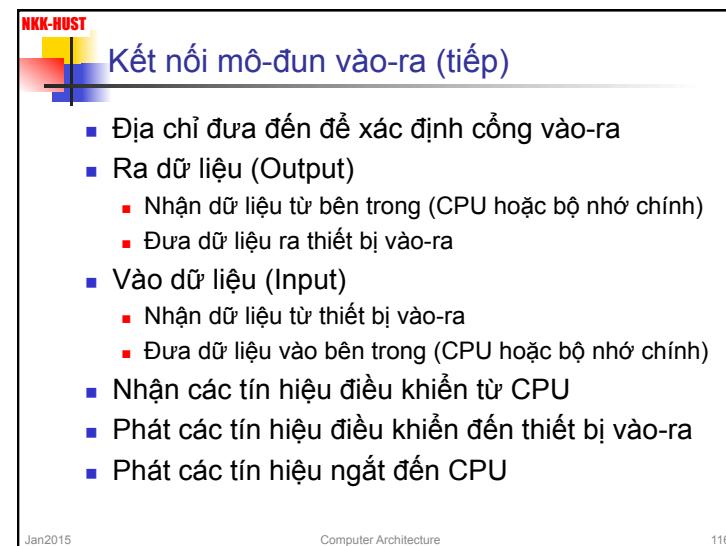
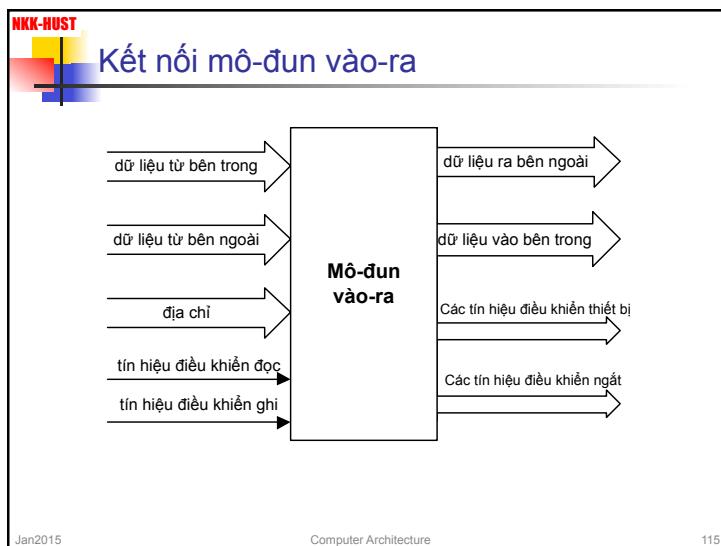
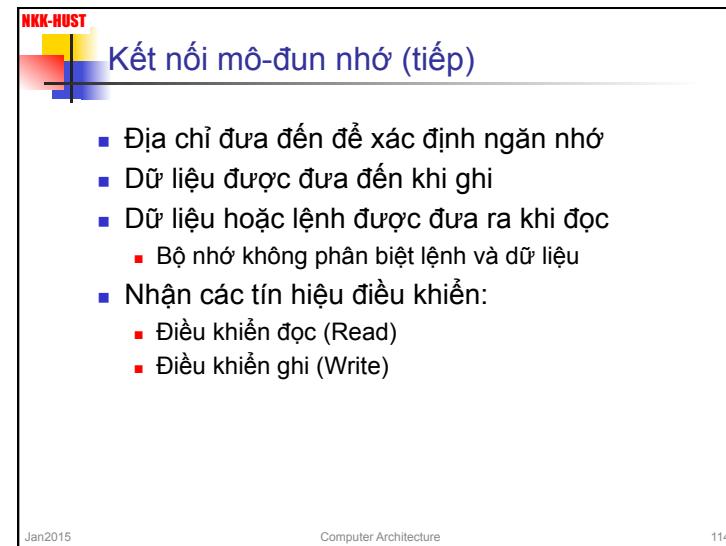
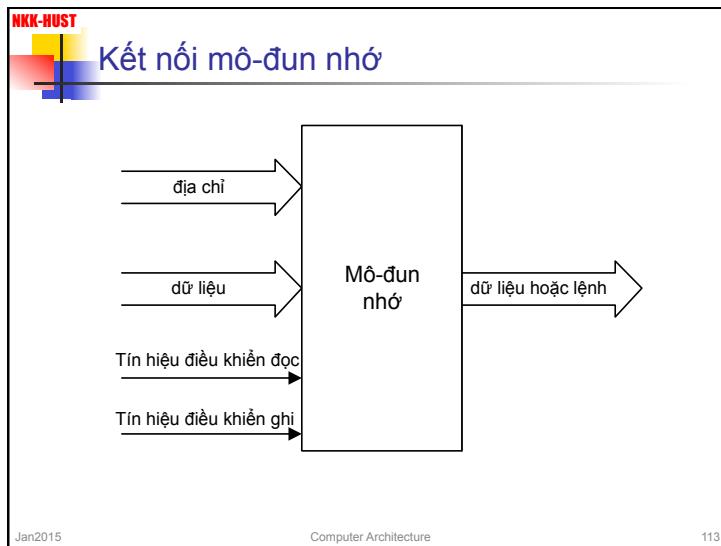
3.3. Bus máy tính

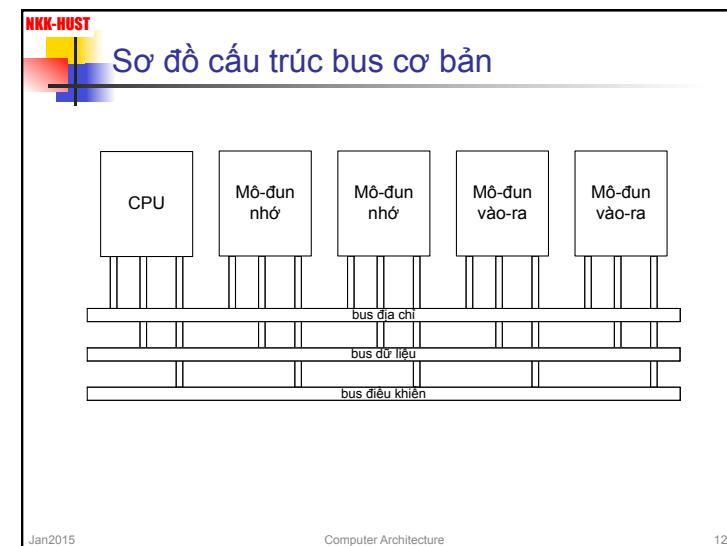
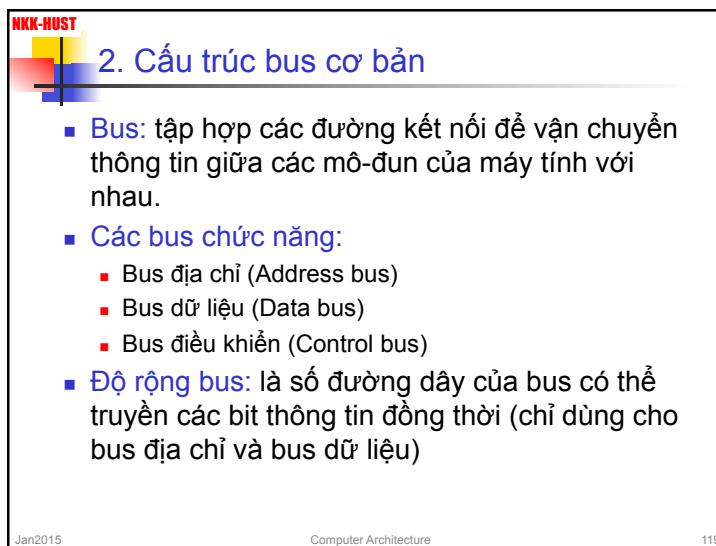
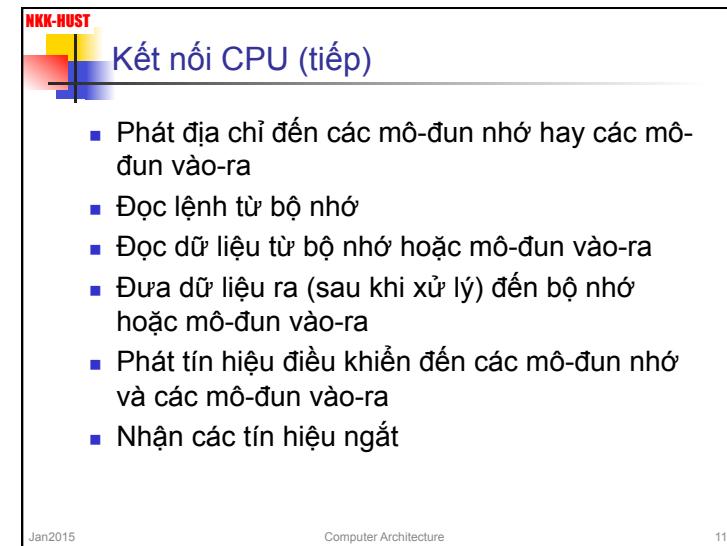
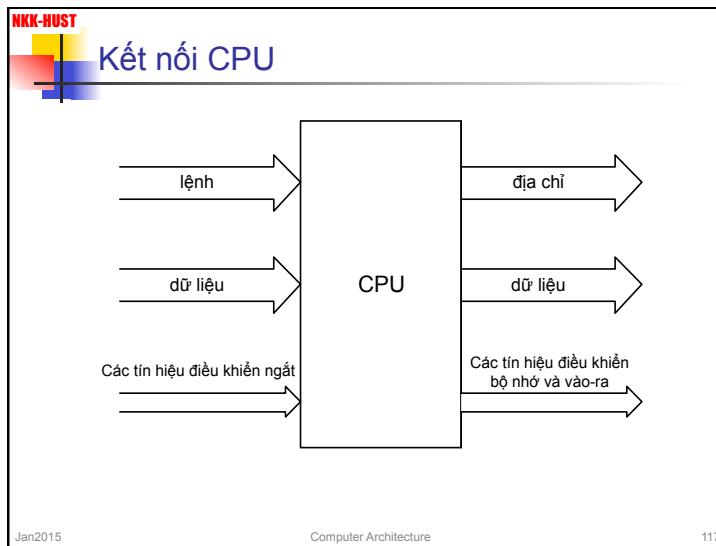
1. Luồng thông tin trong máy tính

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

→ cần được kết nối với nhau

Jan2015 Computer Architecture 112





Bus địa chỉ

- Chức năng: vận chuyển địa chỉ để xác định vị trí ngăn nhớ hay cổng vào-ra
- Độ rộng bus địa chỉ:
 - N bit: $A_{N-1}, A_{N-2}, \dots, A_2, A_1, A_0$
→ Số lượng địa chỉ tối đa được sử dụng là: 2^N địa chỉ (gọi là không gian địa chỉ)
 - Địa chỉ nhỏ nhất: 00 ... 000₍₂₎
 - Địa chỉ lớn nhất: 11 ... 111₍₂₎
- Ví dụ:
 - Máy tính sử dụng bus địa chỉ 32-bit ($A_{31}-A_0$), bộ nhớ chính được đánh địa chỉ cho từng byte
→ Có khả năng đánh địa chỉ cho 2^{32} bytes nhớ = 4GiB

Jan2015

Computer Architecture

121

Bus dữ liệu

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

Jan2015

Computer Architecture

122

Bus điều khiển

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

Jan2015

Computer Architecture

123

Một số tín hiệu điều khiển hiển 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.

Jan2015

Computer Architecture

124

Một số tín hiệu điều khiển hiển thị (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.

Jan2015

Computer Architecture

125

Một số tín hiệu điều khiển hiển thị (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.

Jan2015

Computer Architecture

126

3. Phân cấp bus

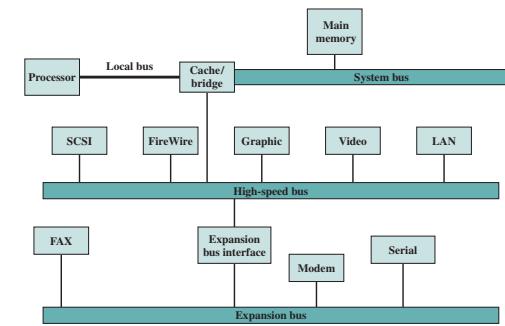
- Đơn bus: Tất cả các mô-đun kết nối vào bus chung
 - Bus chỉ phục vụ được một yêu cầu trao đổi dữ liệu tại một thời điểm → độ trễ lớn
 - Bus phải có tốc độ bằng tốc độ bus của mô-đun nhanh nhất trong hệ thống
- Đa bus: Phân cấp thành nhiều bus cho các mô-đun khác nhau và có tốc độ khác nhau
 - Bus của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

Jan2015

Computer Architecture

127

Phân cấp bus



Jan2015

Computer Architecture

128

NKK-HUST

4. Kết nối điểm-điểm

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

The diagram illustrates two types of point-to-point connections:

- Kết nối QPI:** Shows four cores (Core A, Core B, Core C, Core D) connected via a QPI bus. Each core is also connected to an I/O Hub and memory. The QPI bus connects all cores in a fully meshed manner.
- Kết nối PCIe:** Shows a central Chipset connected to multiple PCIe lanes. These lanes connect to various components like Memory, Legacy I/O, and PCI Express devices. A PCIe switch is also shown connecting multiple PCIe lanes.

Computer Architecture 129

NKK-HUST

Một số bus điển hình trong máy tính

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

Computer Architecture 130

NKK-HUST

Ví dụ bus trong máy tính Intel

The diagram shows the Intel H67 motherboard with its internal bus architecture and key components:

- Processor:** 4th Generation Intel Core Processors, supporting up to 3.0 GHz.
- Graphics:** Has integrated graphics and supports discrete graphics via a mini PCIe slot.
- Memory:** Supports up to 16GB DDR3L RAM.
- Storage:** Supports up to 6x SATA ports (SATA 3.0 and SATA 6.0 Gb/s).
- Networking:** Intel Dual Band Wireless-AC 7265 and Intel PRO/1000 MT Desktop.
- Connectivity:** Includes multiple USB ports (2x USB 3.0, 2x USB 2.0), a LAN port, and a mini PCIe slot.
- Power:** Features Intel Smart Connect Technology, Intel Rapid Start Technology, Intel Auto Shift Technology, and Intel Power Protection Technology.

Computer Architecture 131

NKK-HUST

Hết chương 3

Computer Architecture 132

NKK-HUST

Kiến trúc máy tính

Chương 4

SỐ HỌC MÁY TÍNH

Nguyễn Kim Khánh
Trường Đại học Bách khoa Hà Nội

Jan2015 Computer Architecture 133

NKK-HUST

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

Jan2015 Computer Architecture 134

NKK-HUST

Nội dung chương 4

- 4.1. Biểu diễn số nguyên
- 4.2. Phép cộng và phép trừ số nguyên
- 4.3. Phép nhân và phép chia số nguyên
- 4.4. Số dấu phẩy động

Jan2015 Computer Architecture 135

NKK-HUST

4.1. Biểu diễn số nguyên

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

Jan2015 Computer Architecture 136

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

Jan2015

Computer Architecture

137

NKK-HUST Ví dụ 1

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

$$A = 41 ; B = 150$$

Giải:

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

$$41 = 0010\ 1001$$

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

$$150 = 1001\ 0110$$

Jan2015

Computer Architecture

138

NKK-HUST Ví dụ 2

- Cho các số nguyên không dấu M, N được biểu diễn bằng 8-bit như sau:

- M = 0001 0010
- N = 1011 1001

Xác định giá trị của chúng ?

Giải:

- M = 0001 0010 = $2^4 + 2^1 = 16 + 2 = 18$
- N = 1011 1001 = $2^7 + 2^5 + 2^4 + 2^3 + 2^0$
 $= 128 + 32 + 16 + 8 + 1 = 185$

Jan2015

Computer Architecture

139

NKK-HUST Với n = 8 bit

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

Chú ý:

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

$$\begin{array}{r} 1\ 0000\ 0000 \\ + \underline{0000\ 0011} \end{array}$$

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

Jan2015

Computer Architecture

140

NKK-HUST

Trục số học với $n = 8$ bit

Trục số học:

Trục số học máy tính:

Jan2015 Computer Architecture 141

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 $2^{64} - 1$

Jan2015 Computer Architecture 142

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

Jan2015 Computer Architecture 143

NKK-HUST

Ví dụ

Với $n = 8$ bit, cho A = 0010 0101

- Số bù một của A được tính như sau:

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

→ đảo các bit của A
- Số bù hai của A được tính như sau:

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

→ thực hiện khó khăn

Jan2015 Computer Architecture 144

NKK-HUST

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 = $\begin{array}{r} + \ 1 \\ 1101\ 1011 \end{array}$
- Nhận xét:

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

(bỏ qua bit nhớ ra ngoài)

\rightarrow Số bù hai của A = $-A$

Jan2015

Computer Architecture

145

NKK-HUST

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 $a_{n-1} = 1$

Jan2015

Computer Architecture

146

NKK-HUST

Ví dụ

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

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

Giải:

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

Ta có: $+80 = 0101\ 0000$

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

Vậy: $B = -80 = 1011\ 0000$

Jan2015

Computer Architecture

147

NKK-HUST

Xác định giá trị của số dương

- Dạng tổng quát của số dương:
$$0a_{n-2}\dots a_2a_1a_0$$
- Giá trị của số dương:
$$A = \sum_{i=0}^{n-2} a_i 2^i$$
- Dải biểu diễn cho số dương: $[0, (2^{n-1} - 1)]$

Jan2015

Computer Architecture

148

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: $[-1, -2^{n-1}]$

Jan2015 Computer Architecture 149

Công thức xác định giá trị số âm

$$\begin{aligned} A &= 1 a_{n-2} a_{n-3} \dots a_2 a_1 a_0 \\ -A &= 0 \overline{a_{n-2}} \overline{a_{n-3}} \dots \overline{a_2} \overline{a_1} \overline{a_0} + 1 \\ &= 11 \dots 111 - a_{n-2} a_{n-3} \dots a_2 a_1 a_0 + 1 \\ &= (2^{n-1} - 1) - \left(\sum_{i=0}^{n-2} a_i 2^i \right) + 1 \\ A &= -2^{n-1} + \sum_{i=0}^{n-2} a_i 2^i \end{aligned}$$

Jan2015 Computer Architecture 150

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

Jan2015 Computer Architecture 151

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 = 1001 1011

Giải:

 - P = 0110 0010 = $64+32+2 = +98$
 - Q = 1101 1011 = $-128+64+16+8+2+1 = -37$

Jan2015 Computer Architecture 152

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

Jan2015 Computer Architecture 153

NKK-HUST

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

- Trục số học:

- Trục số học máy tính:

Jan2015 Computer Architecture 154

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 $2^{63} - 1$

Jan2015 Computer Architecture 155

NKK-HUST

Mở rộng bit cho số nguyên

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

+19 =	0001 0011	(8bit)
+19 =	0000 0000 0001 0011	(16bit)

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

- 19 =	1110 1101	(8bit)
- 19 =	1111 1111 1110 1101	(16bit)

 → thêm các bit 1 vào bên trái

Jan2015 Computer Architecture 156

NKK-HUST

4.2. Thực hiện phép cộng/trừ với số nguyên

1. Phép cộng số nguyên không dấu

Bộ cộng n-bit

Jan2015 Computer Architecture 157

NKK-HUST

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: $\text{tổng} > (2^n - 1)$

Jan2015 Computer Architecture 158

NKK-HUST

Ví dụ cộng số nguyên không dấu

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

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

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

Jan2015 Computer Architecture 159

NKK-HUST

2. Phép đảo dấu

- Ta có:

$$\begin{array}{rcl} + 37 & = & 0010\ 0101 \\ \text{bù một} & = & 1101\ 1010 \\ + & & \hline 1 \\ \text{bù hai} & = & 1101\ 1011 = -37 \end{array}$$
- Lấy bù hai của số âm:

$$\begin{array}{rcl} -37 & = & 1101\ 1011 \\ \text{bù một} & = & 0010\ 0100 \\ + & & \hline 1 \\ \text{bù hai} & = & 0010\ 0101 = +37 \end{array}$$
- Kết luận: **Phép đảo dấu số nguyên trong máy tính thực chất là lấy bù hai**

Jan2015 Computer Architecture 160

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

Jan2015

Computer Architecture

161

Ví dụ cộng số nguyên có dấu không tràn

- $(+70) = 0100\ 0110$
 $+ (+42) = 0010\ 1010$
 $+ 112 \quad 0111\ 0000 = +112$
- $(+97) = 0110\ 0001$
 $+ (-52) = 1100\ 1100$ (+52=0011 0100)
 $+ 45 \quad 0010\ 1101 = +45$
- $(-90) = 1010\ 0110$ (+90=0101 1010)
 $+ (+36) = 0010\ 0100$
 $- 54 \quad 1100\ 1010 = -54$
- $(-74) = 1011\ 0110$ (+74=0100 1010)
 $+ (-30) = 1110\ 0010$ (+30=0001 1110)
 $- 104 \quad 1001\ 1000 = -104$

