CHƯƠNG 1: GIỚI THIỆU TỔNG QUAN

### **CHƯƠNG 1: GIỚI THIỆU TỔNG QUAN**

**Mục tiêu của chương này là gì?**

Sau khi học xong chương này, bạn sẽ có thể:

* Phân biệt được sự khác nhau giữa **Kiến trúc máy tính (Architecture)** và **Tổ chức máy tính (Organization)**.
* Hiểu được khái niệm **hệ thống phân cấp (hierarchical system)**.
* Nắm rõ các **chức năng cơ bản** và các **thành phần cấu trúc chính** của một máy tính.

#### **1.1. Tổ Chức và Kiến Trúc Máy Tính**

* **Kiến trúc Máy tính (Computer Architecture):**
  + Đây là những thuộc tính của hệ thống mà **một lập trình viên có thể nhìn thấy được** (visible to a programmer).
  + Các thuộc tính này có tác động trực tiếp đến việc thực thi logic của một chương trình.
  + **Ví dụ về các thuộc tính kiến trúc:** tập lệnh (instruction set), số bit dùng để biểu diễn các kiểu dữ liệu (như số, ký tự), cơ chế I/O, và các kỹ thuật định địa chỉ bộ nhớ.
  + Một thuật ngữ khác thường được dùng thay thế là **Kiến trúc tập lệnh (Instruction Set Architecture - ISA)**.
* **Tổ chức Máy tính (Computer Organization):**
  + Đây là cách các **đơn vị hoạt động (operational units) và các kết nối giữa chúng** hiện thực hóa các đặc tả kiến trúc.
  + Đây là các chi tiết phần cứng mà **lập trình viên không nhìn thấy được** (transparent to the programmer).
  + **Ví dụ về các thuộc tính tổ chức:** các tín hiệu điều khiển, giao diện giữa máy tính và các thiết bị ngoại vi, và công nghệ bộ nhớ được sử dụng.
* **Ví dụ để phân biệt:**
  + Việc một máy tính có lệnh nhân (multiply instruction) hay không là một vấn đề về **kiến trúc**.
  + Lệnh nhân đó được thực hiện bằng một bộ nhân đặc biệt hay bằng cách sử dụng lặp lại bộ cộng có sẵn là một vấn đề về **tổ chức**. Quyết định về tổ chức có thể dựa trên tần suất sử dụng lệnh, tốc độ tương đối của hai cách tiếp cận, và chi phí.
* **Mối quan hệ giữa Kiến trúc và Tổ chức:**
  + Một kiến trúc có thể tồn tại qua nhiều năm và được triển khai trên nhiều mẫu máy tính khác nhau, trong khi tổ chức của nó thay đổi theo công nghệ.
  + **Ví dụ điển hình là kiến trúc IBM System/370**, được giới thiệu lần đầu vào năm 1970 và tồn tại đến ngày nay trên dòng máy chủ mainframe của IBM. Các mẫu máy tính khác nhau trong cùng một họ có cùng kiến trúc nhưng khác nhau về tổ chức, dẫn đến giá cả và hiệu năng khác nhau.

#### **1.2. Cấu Trúc và Chức Năng (Structure and Function)**

Máy tính là một hệ thống phức tạp, và để hiểu nó, chúng ta cần nhận ra **bản chất phân cấp (hierarchical nature)** của nó. Ở mỗi cấp độ, chúng ta quan tâm đến cấu trúc và chức năng:

* **Cấu trúc (Structure):** Là cách các thành phần được kết nối với nhau.
* **Chức năng (Function):** Là hoạt động của từng thành phần riêng lẻ như một phần của cấu trúc.

**A. Các Chức Năng Cơ Bản của Máy Tính**

Về cơ bản, máy tính có thể thực hiện bốn chức năng chính:

1. **Xử lý dữ liệu (Data Processing):** Thực hiện các phép tính và thao tác logic trên dữ liệu.
2. **Lưu trữ dữ liệu (Data Storage):** Lưu trữ dữ liệu tạm thời (short-term) và dài hạn (long-term).
3. **Di chuyển dữ liệu (Data Movement):** Vận chuyển dữ liệu giữa máy tính và môi trường bên ngoài. Quá trình này được gọi là **Nhập/Xuất (Input/Output - I/O)** khi thiết bị được kết nối trực tiếp, hoặc **truyền thông dữ liệu (data communications)** khi di chuyển qua khoảng cách xa.
4. **Điều khiển (Control):** Một đơn vị điều khiển quản lý các tài nguyên của máy tính và điều phối hoạt động của các bộ phận chức năng để đáp ứng các lệnh.

**B. Các Thành Phần Cấu Trúc Chính**

* **Cấu trúc tổng thể của máy tính:**
  + **CPU (Central Processing Unit - Bộ xử lý trung tâm):** Điều khiển hoạt động của máy tính và thực hiện các chức năng xử lý dữ liệu. Thường được gọi đơn giản là bộ xử lý (processor).
  + **Bộ nhớ chính (Main Memory):** Lưu trữ dữ liệu.
  + **I/O (Nhập/Xuất):** Di chuyển dữ liệu giữa máy tính và môi trường bên ngoài.
  + **Kết nối hệ thống (System Interconnection):** Cơ chế cung cấp giao tiếp giữa CPU, bộ nhớ chính và I/O, ví dụ như **bus hệ thống (system bus)**.
* **Cấu trúc của CPU:**
  + **Bộ điều khiển (Control Unit):** Điều khiển hoạt động của CPU và do đó là toàn bộ máy tính.
  + **ALU (Arithmetic and Logic Unit - Đơn vị Số học và Logic):** Thực hiện các chức năng xử lý dữ liệu của máy tính.
  + **Thanh ghi (Registers):** Cung cấp bộ lưu trữ bên trong CPU.
  + **Kết nối CPU (CPU Interconnection):** Cơ chế giao tiếp giữa bộ điều khiển, ALU và các thanh ghi.

#### **1.3. Lịch Sử Tóm Tắt về Máy Tính**

Sự phát triển của máy tính thường được chia thành các thế hệ dựa trên công nghệ phần cứng cơ bản:

* **Thế hệ thứ nhất: Đèn chân không (Vacuum Tubes)**
  + **Công nghệ:** Sử dụng đèn chân không cho các phần tử logic số và bộ nhớ.
  + **Máy tính tiêu biểu:** **IAS computer**, được thiết kế bởi John von Neumann tại Viện Nghiên cứu Cao cấp Princeton.
  + **Khái niệm chương trình được lưu trữ (stored-program concept):** Đây là một ý tưởng nền tảng, trong đó cả **dữ liệu và lệnh đều được lưu trữ trong cùng một bộ nhớ chính**. Hầu hết các máy tính ngày nay đều dựa trên khái niệm này và được gọi là *máy von Neumann*.
  + **Cấu trúc máy IAS:** Gồm có bộ nhớ chính, ALU, bộ điều khiển, và thiết bị I/O. Bộ nhớ của IAS chứa 4096 "từ" (word), mỗi từ 40 bit. Mỗi từ có thể chứa một số 40-bit hoặc hai lệnh 20-bit.
* **Thế hệ thứ hai: Transistor (Bóng bán dẫn)**
  + **Công nghệ:** Thay thế đèn chân không bằng transistor, một thiết bị bán dẫn (solid-state) làm từ silicon. Transistor nhỏ hơn, rẻ hơn, và tỏa ít nhiệt hơn đèn chân không.
  + Thế hệ này chứng kiến sự ra đời của các ALU và bộ điều khiển phức tạp hơn, ngôn ngữ lập trình bậc cao, và phần mềm hệ thống (system software).
  + **Máy tính tiêu biểu:** IBM 7094. Một cải tiến quan trọng là sử dụng **kênh dữ liệu (data channels)** – các module I/O độc lập có bộ xử lý riêng, giúp giảm tải cho CPU.
* **Thế hệ thứ ba: Mạch tích hợp (Integrated Circuits - IC)**
  + **Công nghệ:** Sự phát minh ra mạch tích hợp (IC) vào năm 1958, cho phép chế tạo toàn bộ một mạch trên một mảnh silicon nhỏ. Điều này dẫn đến sự ra đời của ngành vi điện tử (microelectronics).
  + Các thành phần cơ bản của máy tính số là **cổng logic (gates)** và **ô nhớ (memory cells)**.
  + **Định luật Moore:** Gordon Moore (đồng sáng lập Intel) vào năm 1965 đã nhận thấy rằng số lượng transistor trên một chip đơn sẽ tăng gấp đôi sau mỗi 18 tháng. Định luật này đã thúc đẩy sự phát triển nhanh chóng về mật độ, tốc độ, đồng thời giảm chi phí và điện năng tiêu thụ.
  + **Máy tính tiêu biểu:** IBM System/360 và DEC PDP-8.
* **Các thế hệ sau này (Later Generations)**
  + Đặc trưng bởi sự tiến bộ trong công nghệ IC: LSI (Large Scale Integration), VLSI (Very Large Scale Integration), và ULSI (Ultra Large Scale Integration).
  + **Bộ nhớ bán dẫn (Semiconductor Memory):** Vào năm 1970, Fairchild sản xuất bộ nhớ bán dẫn đầu tiên. Đến năm 1974, giá mỗi bit của bộ nhớ bán dẫn đã giảm xuống thấp hơn bộ nhớ lõi từ (magnetic-core memory), dẫn đến việc sử dụng phổ biến và sự ra đời của máy tính cá nhân.
  + **Vi xử lý (Microprocessor):** Năm 1971, Intel phát triển chip 4004, **chip đầu tiên chứa tất cả các thành phần của một CPU trên một chip duy nhất**. Tiếp theo là các vi xử lý 8-bit, 16-bit, 32-bit, và 64-bit, với sự phát triển mạnh mẽ của dòng Intel x86.

CHƯƠNG 2: CÁC VẤN ĐỀ VỀ HIỆU NĂNG

### **CHƯƠNG 2: CÁC VẤN ĐỀ VỀ HIỆU NĂNG**

Mục tiêu của chương này là giúp bạn hiểu rõ các yếu tố cốt lõi ảnh hưởng đến hiệu năng của hệ thống máy tính và các phương pháp đánh giá hiệu năng. Các nội dung chính bao gồm:

* Hiểu các vấn đề then chốt liên quan đến thiết kế máy tính vì hiệu năng.
* Giải thích lý do của việc chuyển dịch sang kiến trúc đa lõi (multicore) và sự cân bằng giữa tài nguyên bộ đệm (cache) và bộ xử lý trên một chip.
* Phân biệt giữa các kiến trúc đa lõi (multicore), nhiều lõi tích hợp (MICs), và GPGPU.
* Nắm được các phương pháp cơ bản để đánh giá hiệu năng máy tính, bao gồm các bộ benchmark của SPEC.
* Hiểu sự khác biệt giữa các phép tính trung bình số học, trung bình điều hòa và trung bình nhân.

#### **2.1. Thiết Kế vì Hiệu Năng (Designing for Performance)**

Trong nhiều năm, chi phí của hệ thống máy tính liên tục giảm trong khi hiệu năng và dung lượng lại không ngừng tăng lên. Các ứng dụng hiện đại trên máy tính cá nhân như **xử lý hình ảnh, dựng hình 3D, nhận dạng giọng nói, và hội nghị truyền hình** đòi hỏi sức mạnh tính toán rất lớn. Tương tự, các doanh nghiệp cũng dựa vào những máy chủ ngày càng mạnh mẽ để xử lý giao dịch và cơ sở dữ liệu.

Mặc dù các khối xây dựng cơ bản của máy tính ngày nay gần như không thay đổi so với máy tính IAS hơn 50 năm trước, các kỹ thuật để tối đa hóa hiệu năng đã trở nên vô cùng tinh vi.