Jan2015

Computer Architecture

162

Ví dụ cộng số nguyên có dấu bị tràn

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

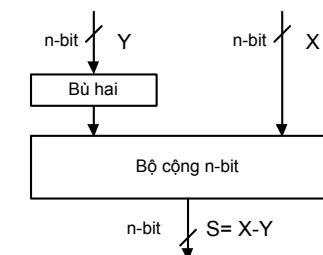
Jan2015

Computer Architecture

163

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



Jan2015

Computer Architecture

164

NKK-HUST

4.3. Phép nhân và phép chia số nguyên

1. Nhân số nguyên không dấu

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

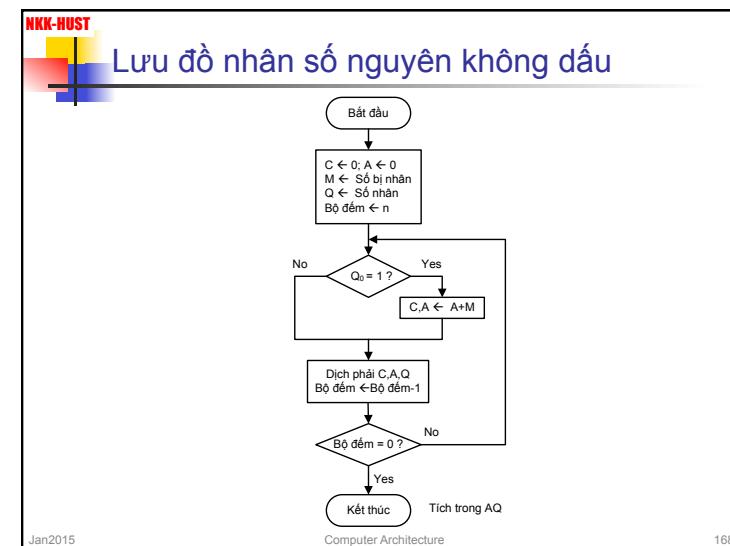
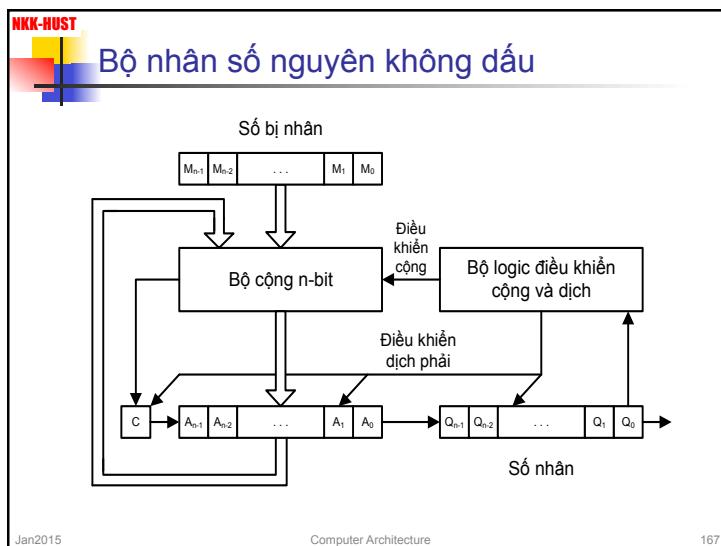
Jan2015 Computer Architecture 165

NKK-HUST

Nhân số nguyên không dấu (tiếp)

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

Jan2015 Computer Architecture 166



NKK-HUST

Ví dụ nhân số nguyên không dấu

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

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

Jan2015

Computer Architecture

169

NKK-HUST

2. Nhân số nguyên có dấu

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

Jan2015

Computer Architecture

170

NKK-HUST

Sử dụng thuật giải nhân không dấu

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

Jan2015

Computer Architecture

171

NKK-HUST

Thuật giải Booth (tham khảo sách COA)

```

graph TD
    START([START]) --> Init[A ← 0, Q-1 ← 0  
M ← Multiplicand  
Q ← Multiplier  
Count ← n]
    Init --> Cond{Q0, Q-1}
    Cond -- 10 --> Sub[A ← A - M]
    Cond -- 01 --> Add[A ← A + M]
    Cond -- 11 --> Shift[Arithmetic shift Right: A, Q, Q-1  
Count ← Count - 1]
    Cond -- 00 --> Shift
    Shift --> Cond
    Sub --> Cond
    Add --> Cond
    Shift --> Cond
    Cond -- No --> End([END])
    Cond -- Yes --> Init
  
```

Jan2015

Computer Architecture

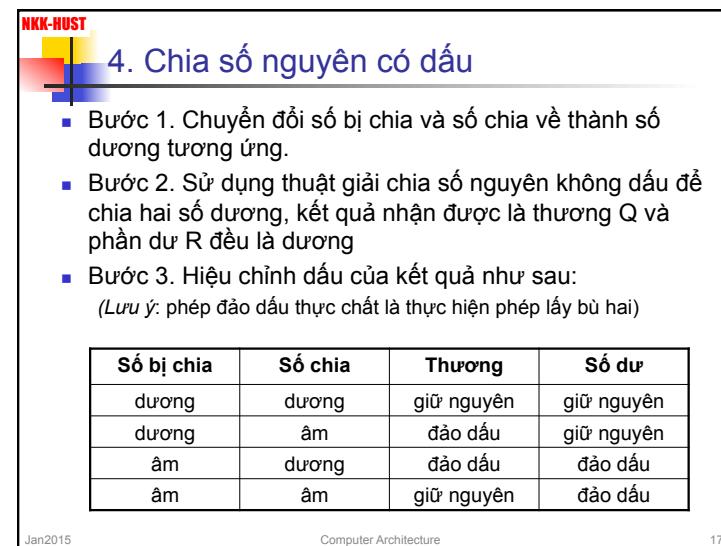
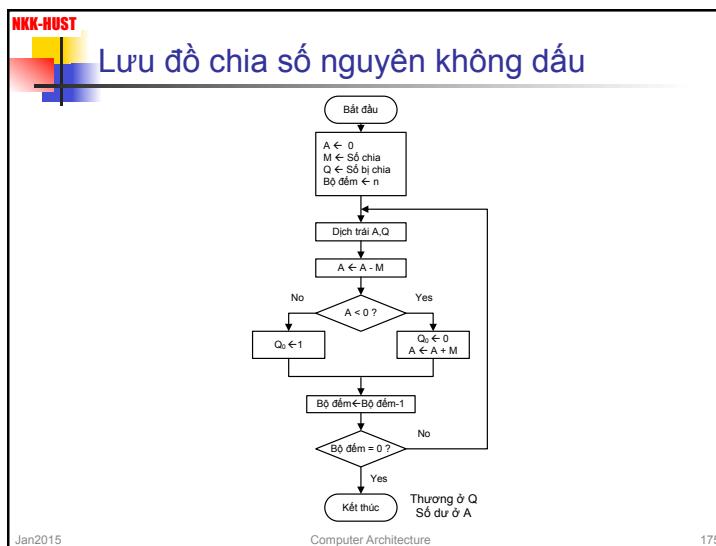
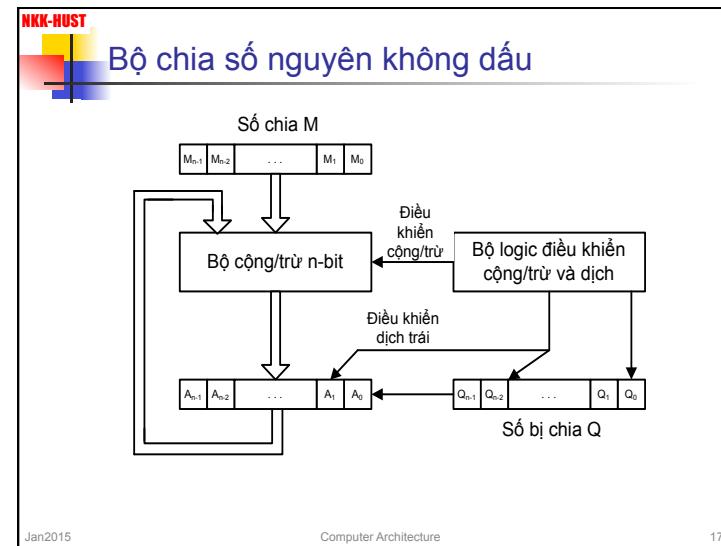
172

NKK-HUST

3. Chia số nguyên không dấu

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

Jan2015 Computer Architecture 173



NKK-HUST

4.4. Số dấu phẩy động

1. Nguyên tắc chung

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

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

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

Jan2015 Computer Architecture 177

NKK-HUST

2. Chuẩn IEEE754-2008

- Cơ số R = 2
- Các dạng:
 - Dạng 32-bit
 - Dạng 64-bit
 - Dạng 128-bit

(a) Binary32 format: Sign (1 bit), Biased exponent (8 bits), Trailing significand field (23 bits). Total 32 bits.

(b) Binary64 format: Sign (1 bit), Biased exponent (11 bits), Trailing significand field (52 bits). Total 64 bits.

(c) Binary128 format: Sign (1 bit), Biased exponent (15 bits), Trailing significand field (112 bits). Total 128 bits.

Jan2015 Computer Architecture 178

NKK-HUST

Dạng 32-bit

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

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

Jan2015 Computer Architecture 179

NKK-HUST

Ví dụ 1

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

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

Jan2015 Computer Architecture 180

Ví dụ 2

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

Giải:

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

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

Jan2015

Computer Architecture

181

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)

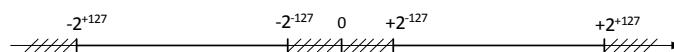
Jan2015

Computer Architecture

182

Dải giá trị biểu diễn

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



Jan2015

Computer Architecture

183

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}

Jan2015

Computer Architecture

184

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}

Jan2015

Computer Architecture

185

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 $E_2 \geq E_1$

Jan2015

Computer Architecture

186

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

Jan2015

Computer Architecture

187

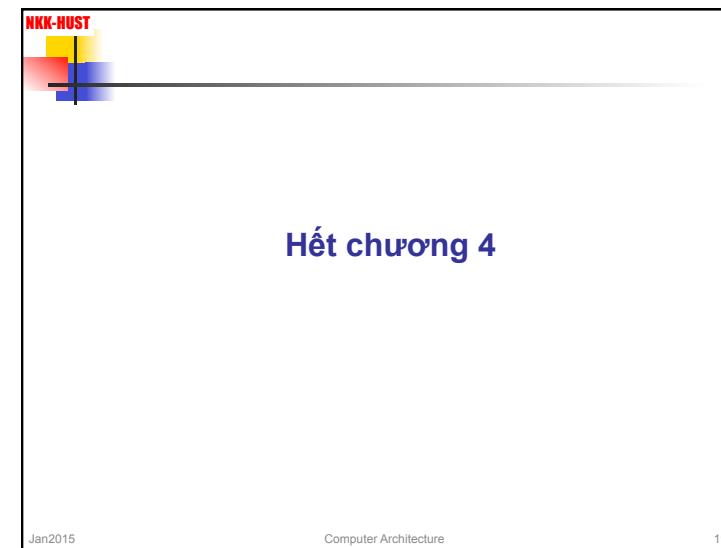
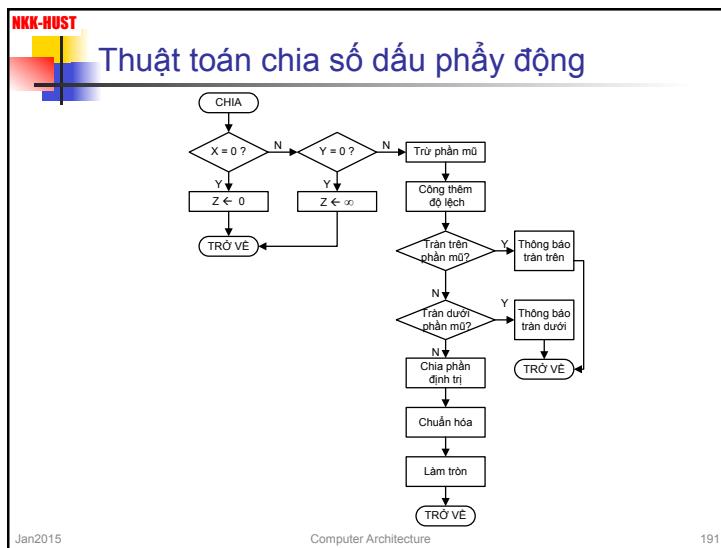
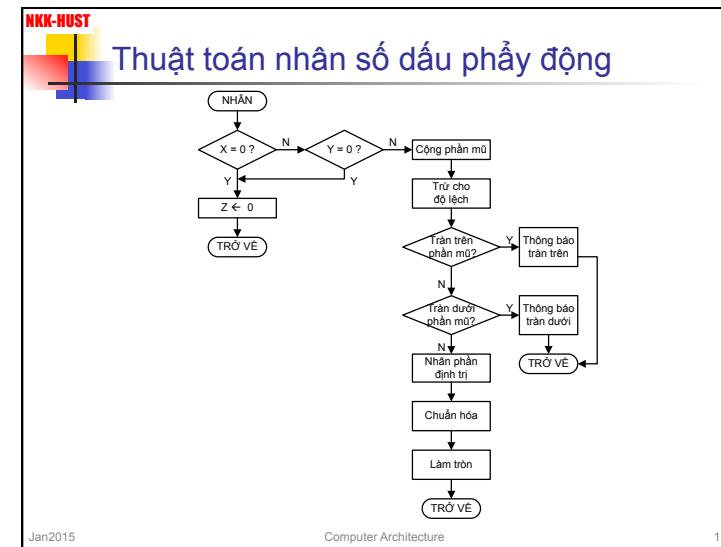
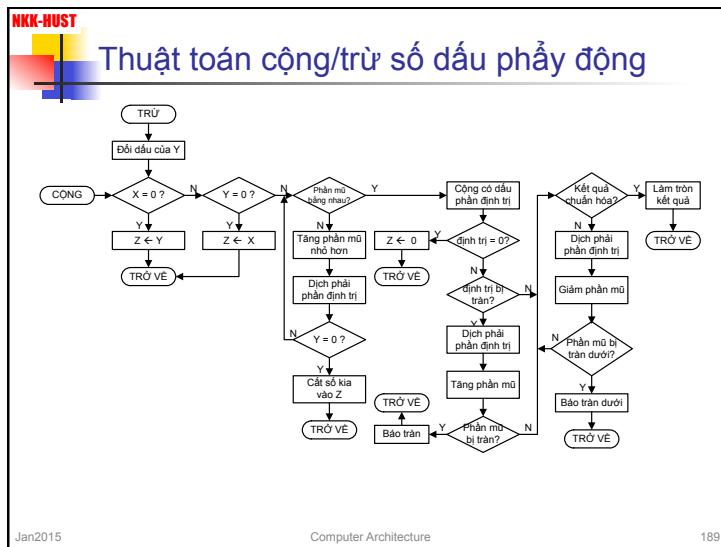
Phép cộng và phép trừ

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

Jan2015

Computer Architecture

188



NKK-HUST

Kiến trúc máy tính

Chương 5 KIẾN TRÚC TẬP LỆNH

Nguyễn Kim Khánh
Trường Đại học Bách khoa Hà Nội

Jan2015 Computer Architecture 193

NKK-HUST

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

Jan2015 Computer Architecture 194

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

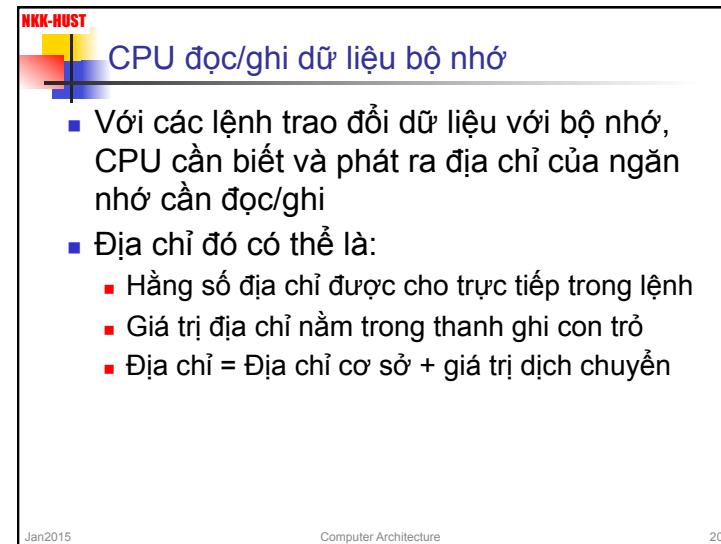
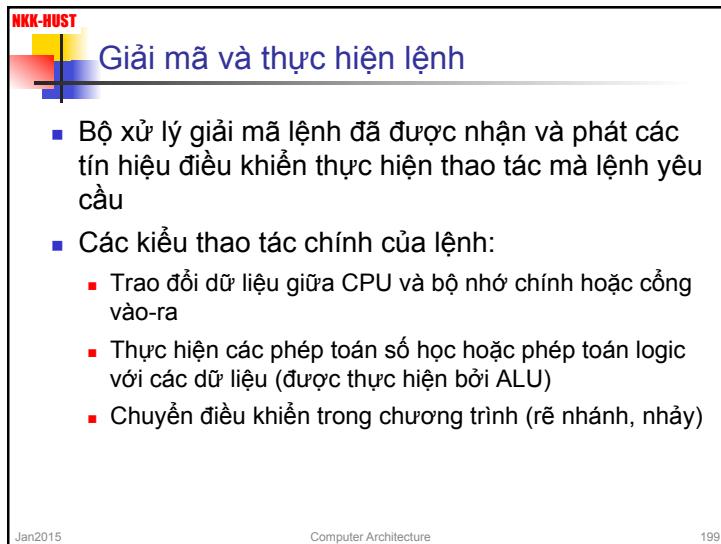
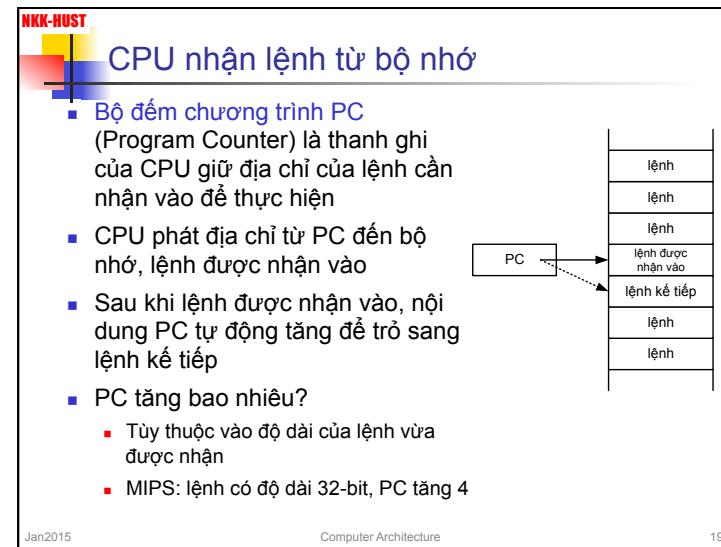
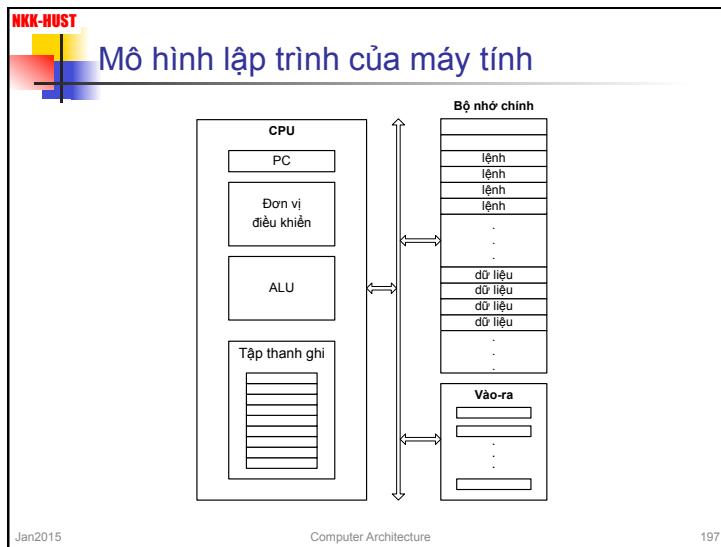
Jan2015 Computer Architecture 195

NKK-HUST

5.1. Giới thiệu chung về kiến trúc tập lệnh

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

Jan2015 Computer Architecture 196



Hằng số địa chỉ

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

Jan2015 Computer Architecture 201

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

Jan2015 Computer Architecture 202

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

Jan2015 Computer Architecture 203

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
- Định ngăn xếp là thông tin nằm ở vị trí trên cùng trong ngăn xếp
- Định ngăn xếp có thể bị thay đổi

Jan2015 Computer Architecture 204

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 trỏ bởi SP
- Khi lấy một thông tin ra khỏi ngăn xếp:
 - Thông tin được đọc từ ngăn nhớ được trỏ bởi SP
 - Tăng nội dung của SP
- Khi ngăn xếp rỗng, SP trở vào đáy

chiều
địa
chi
tăng
dần

SP → đinh ngăn xếp

đáy ngăn xếp

Jan2015 Computer Architecture 205

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)

Jan2015 Computer Architecture 206

NKK-HUST

Ví dụ lưu trữ dữ liệu 32-bit

Số nhi phân	0001 1010 0010 1011 0011 1100 0100 1101
Số Hexa	1A 2B 3C 4D

little-endian	4D 3C 2B 1A 4003	big-endian	1A 2B 3C 4D 4000 4001 4002 4003
---------------	------------------------------	------------	--

Jan2015 Computer Architecture 207

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 gọi nhô dạng text, đó chính là các lệnh của hợp ngữ (assembly language)

Jan2015 Computer Architecture 208

Dạng lệnh hợp ngữ

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

Jan2015 Computer Architecture 209

Các thành phần của lệnh máy

Mã thao tác	Địa chỉ toán hạng
-------------	-------------------

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

Jan2015 Computer Architecture 210

Số lượng địa chỉ toán hạng trong lệnh

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

Jan2015 Computer Architecture 211

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

Jan2015 Computer Architecture 212

NKK-HUST

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

Jan2015 Computer Architecture 213

NKK-HUST

Kiến trúc tập lệnh MIPS

- MIPS viết tắt cho: **M**icroprocessor **W**ithout **I**nterlocked **P**ipeline **S**tages
- Được phát triển bởi John Hennessy và các đồng nghiệp ở đại học Stanford (1984)
- Được thương mại hóa bởi MIPS Technologies
- Năm 2013 công ty này được bán cho Imagination Technologies (imgtec.com)
- Là kiến trúc RISC điển hình, dễ học
- Được sử dụng trong nhiều sản phẩm thực tế
- Các phần tiếp theo trong chương này sẽ nghiên cứu kiến trúc tập lệnh MIPS 32-bit
 - Tài liệu: *MIPS Reference Data Sheet* và *Chapter 2 – COD*

Jan2015 Computer Architecture 214

NKK-HUST

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
$$\text{add } a, b, c \quad \# \quad a = b + c$$
- Hầu hết các lệnh số học/logic có dạng trên
- Các lệnh số học sử dụng toán hạng thanh ghi hoặc hằng số

Jan2015 Computer Architecture 215

NKK-HUST

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”

Jan2015 Computer Architecture 216

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

Jan2015

Computer Architecture

217

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)
```

Jan2015

Computer Architecture

218

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 công 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*)

Jan2015

Computer Architecture

219

Địa chỉ byte nhớ và word nhớ

Dữ liệu hoặc lệnh	Địa chỉ byte (theo Hexa)	Dữ liệu hoặc lệnh	Địa chỉ word (theo Hexa)
byte (8-bit)	0x0000 0000	word (32-bit)	0x0000 0000
byte	0x0000 0001	word	0x0000 0004
byte	0x0000 0002	word	0x0000 0008
byte	0x0000 0003	word	0x0000 000C
byte	0x0000 0004	word	0x0000 0010
byte	0x0000 0005	word	0x0000 0014
byte	0x0000 0006	word	0x0000 0018
byte	0x0000 0007	.	.
.	.	.	.
byte	0xFFFF FFFB	word	0xFFFF FFF4
byte	0xFFFF FFFC	word	0xFFFF FFF8
byte	0xFFFF FFFD	word	0xFFFF FFFC
byte	0xFFFF FFEE	.	.
byte	0xFFFF FFFF	2 ³² bytes	2 ³⁰ words

Jan2015

Computer Architecture

220

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

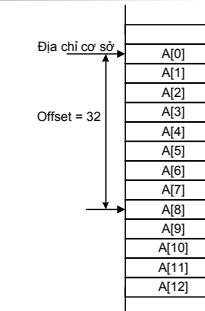
Jan2015

Computer Architecture

221

Ví dụ toán hạng bộ nhớ

- Mã C:
 $g = h + A[8];$
 - Cho g ở \$s1, h ở \$s2
 - \$s3 chứa địa chỉ cơ sở của mảng A



Jan2015

Computer Architecture

222

Ví dụ toán hạng bộ nhớ

- Mã C:
 $g = h + A[8];$
 - Cho g ở \$s1, h ở \$s2
 - \$s3 chứa địa chỉ cơ sở của mảng A
- Mã hợp ngữ MIPS:
 $# Chỉ số 8, do đó offset = 32$
 $lw \ t0, \ 32($s3) \ # \ t0 = A[8]$
 $add \ $s1, \ /$s2, \ /$t0 \ # \ g = h+A[8]$
 - offset
 - base register

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

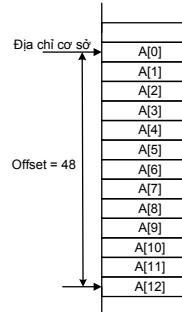
Jan2015

Computer Architecture

223

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



Jan2015

Computer Architecture

224

NKK-HUST

Ví dụ toán hạng bộ nhớ (tiếp)

- Mã C:
 $A[12] = h + A[8];$
 - $h \in \$s2$
 - $\$s3$ 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]
```

Jan2015 Computer Architecture 225

NKK-HUST

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

Jan2015 Computer Architecture 226

NKK-HUST

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
```

Jan2015 Computer Architecture 227

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

Jan2015 Computer Architecture 228

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

Jan2015

Computer Architecture

229

5.3. Mã máy (Machine code)

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

Jan2015

Computer Architecture

230

Các kiểu lệnh máy của MIPS

Lệnh kiểu R	op	rs	rt	rd	shamt	funct
	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

Lệnh kiểu I	op	rs	rt	imm
	6 bits	5 bits	5 bits	16 bits

Lệnh kiểu J	op	address
	6 bits	26 bits

Jan2015

Computer Architecture

231

Lệnh kiểu R (Registers)

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

- 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

Jan2015

Computer Architecture

232

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
<i>add \$t0, \$s1, \$s2</i>					
0	\$s1	\$s2	\$t0	0	add
0	17	18	8	0	32
000000	10001	10010	01000	00000	100000
(0x02324020)					
<i>sub \$s0, \$t3, \$t5</i>					
0	\$t3	\$t5	\$s0	0	sub
0	11	13	16	0	34
000000	01011	01101	10000	00000	100010
(0x016D8022)					

Jan2015 Computer Architecture 233

NKK-HUST

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)			
<ul style="list-style-type: none"> ▪ 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)			

Jan2015 Computer Architecture 234

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

Jan2015 Computer Architecture 235

NKK-HUST

Ví dụ mã máy của lệnh addi

op	rs	rt	imm
6 bits	5 bits	5 bits	16 bits
<i>addi \$s0, \$s1, 5</i>			
8	\$s1	\$s0	5
8	17	16	5
001000	10001	10000	0000 0000 0000 0101
(0x22300005)			
<i>addi \$t1, \$s2, -12</i>			
8	\$s2	\$t1	-12
8	18	9	-12
001000	10010	01001	1111 1111 1111 0100
(0x2249FFF4)			

Jan2015 Computer Architecture 236

NKK-HUST

Ví dụ mã máy của lệnh load và lệnh store

op	rs	rt	imm
6 bits	5 bits	5 bits	16 bits
<i>lw \$t0, 32(\$s3)</i>			
35	\$s3	\$t0	32
35	19	8	32
100011	10011	01000	0000 0000 0010 0000
(0x8E680020)			
<i>sw \$s1, 4(\$t1)</i>			
43	\$t1	\$s1	4
43	9	17	4
101011	01001	10001	0000 0000 0000 0100
(0xAD310004)			

Jan2015 Computer Architecture 237

NKK-HUST

Lệnh kiểu J (Jump)

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

op	address
6 bits	26 bits

Jan2015 Computer Architecture 238

NKK-HUST

5.4. Cơ bản về lập trình hợp ngữ

- Các lệnh logic
- Nạp hằng số vào thanh ghi
- Tạo các cấu trúc điều khiển
- Lập trình mảng dữ liệu
- Chương trình con
- Dữ liệu ký tự
- Lệnh nhân và lệnh chia
- Các lệnh với số dấu phẩy động

Jan2015 Computer Architecture 239

NKK-HUST

1. Các lệnh logic

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

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

Jan2015 Computer Architecture 240

Ví dụ lệnh logic kiểu R	
Nội dung các thanh ghi nguồn	
\$s1	0100 0110 1010 0001 1100 0000 1011 0111
\$s2	1111 1111 1111 1111 0000 0000 0000 0000
Mã hợp ngữ	Kết quả thanh ghi đích
and \$s3, \$s1, \$s2	\$s3
or \$s4, \$s1, \$s2	\$s4
xor \$s5, \$s1, \$s2	\$s5
nor \$s6, \$s1, \$s2	\$s6

NKK-HUST

Ví dụ lệnh logic kiểu R

Nội dung các thanh ghi nguồn

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

Mã hợp ngữ

Kết quả thanh ghi đích

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

\$s3	0100	0110	1010	0001	0000	0000	0000	0000
\$s4	1111	1111	1111	1111	1100	0000	1011	0111

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

\$s4	1111	1111	1111	1111	1100	0000	1011	0111
\$s5	1011	1001	0101	1110	1100	0000	1011	0111

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

\$s5	1011	1001	0101	1110	1100	0000	1011	0111
\$s6	0000	0000	0000	0000	0011	1111	0100	1000

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

Computer Architecture

Jan2015

24

NKK-HUST

Ví dụ lệnh logic kiểu I

Giá trị các toán hạng nguồn

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

← Zero-extended →

Mã hợp ngữ

Kết quả thanh ghi đích

andi \$s2,\$s1,0xF034	\$s2
ori \$s3,\$s1,0xF034	\$s3
xori \$s4,\$s1,0xF034	\$s4

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

NKK-HUST

Ví dụ lệnh logic kiểu I

Giá trị các toán hạng nguồn

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

← Zero-extended →

Mã hợp ngữ

Kết quả thanh ghi đích

andi	\$s2,\$s1,0xFA34	\$s2	0000	0000	0000	0000	0000	0000	0100
ori	\$s3,\$s1,0xFA34	\$s3	0000	0000	0000	0000	1111	1010	1111
xori	\$s4,\$s1,0xFA34	\$s4	0000	0000	0000	0000	1111	1010	1100

Jan2015

Computer Architecture

24

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
 - $a \text{ NOR } b == \text{NOT} (a \text{ OR } b)$
- nor \$t0, \$t1, \$zero # \$t0 = not (\$t1)

Jan2015

Computer Architecture

245

Lệnh logic dịch bit

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

- shamt: chỉ ra dịch bao nhiêu vị trí (shift amount)
- rs: không sử dụng, thiết lập = 00000
- Thanh ghi đích rd nhận giá trị thanh ghi nguồn rt đã được dịch trái hoặc dịch phải, rt không thay đổi nội dung
- **sll** – shift left logical (dịch trái logic)
 - Dịch trái các bit và 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)

Jan2015

Computer Architecture

246

Ví dụ lệnh dịch trái sll

Lệnh hợp ngữ:

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

Mã máy:

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

000000	00000	10000	01010	00100	000000
(0x00105100)					

Ví dụ kết quả thực hiện lệnh:

\$s0	0000	0000	0000	0000	0000	0000	1101	= 13
------	------	------	------	------	------	------	------	------

\$t2	0000	0000	0000	0000	0000	0000	1101	0000	= 208 (13x16)
------	------	------	------	------	------	------	------	------	------------------

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

Jan2015

Computer Architecture

247

Ví dụ lệnh dịch phải srl

Lệnh hợp ngữ:

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

Mã máy:

op	rs	rt	rd	shamt	funct
0	0	17	18	2	2

000000	00000	10001	10010	00010	000010
(0x00119082)					

Ví dụ kết quả thực hiện lệnh:

\$s1	0000	0000	0000	0000	0000	0000	0101	0110	= 86
------	------	------	------	------	------	------	------	------	------

\$s2	0000	0000	0000	0000	0000	0000	0001	0101	= 21 [86/4]
------	------	------	------	------	------	------	------	------	----------------

Jan2015

Computer Architecture

248

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

Jan2015 Computer Architecture 249

NKK-HUST

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

Jan2015 Computer Architecture 250

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

or

0000	0000	0000	0000	0100	0000	0011	1011
------	------	------	------	------	------	------	------

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

\$s0	0010	0001	1010	0000	0400	0000	0011	1011
------	------	------	------	------	------	------	------	------

Jan2015 Computer Architecture 251

NKK-HUST

3. Tạo các cấu trúc điều khiển

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

Jan2015 Computer Architecture 252

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

Jan2015 Computer Architecture 253

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

Jan2015 Computer Architecture 254

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
```

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

Jan2015 Computer Architecture 255

NKK-HUST

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

Jan2015 Computer Architecture 256

Dịch câu lệnh if/else

Mã C:

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

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

Mã hợp ngữ MIPS:

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

Jan2015 Computer Architecture 257

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;
```

Jan2015 Computer Architecture 258

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:
```

Jan2015 Computer Architecture 259

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

Jan2015 Computer Architecture 260

Dịch câu lệnh vòng lặp while

- Mã C:


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

 - i ở \$s3, k ở \$s5
 - địa chỉ cơ sở của mảng A ở \$s6
- Mã hợp ngữ MIPS:


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

Jan2015 Computer Architecture 261

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;
}
```

Jan2015 Computer Architecture 262

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: ...
```

Jan2015 Computer Architecture 263

Khối lệnh cơ sở (basic block)

- Khối lệnh cơ sở là dãy các lệnh với
 - Không có lệnh rẽ nhánh nhúng trong đó (ngoại trừ ở cuối)
 - Không có đích rẽ nhánh tới (ngoại trừ ở vị trí đầu tiên)
- Chương trình dịch xác định khối cơ sở để tối ưu hóa
- Các bộ xử lý tiên tiến có thể tăng tốc độ thực hiện khối cơ sở

Jan2015 Computer Architecture 264

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, $s0, $s1 # nếu ($s1 < $s2)
bne $t0, $zero, L1 # rẽ nhánh đến L1
...
L1:
```

Jan2015 Computer Architecture 265

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**
 - $+4,294,967,295 > +1 \rightarrow \$t0 = 0$

Jan2015 Computer Architecture 266

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;
}
```

Jan2015 Computer Architecture 267

NKK-HUST

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 = 0
        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:
```

Jan2015 Computer Architecture 268

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

Jan2015

Computer Architecture

269

Ví dụ về mảng

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

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

Jan2015

Computer Architecture

270

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
```

Jan2015

Computer Architecture

271

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
```

Jan2015

Computer Architecture

272

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:
```

Jan2015

Computer Architecture

273

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

Jan2015

Computer Architecture

274

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: cát giữ các biến
 - Cần phải cát/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)

Jan2015

Computer Architecture

275

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
- Trả về từ thủ tục: jump register

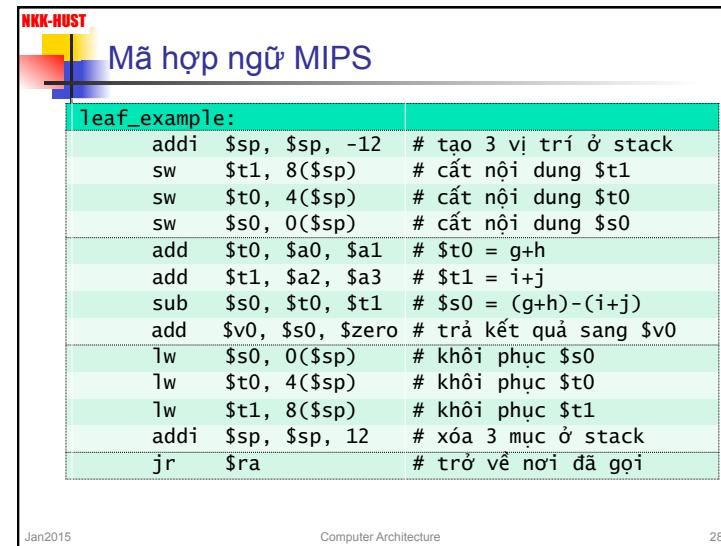
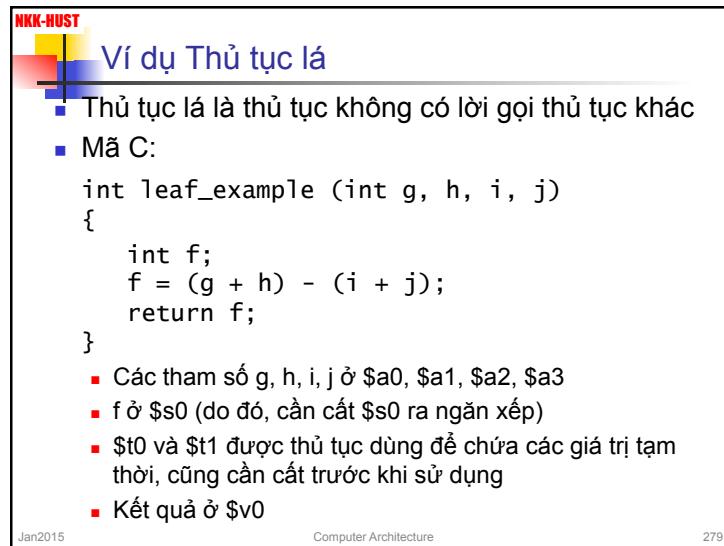
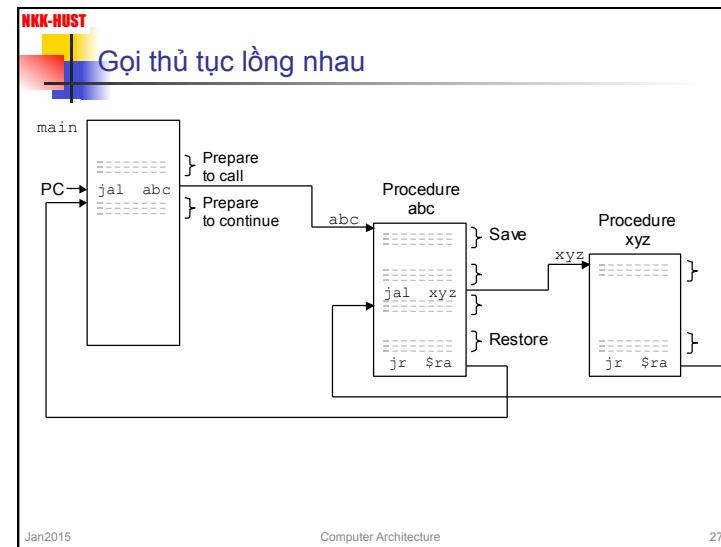
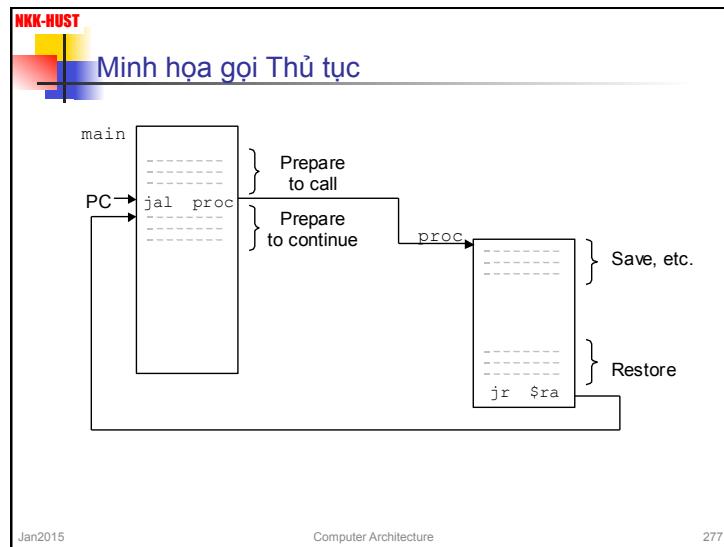

```
jr $ra
```

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

Jan2015

Computer Architecture

276



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

Jan2015

Computer Architecture

281

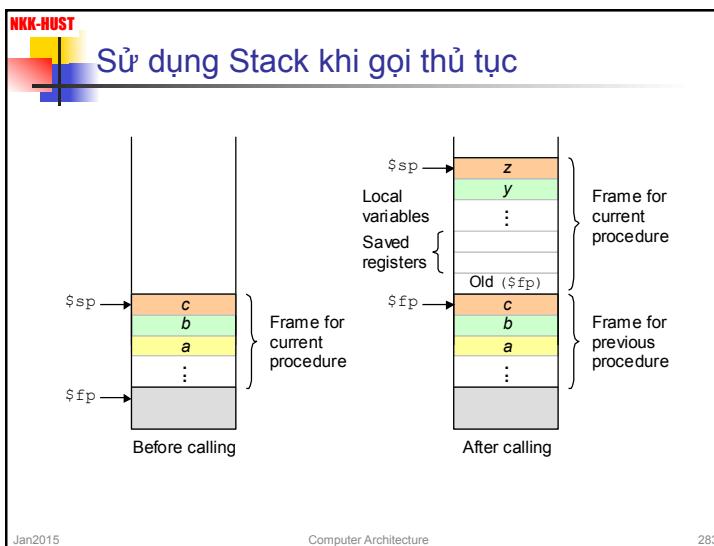
Mã hợp ngữ MIPS

fact:	
addi \$sp, \$sp, -8	# dành stack cho 2 mục
sw \$ra, 4(\$sp)	# cất địa chỉ trả về
sw \$a0, 0(\$sp)	# cất tham số n
slti \$t0, \$a0, 1	# kiểm tra n < 1
beq \$t0, \$zero, L1	
addi \$v0, \$zero, 1	# nếu đúng, kết quả là 1
addi \$sp, \$sp, 8	# lấy 2 mục từ stack
jr \$ra	# và trả về
L1: addi \$a0, \$a0, -1	# nếu không, giảm n
jal fact	# gọi đệ 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

Jan2015

Computer Architecture

282



Jan2015

Computer Architecture

283

- 6. Dữ liệu ký tự**
- Các tập ký tự được mã hóa theo byte
 - ASCII: 128 ký tự
 - 95 ký tự 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

Jan2015

Computer Architecture

284

NKK-HUST

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 dấu thành 32 bits trong rt
- **lbu rt, offset(rs)** **lhu rt, offset(rs)**
 - Mở rộng zero thành 32 bits trong rt
- **sb rt, offset(rs)** **sh rt, offset(rs)**
 - Chỉ lưu byte/halfword bên phải

Jan2015 Computer Architecture 285

NKK-HUST

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



Jan2015                          Computer Architecture                          286


```

NKK-HUST

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
```

Jan2015 Computer Architecture 287

NKK-HUST

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

Jan2015 Computer Architecture 288

8. Các lệnh với số dấu phẩy động (FP)

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

Jan2015

Computer Architecture

289

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

Jan2015

Computer Architecture

290

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

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

- 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 [-2¹⁵, +2¹⁵ - 1]

Jan2015

Computer Architecture

291

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

Jan2015

Computer Architecture

292

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

op	address
6 bits	26 bits

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

Jan2015 Computer Architecture 293

NKK-HUST

Ví dụ mã hóa lệnh j và jr

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

jump target address

31	op	25	0
0 0 0 0 1 0			

j = 2

From PC

x x x x	0 0
---------	-----

Effective target address (32 bits)