**A. Tốc độ Vi xử lý (Microprocessor Speed)**

* **Sự phát triển theo Định luật Moore:** Các nhà sản xuất chip liên tục cho ra đời các thế hệ chip mới với số lượng transistor gấp bốn lần sau mỗi ba năm. Điều này giúp cải thiện hiệu năng từ bốn đến năm lần sau mỗi ba năm.
* **Các kỹ thuật tăng hiệu năng:** Tốc độ của vi xử lý sẽ không đạt được tiềm năng tối đa nếu không được cung cấp một luồng công việc liên tục dưới dạng các lệnh máy. Các kỹ thuật được tích hợp trong bộ xử lý hiện đại để đảm bảo luồng lệnh trôi chảy bao gồm:
  + **Pipelining (Đường ống lệnh):** Cho phép bộ xử lý làm việc đồng thời trên nhiều lệnh bằng cách thực hiện các giai đoạn khác nhau của nhiều lệnh cùng một lúc.
  + **Branch Prediction (Dự đoán rẽ nhánh):** Bộ xử lý dự đoán các nhánh lệnh có khả năng được thực thi tiếp theo để tìm nạp trước các lệnh chính xác, giữ cho bộ xử lý luôn bận rộn.
  + **Superscalar Execution (Thực thi siêu vô hướng):** Khả năng phát hành nhiều hơn một lệnh trong mỗi chu kỳ xung nhịp của bộ xử lý, sử dụng nhiều đường ống song song.
  + **Data Flow Analysis (Phân tích luồng dữ liệu):** Bộ xử lý phân tích sự phụ thuộc giữa các lệnh để tạo ra một lịch trình thực thi tối ưu, không phụ thuộc vào thứ tự chương trình gốc.
  + **Speculative Execution (Thực thi suy đoán):** Bộ xử lý thực thi trước các lệnh có khả năng sẽ cần đến, dựa trên dự đoán rẽ nhánh và phân tích luồng dữ liệu, để giữ cho các đơn vị thực thi của nó luôn bận rộn.

**B. Cân bằng Hiệu năng (Performance Balance)**

Trong khi sức mạnh của bộ xử lý đã tăng vọt, các thành phần quan trọng khác của máy tính lại không theo kịp, dẫn đến nhu cầu **cân bằng hiệu năng** giữa các thành phần.

* **Vấn đề tốc độ giữa bộ xử lý và bộ nhớ chính:** Tốc độ truyền dữ liệu giữa bộ nhớ chính và bộ xử lý đã tụt hậu nghiêm trọng so với tốc độ xử lý, tạo ra một nút thắt cổ chai quan trọng.
* **Các giải pháp cho vấn đề này bao gồm:**
  + Tăng số bit được truy xuất cùng một lúc bằng cách làm cho DRAM "rộng hơn" và sử dụng các đường bus dữ liệu rộng.
  + Thay đổi giao diện DRAM để hiệu quả hơn bằng cách tích hợp bộ đệm (cache) trên chip DRAM.
  + Giảm tần suất truy cập bộ nhớ bằng cách kết hợp các cấu trúc cache ngày càng phức tạp và hiệu quả giữa bộ xử lý và bộ nhớ chính.
  + Tăng băng thông kết nối giữa bộ xử lý và bộ nhớ bằng cách sử dụng các bus tốc độ cao hơn và một hệ thống bus phân cấp.
* **Xử lý các thiết bị I/O:** Các thiết bị ngoại vi hiện đại như đồ họa, mạng tốc độ cao, và ổ đĩa cứng tạo ra nhu cầu thông lượng dữ liệu khổng lồ. Các chiến lược để giải quyết vấn đề này bao gồm các sơ đồ đệm, bus kết nối tốc độ cao hơn, và cấu hình đa bộ xử lý.

**C. Cải tiến trong Tổ chức và Kiến trúc Chip**

Theo truyền thống, yếu tố chính trong việc tăng hiệu năng là tăng tốc độ xung nhịp và mật độ logic. Tuy nhiên, khi tốc độ xung nhịp và mật độ logic tăng lên, một số trở ngại trở nên đáng kể hơn:

* **Năng lượng (Power):** Mật độ năng lượng (Watts/cm²) tăng theo mật độ logic và tốc độ xung nhịp, gây khó khăn trong việc tản nhiệt.
* **Độ trễ RC (RC Delay):** Tốc độ của các electron trên chip bị giới hạn bởi điện trở (R) và điện dung (C) của các dây kim loại kết nối. Khi các thành phần nhỏ lại, dây dẫn mỏng hơn (tăng R) và gần nhau hơn (tăng C), làm tăng độ trễ.
* **Độ trễ và thông lượng bộ nhớ (Memory Latency and Throughput):** Tốc độ truy cập và truyền tải bộ nhớ tụt hậu so với tốc độ bộ xử lý.

Do đó, các nhà thiết kế ngày càng tập trung vào các cách tiếp cận về tổ chức và kiến trúc để cải thiện hiệu năng. Hai chiến lược chính đã được sử dụng là **tăng dung lượng cache** và làm cho **logic thực thi lệnh trở nên phức tạp hơn** để cho phép thực thi lệnh song song (pipelining và superscalar).

Tuy nhiên, khi các phương pháp này đạt đến giới hạn, câu trả lời trong những năm gần đây là phát triển **chip máy tính đa lõi (multicore)**.

#### **2.2. Đa lõi (Multicore), MICs, và GPGPUs**

* **Multicore (Đa lõi):** Thay vì xây dựng một bộ xử lý phức tạp hơn, chiến lược là đặt hai hoặc nhiều bộ xử lý đơn giản hơn (lõi) trên cùng một chip với một bộ đệm lớn dùng chung. Việc tăng gấp đôi số lượng bộ xử lý gần như tăng gấp đôi hiệu năng nếu phần mềm có thể hỗ trợ hiệu quả. Ngoài ra, với nhiều bộ xử lý, các bộ đệm lớn hơn được hợp lý hóa vì logic bộ nhớ tiêu thụ ít năng lượng hơn nhiều so với logic xử lý.
* **MIC (Many Integrated Core - Nhiều lõi tích hợp):** Các nhà sản xuất chip hiện đang có một bước tiến lớn về số lượng lõi trên mỗi chip, với hơn 50 lõi trên mỗi chip. Thuật ngữ MIC được sử dụng để mô tả sự nhảy vọt về hiệu năng này.
* **GPGPU (General-Purpose Computing on GPUs - Điện toán đa năng trên GPU):** Một lựa chọn thiết kế khác là một con chip với nhiều bộ xử lý đa năng cộng với các đơn vị xử lý đồ họa (GPU) và các lõi chuyên dụng khác. GPU là một lõi được thiết kế để thực hiện các hoạt động song song trên dữ liệu đồ họa và ngày càng được sử dụng như các bộ xử lý vector cho nhiều ứng dụng đòi hỏi các phép tính lặp đi lặp lại.

#### **2.3. Hai Định luật cung cấp Cái nhìn sâu sắc: Định luật Amdahl và Định luật Little**

**A. Định luật Amdahl (Amdahl’s Law)**

Định luật Amdahl, do Gene Amdahl đề xuất năm 1967, giải quyết vấn đề tăng tốc tiềm năng của một chương trình khi sử dụng nhiều bộ xử lý so với một bộ xử lý duy nhất.

* **Công thức:** Giả sử một chương trình có một phần (f) có thể được song song hóa và một phần (1-f) là tuần tự. Tốc độ tăng tốc (Speedup) khi sử dụng N bộ xử lý là: **Speedup = 1 / [(1 - f) + (f / N)]**
* **Kết luận:**
  1. Khi f nhỏ (phần song song hóa được ít), việc sử dụng nhiều bộ xử lý có ít tác dụng.
  2. Khi N tiến đến vô cùng, tốc độ tăng tốc bị giới hạn bởi **1/(1-f)**. Điều này cho thấy có sự giảm dần lợi ích khi sử dụng thêm nhiều bộ xử lý hơn.
* **Ý nghĩa:** Định luật Amdahl minh họa rằng phần mềm chạy trên các máy đa lõi phải được điều chỉnh cho một môi trường thực thi song song cao để khai thác sức mạnh của xử lý song song.

**B. Định luật Little (Little’s Law)**

Đây là một mối quan hệ cơ bản và đơn giản với các ứng dụng rộng rãi, áp dụng cho hầu hết các hệ thống ở trạng thái ổn định.

* **Công thức:** **L = λ \* W**
  + **L:** Số lượng trung bình các mục trong hệ thống tại bất kỳ thời điểm nào.
  + **λ:** Tốc độ trung bình các mục đến hệ thống.
  + **W:** Thời gian trung bình một mục ở lại trong hệ thống.
* **Ứng dụng:** Định luật này có thể được áp dụng cho một hệ thống hàng đợi (queuing system), chẳng hạn như một bộ xử lý cung cấp dịch vụ cho các tiến trình, hoặc một thiết bị I/O cung cấp dịch vụ đọc/ghi cho các yêu cầu I/O. Định luật này rất hữu ích và đã được quan tâm trở lại do các vấn đề về hiệu năng liên quan đến máy tính đa lõi.

#### **2.4. Các Thước đo Cơ bản về Hiệu năng Máy tính**

**A. Tốc độ xung nhịp (Clock Speed)**

* Các hoạt động của bộ xử lý được điều khiển bởi một **xung nhịp hệ thống (system clock)**, được đo bằng chu kỳ mỗi giây, hay Hertz (Hz).
* **Tốc độ xung nhịp (clock rate)** là tốc độ của các xung nhịp. Tuy nhiên, việc so sánh tốc độ xung nhịp trên các bộ xử lý khác nhau không nói lên toàn bộ câu chuyện về hiệu năng vì hầu hết các lệnh đòi hỏi nhiều chu kỳ xung nhịp để hoàn thành.

**B. Tốc độ thực thi lệnh (Instruction Execution Rate)**

* Một thước đo quan trọng là **số chu kỳ trung bình trên mỗi lệnh (CPI - cycles per instruction)** cho một chương trình.
* Thời gian xử lý (T) để thực thi một chương trình có thể được biểu thị là: **T = Ic \* CPI \* τ**, trong đó Ic là số lượng lệnh và τ là thời gian chu kỳ xung nhịp.
* Một thước đo hiệu năng phổ biến là **MIPS (millions of instructions per second - hàng triệu lệnh mỗi giây)**, có thể được biểu thị là: **MIPS rate = f / (CPI \* 10^6)**, trong đó f là tần số xung nhịp.
* Hiệu năng dấu phẩy động được biểu thị bằng **MFLOPS (millions of floating-point operations per second - hàng triệu phép toán dấu phẩy động mỗi giây)**.

#### **2.5. Tính toán giá trị Trung bình (Calculating the Mean)**

Ba công thức phổ biến được sử dụng để tính giá trị trung bình là trung bình số học, trung bình nhân, và trung bình điều hòa.

* **Trung bình số học (Arithmetic Mean - AM):** Phù hợp nếu tổng của tất cả các phép đo là một giá trị có ý nghĩa, ví dụ như thời gian thực thi chương trình.
* **Trung bình điều hòa (Harmonic Mean - HM):** Phù hợp với các đại lượng dạng tỷ lệ (rate), như tốc độ thực thi lệnh (MIPS hoặc MFLOPS), vì nó tỷ lệ nghịch với tổng thời gian thực thi.
* **Trung bình nhân (Geometric Mean - GM):** Hữu ích cho việc so sánh hiệu năng tương đối của các máy. Nó cung cấp kết quả nhất quán khi đo lường các giá trị đã được chuẩn hóa so với một máy tham chiếu, và đây là lý do SPEC đã chọn sử dụng nó.

#### **2.6. Benchmarks và SPEC**

* Các thước đo như MIPS và MFLOPS đã tỏ ra không đủ để đánh giá hiệu năng của bộ xử lý do sự khác biệt trong tập lệnh.
* **Benchmark** là một chương trình được viết bằng ngôn ngữ bậc cao, có thể biên dịch và thực thi trên bất kỳ máy tính nào. Các bộ benchmark cung cấp một bài kiểm tra đại diện cho một máy tính trong một lĩnh vực ứng dụng cụ thể.
* **SPEC (Standard Performance Evaluation Corporation)** là một hiệp hội công nghiệp định nghĩa và duy trì các bộ benchmark được biết đến rộng rãi nhất.
* **SPEC CPU2006** là bộ benchmark tiêu chuẩn công nghiệp cho các ứng dụng chuyên sâu về bộ xử lý. Nó bao gồm 12 benchmark số nguyên và 17 benchmark dấu phẩy động, được rút ra từ các ứng dụng thực tế.
* Quá trình đánh giá bao gồm việc chạy mỗi chương trình benchmark ba lần, chọn giá trị trung vị, sau đó chuẩn hóa kết quả bằng cách tính tỷ lệ thời gian chạy của máy tham chiếu so với thời gian chạy của hệ thống đang kiểm tra. Cuối cùng, **trung bình nhân (geometric mean)** của các tỷ lệ được tính để đưa ra chỉ số tổng thể.