31	op	25	rs	20	rt	15	rd	10	sh	5	fn	0
0 0 0 0 0 0	1 1	1 1	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	0 0 1	0 0 0	

ALU instruction Source register Unused Unused Unused jr = 8

Jan2015 Computer Architecture 294

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: ...                  0x8018
```

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		

Jan2015 Computer Architecture 295

NKK-HUST

Rẽ nhánh xa

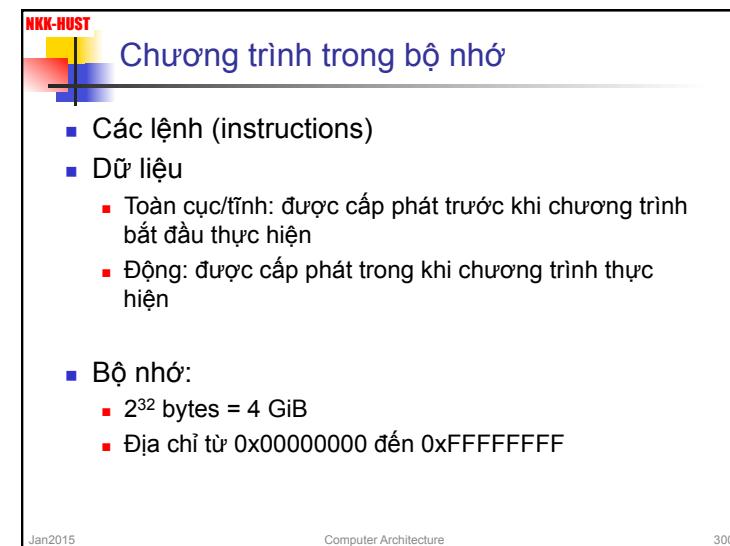
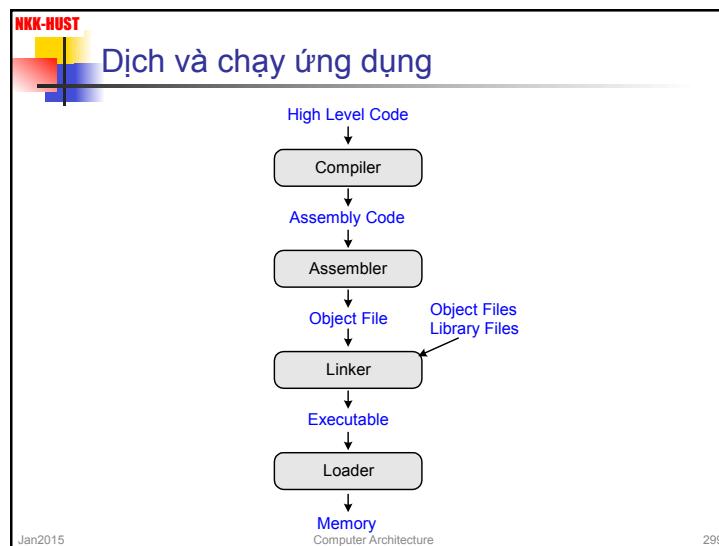
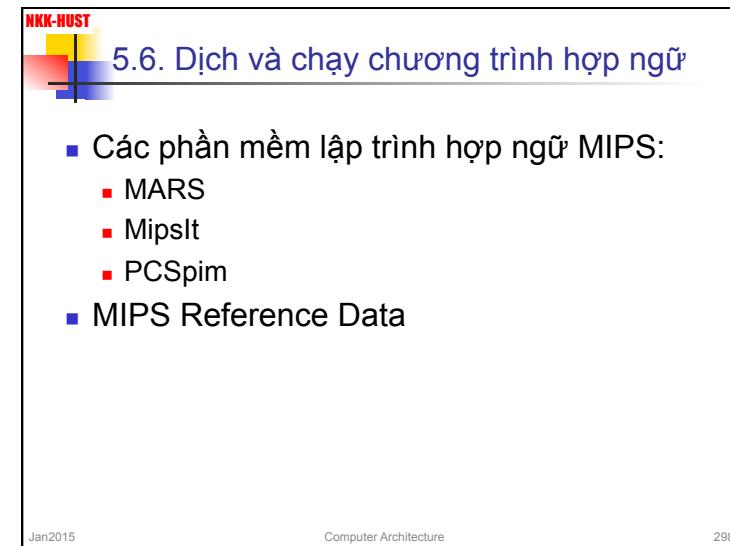
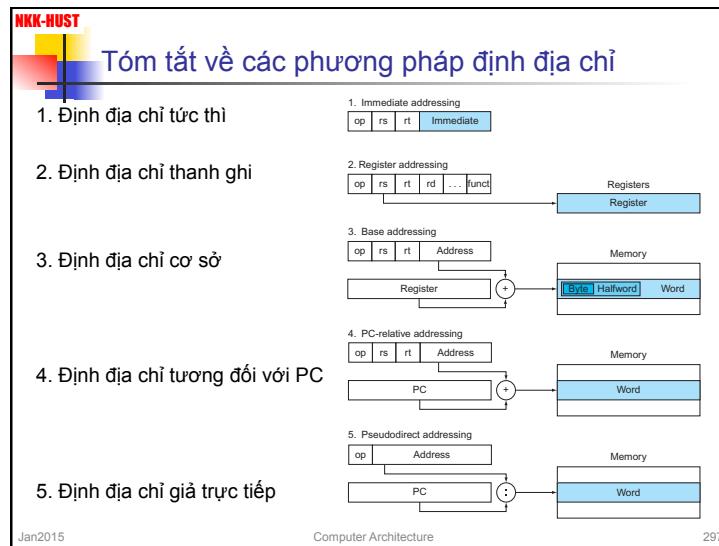
- Nếu đích rẽ nhánh là quá xa để mã hóa với offset 16-bit, assembler sẽ viết lại code
- Ví dụ

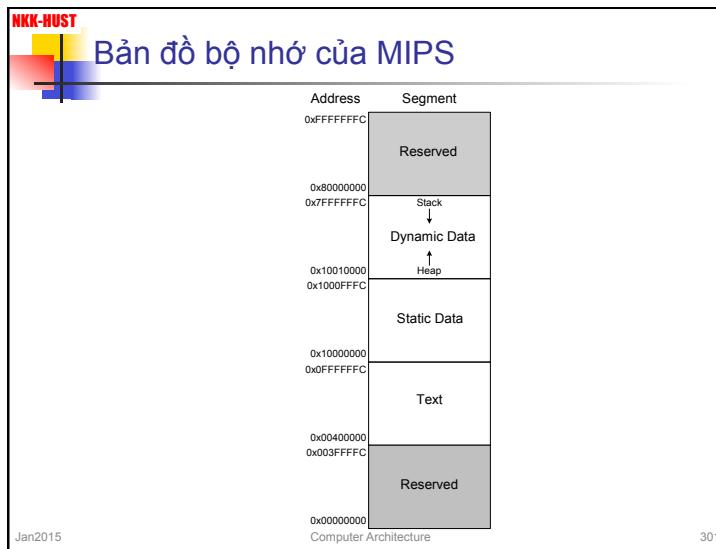
```
beq $s0, $s1, L1          (lệnh kế tiếp)
...
L1:
```

sẽ được thay bằng đoạn lệnh sau:

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

Jan2015 Computer Architecture 296





NKK-HUST

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);
}
```

Jan2015 Computer Architecture 302

NKK-HUST

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
```

Jan2015 Computer Architecture 303

NKK-HUST

Bảng ký hiệu

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

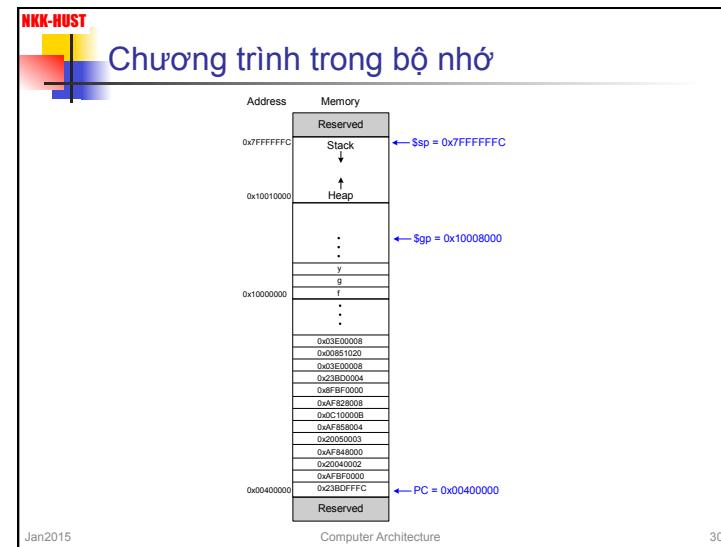
Jan2015 Computer Architecture 304

NKK-HUST

Chương trình thực thi

Executable file header	Text Size	Data Size
	0x34 (52 bytes)	0xC (12 bytes)
Text segment	Address	Instruction
	0x00400000	0x23BDFFFFC
	0x00400004	sw \$ra, 0(\$sp)
	0x00400008	addi \$a0, \$0, 2
	0x0040000C	sw \$a0, 0x8000(\$gp)
	0x00400010	addi \$a1, \$0, 3
	0x00400014	sw \$a1, 0x8004(\$gp)
	0x00400018	jal 0x00400002C
	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

Jan2015 Computer Architecture 305

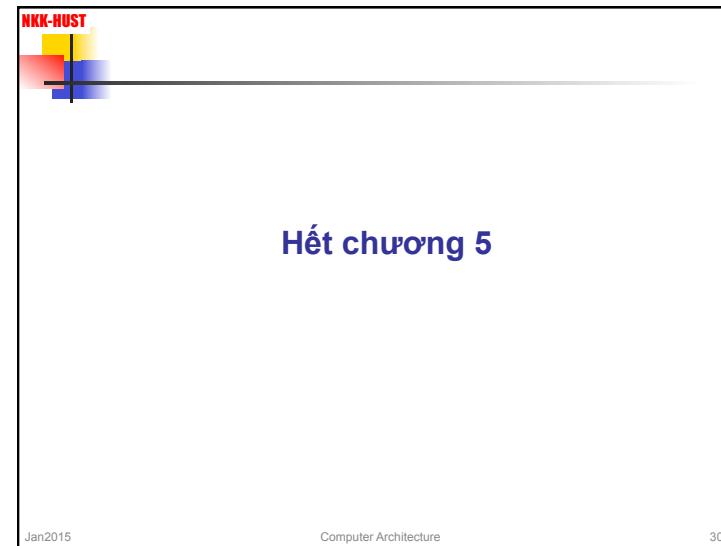


NKK-HUST

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

Jan2015 Computer Architecture 307



NKK-HUST

Kiến trúc máy tính

Chương 6 BỘ XỬ LÝ

Nguyễn Kim Khánh
Trường Đại học Bách khoa Hà Nội

Jan2015 Computer Architecture 309

NKK-HUST

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

Jan2015 Computer Architecture 310

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 đmouseoverong ống lệnh
- 6.4. Ví dụ thiết kế bộ xử lý theo kiến trúc MIPS***

Jan2015 Computer Architecture 311

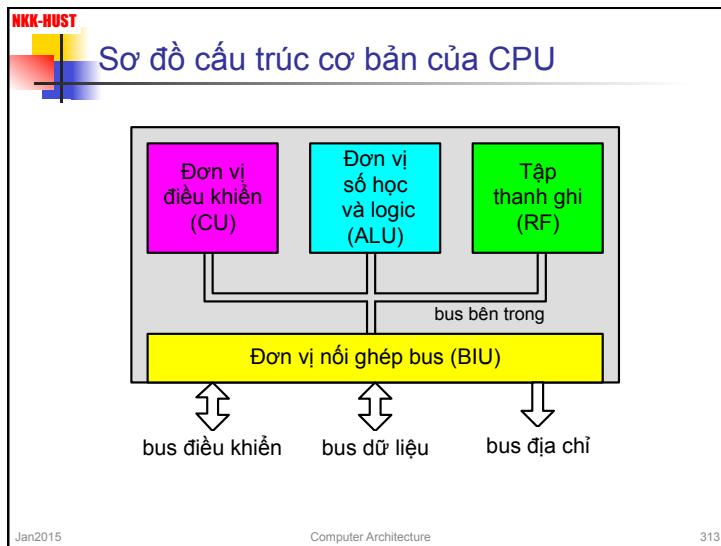
NKK-HUST

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

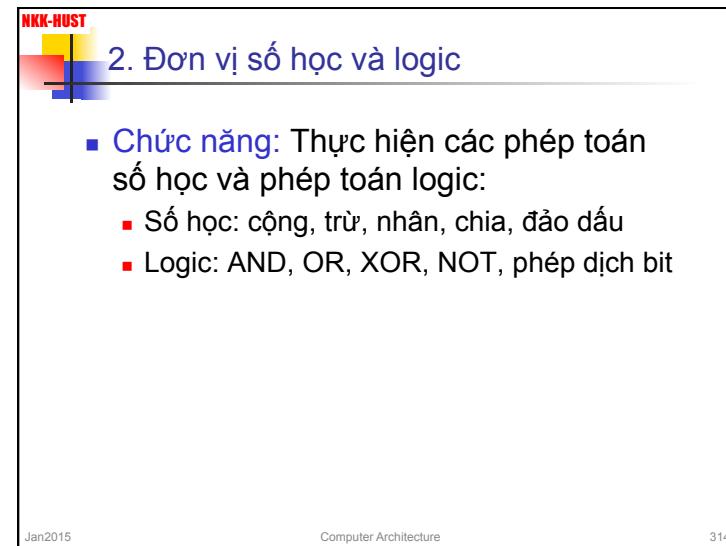
Jan2015 Computer Architecture 312



Jan2015

Computer Architecture

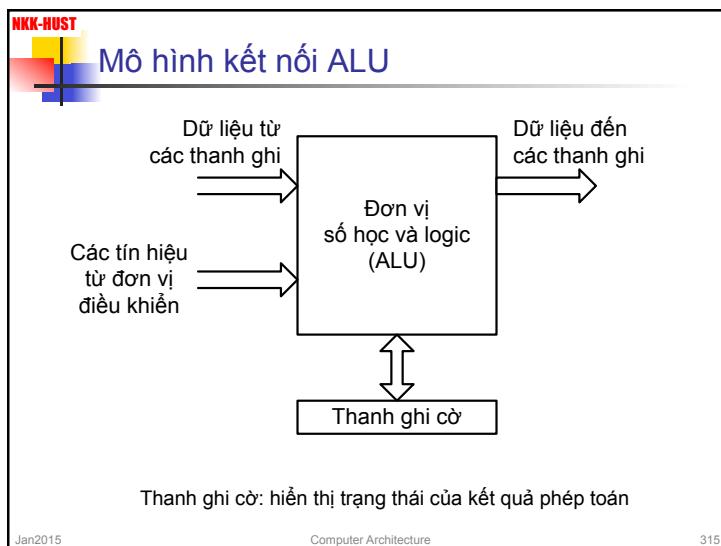
313



Jan2015

Computer Architecture

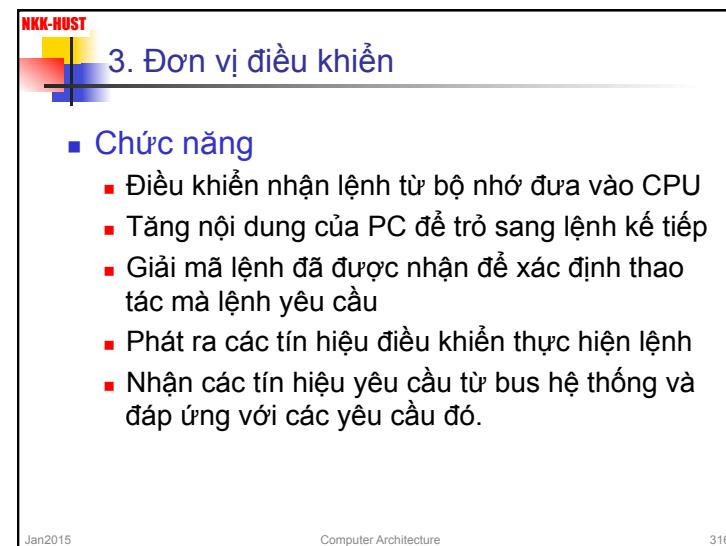
314



Jan2015

Computer Architecture

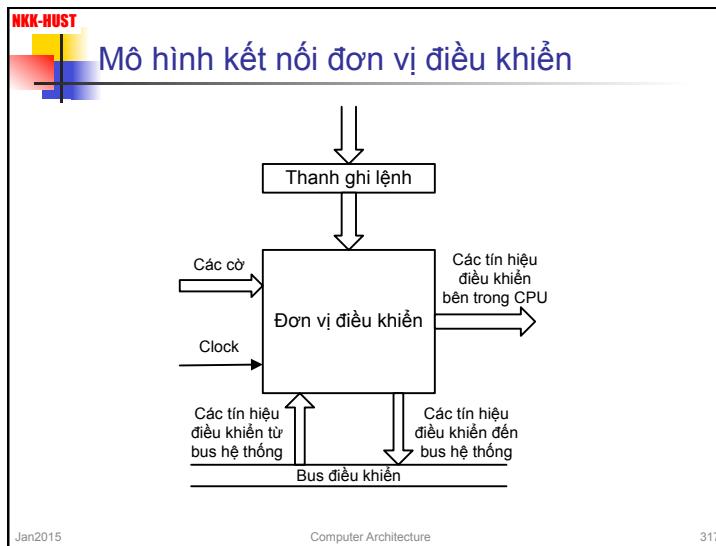
315



Jan2015

Computer Architecture

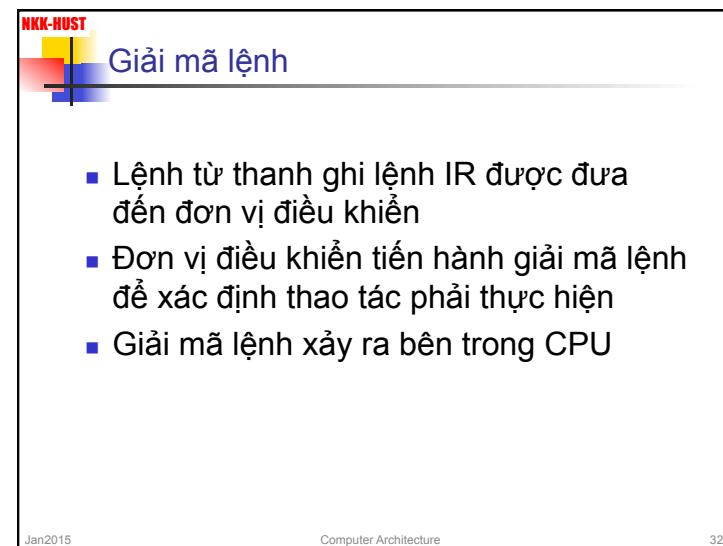
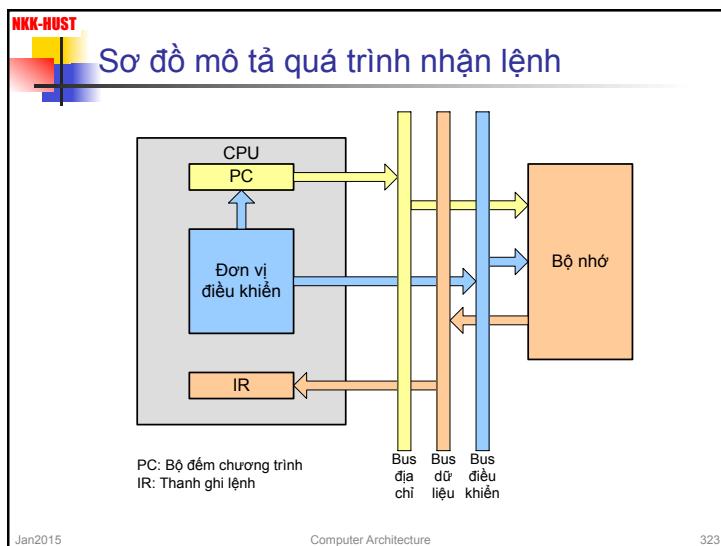
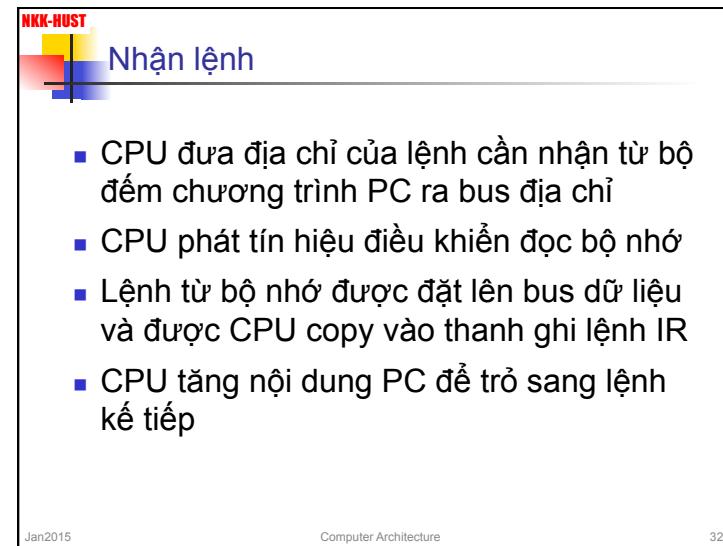
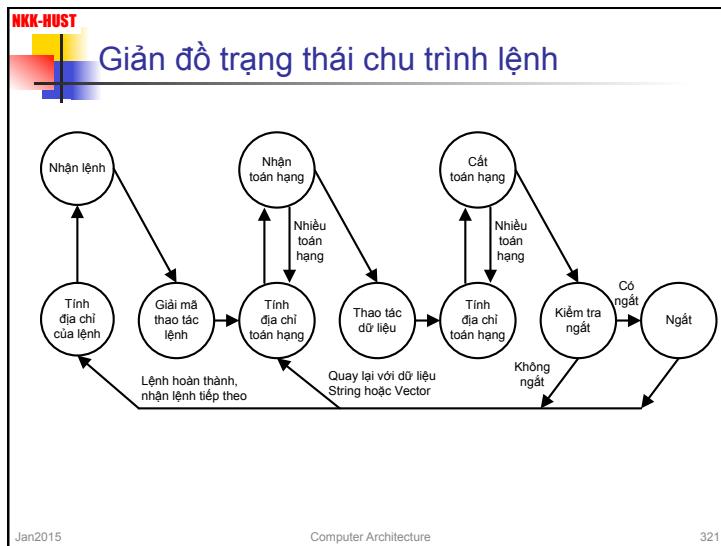
316



- 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
- Jan2015 Computer Architecture 318

- 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
- Jan2015 Computer Architecture 319

- 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
- Jan2015 Computer Architecture 320



Nhận dữ liệu từ bộ nhớ

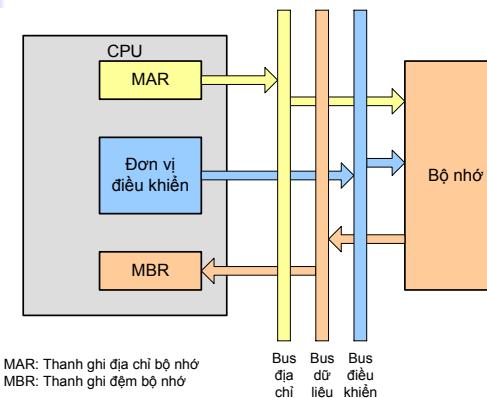
- CPU đưa địa chỉ của toán hạng ra bus địa chỉ
- CPU phát tín hiệu điều khiển đọc
- Toán hạng được đọc vào CPU
- Tương tự như nhận lệnh

Jan2015

Computer Architecture

325

Sơ đồ mô tả nhận dữ liệu từ bộ nhớ



Jan2015

Computer Architecture

326

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)
 - ...

Jan2015

Computer Architecture

327

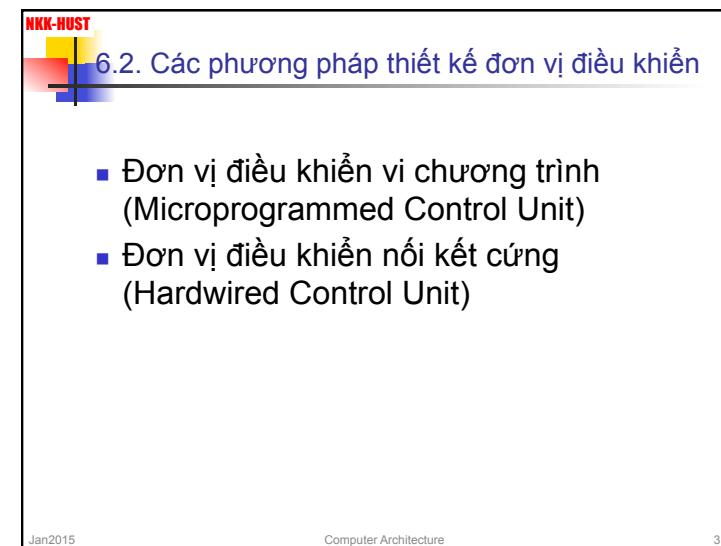
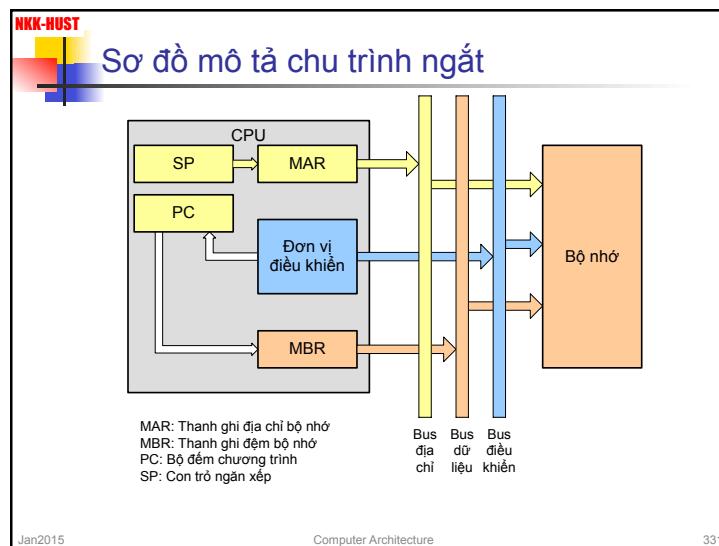
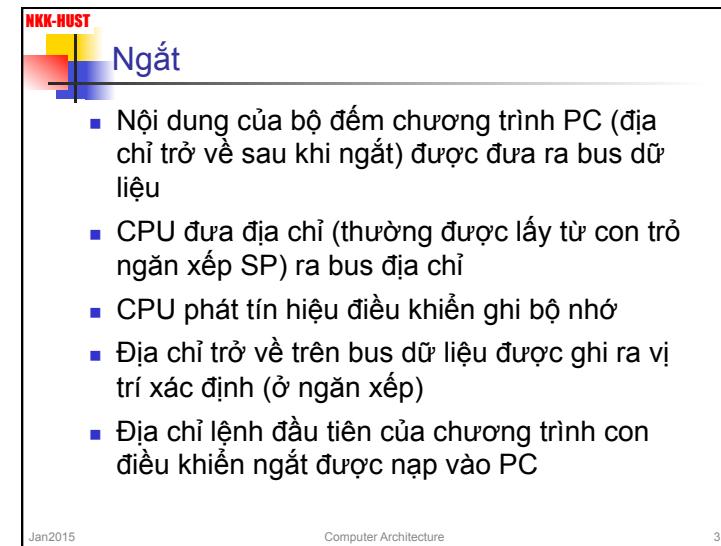
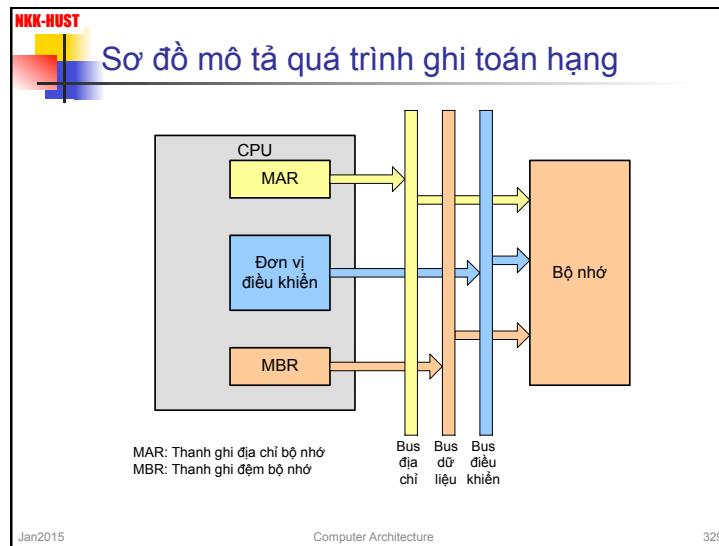
Ghi toán hạng

- 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

Jan2015

Computer Architecture

328



NKK-HUST

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

Jan2015 Computer Architecture 333

NKK-HUST

2. Đơn vị điều khiển nối kết cứng

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

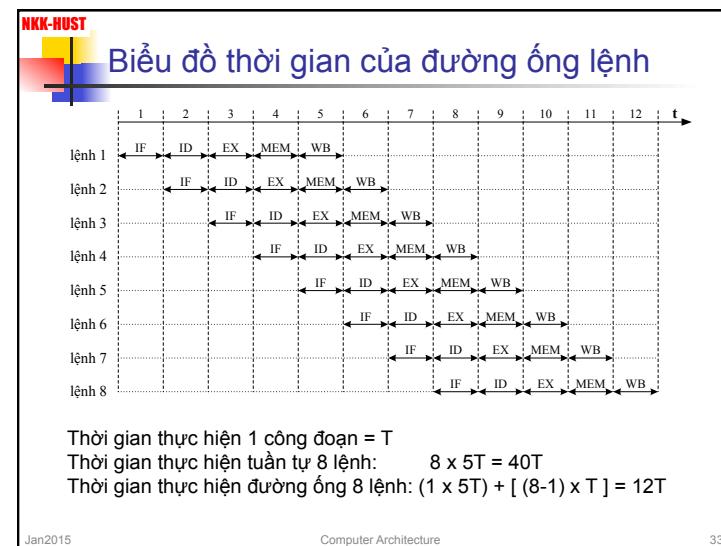
Jan2015 Computer Architecture 334

NKK-HUST

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 lén nhau (như dây chuyền lắp ráp)
- Chẳng hạn bộ xử lý MIPS có 5 công đoạn:
 - IF: Instruction fetch from memory – Nhận lệnh từ bộ nhớ
 - ID: Instruction decode & register read – Giải mã lệnh và đọc thanh ghi
 - EX: Execute operation or calculate address – Thực hiện thao tác hoặc tính toán địa chỉ
 - MEM: Access memory operand – Truy nhập toán hạng bộ nhớ
 - WB: Write result back to register – Ghi kết quả trả về thanh ghi

Jan2015 Computer Architecture 335



Các mối trở ngại (Hazard) của đường ống lệnh

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

Jan2015

Computer Architecture

337

Hazard cấu trúc

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

Jan2015

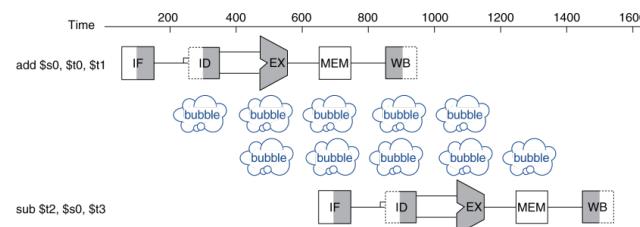
Computer Architecture

338

Hazard dữ liệu

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

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



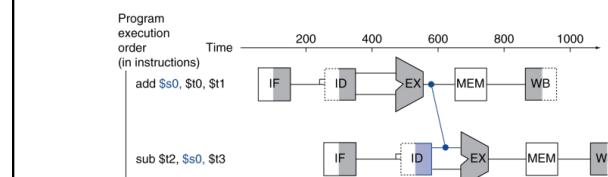
Jan2015

Computer Architecture

339

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



Jan2015

Computer Architecture

340

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) Time: 200 400 600 800 1000 1200 1400

IF ID EX MEM WB

Iw \$s0, 20(\$t1)
sub \$t2, \$s0, \$t3

stall

Computer Architecture 341

Lập lịch mã để tránh trì hoãn

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

$$a = b + e; \quad c = b + f;$$

$lw \quad \$t1, 0(\$t0)$ $lw \quad \$t2, 4(\$t0)$ $add \quad \$t3, \$t1, \$t2$ $sw \quad \$t3, 12(\$t0)$ $lw \quad \$t4, 8(\$t0)$ $add \quad \$t5, \$t1, \$t4$ $sw \quad \$t5, 16(\$t0)$	$lw \quad \$t1, 0(\$t0)$ $lw \quad \$t2, 4(\$t0)$ $lw \quad \$t4, 8(\$t0)$ $add \quad \$t3, \$t1, \$t2$ $sw \quad \$t3, 12(\$t0)$ $add \quad \$t5, \$t1, \$t4$ $sw \quad \$t5, 16(\$t0)$
13 cycles	11 cycles

stall

stall

Computer Architecture 342

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 luân 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

Program execution order (in instructions) Time: 200 400 600 800 1000 1200 1400

add \$4, \$5, \$6
beq \$1, \$2, 40
or \$7, \$8, \$9

Instruction fetch
Reg ALU Data access Reg

bubble

Computer Architecture 343

Trì hoãn khi rẽ nhánh

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

Program execution order (in instructions) Time: 200 400 600 800 1000 1200 1400

add \$4, \$5, \$6
beq \$1, \$2, 40
or \$7, \$8, \$9

Instruction fetch
Reg ALU Data access Reg

bubble

Instruction fetch
Reg ALU Data access Reg

Computer Architecture 344

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)

Jan2015

Computer Architecture

345

MIPS với dự đoán rẽ nhánh không xảy ra

The diagram illustrates the execution of three instructions: add \$4, \$5, \$6, beq \$1, \$2, 40, and lw \$3, 300(\$0). The top part, labeled 'Prediction correct', shows the flow for a correct prediction where the branch taken is followed by the load instruction. The bottom part, labeled 'Prediction incorrect', shows the flow for an incorrect prediction where the branch taken is followed by a bubble instruction (or \$7, \$8, \$9) before the load instruction. Both diagrams include a timeline from 200 to 1400 ps and show the stages: Instruction fetch, Register (Reg), ALU, Data access, and Register (Reg).

Jan2015

Computer Architecture

346

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

Jan2015

Computer Architecture

347

Tăng cường khả năng song song mức lệnh

- Tăng số công đoạn của đường ống

Lệnh 1	IF	R	ALU	DA	R
Lệnh 2	IF	R	ALU	DA	R
Lệnh 3	IF	R	ALU	DA	R
Lệnh 4	IF	R	ALU	DA	R
Lệnh 5	IF	R	ALU	DA	R
Lệnh 6	IF	R	ALU	DA	R

- Siêu vô hướng (Superscalar)

Lệnh 1	IF1	R1	ALU1	DA1	R1
Lệnh 2	IF2	R2	ALU2	DA2	R2
Lệnh 3	IF3	R3	ALU3	DA3	R3
Lệnh 4	IF4	R4	ALU4	DA4	R4
Lệnh 5	IF5	R5	ALU5	DA5	R5
Lệnh 6	IF6	R6	ALU6	DA6	R6

Jan2015

Computer Architecture

348

NKK-HUST

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

Bộ xử lý MIPS – Chapter 4 – [2]

Jan2015 Computer Architecture 349

NKK-HUST

Hết chương 6

Jan2015 Computer Architecture 350

NKK-HUST

Kiến trúc máy tính

Chương 7

BỘ NHỚ MÁY TÍNH

Nguyễn Kim Khánh
Trường Đại học Bách khoa Hà Nội

Jan2015 Computer Architecture 351

NKK-HUST

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

Jan2015 Computer Architecture 352

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

Jan2015 Computer Architecture 353

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

Jan2015 Computer Architecture 354

NKK-HUST

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)

Jan2015 Computer Architecture 355

NKK-HUST

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

Jan2015 Computer Architecture 356

NKK-HUST

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

Jan2015 Computer Architecture 357

NKK-HUST

2. Phân cấp bộ nhớ

```

graph LR
    subgraph CPU [Bộ vi xử lý]
        direction TB
        A[CPU] --> B[Cache]
        B --> C[Bộ nhớ chính]
    end
    C <--> D[Thiết bị lưu trữ (HDD, SSD)]
    D <--> E[Bộ nhớ mạng]
    