CHƯƠNG 3: TỔNG QUAN VỀ CHỨC NĂNG VÀ KẾT NỐI

### **CHƯƠNG 3: CÁI NHÌN TỔNG QUAN VỀ CHỨC NĂNG VÀ KẾT NỐI MÁY TÍNH**

**Mục tiêu của chương này là gì?**

Sau khi học xong chương này, bạn sẽ có thể:

* Hiểu các thành phần cơ bản của một chu kỳ lệnh và vai trò của ngắt (interrupts).
* Mô tả khái niệm kết nối trong một hệ thống máy tính.
* Đánh giá những ưu điểm tương đối của kết nối điểm-tới-điểm so với kết nối bus.

#### **3.1. Các thành phần của máy tính**

Hầu hết các thiết kế máy tính đương đại đều dựa trên các khái niệm được phát triển bởi John von Neumann tại Viện Nghiên cứu Cao cấp Princeton. Kiến trúc này, được gọi là **kiến trúc von Neumann**, dựa trên ba khái niệm chính:

1. **Dữ liệu và lệnh được lưu trữ trong cùng một bộ nhớ đọc-ghi duy nhất**.
2. Nội dung của bộ nhớ này có thể được định địa chỉ theo vị trí, bất kể loại dữ liệu chứa trong đó.
3. Việc thực thi diễn ra theo trình tự (tuần tự), từ lệnh này đến lệnh tiếp theo (trừ khi có sự thay đổi rõ ràng).

Lý do đằng sau các khái niệm này là một tập hợp nhỏ các thành phần logic cơ bản có thể được kết hợp theo nhiều cách khác nhau để lưu trữ dữ liệu nhị phân và thực hiện các phép toán số học và logic trên dữ liệu đó. Thay vì phải đấu nối lại phần cứng cho mỗi chương trình mới (chương trình có dây cứng - hardwired program), người ta xây dựng một cấu hình phần cứng đa năng. Hệ thống này chấp nhận cả dữ liệu và tín hiệu điều khiển để tạo ra kết quả. Một chuỗi các mã lệnh, được gọi là **phần mềm (software)**, được cung cấp để tạo ra các tín hiệu điều khiển này.

Một hệ thống máy tính có các thành phần cấp cao sau đây (như trong Hình 3.2 của sách giáo khoa):

* **CPU (Central Processing Unit - Bộ xử lý trung tâm):** Bao gồm một bộ thông dịch lệnh và một module chức năng số học và logic đa năng. CPU trao đổi dữ liệu với bộ nhớ.
* **Bộ nhớ chính (Main Memory):** Lưu trữ cả dữ liệu và lệnh.
* **Các thành phần I/O (Input/Output - Nhập/Xuất):** Chịu trách nhiệm nhận dữ liệu và lệnh và báo cáo kết quả.

Để trao đổi dữ liệu, CPU sử dụng các thanh ghi sau:

* **MAR (Memory Address Register - Thanh ghi địa chỉ bộ nhớ):** Chỉ định địa chỉ trong bộ nhớ cho lần đọc hoặc ghi tiếp theo.
* **MBR (Memory Buffer Register - Thanh ghi đệm bộ nhớ):** Chứa dữ liệu sẽ được ghi vào bộ nhớ hoặc nhận dữ liệu được đọc từ bộ nhớ.
* **I/OAR (I/O Address Register - Thanh ghi địa chỉ I/O):** Chỉ định một thiết bị I/O cụ thể.
* **I/OBR (I/O Buffer Register - Thanh ghi đệm I/O):** Được sử dụng để trao đổi dữ liệu giữa một module I/O và CPU.

#### **3.2. Chức năng của máy tính**

Chức năng cơ bản của một máy tính là thực thi một chương trình, bao gồm một tập hợp các lệnh được lưu trữ trong bộ nhớ. Quá trình này được gọi là **chu kỳ lệnh (instruction cycle)**, bao gồm hai bước chính: **chu kỳ tìm nạp (fetch cycle)** và **chu kỳ thực thi (execute cycle)**.

**A. Tìm nạp và Thực thi lệnh (Instruction Fetch and Execute)**

* **Chu kỳ Tìm nạp:**
  + Bắt đầu mỗi chu kỳ lệnh, bộ xử lý tìm nạp một lệnh từ bộ nhớ.
  + **Thanh ghi PC (Program Counter - Bộ đếm chương trình)** giữ địa chỉ của lệnh tiếp theo sẽ được tìm nạp.
  + Sau mỗi lần tìm nạp, PC được tăng lên để nó sẽ tìm nạp lệnh tiếp theo trong chuỗi.
  + Lệnh được tìm nạp sẽ được nạp vào **thanh ghi lệnh (Instruction Register - IR)**.
* **Chu kỳ Thực thi:**
  + Bộ xử lý diễn giải lệnh trong IR và thực hiện hành động cần thiết. Các hành động này thuộc bốn loại:
    1. **Bộ xử lý - bộ nhớ (Processor-memory):** Truyền dữ liệu từ bộ xử lý đến bộ nhớ hoặc ngược lại.
    2. **Bộ xử lý - I/O (Processor-I/O):** Truyền dữ liệu đến hoặc từ một thiết bị ngoại vi.
    3. **Xử lý dữ liệu (Data processing):** Thực hiện một số phép toán số học hoặc logic trên dữ liệu.
    4. **Điều khiển (Control):** Một lệnh có thể chỉ định rằng chuỗi thực thi bị thay đổi.

**B. Ngắt (Interrupts)**

Hầu hết tất cả các máy tính đều cung cấp một cơ chế mà các module khác (như I/O, bộ nhớ) có thể làm gián đoạn quá trình xử lý bình thường của bộ xử lý. Ngắt chủ yếu được cung cấp để **cải thiện hiệu quả xử lý**, vì các thiết bị ngoại vi thường chậm hơn nhiều so với bộ xử lý.

* **Hoạt động không có ngắt:** Nếu không có ngắt, khi bộ xử lý ra lệnh I/O, nó phải tạm dừng và chờ cho đến khi hoạt động I/O hoàn tất. Đây là một sự lãng phí thời gian của bộ xử lý.
* **Hoạt động có ngắt:** Với ngắt, bộ xử lý có thể thực hiện các lệnh khác trong khi một hoạt động I/O đang diễn ra.  
  + Khi một thiết bị ngoại vi sẵn sàng, module I/O của nó sẽ gửi một **tín hiệu yêu cầu ngắt** đến bộ xử lý.
  + Bộ xử lý tạm ngưng hoạt động của chương trình hiện tại, chuyển sang một chương trình dịch vụ (**trình xử lý ngắt - interrupt handler**), và sau đó tiếp tục thực thi ban đầu sau khi thiết bị được phục vụ.
  + Để xử lý ngắt, một **chu kỳ ngắt (interrupt cycle)** được thêm vào chu kỳ lệnh. Trong chu kỳ này, bộ xử lý kiểm tra xem có ngắt nào đang chờ xử lý không.
* **Xử lý nhiều ngắt:**
  + **Vô hiệu hóa ngắt:** Một phương pháp là vô hiệu hóa các ngắt trong khi một ngắt đang được xử lý, xử lý chúng theo thứ tự tuần tự nghiêm ngặt.
  + **Ngắt ưu tiên:** Một phương pháp khác là định nghĩa các mức ưu tiên cho các ngắt, cho phép một ngắt có mức ưu tiên cao hơn làm gián đoạn một trình xử lý ngắt có mức ưu tiên thấp hơn.

**C. Chức năng I/O**

Một module I/O có thể trao đổi dữ liệu trực tiếp với bộ xử lý. Trong một số trường hợp, việc trao đổi I/O trực tiếp với bộ nhớ sẽ hiệu quả hơn. Trong trường hợp này, bộ xử lý cấp cho một module I/O quyền đọc hoặc ghi vào bộ nhớ mà không cần sự tham gia của bộ xử lý. Hoạt động này được gọi là **truy cập bộ nhớ trực tiếp (Direct Memory Access - DMA)**.

#### **3.3. Cấu trúc kết nối**

Một máy tính bao gồm một tập hợp các module (bộ xử lý, bộ nhớ, I/O) giao tiếp với nhau. Tập hợp các đường dẫn kết nối các module khác nhau được gọi là **cấu trúc kết nối (interconnection structure)**. Cấu trúc này phải hỗ trợ các loại truyền dữ liệu sau:

* **Bộ nhớ đến bộ xử lý:** Bộ xử lý đọc một lệnh hoặc một đơn vị dữ liệu từ bộ nhớ.
* **Bộ xử lý đến bộ nhớ:** Bộ xử lý ghi một đơn vị dữ liệu vào bộ nhớ.
* **I/O đến bộ xử lý:** Bộ xử lý đọc dữ liệu từ một thiết bị I/O thông qua một module I/O.
* **Bộ xử lý đến I/O:** Bộ xử lý gửi dữ liệu đến một thiết bị I/O.
* **I/O đến hoặc từ bộ nhớ:** Một module I/O được phép trao đổi dữ liệu trực tiếp với bộ nhớ bằng cách sử dụng DMA.

#### **3.4. Kết nối Bus**

Một **bus** là một đường dẫn giao tiếp kết nối hai hoặc nhiều thiết bị và là một phương tiện truyền dẫn chung. Một **bus hệ thống** kết nối các thành phần chính của máy tính (bộ xử lý, bộ nhớ, I/O). Một bus hệ thống thường bao gồm ba nhóm chức năng của các đường dây:

1. **Các đường dữ liệu (Data lines):** Cung cấp một đường dẫn để di chuyển dữ liệu giữa các module hệ thống. Chúng được gọi chung là **bus dữ liệu (data bus)**.
2. **Các đường địa chỉ (Address lines):** Được sử dụng để chỉ định nguồn hoặc đích của dữ liệu trên bus dữ liệu.
3. **Các đường điều khiển (Control lines):** Được sử dụng để kiểm soát việc truy cập và sử dụng các đường dữ liệu và địa chỉ. Chúng truyền cả thông tin lệnh và thông tin thời gian.