```

Từ trái sang phải:

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

Jan2015 Computer Architecture 358

NKK-HUST

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

Jan2015 Computer Architecture 359

NKK-HUST

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

Jan2015 Computer Architecture 360

NKK-HUST

7.2. Bộ nhớ chính

1. Bộ nhớ bán dẫn

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

Jan2015 Computer Architecture 361

NKK-HUST

ROM (Read Only Memory)

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

Jan2015 Computer Architecture 362

NKK-HUST

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

Jan2015 Computer Architecture 363

NKK-HUST

Bộ nhớ Flash

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

Jan2015 Computer Architecture 364

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)

Jan2015

Computer Architecture

365

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

Jan2015

Computer Architecture

366

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

Jan2015

Computer Architecture

367

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

Jan2015

Computer Architecture

368

Tổ chức của chip nhớ

- Sơ đồ cơ bản của chip nhớ

Jan2015 Computer Architecture 369

Các tín hiệu của chip nhớ

- Các đường địa chỉ: $A_{n-1} \div A_0 \rightarrow$ có 2^n từ nhớ
- Các đường dữ liệu: $D_{m-1} \div D_0 \rightarrow$ độ dài từ nhớ = m bit
- Dung lượng chip nhớ = $2^n \times m$ bit
- Các đường điều khiển:
 - Tín hiệu chọn chip CS (Chip Select)
 - Tín hiệu điều khiển đọc OE (Output Enable)
 - Tín hiệu điều khiển ghi WE (Write Enable)
 (Các tín hiệu điều khiển thường tích cực với mức 0)

Jan2015 Computer Architecture 370

Tổ chức của DRAM

- Dùng n đường địa chỉ dòn kẽnh \rightarrow 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

Jan2015 Computer Architecture 371

Ví dụ chip nhớ

(a) 8-Mbit EPROM (b) 16-Mbit DRAM

Jan2015 Computer Architecture 372



Thiết kế mô-đun nhớ bán dẫn

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

Jan2015

Computer Architecture

373



Tăng độ dài từ nhớ

VD1:

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

Giải:

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

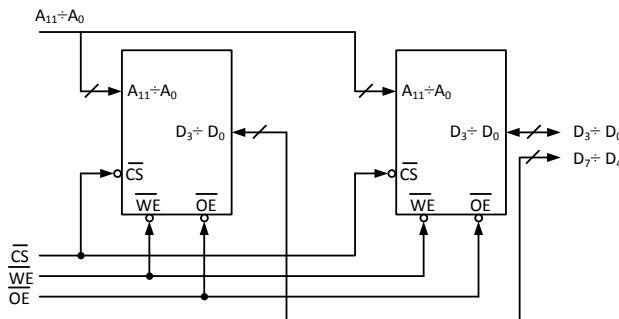
Jan2015

Computer Architecture

374



Sơ đồ ví dụ tăng độ dài từ nhớ



Jan2015

Computer Architecture

375



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

Jan2015

Computer Architecture

376

NKK-HUST

Tăng số lượng từ nhớ

VD2:

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

Giải:

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

Jan2015 Computer Architecture 377

NKK-HUST

Sơ đồ ví dụ tăng số lượng từ nhớ

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

Jan2015 Computer Architecture 378

NKK-HUST

Bộ giải mã 2 \rightarrow 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

Jan2015 Computer Architecture 379

NKK-HUST

Thiết kế kết hợp

Ví dụ 3:

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

Phương pháp thực hiện:

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

Jan2015 Computer Architecture 380

2. Các đặc trưng cơ bản của bộ nhớ chính

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

Jan2015

Computer Architecture

381

Tổ chức bộ nhớ đan xen (interleaved memory)

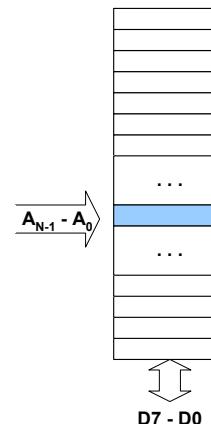
- Độ rộng của bus dữ liệu để trao đổi với bộ nhớ: $m = 8, 16, 32, 64, 128 \dots$ bit
- Các ngăn nhớ được tổ chức theo byte
→ tổ chức bộ nhớ vật lý khác nhau

Jan2015

Computer Architecture

382

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

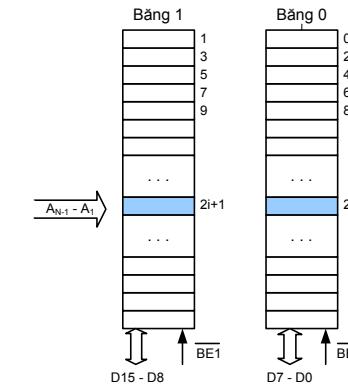


Jan2015

Computer Architecture

383

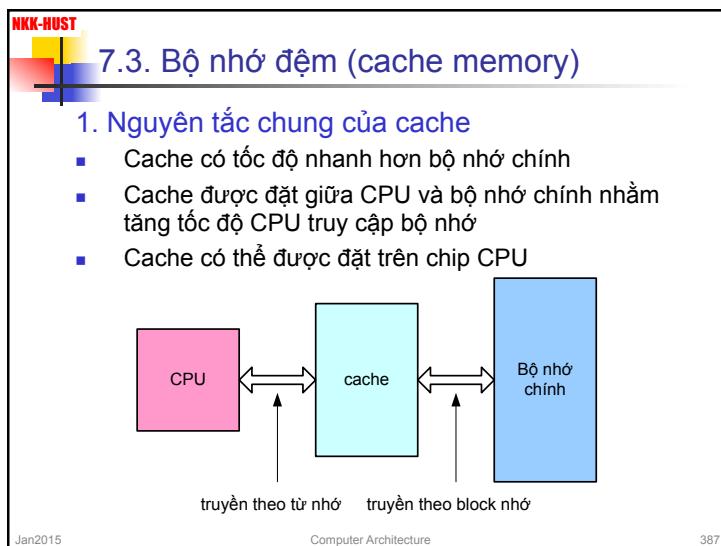
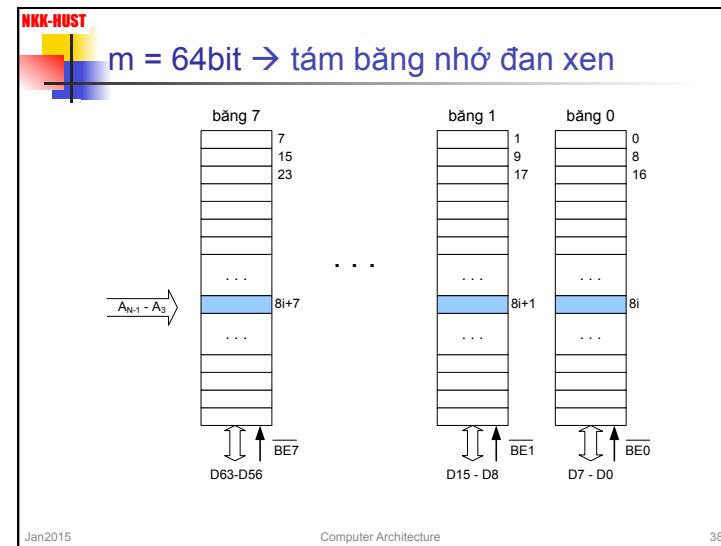
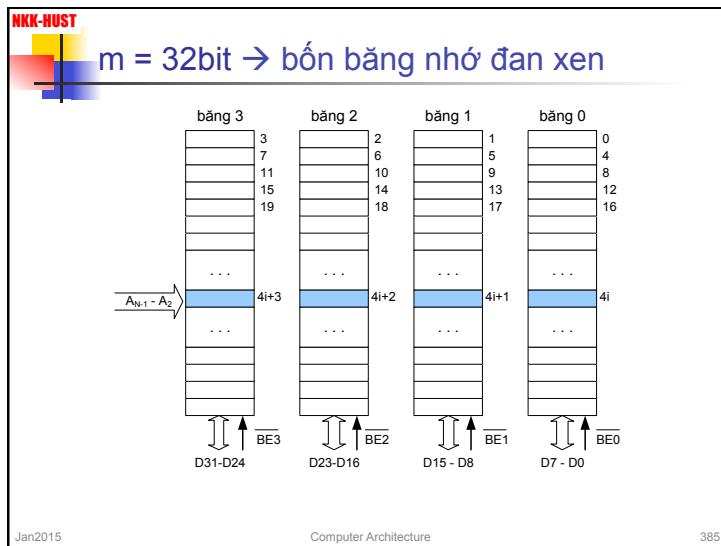
$m = 16\text{bit} \rightarrow$ hai băng nhớ đan xen



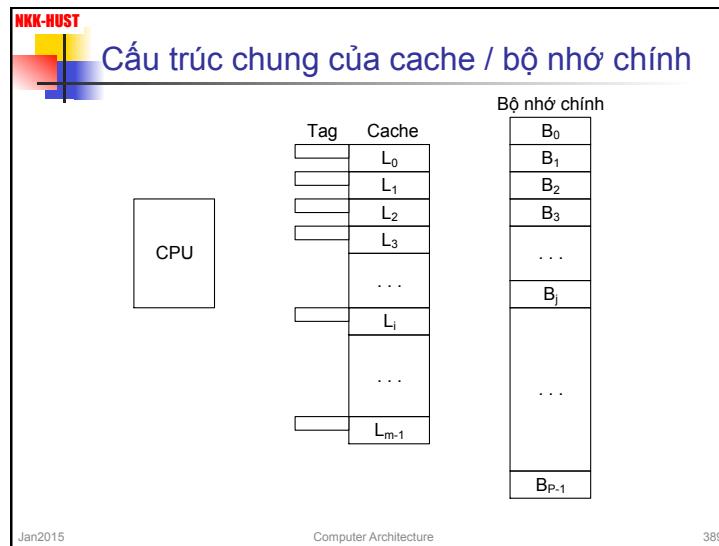
Jan2015

Computer Architecture

384



- NKK-HUST**
- ### 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
- Jan2015 Computer Architecture 388



- 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
- Jan2015 Computer Architecture 390

- 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).
- Jan2015 Computer Architecture 391

- 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)
- Jan2015 Computer Architecture 392

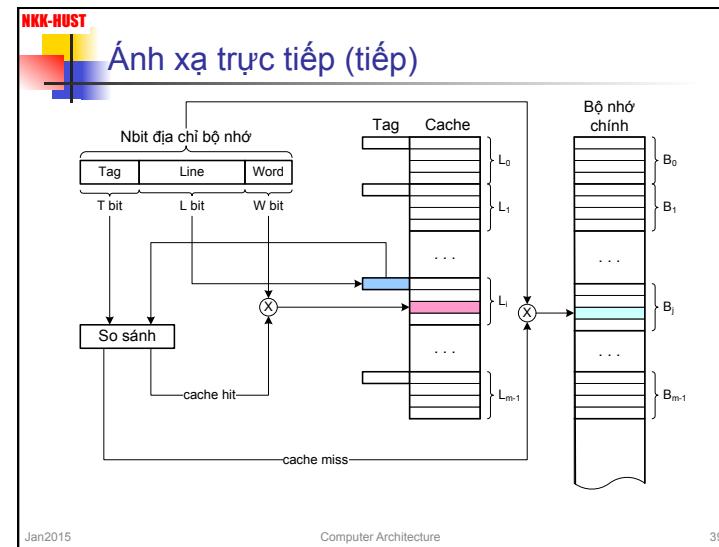
Ánh xạ trực tiếp

- 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.

Jan2015

Computer Architecture

393



Jan2015

Computer Architecture

394

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:
 $T = N - (W+L)$

Jan2015

Computer Architecture

395

- 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 cao 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

Jan2015

Computer Architecture

396

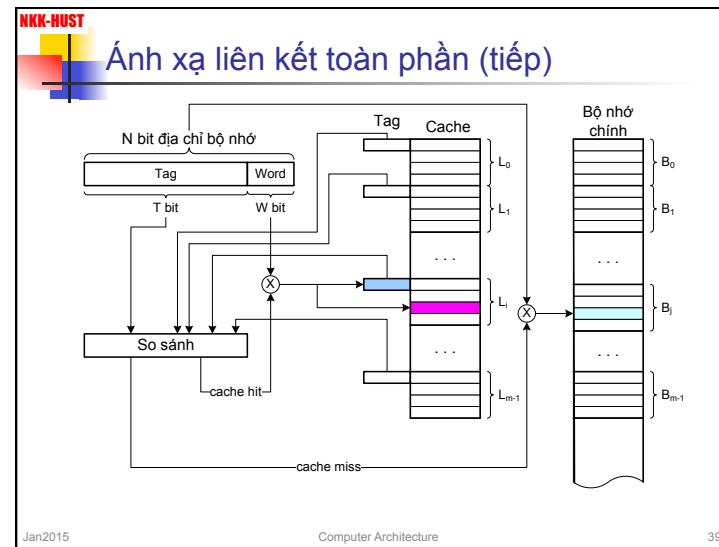
Ánh xạ liên kết toàn phần

- 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

Jan2015

Computer Architecture

397



Jan2015

Computer Architecture

398

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 cao 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

Jan2015

Computer Architecture

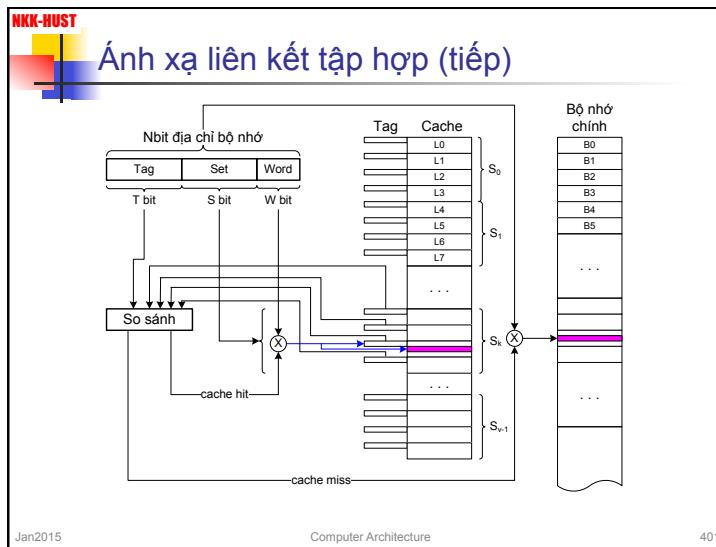
399

- 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:**
 - B₀ \rightarrow S₀
 - B₁ \rightarrow S₁
 - B₂ \rightarrow S₂
 -

Jan2015

Computer Architecture

400



- NKK-HUST**
- ### 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 nào 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
- Jan2015 Computer Architecture 402

- NKK-HUST**
- ### 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
- Jan2015 Computer Architecture 403

- NKK-HUST**
- ### Với ánh xạ trực tiếp
- Bộ nhớ chính = 4GiB = 2^{32} byte $\rightarrow N = 32$ bit
 - Cache = 256 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
- | | | |
|--------|--------|-------|
| Tag | Line | Word |
| 14 bit | 13 bit | 5 bit |
- Jan2015 Computer Architecture 404

Với ánh xạ liên kết toàn phần

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

Tag	Word
27 bit	5 bit

Jan2015 Computer Architecture 405

Với ánh xạ liên kết tập hợp 4 đường

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

Tag	Set	Word
16 bit	11 bit	5 bit

Jan2015 Computer Architecture 406

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

Jan2015 Computer Architecture 407

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

Jan2015 Computer Architecture 408

NKK-HUST 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

Jan2015

Computer Architecture

409

NKK-HUST 7.4. Bộ nhớ ngoài

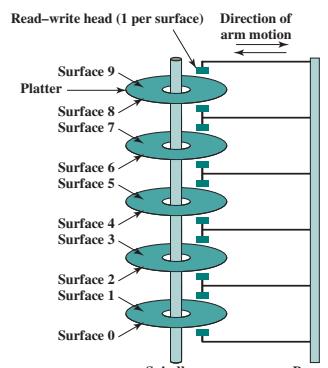
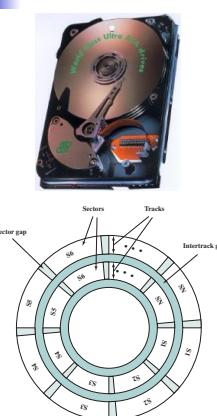
- Tồn tại dưới dạng các thiết bị lưu trữ
- Các kiểu bộ nhớ ngoài
 - Băng từ: ít sử dụng
 - Đĩa từ: Ổ đĩa cứng HDD (Hard Disk Drive)
 - Đĩa quang: CD, DVD
 - Bộ nhớ Flash:
 - Ổ nhớ thẻ rắn SSD (Solid State Drive)
 - USB flash
 - Thẻ nhớ

Jan2015

Computer Architecture

410

NKK-HUST Ô đĩa cứng (HDD –Hard Disk Drive)



Jan2015

Computer Architecture

411

NKK-HUST HDD

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

Jan2015

Computer Architecture

412

ĐỒ 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



Jan2015 Computer Architecture 413

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

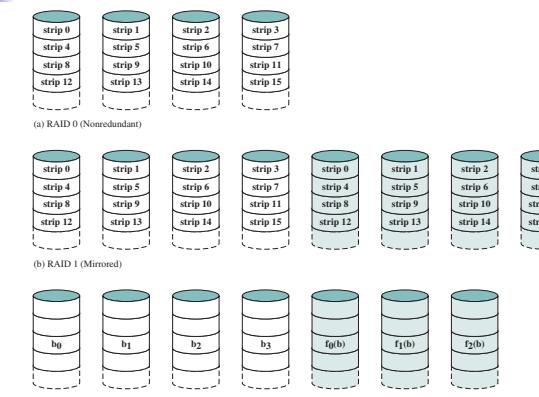
Jan2015 Computer Architecture 414

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)

Jan2015 Computer Architecture 415

RAID 0, 1, 2

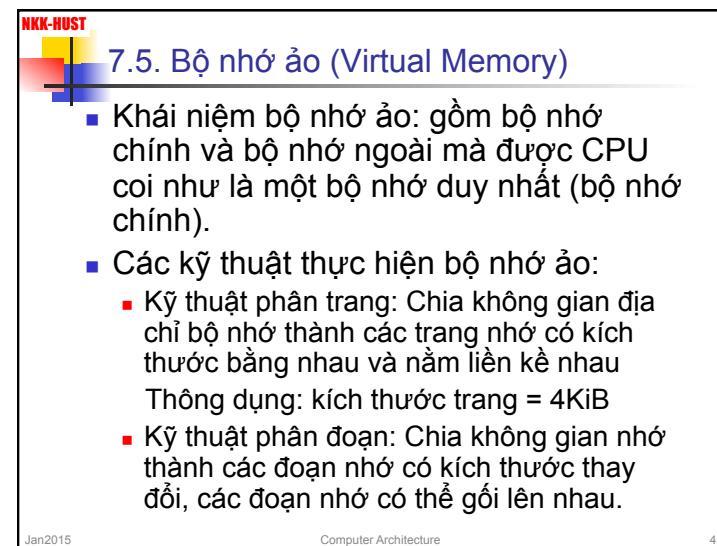
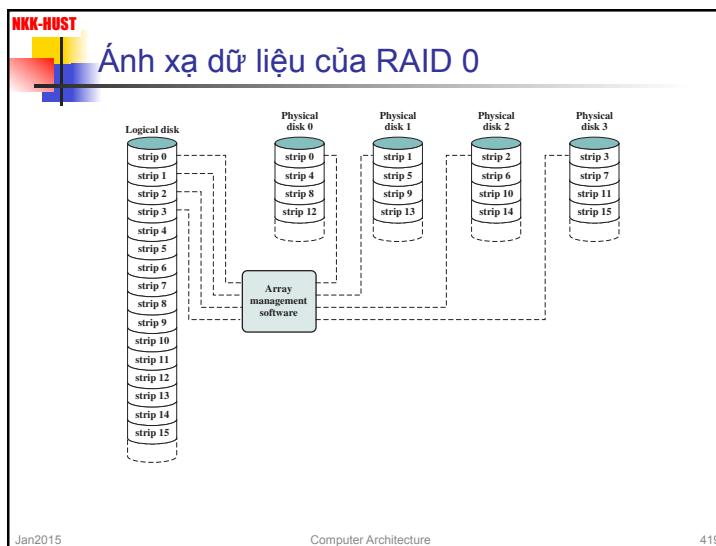
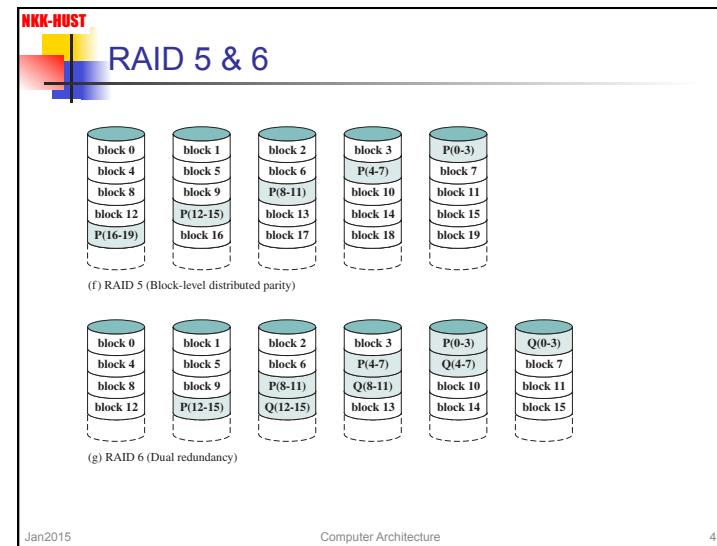
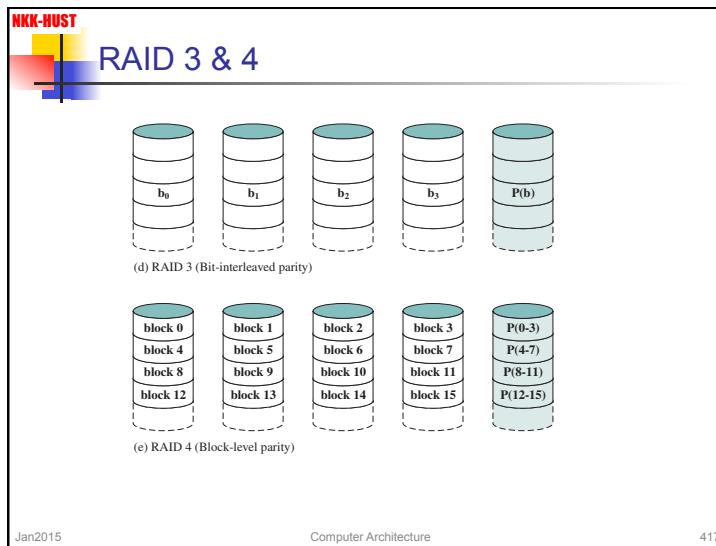


(a) RAID 0 (Nonredundant): Shows four separate cylinders labeled strip 0 through strip 15, representing data striped across four disks.

(b) RAID 1 (Mirrored): Shows two pairs of cylinders, each pair being a mirror of the other, representing data mirrored across two disks.

(c) RAID 2 (Redundancy through Hamming code): Shows four cylinders labeled b0 through b3, with associated error correction cylinders labeled f0(b), f1(b), and f2(b), representing data encoded with Hamming code for error correction.

Jan2015 Computer Architecture 416

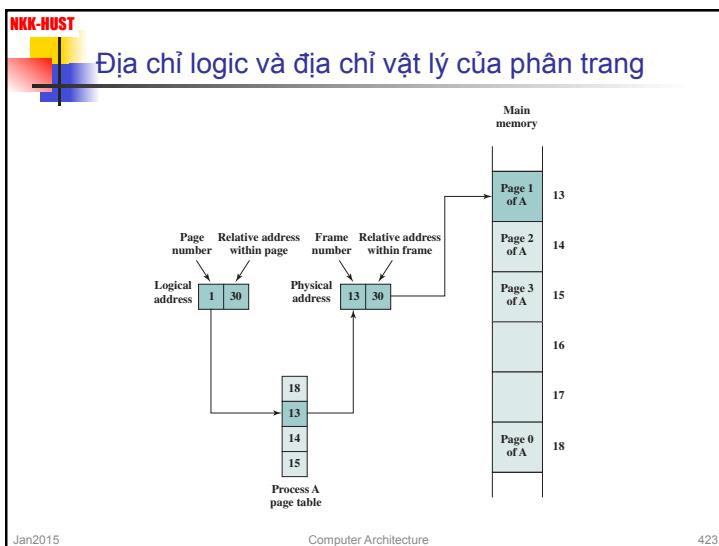
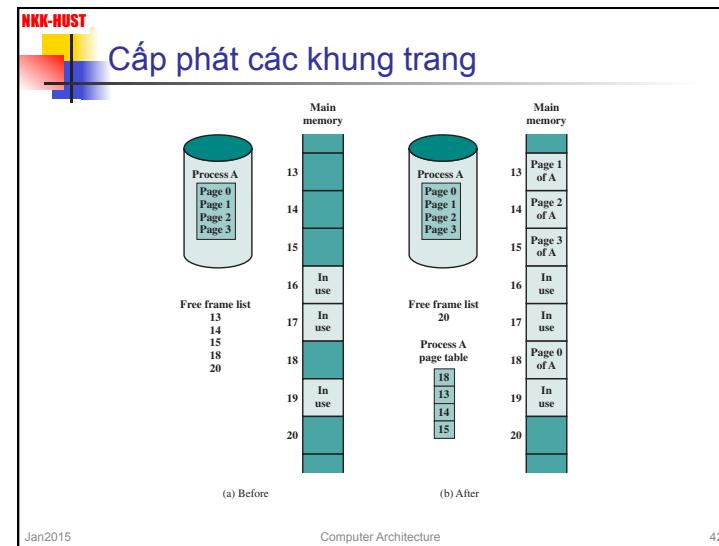


NKK-HUST

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

Jan2015 Computer Architecture 421



- NKK-HUST**
- ## 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
- Jan2015 Computer Architecture 424

NKK-HUST

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

Jan2015 Computer Architecture 425

NKK-HUST

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

Jan2015 Computer Architecture 426

NKK-HUST

Cấu trúc bảng trang

Virtual address
n bits
Page # | Offset

n bits

Hash function
m bits

Inverted page table
(one entry for each physical memory frame)

Page # | Process ID | Chain | Frame #
0
i
j
 $2^n - 1$

m bits
Offset
Real address

Jan2015 Computer Architecture 427

NKK-HUST

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

Jan2015 Computer Architecture 428

NKK-HUST

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

Jan2015 Computer Architecture 429

NKK-HUST

Hết chương 7

Jan2015 Computer Architecture 430

NKK-HUST

Kiến trúc máy tính

Chương 8

HỆ THỐNG VÀO-RA

Nguyễn Kim Khánh
Trường Đại học Bách khoa Hà Nội

Jan2015 Computer Architecture 431

NKK-HUST

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

Jan2015 Computer Architecture 432

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

Jan2015 Computer Architecture 433

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

Jan2015 Computer Architecture 434

Đặc điểm của hệ thống vào-ra

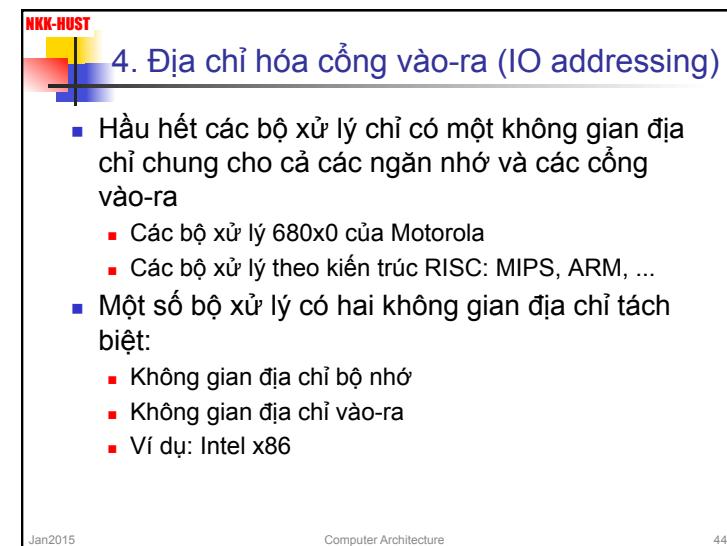
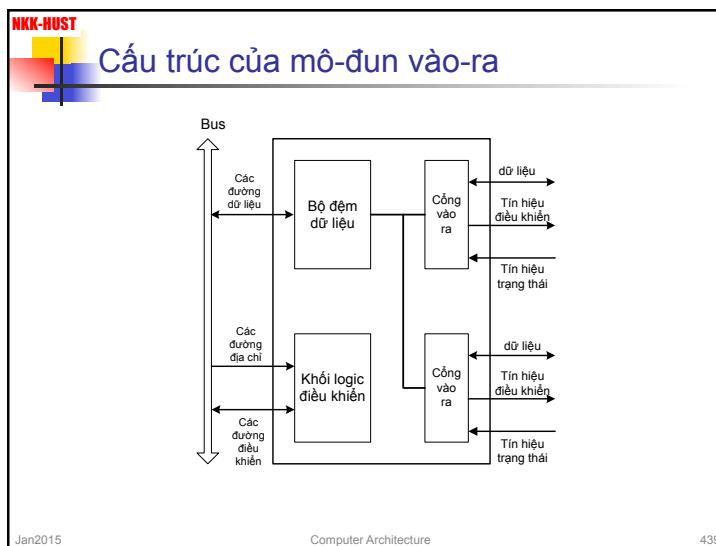
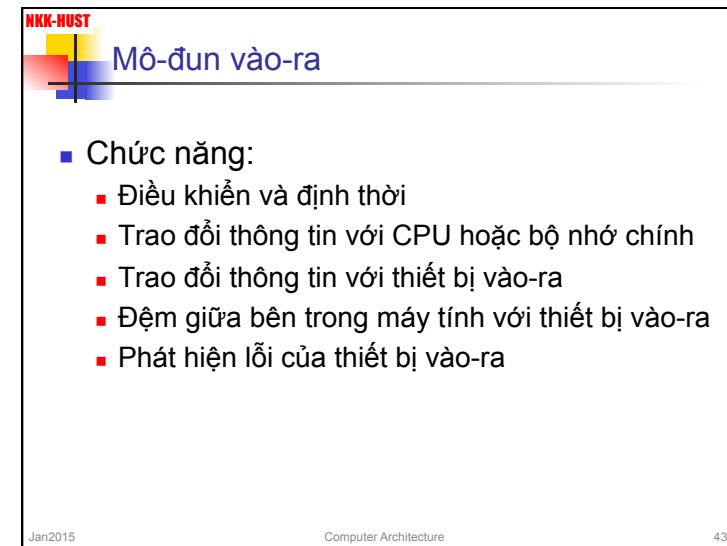
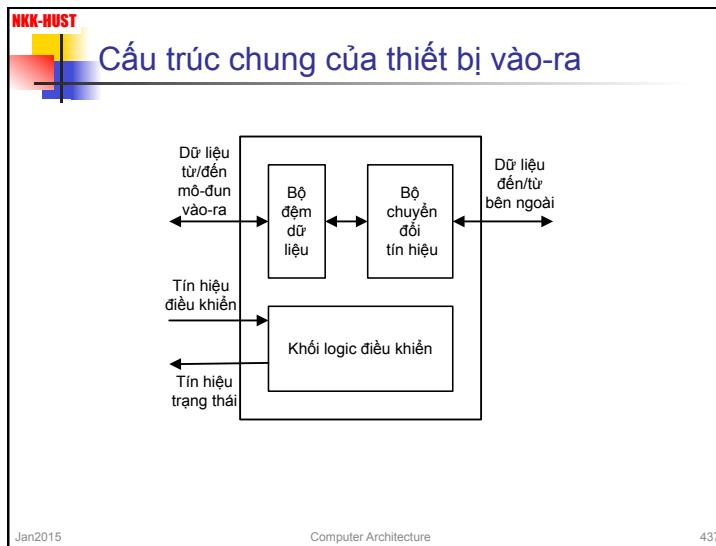
- Tồn tại đa dạng các thiết bị vào-ra khác nhau về:
 - Nguyên tắc hoạt động
 - Tốc độ
 - Khuôn dạng dữ liệu
- Tất cả các thiết bị vào-ra đều chậm hơn CPU và RAM
→ Cần có các mô-đun vào-ra để nối ghép các thiết bị với CPU và bộ nhớ chính

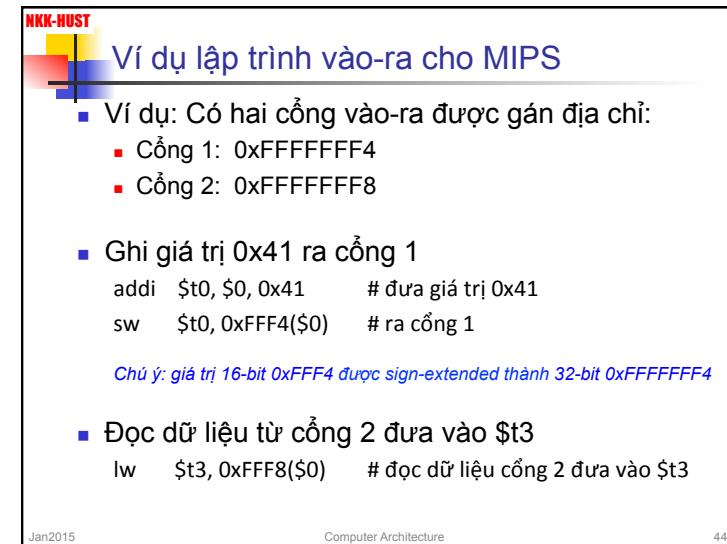
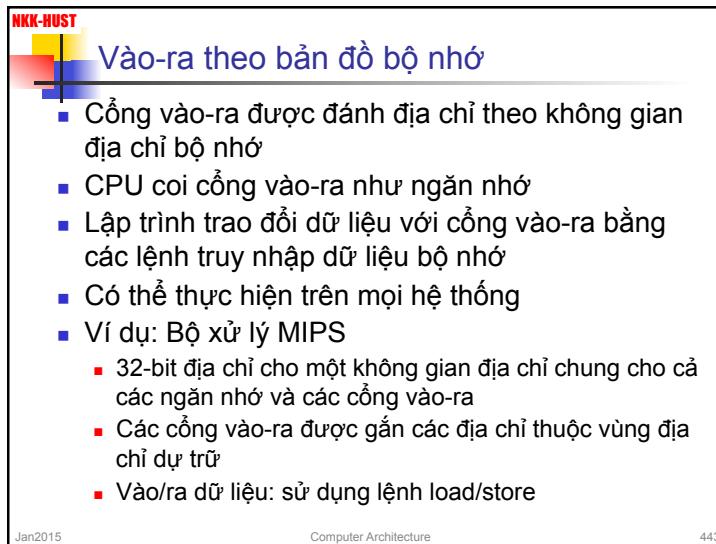
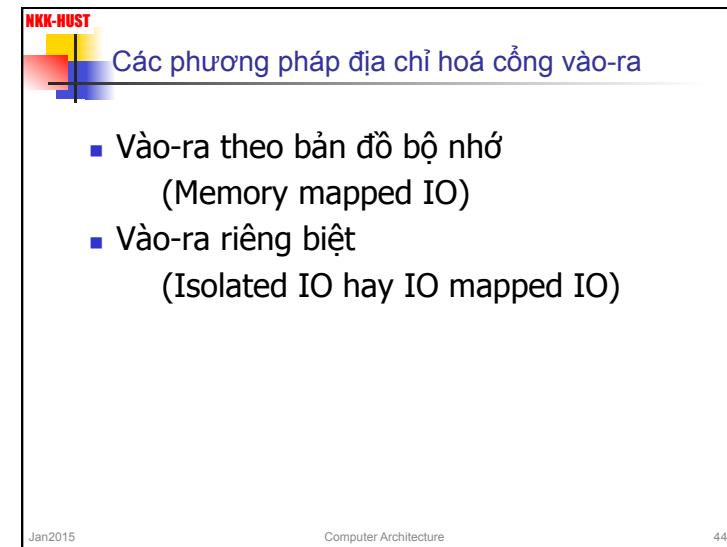
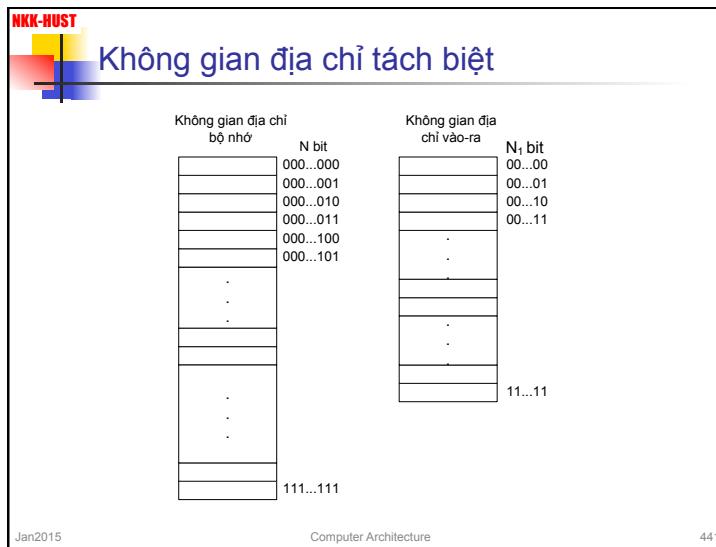
Jan2015 Computer Architecture 435

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

Jan2015 Computer Architecture 436





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

Jan2015 Computer Architecture 445

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)

Jan2015 Computer Architecture 446

NKK-HUST

Ba kỹ thuật thực hiện vào một khối dữ liệu

The diagram illustrates three methods for reading a block of data:

- (a) Programmed I/O:** Shows a sequential process where the CPU issues a read command to the I/O module, checks its status, reads data from the I/O module, and writes it to memory. This loop repeats until all data is read.
- (b) Interrupt-Driven I/O:** Similar to programmed I/O, but the CPU continues executing other tasks ("Do something else") while waiting for the I/O module to signal an interrupt when data is ready. The CPU then handles the interrupt to read the data.
- (c) Direct Memory Access (DMA):** The CPU issues a read block command to the I/O module and a read status command to the DMA module. The DMA module handles the data transfer directly between the I/O module and memory, bypassing the CPU's direct intervention.

Jan2015 Computer Architecture 447

NKK-HUST

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