CHƯƠNG 4: BỘ NHỚ ĐỆM (CACHE MEMORY)

### **CHƯƠNG 4: BỘ NHỚ ĐỆM (CACHE MEMORY)**

**Mục tiêu của chương này là gì?**

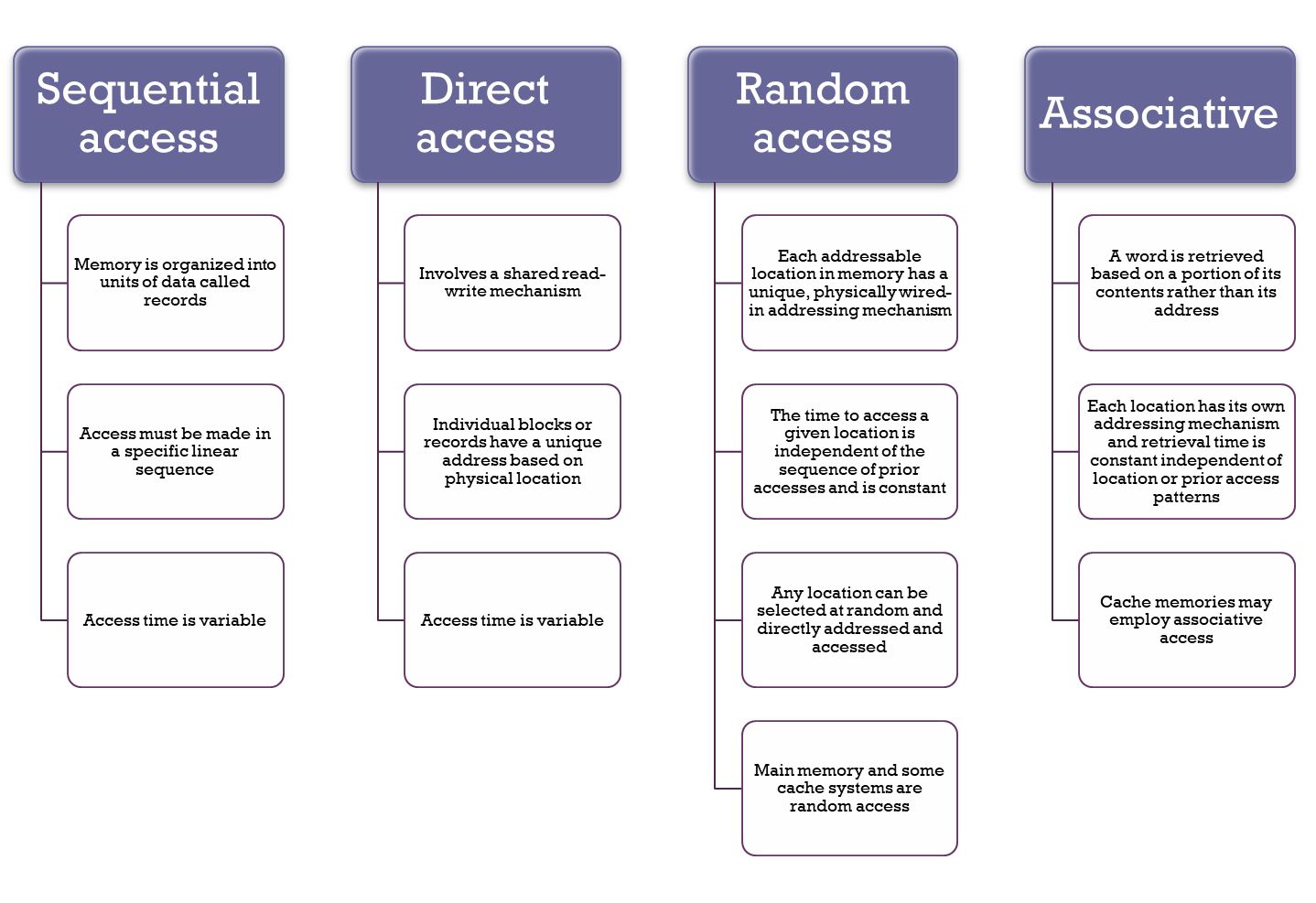
Sau khi học xong chương này, bạn sẽ có thể:

* Trình bày tổng quan về các đặc điểm chính của hệ thống bộ nhớ máy tính và việc sử dụng hệ thống phân cấp bộ nhớ.
* Mô tả các khái niệm cơ bản và mục đích của bộ nhớ đệm.
* Thảo luận về các yếu tố thiết kế chính của bộ nhớ đệm.
* Phân biệt giữa các kỹ thuật ánh xạ trực tiếp, ánh xạ liên kết và ánh xạ tập-liên kết.
* Giải thích lý do sử dụng nhiều cấp bộ nhớ đệm.

#### **4.1. Tổng Quan về Hệ Thống Bộ Nhớ Máy Tính**

* Phân biệt: bộ nhớ trong: Không thể thay thế. bộ nhớ ngoài có thể thay thế.
* Bộ nhớ đời thấp x32bit, giờ đa số các máy là x64bit
* Bộ nhớ k bền vững: RAM- Reason: Mất điện-> Mất data
* ROM: Bền vững-> Mất điện ->> K mất data.

**A. Các Đặc Điểm của Hệ Thống Bộ Nhớ**

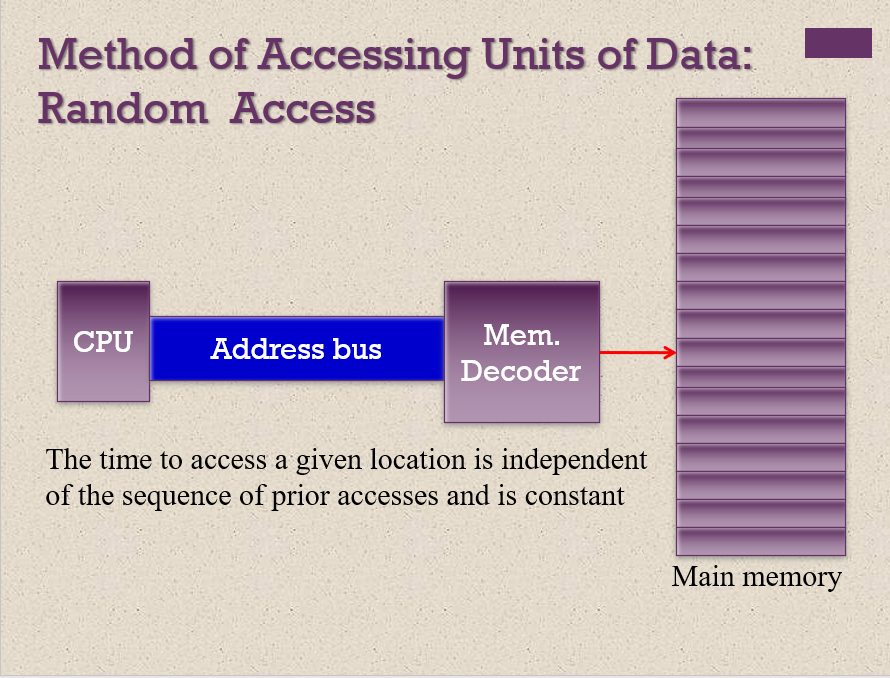
****

Hệ thống bộ nhớ máy tính có thể được phân loại dựa trên các đặc điểm chính sau đây:

* **Vị trí (Location):** Bộ nhớ có thể là **bộ nhớ trong (internal)**, như thanh ghi (register) của bộ xử lý, bộ nhớ đệm (cache) và bộ nhớ chính (main memory), hoặc **bộ nhớ ngoài (external)**, như đĩa quang, đĩa từ và băng từ.
* **Dung lượng (Capacity):** Dung lượng bộ nhớ trong thường được tính bằng byte hoặc word.
* **Đơn vị truyền (Unit of Transfer):** Đối với bộ nhớ trong, đơn vị truyền tải bằng với số lượng đường dây điện vào và ra khỏi module bộ nhớ. Đối với bộ nhớ ngoài, dữ liệu thường được truyền theo các đơn vị lớn hơn gọi là **khối (blocks)**.
* **Phương thức truy cập (Access Method):**
  + **Truy cập tuần tự (Sequential access):** Dữ liệu phải được truy cập theo một trình tự tuyến tính cụ thể, ví dụ như băng từ.
  + **Truy cập trực tiếp (Direct access):** Các khối riêng lẻ có một địa chỉ duy nhất dựa trên vị trí vật lý. Việc truy cập được thực hiện bằng cách đi thẳng đến một vùng chung rồi tìm kiếm tuần tự đến vị trí cuối cùng, ví dụ như đĩa từ.
  + **Truy cập ngẫu nhiên (Random access):** Mỗi vị trí có thể định địa chỉ đều có cơ chế dây nối vật lý riêng. Thời gian truy cập một vị trí nhất định không phụ thuộc vào chuỗi truy cập trước đó và là hằng số. Bộ nhớ chính và một số hệ thống đệm là truy cập ngẫu nhiên.
  + **Truy cập liên kết (Associative access):** Dữ liệu được truy xuất dựa trên một phần nội dung của nó thay vì địa chỉ.

VD: việc mở word -> Mở theo trực tiếp.

+NOTE: truy cập bằng phương pháp kết hợp chỉ dành cho RAM.



**->>> Việc ram làm việc ở vùng(WORLD) nào cho tác vụ nào DO- hệ điều hành quyết định**

* **Hiệu năng (Performance):** Được đo bằng **thời gian truy cập (access time)**, **thời gian chu kỳ bộ nhớ (memory cycle time)**, và **tốc độ truyền (transfer rate)**.
* hiệu năng được thể hiện bằng thời gian phản hồi 1 tác vụ nào đó.
* thời gian để bộ nhớ phản hồi còn phụ thuộc vào vị trí lưu trữ của tác vụ, dữ liệu đó.
* TRUY XUẤT nhanh hay chậm còn do bộ lọc và các thuật toán dò tìm, tối ưu hệ thống.
* memory cycle time: việc làm mới 1 chu kỳ bộ nhớ. Ví dụ xóa ram ở các ứng dụng nền trong các máy điện thoại android.

-transfer rate: tốc độ truyền tải dữ liệu/giây vào/ra của bộ nhớ: ổ cứng, usb, thẻ sd,..

* **Loại vật lý (Physical Type):** Các loại phổ biến nhất hiện nay là **bộ nhớ bán dẫn (semiconductor)**, **bộ nhớ bề mặt từ (magnetic surface)**, và **quang học (optical)**.

- Bộ nhớ bề mặt từ: băng cassete

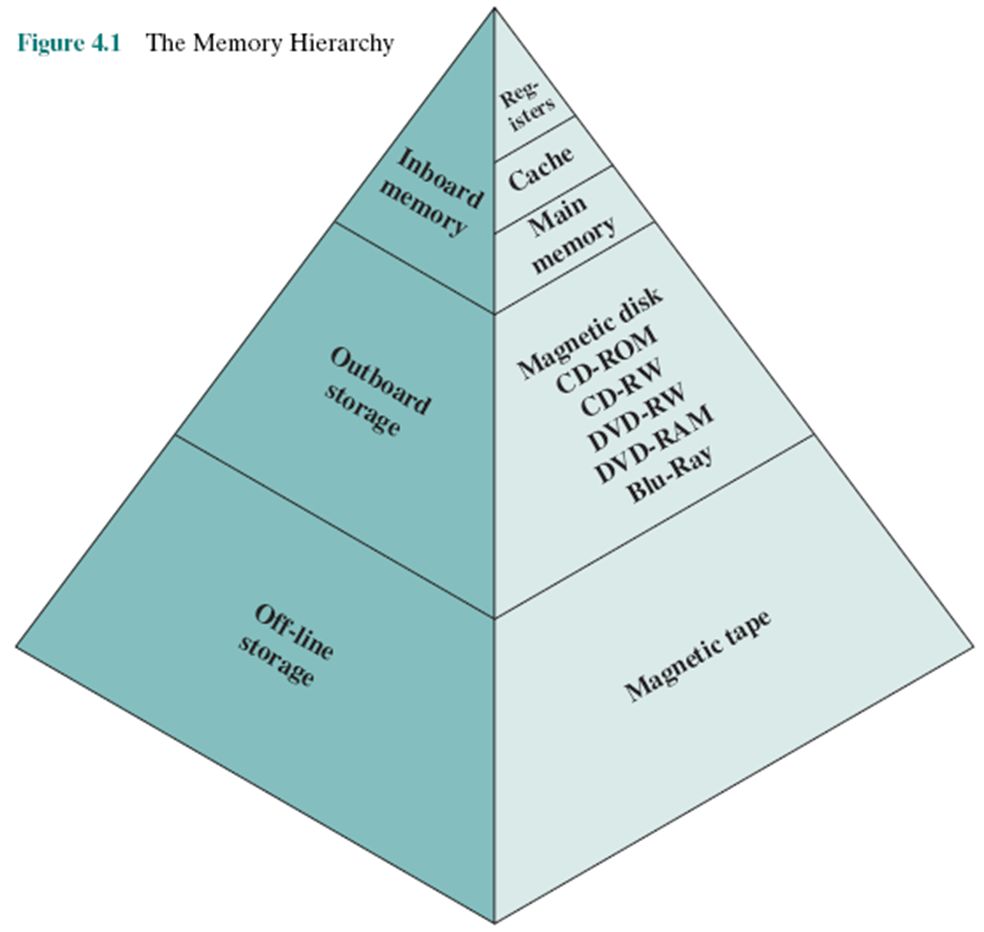
-Bộ nhớ quang học: các đĩa CD,..

-Bộ nhớ bán dẫn: sử dụng các chất bán dẫn, ưu điểm:

lưu trữ được nhiều dữ liệu, hiệu năng tiết kiệm, đường truyền lớn. ->>Tốn tiền nhiều hơn các loại khác.