```

    graph TD
        A[Đọc trạng thái mô-đun vào-ra] --> B{Sẵn sàng?}
        B -- N --> A
        B -- Y --> C[Trao đổi dữ liệu]
        C --> D[Next instruction]
    
```

Jan2015 Computer Architecture 448

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

Jan2015

Computer Architecture

449

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)

Jan2015

Computer Architecture

450

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

Jan2015

Computer Architecture

451

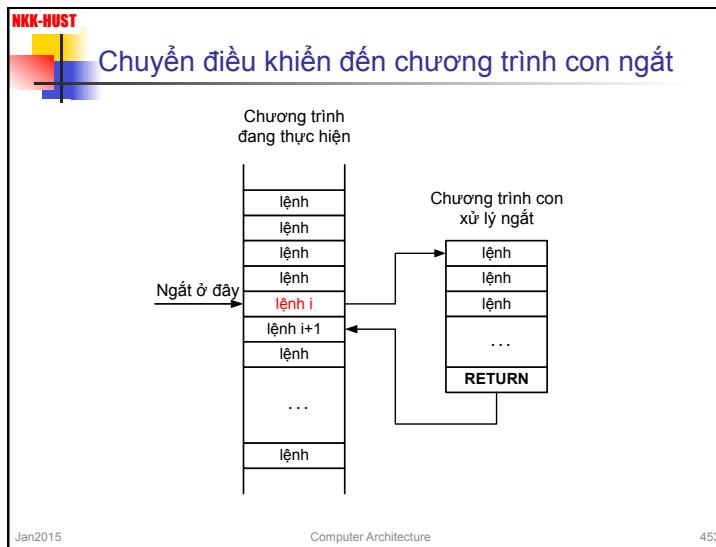
2. Vào-ra điều khiển bằng ngắt

- 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

Jan2015

Computer Architecture

452



- NKK-HUST
- ### Hoạt động vào dữ liệu: nhìn từ mô-đun vào-ra
- Mô-đun vào-ra nhận tín hiệu điều khiển đọc từ CPU
 - Mô-đun vào-ra nhận dữ liệu từ thiết bị vào-ra, trong khi đó CPU làm việc khác
 - Khi đã có dữ liệu → mô-đun vào-ra phát tín hiệu ngắt CPU
 - CPU yêu cầu dữ liệu
 - Mô-đun vào-ra chuyển dữ liệu đến CPU
- Jan2015 Computer Architecture 454

- NKK-HUST
- ### 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
- Jan2015 Computer Architecture 455

- NKK-HUST
- ### 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 ?
- Jan2015 Computer Architecture 456

NKK-HUST

Các phương pháp nối ghép ngắn

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

Jan2015 Computer Architecture 457

NKK-HUST

Nhiều đường yêu cầu ngắn

The diagram illustrates a system where four parallel input modules (Mô-đun vào-ra) each generate an interrupt signal (INTR0, INTR1, INTR2, INTR3) that is connected to a CPU's interrupt queue (Thanh ghi yêu cầu ngắn). This allows the CPU to handle multiple interrupt requests simultaneously.

Jan2015 Computer Architecture 458

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

NKK-HUST

Hỏi vòng bằng phần mềm

The diagram shows a software poll mechanism. The CPU has a 'Cờ ngắt' (flag) and an 'INTR' line. It sequentially checks five input modules (Mô-đun vào-ra) connected in series. When one module detects an event, it asserts the INTR line, which triggers the CPU's flag. The CPU then performs the required task and continues the poll cycle.

Jan2015 Computer Architecture 459

- 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

NKK-HUST

Hỏi vòng bằng phần cứng

The diagram shows a hardware poll mechanism using a shared bus. The CPU has a 'Cờ ngắt' and an 'INTR' line. It sequentially polls four input modules (Mô-đun vào-ra) connected to a shared bus. When one module detects an event, it asserts the INTR line, which triggers the CPU's flag. The CPU then performs the required task and continues the poll cycle.

Jan2015 Computer Architecture 460

Hỏi vòng bằng phần cứng (tiếp)

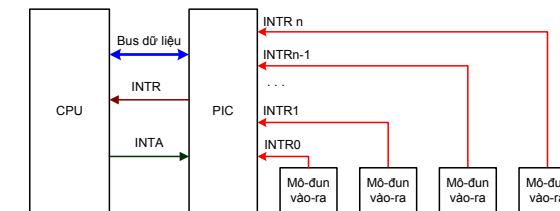
- CPU phát tín hiệu chấp nhận ngắt (INTA) đến mô-đun vào-ra đầu tiên
- Nếu mô-đun vào-ra đó không gây ra ngắt thì nó gửi tín hiệu đến mô-đun kế tiếp cho đến khi xác định được mô-đun gây ngắt
- Thứ tự các mô-đun vào-ra kết nối trong chuỗi xác định thứ tự ưu tiên

Jan2015

Computer Architecture

461

Bộ điều khiển ngắt lập trình được



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

Jan2015

Computer Architecture

462

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

Jan2015

Computer Architecture

463

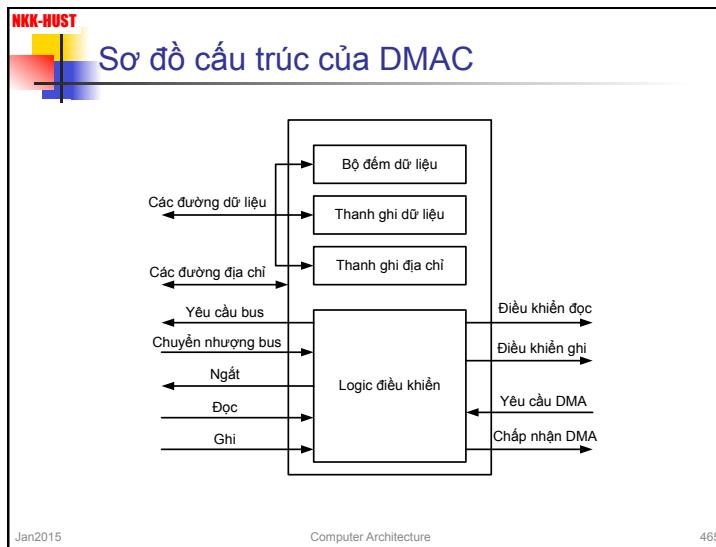
3. DMA (Direct Memory Access)

- 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

Jan2015

Computer Architecture

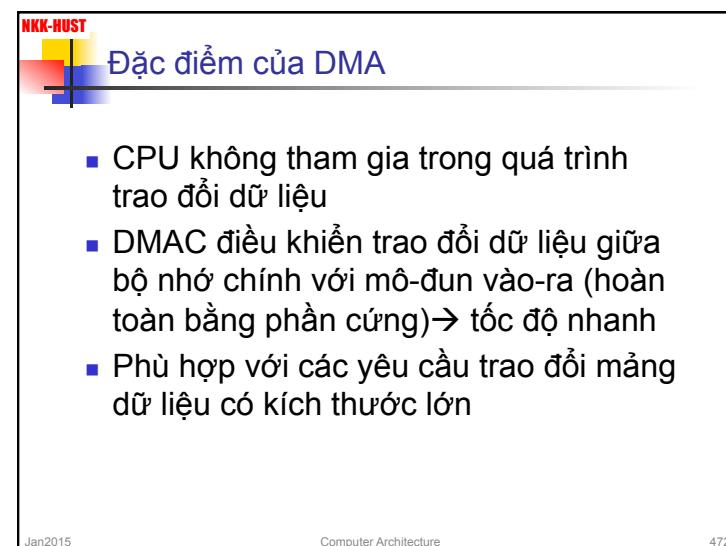
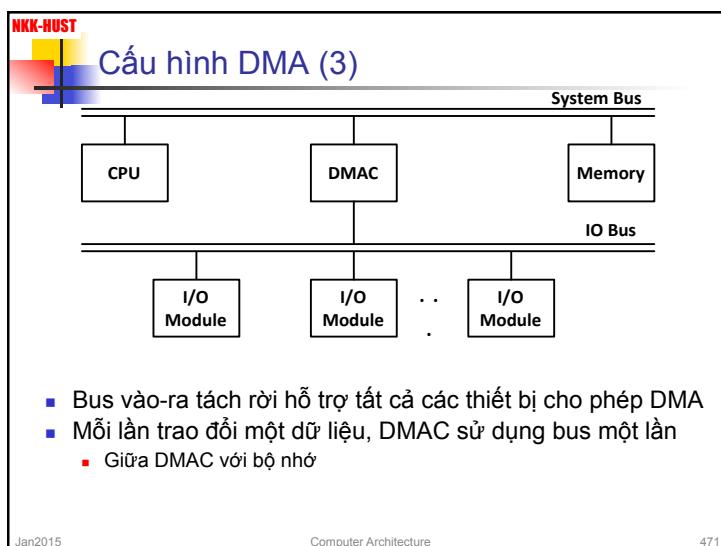
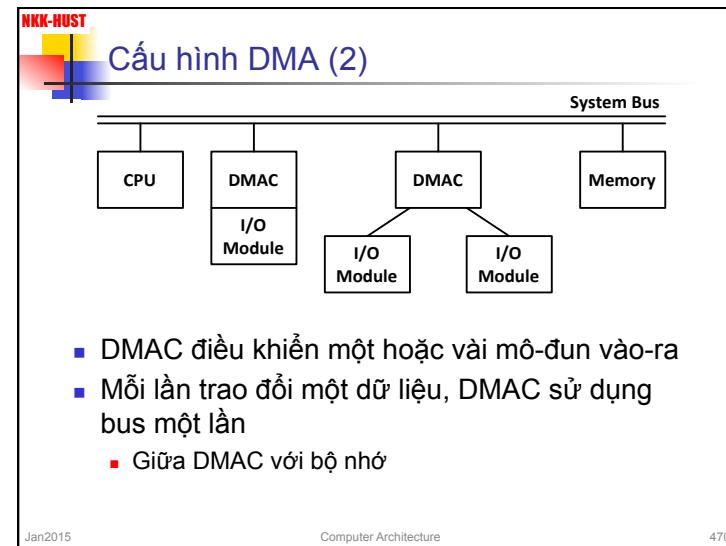
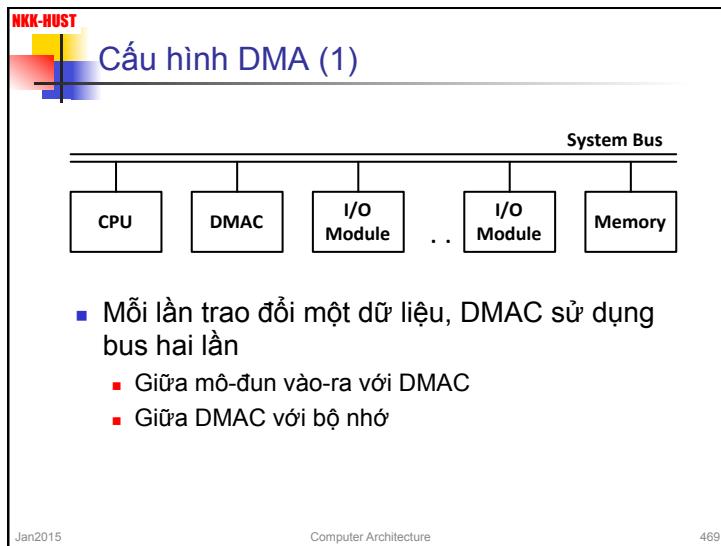
464



- NKK-HUST**
- ### 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
- Jan2015 Computer Architecture 466

- NKK-HUST**
- ### 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
- Jan2015 Computer Architecture 467

- NKK-HUST**
- ### 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.
- Jan2015 Computer Architecture 468



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

Jan2015

Computer Architecture

473

8.3. Nối ghép thiết bị vào-ra

1. Các kiểu nối ghép vào-ra

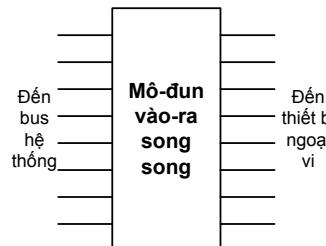
- Nối ghép song song
- Nối ghép nối tiếp

Jan2015

Computer Architecture

474

Nối ghép song song



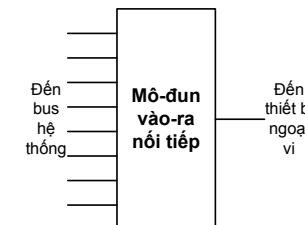
- Truyền nhiều bit song song
- Tốc độ nhanh
- Cần nhiều đường truyền dữ liệu

Jan2015

Computer Architecture

475

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

Jan2015

Computer Architecture

476

2. Các cấu hình nối ghép

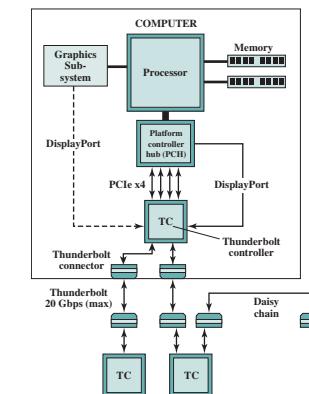
- Điểm tới điểm (Point to Point)
 - Thông qua một cổng vào-ra nối ghép với một thiết bị
- Điểm tới đa điểm (Point to Multipoint)
 - Thông qua một cổng vào-ra cho phép nối ghép được với nhiều thiết bị
 - Ví dụ:
 - USB (Universal Serial Bus): 127 thiết bị
 - IEEE 1394 (FireWire): 63 thiết bị
 - Thunderbolt

Jan2015

Computer Architecture

477

Thunderbolt



Jan2015

Computer Architecture

478

Hết chương 8

Jan2015

Computer Architecture

479

Kiến trúc máy tính

Chương 9 CÁC KIẾN TRÚC SONG SONG

Nguyễn Kim Khánh
Trường Đại học Bách khoa Hà Nội

Jan2015

Computer Architecture

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**

Jan2015 Computer Architecture 481

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

Jan2015 Computer Architecture 482

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

Jan2015 Computer Architecture 483

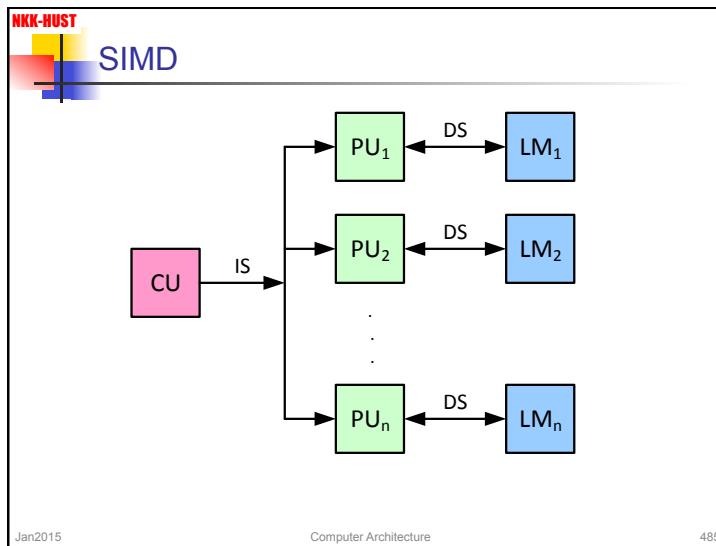
SISD