* **Đặc tính vật lý:**
  + **Khả biến/Bất biến (Volatile/nonvolatile):** Bộ nhớ khả biến (volatile) mất thông tin khi mất điện, trong khi bộ nhớ bất biến (nonvolatile) thì không.
  + **Có thể xóa/Không thể xóa (Erasable/nonerasable):** Bộ nhớ không thể xóa, hay còn gọi là **bộ nhớ chỉ đọc (ROM)**, không thể bị thay đổi.
* **Tổ chức (Organization):** Đề cập đến sự sắp xếp vật lý của các bit để tạo thành các từ (word).

**B. Hệ Thống Phân Cấp Bộ Nhớ (The Memory Hierarchy)**

****

Các ràng buộc thiết kế đối với bộ nhớ của máy tính có thể được tóm gọn trong ba câu hỏi: **Bao nhiêu? Nhanh thế nào? Đắt thế nào?**. Các đặc tính này có sự đánh đổi với nhau:

* Thời gian truy cập nhanh hơn, chi phí mỗi bit lớn hơn.
* Dung lượng lớn hơn, chi phí mỗi bit nhỏ hơn.
* Dung lượng lớn hơn, thời gian truy cập chậm hơn.

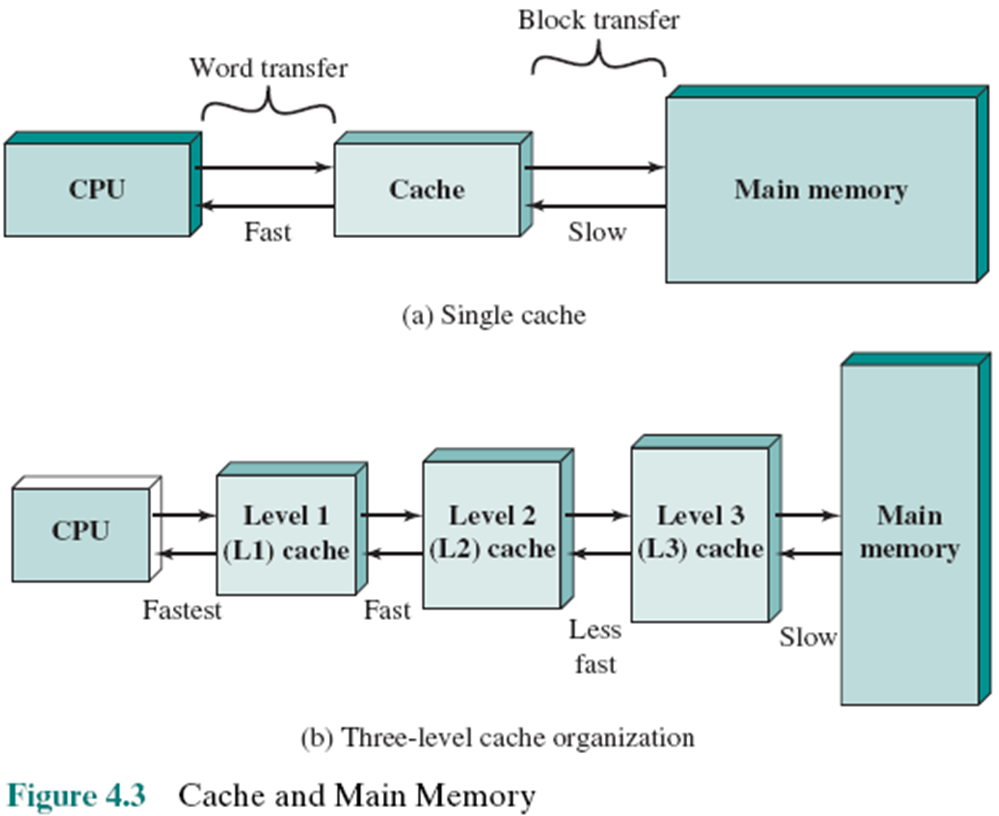
Để giải quyết tình thế tiến thoái lưỡng nan này, các nhà thiết kế sử dụng một **hệ thống phân cấp bộ nhớ**. Khi đi xuống các cấp thấp hơn trong hệ thống phân cấp (Hình 4.1 trong sách giáo khoa), các yếu tố sau sẽ xảy ra:

a. Chi phí mỗi bit giảm. b. Dung lượng tăng.

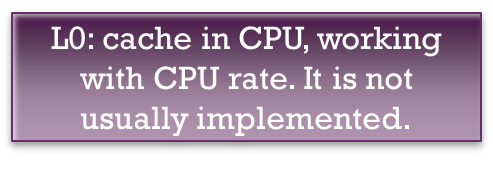
c. Thời gian truy cập tăng. d. Tần suất truy cập của bộ xử lý giảm.

Chìa khóa cho sự thành công của tổ chức này là **tính cục bộ của tham chiếu (locality of reference)**. Nguyên tắc này nói rằng các tham chiếu bộ nhớ của bộ xử lý có xu hướng tập trung lại. Do đó, có thể tổ chức dữ liệu trên các cấp sao cho phần trăm truy cập vào mỗi cấp thấp hơn kế tiếp sẽ ít hơn đáng kể so với cấp trên nó.





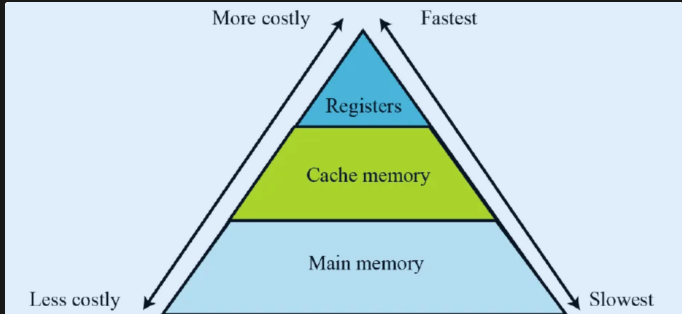
**Cache**: A small size, expensive, memory which has high-speed access is located between CPU and RAM (large memory size, cheaper, and lower-speed Memory).



Vấn đề chi chí: Thứ bộ nhớ nào quan trọng nhất là bộ nhớ

* Chi phí đắt nhất
* Hiệu suất cao nhất
* Tốc độ phản hồi nhanh nhất

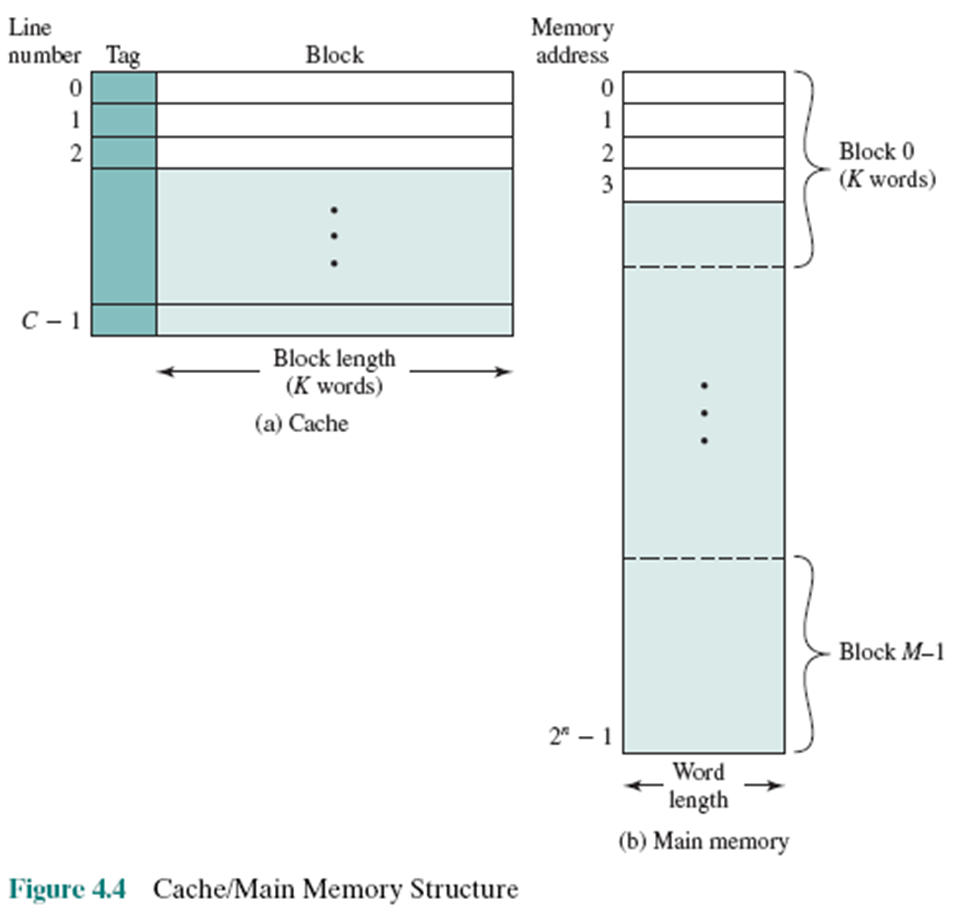
các mức độ chi phí thường thấy(most important of memory):

ROM>RAM>Register>Cache

#### **4.2. Nguyên Tắc của Bộ Nhớ Đệm**

Bộ nhớ đệm được thiết kế để kết hợp thời gian truy cập của bộ nhớ tốc độ cao đắt tiền với kích thước bộ nhớ lớn của bộ nhớ tốc độ thấp rẻ hơn.

* **Nguyên tắc hoạt động:** Khi bộ xử lý cố gắng đọc một từ trong bộ nhớ, một kiểm tra sẽ được thực hiện để xem từ đó có trong bộ nhớ đệm hay không.
  + **Cache hit (trúng đệm):** Nếu có, từ đó được chuyển đến bộ xử lý.
  + **Cache miss (trượt đệm):** Nếu không, một **khối (block)** của bộ nhớ chính (gồm một số từ cố định) sẽ được đọc vào bộ nhớ đệm, sau đó từ đó mới được chuyển đến bộ xử lý.



* **Cấu trúc:** Bộ nhớ chính gồm tới 2^n từ có thể định địa chỉ, và được chia thành các khối có độ dài cố định là K từ. Bộ nhớ đệm bao gồm m khối, gọi là các **dòng (lines)**. Mỗi dòng chứa K từ, cộng với một **thẻ (tag)** để xác định khối bộ nhớ chính nào đang được lưu trữ.

**note:**

**Mapping**- ánh xạ: Cách gán ghép chủ quan, cố tính **mỗi** phần tử trong tập hợp A sẽ **tương ứng với một phần tử duy nhất** trong tập hợp B

Trong trường hợp này: A = tập các lines của cache, B: tập các blocks trong main memory. Mỗi line sẽ chứa nội dung của một block.

**Tag là gì:** Tag là dữ liệu, thông tin được đính kèm. Tag của mỗi line chứa dữ liệu cho biết nội dung của line này ứng với block nào trong main memory. Data trong tag mô tả ánh xạ từ Cache sang Memory

**Tại sao mỗi line phải có phần tag?**: Một line có thể chứa cả data lẫn intructions. Cache có dung lượng nhỏ nên chúng ta có thể hiểu việc chạy chương trình giống như cơ chế làm theo từng mẻ. Data trong cache chỉ mang tính tạm thời, data trong main memory mới là data chính thức. Như vậy, những data đã bị modified trong cache cần phải được cập nhật về main memory. Chính dữ liệu trong tag giúp việc cập nhật này được thực thi chình xác, dúng chỗ

#### **4.3. Các Yếu Tố Thiết Kế Bộ Nhớ Đệm**

**A. Địa chỉ trong Bộ đệm (Cache Addresses)**

Khi bộ nhớ ảo được sử dụng, nhà thiết kế có thể đặt bộ đệm giữa bộ xử lý và **đơn vị quản lý bộ nhớ (Memory Management Unit)** hoặc giữa MMU và bộ nhớ chính.