```

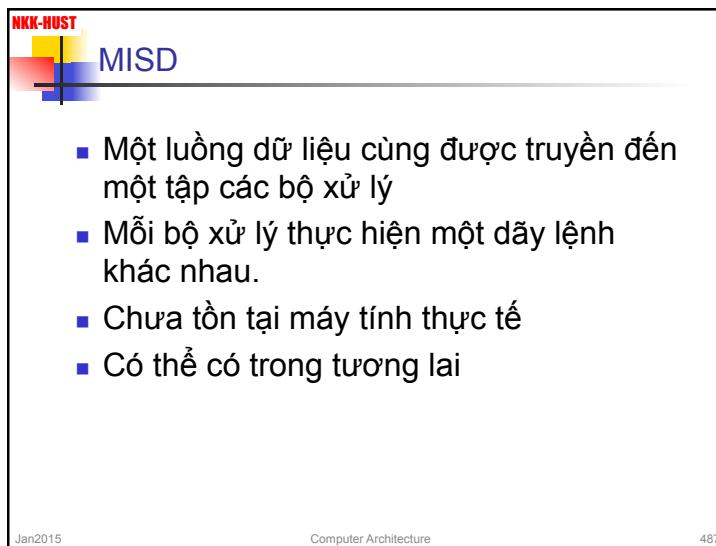
graph LR
    CU[CU] -- IS --> PU[PU]
    PU -- DS --> MU[MU]
    MU -- DS --> PU
  
```

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

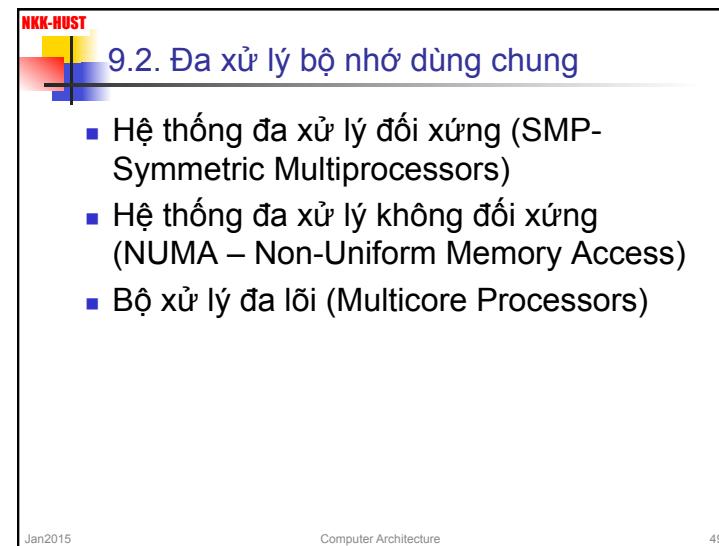
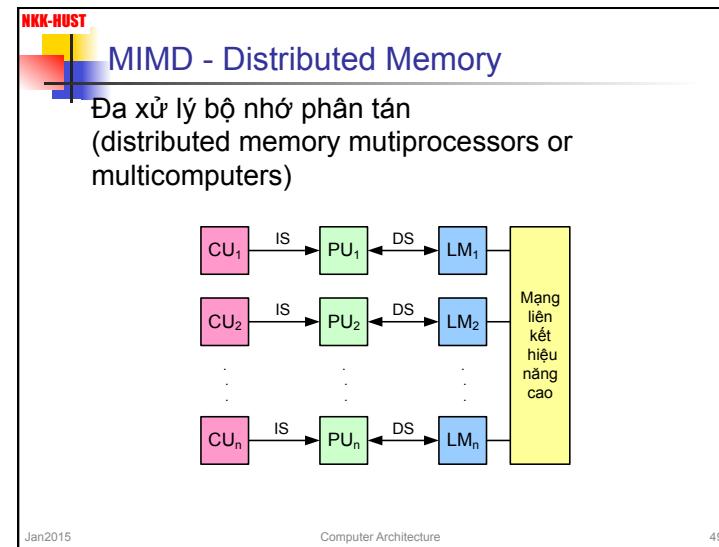
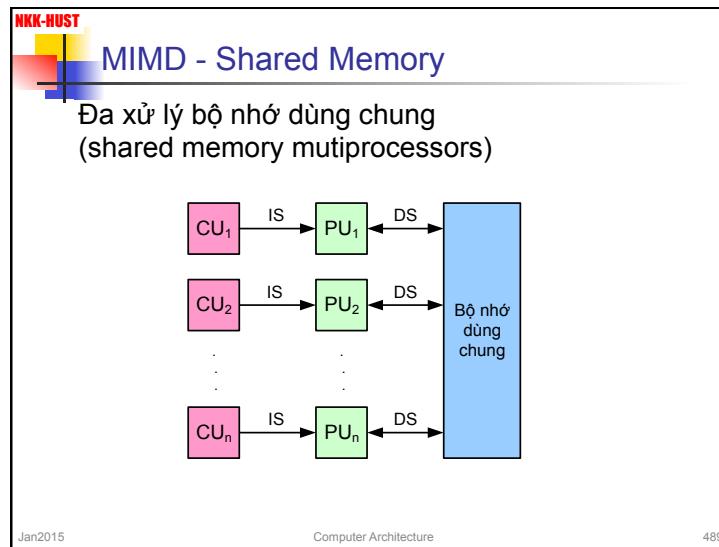
Jan2015 Computer Architecture 484

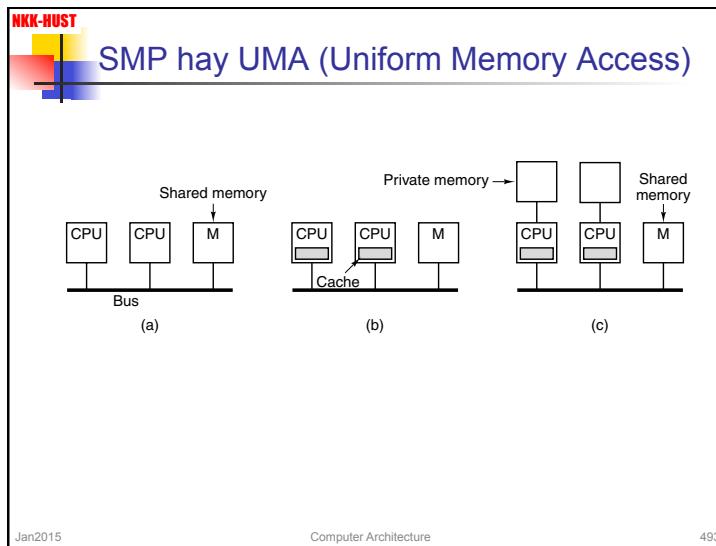


- NKK-HUST**
- ### SIMD (tiếp)
- Đơn dòng lệnh điều khiển đồng thời các đơn vị xử lý PUs
 - Mỗi phần tử xử lý có một bộ nhớ dữ liệu riêng LM (local memory)
 - Mỗi lệnh được thực hiện trên một tập các dữ liệu khác nhau
 - Các mô hình SIMD
 - Vector Computer
 - Array processor
- Jan2015 Computer Architecture 486

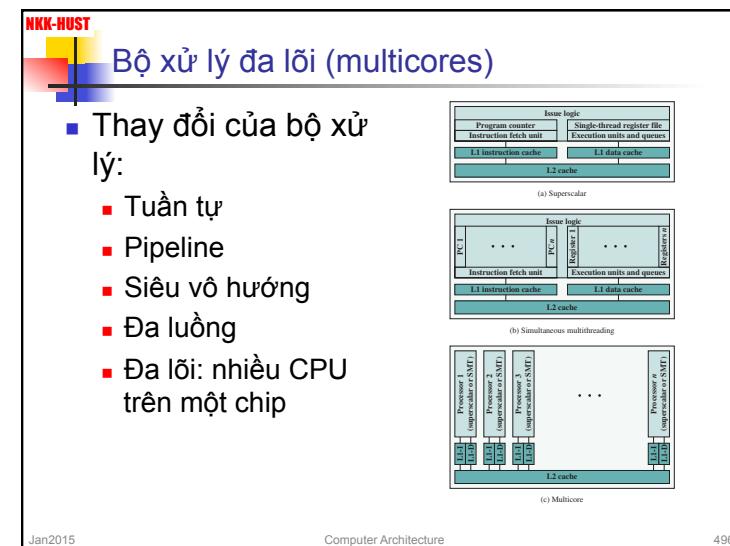
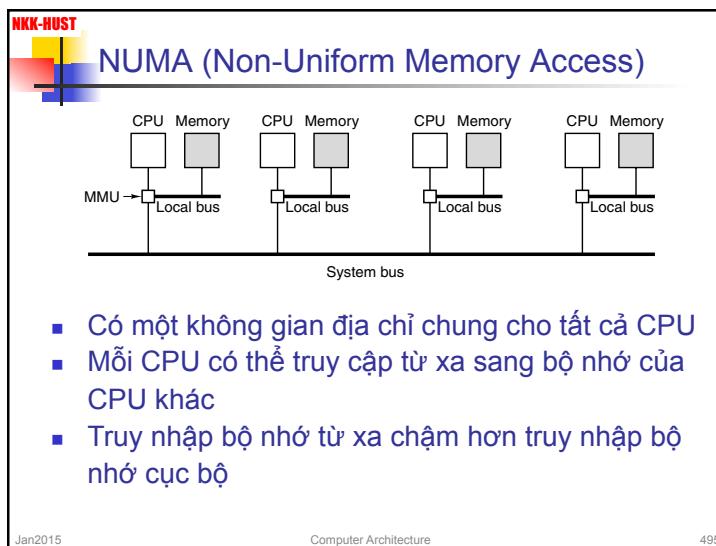


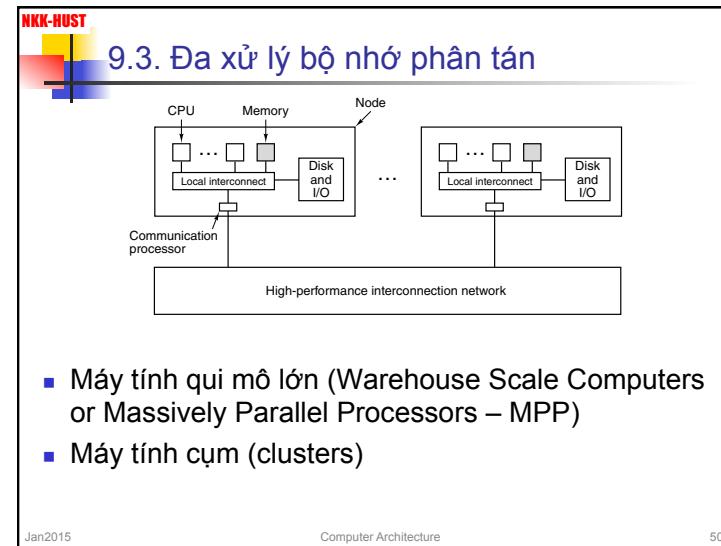
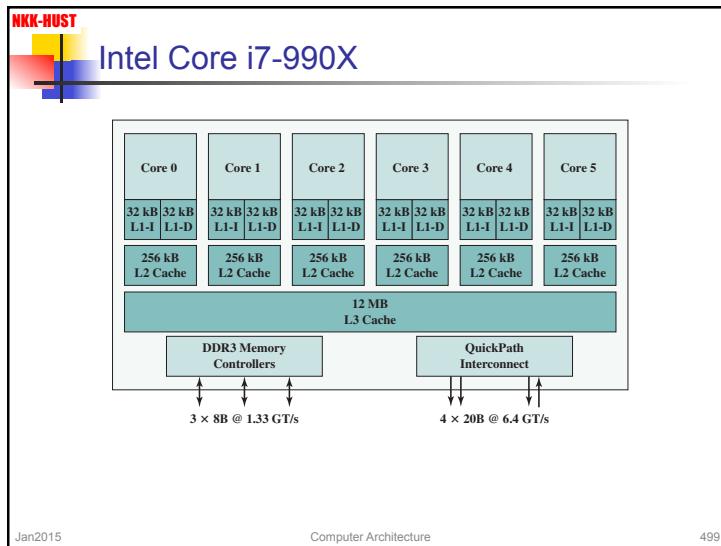
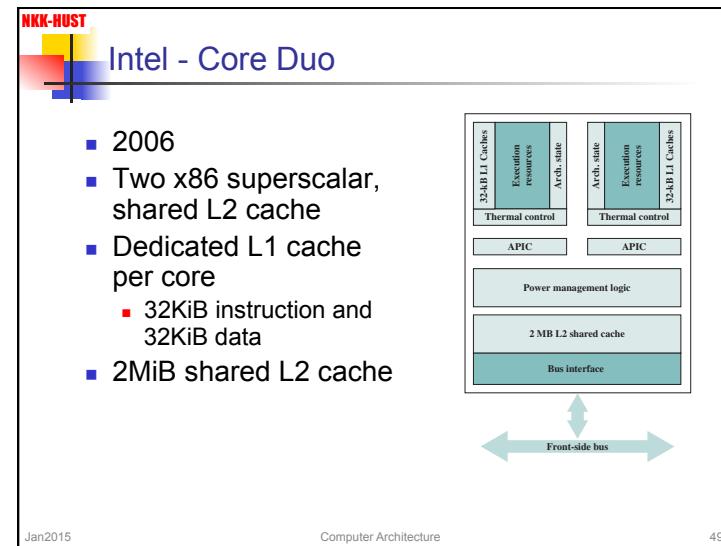
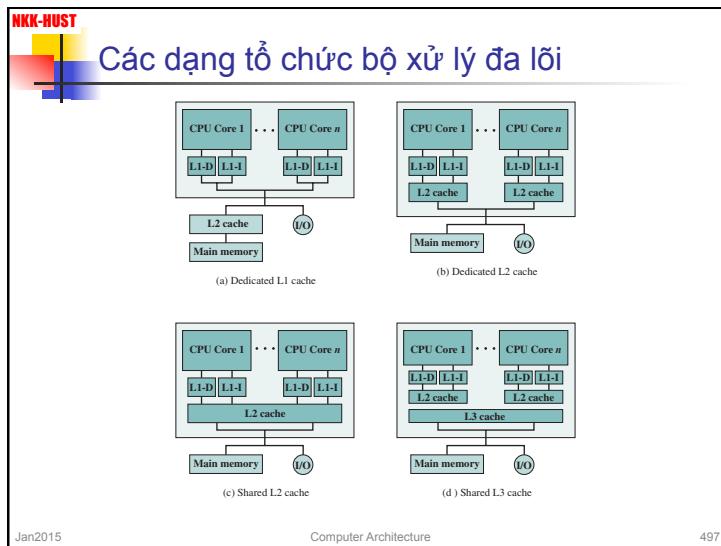
- NKK-HUST**
- ### MIMD
- Tập các bộ xử lý
 - Các bộ xử lý đồng thời thực hiện các dãy lệnh khác nhau trên các dữ liệu khác nhau
 - Các mô hình MIMD
 - Multiprocessors (Shared Memory)
 - Multicomputers (Distributed Memory)
- Jan2015 Computer Architecture 488

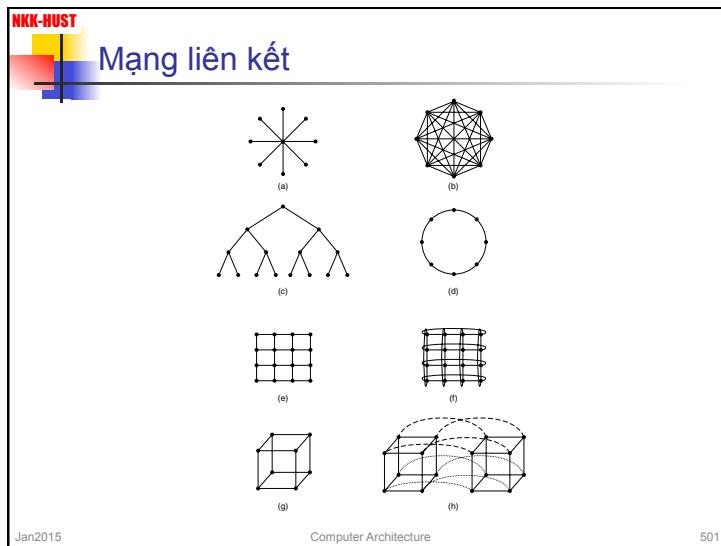




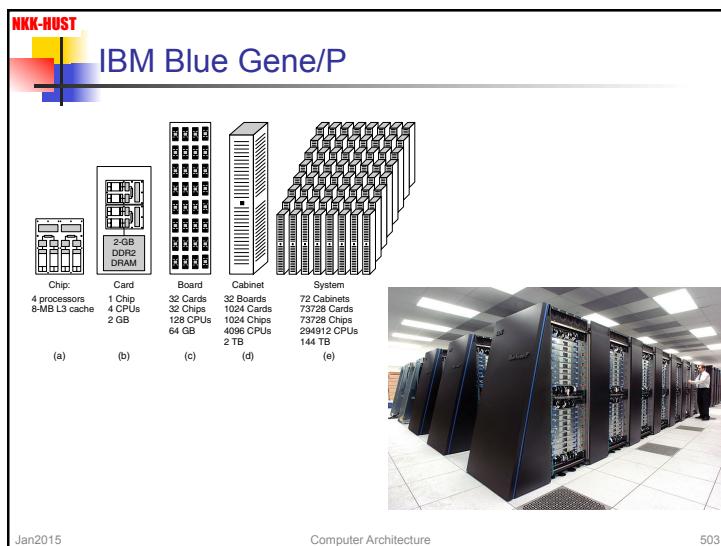
- NKK-HUST**
- ### SMP (tiếp)
- Một máy tính có $n \geq 2$ bộ xử lý giống nhau
 - Các bộ xử lý dùng chung bộ nhớ và hệ thống vào-ra
 - Thời gian truy cập bộ nhớ là bằng nhau với các bộ xử lý
 - Các bộ xử lý có thể thực hiện chức năng giống nhau
 - Hệ thống được điều khiển bởi một hệ điều hành phân tán
 - Hiệu năng: Các công việc có thể thực hiện song song
 - Khả năng chịu lỗi
- Jan2015 Computer Architecture 494







- 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
- Jan2015 Computer Architecture 502



- 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
- Jan2015 Computer Architecture 504