* **Bộ đệm logic (Logical cache)** hay **bộ đệm ảo (virtual cache)** lưu trữ dữ liệu bằng địa chỉ ảo. Bộ xử lý truy cập trực tiếp vào bộ đệm mà không cần qua MMU. Ưu điểm là tốc độ truy cập nhanh hơn, nhưng có nhược điểm là các không gian địa chỉ ảo khác nhau có thể gây xung đột.
* **Bộ đệm vật lý (Physical cache)** lưu trữ dữ liệu bằng địa chỉ vật lý của bộ nhớ chính.

cách hoạt động:  **user mở task, windows cấp mở 1 vị trí bộ nhớ ảo.**

* bộ nhớ ảo ánh xạ vào RAM
* RAM xác nhận có vùng trống sẽ chuyển từ vùng bộ nhớ ảo -> thật
* khởi tạo bộ đệm vật lý (Physical Cache)
* Khởi tạo và xác nhận có luồng trống để xử lý task.

->Báo về win để win cho CPU xử lý

**B. Kích thước Bộ đệm (Cache Size)**

Kích thước của bộ đệm phải đủ nhỏ để chi phí trung bình mỗi bit gần với chi phí của bộ nhớ chính, và đủ lớn để thời gian truy cập trung bình gần với thời gian của bộ đệm. Bộ đệm lớn có xu hướng chậm hơn một chút so với bộ đệm nhỏ do số lượng cổng logic tham gia vào việc định địa chỉ.

**C. Hàm Ánh xạ (Mapping Function)**

Có ba kỹ thuật được sử dụng để ánh xạ các khối bộ nhớ chính vào các dòng của bộ đệm:

1. **Ánh xạ trực tiếp (Direct Mapping):** Mỗi khối của bộ nhớ chính chỉ được ánh xạ vào một dòng bộ nhớ đệm duy nhất.
   * **Công thức:** i = j modulo m (trong đó i là số dòng đệm, j là số khối bộ nhớ chính, m là số dòng trong bộ đệm).
   * **Ưu điểm:** Đơn giản và rẻ tiền để triển khai.
   * **Nhược điểm:** Vị trí bộ đệm cố định cho mỗi khối, có thể dẫn đến hiện tượng **thrashing** (đánh bật liên tục) nếu một chương trình tham chiếu lặp lại từ hai khối khác nhau ánh xạ vào cùng một dòng, làm cho tỷ lệ trúng (hit ratio) thấp.
2. **Ánh xạ liên kết (Associative Mapping):** Cho phép mỗi khối bộ nhớ chính được nạp vào bất kỳ dòng nào của bộ đệm.
   * **Hoạt động:** Để xác định một khối có trong bộ đệm hay không, logic điều khiển bộ đệm phải kiểm tra đồng thời thẻ của mọi dòng để tìm sự trùng khớp.
   * **Ưu điểm:** Linh hoạt trong việc chọn khối nào để thay thế khi một khối mới được đọc vào bộ đệm.
   * **Nhược điểm:** Mạch logic phức tạp để kiểm tra song song tất cả các thẻ.
3. **Ánh xạ tập-liên kết (Set-Associative Mapping):** Là sự kết hợp giữa hai phương pháp trên, giảm bớt nhược điểm của chúng.
   * **Hoạt động:** Bộ đệm được chia thành các **tập (sets)**, mỗi tập chứa một số dòng. Một khối bộ nhớ chính có thể được ánh xạ vào bất kỳ dòng nào trong một tập cụ thể.
   * **Tổ chức:** Ánh xạ tập-liên kết k-chiều (k-way) cải thiện đáng kể tỷ lệ trúng so với ánh xạ trực tiếp. Tổ chức 2-chiều là phổ biến nhất.

**D. Thuật toán Thay thế (Replacement Algorithms)**

Khi bộ đệm đã đầy, một khối hiện có phải được thay thế.

* **Ánh xạ trực tiếp:** Không có lựa chọn.
* **Ánh xạ liên kết và tập-liên kết:** Cần một thuật toán thay thế.
  + **Least Recently Used (LRU - Ít được sử dụng gần đây nhất):** Thay thế khối đã ở trong bộ đệm lâu nhất mà không có tham chiếu nào đến nó. Đây là thuật toán phổ biến nhất.
  + **First-In-First-Out (FIFO - Vào trước, ra trước):** Thay thế khối đã ở trong bộ đệm lâu nhất.
  + **Least Frequently Used (LFU - Ít được sử dụng thường xuyên nhất):** Thay thế khối có số lần tham chiếu ít nhất.
  + **Ngẫu nhiên (Random):** Chọn một dòng ngẫu nhiên để thay thế. Hiệu năng chỉ kém hơn một chút so với các thuật toán dựa trên việc sử dụng.

**E. Chính sách Ghi (Write Policy)**

* **Write-through (Ghi thẳng):** Tất cả các hoạt động ghi được thực hiện đồng thời vào cả bộ nhớ chính và bộ đệm, đảm bảo bộ nhớ chính luôn hợp lệ. Nhược điểm là tạo ra lưu lượng bộ nhớ đáng kể.
* **Write-back (Ghi ngược):** Các cập nhật chỉ được thực hiện trong bộ đệm. Một **bit bẩn (dirty bit)** được đặt khi có cập nhật. Khi một khối bị thay thế, nó chỉ được ghi lại vào bộ nhớ chính nếu bit bẩn được đặt. Nhược điểm là các phần của bộ nhớ chính có thể không hợp lệ, đòi hỏi các truy cập I/O phải đi qua bộ đệm.

**F. Kích thước Dòng (Line Size)**

Khi kích thước khối tăng lên, tỷ lệ trúng ban đầu sẽ tăng do tính cục bộ, nhưng sau đó sẽ bắt đầu giảm vì khối trở nên quá lớn và xác suất sử dụng thông tin mới được tìm nạp sẽ thấp hơn xác suất sử dụng lại thông tin phải thay thế.

**G. Số lượng Bộ đệm (Number of Caches)**

* **Bộ đệm đa cấp (Multilevel Caches):** Hầu hết các thiết kế đương đại bao gồm cả **bộ đệm trên chip (on-chip cache)** và **bộ đệm ngoài (external cache)**. Một tổ chức đơn giản là **bộ đệm hai cấp**, với bộ đệm trong là **cấp 1 (L1)** và bộ đệm ngoài là **cấp 2 (L2)**. Nhiều bộ vi xử lý hiện đại đã tích hợp cả L2 và **L3** trên chip.
* **Bộ đệm hợp nhất và phân chia (Unified vs. Split Caches):**
  + **Bộ đệm hợp nhất (Unified Cache):** Một bộ đệm duy nhất được sử dụng cho cả lệnh và dữ liệu. Ưu điểm là tỷ lệ trúng cao hơn vì nó tự động cân bằng tải.
  + **Bộ đệm phân chia (Split Cache):** Chia bộ đệm thành hai: một cho lệnh và một cho dữ liệu. Ưu điểm là loại bỏ sự tranh chấp bộ đệm giữa đơn vị tìm nạp/giải mã lệnh và đơn vị thực thi.

CHƯƠNG 5: BỘ NHỚ TRONG (INTERNAL MEMORY)

### **CHƯƠNG 5: BỘ NHỚ TRONG (INTERNAL MEMORY)**

**Mục tiêu của chương này là gì?**

Sau khi học xong chương này, bạn sẽ có thể:

* Trình bày tổng quan về các loại bộ nhớ bán dẫn (semiconductor memory) chính.
* Hiểu hoạt động của mã có khả năng phát hiện và sửa lỗi bit đơn trong các từ 8-bit.
* Tóm tắt các thuộc tính của các tổ chức DDR DRAM đương đại.
* Hiểu sự khác biệt giữa bộ nhớ flash NOR và NAND.
* Trình bày tổng quan về các công nghệ bộ nhớ thể rắn bất biến (nonvolatile solid-state memory) mới hơn.

#### **5.1. Bộ Nhớ Bán Dẫn (Semiconductor Main Memory)**

Bộ nhớ bán dẫn (semiconductor memory) là loại bộ nhớ truy cập ngẫu nhiên (random-access) phổ biến nhất hiện nay. Phần tử cơ bản của bộ nhớ bán dẫn là **ô nhớ (memory cell)**. Mỗi ô nhớ có các thuộc tính sau:

* Có hai trạng thái ổn định (hoặc bán ổn định), được dùng để biểu diễn hai giá trị nhị phân 1 và 0.
* Có khả năng được ghi vào (ít nhất một lần) để thiết lập trạng thái.
* Có khả năng được đọc để cảm nhận trạng thái.

**A. DRAM và SRAM**

Hai loại bộ nhớ RAM truyền thống được sử dụng trong máy tính là DRAM và SRAM. Cả hai đều là bộ nhớ khả biến (volatile), nghĩa là chúng sẽ mất dữ liệu khi mất nguồn điện.

ram tĩnh-ram động: Ram tĩnh, không thể thay thế

Ram động, có thể thay thế

* **DRAM (Dynamic RAM - RAM động):**
  + Sử dụng các ô nhớ lưu trữ dữ liệu dưới dạng **điện tích trên các tụ điện (capacitors)**. Sự có mặt hay vắng mặt của điện tích được hiểu là 1 hoặc 0.
  + Vì các tụ điện có xu hướng tự xả điện, DRAM cần được **làm mới điện tích (charge refreshing)** định kỳ để duy trì dữ liệu. Thuật ngữ "động" (dynamic) dùng để chỉ xu hướng rò rỉ điện tích này.
  + Ô nhớ DRAM đơn giản và nhỏ hơn ô nhớ SRAM, giúp nó có **mật độ cao hơn** và **chi phí thấp hơn**. Do đó, DRAM được ưa chuộng cho các yêu cầu bộ nhớ lớn và được dùng làm **bộ nhớ chính (main memory)**.
* **SRAM (Static RAM - RAM tĩnh):**
  + Lưu trữ giá trị nhị phân bằng cách sử dụng các cấu hình cổng logic **flip-flop** truyền thống.
  + SRAM sẽ giữ dữ liệu của nó miễn là có nguồn điện cung cấp, **không cần làm mới**.
  + SRAM **nhanh hơn** DRAM. Vì các đặc tính này, SRAM được dùng cho **bộ nhớ đệm (cache memory)**, cả trên chip (on-chip) và ngoài chip (off-chip).

**B. Các loại ROM (Read-Only Memory)**

ROM là bộ nhớ bất biến (nonvolatile), nghĩa là không cần nguồn điện để duy trì giá trị bit trong bộ nhớ. Dữ liệu trong ROM không thể thay đổi, hoặc việc ghi dữ liệu mới rất khó khăn.

* **ROM (Read-Only Memory):** Chứa một mẫu dữ liệu vĩnh viễn không thể thay đổi. Dữ liệu được ghi vào chip trong quá trình sản xuất.
* **PROM (Programmable ROM):** Là bộ nhớ bất biến và chỉ có thể được ghi vào một lần duy nhất. Việc ghi được thực hiện bằng điện và có thể được thực hiện bởi nhà cung cấp hoặc khách hàng sau khi chip đã được sản xuất.
* **Bộ nhớ chỉ đọc đa phần (Read-Mostly Memory):** Hữu ích cho các ứng dụng mà thao tác đọc xảy ra thường xuyên hơn nhiều so với thao tác ghi nhưng vẫn cần lưu trữ bất biến.
  + **EPROM (Erasable PROM):** Có thể đọc và ghi bằng điện. Tuy nhiên, trước khi ghi, tất cả các ô nhớ phải được **xóa bằng tia cực tím (ultraviolet radiation)**. Quá trình xóa này có thể mất tới 20 phút.
  + **EEPROM (Electrically Erasable PROM):** Có thể được ghi vào bất kỳ lúc nào mà không cần xóa nội dung trước đó; chỉ các byte được định địa chỉ mới được cập nhật. Thao tác ghi chậm hơn đáng kể so với thao tác đọc.
  + **Bộ nhớ Flash (Flash Memory):** Trung gian giữa EPROM và EEPROM về chi phí và chức năng. Nó sử dụng công nghệ xóa bằng điện, có thể xóa toàn bộ bộ nhớ trong vài giây hoặc xóa theo từng khối thay vì toàn bộ chip. Tuy nhiên, nó không hỗ trợ xóa ở cấp độ byte.

#### **5.2. Sửa Lỗi (Error Correction)**

Hệ thống bộ nhớ bán dẫn có thể gặp lỗi, được phân loại thành **lỗi cứng (hard failures)** và **lỗi mềm (soft errors)**.

* **Lỗi cứng** là một khiếm khuyết vật lý vĩnh viễn khiến ô nhớ không thể lưu trữ dữ liệu một cách đáng tin cậy.

ex: lỗi phần cứng từ nhà sản xuất.

* **Lỗi mềm** là một sự kiện ngẫu nhiên, không phá hủy, làm thay đổi nội dung của một hoặc nhiều ô nhớ mà không làm hỏng bộ nhớ.

ex: lỗi win, treo máy, app. xung đột dữ liệu gây nghẽn phần mềm.

Hầu hết các hệ thống bộ nhớ chính hiện đại đều bao gồm logic để phát hiện và sửa lỗi. Quá trình này hoạt động như sau:

1. Khi dữ liệu được ghi vào bộ nhớ, một phép tính được thực hiện trên dữ liệu để tạo ra một **mã (code)**. Cả dữ liệu và mã đều được lưu trữ.
2. Khi dữ liệu được đọc ra, mã được sử dụng để phát hiện và có thể sửa lỗi. Một bộ mã mới được tạo ra từ dữ liệu vừa đọc và so sánh với mã được lấy ra.
3. Kết quả so sánh có thể là: không có lỗi, lỗi được phát hiện và có thể sửa, hoặc lỗi được phát hiện nhưng không thể sửa.

Mã đơn giản nhất có khả năng sửa lỗi là **mã Hamming (Hamming code)**. Để sửa một lỗi bit đơn trong một từ dữ liệu M-bit, cần K bit kiểm tra, trong đó K phải thỏa mãn bất đẳng thức sau: **2^(K-1) ≥ M + K**

Ví dụ, đối với dữ liệu 8-bit (M=8), cần 4 bit kiểm tra (K=4). Phổ biến hơn, bộ nhớ bán dẫn được trang bị mã **sửa lỗi đơn, phát hiện lỗi kép (SEC-DED)**, yêu cầu thêm một bit so với mã SEC.

note: ECC tính năng tự động sửa lỗi-RAM. nó đắt hơn ram thường.

### **1. Vị trí bit kiểm tra**

* Các bit kiểm tra được đặt tại vị trí là lũy thừa của 2:  
   C1C1C1 ở vị trí 1,  
   C2C2C2 ở vị trí 2,  
   C4C4C4 ở vị trí 4,  
   C8C8C8 ở vị trí 8, …
* Các vị trí còn lại chứa bit dữ liệu (D1, D2, …).

Vị trí Ci tức Check bit, được định vị bằng số lũy thừa của 2, 2^0 2^1 2^2,....

còn lại là Di tức Data bit, định vị băng cách chừa lại những Check bit

cấu trúc định vị và thực hiện như bảng:

| **Vị trí** | **Nhị phân** | **Nội dung** | **Được check bởi** |
| --- | --- | --- | --- |
|  |  |  |  |
| **1** | **0001** | **C1** | **–** |
| **2** | **0010** | **C2** | **–** |
| **3** | **0011** | **D1** | **C1, C2** |
| **4** | **0100** | **C4** | **–** |
| **5** | **0101** | **D2** | **C1, C4** |
| **6** | **0110** | **D3** | **C2, C4** |
| **7** | **0111** | **D4** | **C1, C2, C4** |
| **8** | **1000** | **C8** | **–** |
| **9** | **1001** | **D5** | **C1, C8** |
| **10** | **1010** | **D6** | **C2, C8** |
| **11** | **1011** | **D7** | **C1, C2, C8** |
| **12** | **1100** | **D8** | **C4, C8** |

#### **5.3. DDR DRAM**

Để giải quyết vấn đề tắc nghẽn cổ chai giữa bộ xử lý hiệu năng cao và bộ nhớ trong, các kiến trúc DRAM tiên tiến đã được phát triển, trong đó nổi bật là **SDRAM (Synchronous DRAM)** và **DDR DRAM (Double-Data-Rate DRAM)**.

* **SDRAM (Synchronous DRAM):**
  + Không giống DRAM truyền thống (bất đồng bộ), SDRAM trao đổi dữ liệu với bộ xử lý **đồng bộ với tín hiệu xung nhịp bên ngoài** và chạy ở tốc độ tối đa của bus xử lý/bộ nhớ mà không cần trạng thái chờ.
  + Trong khi DRAM truyền thống thực hiện các chức năng nội bộ, bộ xử lý phải chờ, làm chậm hệ thống. Với SDRAM, bộ xử lý có thể thực hiện các tác vụ khác trong khi SDRAM đang xử lý yêu cầu.
  + SDRAM sử dụng **chế độ burst (burst mode)** để loại bỏ thời gian thiết lập địa chỉ sau lần truy cập đầu tiên, cho phép truyền một loạt bit dữ liệu một cách nhanh chóng.
* **DDR DRAM (Double-Data-Rate SDRAM):**
  + Cung cấp một số tính năng giúp tăng đáng kể tốc độ dữ liệu so với SDRAM.
  + DDR đạt được tốc độ dữ liệu cao hơn bằng cách **truyền dữ liệu trên cả cạnh lên (rising edge) và cạnh xuống (falling edge) của xung nhịp**, giúp tăng gấp đôi tốc độ dữ liệu.
  + Nó cũng sử dụng tốc độ xung nhịp cao hơn trên bus và một sơ đồ đệm (buffering scheme).
  + Các thế hệ DDR (DDR1, DDR2, DDR3, DDR4) đã tăng kích thước bộ đệm prefetch từ 2-bit lên 8-bit để tăng tốc độ truyền dữ liệu hiệu quả. DDR4 giới thiệu khái niệm **nhóm ngân hàng (bank group)** để tăng hiệu năng song song mà không cần tăng thêm kích thước bộ đệm.

CHÂN CẮM: PIN CONNECT



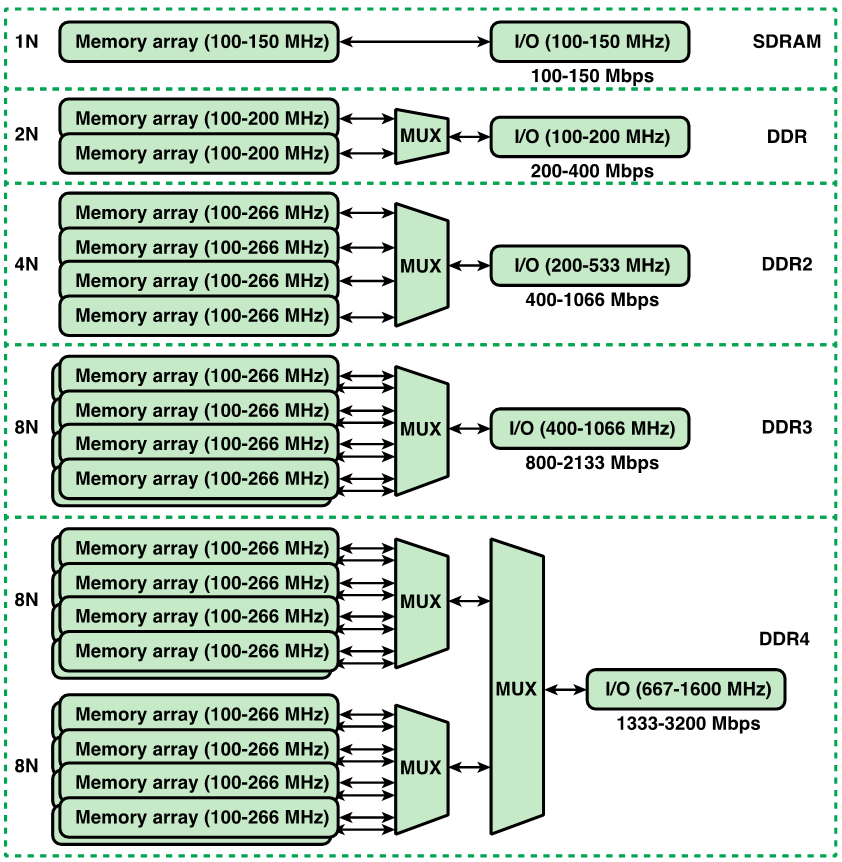
Bản dịch rút gọn:

Bảng 6.3 mô tả các chân kết nối. SDRAM dùng chế độ *burst* để giảm thời gian thiết lập địa chỉ và nạp lại hàng/cột sau truy cập đầu tiên. Ở chế độ này, nhiều bit dữ liệu được truyền liên tiếp nhanh chóng nếu cùng hàng. Kiến trúc nhiều bank bên trong cho phép song song trên chip.

Thanh ghi *mode* và logic điều khiển là điểm khác biệt chính so với DRAM thường. Nó cho phép cấu hình SDRAM theo nhu cầu hệ thống: chọn độ dài *burst* (số đơn vị dữ liệu truyền đồng bộ) và độ trễ giữa lệnh đọc và truyền dữ liệu.

SDRAM đạt hiệu quả cao nhất khi truyền khối dữ liệu lớn liên tục, phù hợp với xử lý văn bản, bảng tính và đa phương tiện.

XUNG NHỊP ram



Bản dịch rút gọn:

Để hiểu bộ đệm *prefetch*, cần xem theo góc độ truyền một từ. Kích thước bộ đệm quyết định số từ dữ liệu được lấy mỗi lần thực hiện lệnh cột trong DDR. Vì lõi DRAM chậm hơn giao tiếp, dữ liệu được truy cập song song rồi ghép nối ra ngoài qua MUX. DDR lấy trước 2 từ, nên mỗi lần đọc/ghi xử lý 2 từ và truyền trong một chu kỳ xung nhịp ở cả hai cạnh, khiến tốc độ I/O gấp đôi lõi DRAM.

Dung lượng SDRAM tăng qua các thế hệ nhưng tốc độ lõi ít đổi. Để tăng băng thông, JEDEC mở rộng bộ đệm: DDR2 dùng *prefetch* 4 từ, DDR3 dùng 8 từ. Nhược điểm: *prefetch* quyết định độ dài burst tối thiểu (ví dụ DDR3 khó dùng burst 4 từ). Vì vậy, DDR4 không nâng lên 16 mà giới thiệu khái niệm *bank group*: mỗi nhóm độc lập, cho phép các chu kỳ cột song song, đạt hiệu năng như *prefetch* lớn hơn nhưng vẫn giữ kích thước bằng DDR3. DDR4 hỗ trợ tối đa 4 *bank group*.

CHƯƠNG 6: BỘ NHỚ NGOÀI (EXTERNAL MEMORY)

### **CHƯƠNG 6: BỘ NHỚ NGOÀI (EXTERNAL MEMORY)**

**Mục tiêu của chương này là gì?**

Sau khi học xong chương này, bạn sẽ có thể:

* Hiểu các thuộc tính chính của **đĩa từ (magnetic disks)**.
* Hiểu các vấn đề về hiệu năng liên quan đến việc truy cập đĩa từ.
* Giải thích khái niệm **RAID (Redundant Array of Independent Disks)** và mô tả các cấp độ khác nhau của nó.
* So sánh và đối chiếu ổ đĩa cứng (HDD) và ổ đĩa thể rắn (SSD).
* Mô tả tổng quan về hoạt động của bộ nhớ flash.
* Hiểu sự khác biệt giữa các phương tiện lưu trữ đĩa quang khác nhau.
* Trình bày tổng quan về công nghệ lưu trữ băng từ.

#### **6.1. Đĩa Từ (Magnetic Disk)**

Đĩa từ là nền tảng của bộ nhớ ngoài trên hầu hết các hệ thống máy tính.

**A. Cơ chế Đọc và Ghi Từ tính**

* Đĩa là một đĩa tròn làm từ vật liệu không từ tính (gọi là **chất nền - substrate**), được phủ một lớp vật liệu có thể từ hóa. Các chất nền bằng thủy tinh ngày càng phổ biến vì chúng cải thiện độ đồng đều của màng từ, giảm khuyết tật bề mặt, hỗ trợ chiều cao bay thấp hơn của đầu đọc/ghi, cứng hơn và chịu sốc tốt hơn.
* Dữ liệu được ghi và truy xuất thông qua một cuộn dây dẫn gọi là **đầu đọc/ghi (head)**.
* **Cơ chế ghi:** Dòng điện chạy qua cuộn dây trong đầu ghi tạo ra một từ trường, từ hóa một vùng nhỏ trên bề mặt đĩa đang quay bên dưới nó. Đảo chiều dòng điện sẽ đảo chiều từ hóa trên môi trường ghi.
* **Cơ chế đọc:**
  + **Truyền thống (Cảm ứng):** Một từ trường di chuyển tương đối với cuộn dây sẽ tạo ra dòng điện trong cuộn dây. Khi bề mặt đĩa quay dưới đầu đọc, nó tạo ra một dòng điện có cùng cực tính với dòng đã được ghi.
  + **Hiện đại (Từ trở - Magnetoresistive - MR):** Các hệ thống đĩa cứng hiện đại sử dụng một đầu đọc riêng biệt có cảm biến từ trở (MR). Điện trở của vật liệu MR thay đổi tùy thuộc vào chiều từ hóa của môi trường di chuyển bên dưới nó. Bằng cách cho dòng điện chạy qua cảm biến MR, sự thay đổi điện trở được phát hiện dưới dạng tín hiệu điện áp. Thiết kế MR cho phép hoạt động ở tần số cao hơn, dẫn đến mật độ lưu trữ và tốc độ hoạt động lớn hơn.

**B. Tổ chức và Định dạng Dữ liệu**

* Dữ liệu được tổ chức trên đĩa thành một tập hợp các vòng tròn đồng tâm gọi là **rãnh (tracks)**. Giữa các rãnh có các **khoảng trống liên rãnh (intertrack gaps)** để giảm thiểu lỗi.
* Dữ liệu được truyền đến và đi từ đĩa theo các cung tròn gọi là **cung từ (sectors)**. Giữa các cung từ có các **khoảng trống liên cung từ (intersector gaps)**. Hầu hết các hệ thống hiện đại đều sử dụng các cung từ có độ dài cố định, gần như phổ biến là 512 byte.
* **Tốc độ góc không đổi (Constant Angular Velocity - CAV):** Đĩa quay ở một tốc độ cố định. Để bù đắp cho việc các bit ở gần tâm di chuyển chậm hơn so với các bit ở rìa ngoài, khoảng cách giữa các bit được điều chỉnh để đầu đọc/ghi có thể đọc tất cả các bit ở cùng một tốc độ. Nhược điểm của CAV là lượng dữ liệu có thể lưu trữ trên các rãnh bên ngoài dài cũng chỉ bằng lượng dữ liệu trên các rãnh bên trong ngắn.
* **Ghi nhiều vùng (Multiple Zone Recording - MZR):** Để tăng dung lượng lưu trữ, bề mặt đĩa được chia thành nhiều vùng đồng tâm. Các vùng ở xa tâm hơn sẽ chứa nhiều bit (nhiều cung từ) hơn các vùng gần tâm, cho phép mật độ bit tuyến tính gần như không đổi trên tất cả các rãnh.
* **Định dạng đĩa:** Đĩa được định dạng với dữ liệu điều khiển bổ sung để xác định điểm bắt đầu của rãnh và điểm bắt đầu/kết thúc của mỗi cung từ. Ví dụ, định dạng đĩa Winchester (Hình 6.4 trong sách) cho thấy mỗi cung từ chứa 512 byte dữ liệu người dùng cùng với thông tin điều khiển như trường ID (để định vị cung từ) và mã phát hiện lỗi (CRC).

**C. Đặc tính Vật lý**

* **Chuyển động của đầu đọc/ghi:**
  + **Đầu đọc/ghi cố định (Fixed-head disk):** Có một đầu đọc/ghi cho mỗi rãnh. Các hệ thống này hiện nay rất hiếm.
  + **Đầu đọc/ghi di động (Movable-head disk):** Chỉ có một đầu đọc/ghi. Cần di chuyển cánh tay (arm) để định vị đầu đọc/ghi trên rãnh mong muốn.
* **Tính di động của đĩa:**
  + **Đĩa không thể tháo rời (Nonremovable disk):** Được gắn vĩnh viễn trong ổ đĩa, ví dụ như ổ cứng trong máy tính cá nhân.
  + **Đĩa có thể tháo rời (Removable disk):** Có thể tháo ra và thay thế, ví dụ như đĩa mềm và đĩa ZIP.
* **Nhiều đĩa (Multiple platters):** Các ổ đĩa có thể chứa nhiều đĩa xếp chồng lên nhau. Một bộ các rãnh ở cùng một vị trí tương đối trên các đĩa khác nhau được gọi là một **trụ (cylinder)**.
* **Cơ chế đầu đọc/ghi:**
  + **Đĩa Winchester:** Đầu đọc/ghi được thiết kế để hoạt động gần bề mặt đĩa hơn so với các loại đĩa cứng thông thường, cho phép mật độ dữ liệu cao hơn. Đầu đọc/ghi là một lá kim loại khí động học (aerodynamic foil) nằm nhẹ trên bề mặt đĩa khi đĩa đứng yên và bay lên trên bề mặt khi đĩa quay.

**D. Các Tham số Hiệu năng của Đĩa**

* **Thời gian tìm kiếm (Seek time):** Thời gian cần thiết để di chuyển cánh tay đĩa đến rãnh mong muốn. Thời gian tìm kiếm trung bình điển hình trên các ổ đĩa cứng hiện đại là dưới 10 ms.
* **Độ trễ quay (Rotational delay/latency):** Thời gian cần thiết để phần đầu của cung từ mong muốn quay đến dưới đầu đọc/ghi.
* **Thời gian truy cập (Access time):** Tổng của thời gian tìm kiếm và độ trễ quay.
* **Thời gian truyền (Transfer time):** Thời gian cần thiết để hoạt động đọc hoặc ghi được thực hiện khi cung từ di chuyển dưới đầu đọc/ghi. Công thức tính là: T = b / (r \* N), trong đó b là số byte cần truyền, N là số byte trên một rãnh, và r là tốc độ quay (vòng/giây).
* **Tổng thời gian đọc/ghi trung bình** có thể được biểu thị bằng công thức T\_total = T\_s + 1/(2r) + b/(r\*N), trong đó T\_s là thời gian tìm kiếm trung bình.

#### **6.2. RAID (Redundant Array of Independent Disks)**

RAID là một lược đồ thiết kế cơ sở dữ liệu trên nhiều đĩa được tiêu chuẩn hóa. Nó bao gồm bảy cấp độ, từ 0 đến 6. Các đặc điểm chung của RAID là:

1. Là một tập hợp các ổ đĩa vật lý được hệ điều hành xem như một ổ đĩa logic duy nhất.
2. Dữ liệu được phân bổ trên các ổ đĩa vật lý của một mảng theo một lược đồ gọi là **striping**.
3. Dung lượng đĩa dự phòng được sử dụng để lưu trữ thông tin chẵn lẻ (parity), đảm bảo khả năng phục hồi dữ liệu trong trường hợp một đĩa bị lỗi.

Dưới đây là tóm tắt về các cấp độ RAID khác nhau:

* **RAID 0 (Không dự phòng):** Dữ liệu được chia nhỏ (striped) trên tất cả các đĩa trong mảng. Nó không bao gồm sự dự phòng để cải thiện hiệu năng, nhưng có thể cung cấp tốc độ truyền dữ liệu cao hoặc tốc độ yêu cầu I/O cao tùy thuộc vào kích thước của các dải (strip).
* **RAID 1 (Phản chiếu - Mirrored):** Dữ liệu được sao chép trên hai ổ đĩa (phản chiếu). Một yêu cầu đọc có thể được phục vụ bởi một trong hai đĩa có thời gian truy cập tối thiểu. Yêu cầu ghi đòi hỏi cả hai đĩa phải được cập nhật. Phục hồi từ một sự cố rất đơn giản. Nhược điểm chính là chi phí vì nó yêu cầu gấp đôi dung lượng đĩa.
* **RAID 2 (Dự phòng thông qua mã Hamming):** Sử dụng kỹ thuật truy cập song song. Các dải dữ liệu rất nhỏ. Một mã sửa lỗi (thường là mã Hamming) được tính toán trên các bit tương ứng trên mỗi đĩa dữ liệu, và các bit của mã này được lưu trữ trên các đĩa chẵn lẻ. RAID 2 không được triển khai trên thực tế vì nó quá thừa thãi so với độ tin cậy cao của các ổ đĩa hiện đại.
* **RAID 3 (Chẵn lẻ xen kẽ bit - Bit-interleaved parity):** Yêu cầu chỉ một đĩa dự phòng duy nhất. Một bit chẵn lẻ đơn giản được tính toán cho tập hợp các bit riêng lẻ ở cùng một vị trí trên tất cả các đĩa dữ liệu. Có thể đạt được tốc độ truyền dữ liệu rất cao nhưng chỉ có thể thực hiện một yêu cầu I/O tại một thời điểm.
* **RAID 4 (Chẵn lẻ xen kẽ khối - Block-interleaved parity):** Sử dụng kỹ thuật truy cập độc lập, cho phép các yêu cầu I/O riêng biệt được thỏa mãn song song. Các dải dữ liệu tương đối lớn. Một dải chẵn lẻ xen kẽ bit được tính toán trên các dải tương ứng trên mỗi đĩa dữ liệu và được lưu trữ trên một đĩa chẵn lẻ chuyên dụng. Phải chịu "chi phí ghi" (write penalty) đáng kể vì mỗi lần ghi đều phải truy cập vào đĩa chẵn lẻ, có thể tạo ra một nút thắt cổ chai.
* **RAID 5 (Chẵn lẻ phân tán xen kẽ khối - Block-interleaved distributed parity):** Tương tự như RAID 4, nhưng phân phối các dải chẵn lẻ trên tất cả các đĩa để tránh nút thắt cổ chai tiềm năng của RAID 4.
* **RAID 6 (Dự phòng kép - Dual redundancy):** Thực hiện hai phép tính chẵn lẻ khác nhau và lưu trữ chúng trong các khối riêng biệt trên các đĩa khác nhau. Điều này cung cấp tính sẵn sàng dữ liệu cực kỳ cao và có thể chịu được việc hai đĩa bị lỗi. Tuy nhiên, nó phải chịu một chi phí ghi đáng kể vì mỗi lần ghi ảnh hưởng đến hai khối chẵn lẻ.

#### **6.3. Ổ Đĩa Thể Rắn (Solid State Drives - SSD)**

SSD là một thiết bị bộ nhớ được làm bằng các thành phần thể rắn có thể được sử dụng để thay thế ổ đĩa cứng (HDD). Các ổ SSD hiện tại sử dụng bộ nhớ flash NAND.

**A. So sánh SSD và HDD**

So với HDD, SSD có những ưu điểm sau:

* **Hiệu năng I/O mỗi giây (IOPS) cao.**
* **Độ bền cao hơn:** Ít bị ảnh hưởng bởi sốc vật lý và rung động.
* **Tuổi thọ dài hơn:** Không bị hao mòn cơ học.
* **Tiêu thụ điện năng thấp hơn.**
* **Hoạt động êm hơn và mát hơn.**
* **Thời gian truy cập và độ trễ thấp hơn.**

Hiện tại, HDD có lợi thế về chi phí trên mỗi bit và dung lượng, nhưng những khác biệt này đang thu hẹp dần.

**B. Tổ chức SSD**

Một SSD bao gồm các thành phần sau:

* **Bộ điều khiển (Controller):** Cung cấp giao diện cấp thiết bị SSD và thực thi phần sụn.
* **Logic định địa chỉ (Addressing):** Thực hiện chức năng lựa chọn trên các thành phần bộ nhớ flash.
* **Bộ đệm/Cache dữ liệu (Data buffer/cache):** Các thành phần RAM tốc độ cao được sử dụng để khớp tốc độ và tăng thông lượng dữ liệu.
* **Sửa lỗi (Error correction):** Logic để phát hiện và sửa lỗi.
* **Các thành phần bộ nhớ Flash:** Các chip flash NAND riêng lẻ.

**C. Các Vấn đề Thực tế**

* **Hiệu năng chậm dần:** Hiệu năng của SSD có xu hướng chậm lại khi thiết bị được sử dụng. Điều này là do bộ nhớ flash được truy cập theo các khối lớn, và để ghi một trang nhỏ, toàn bộ khối phải được đọc vào bộ đệm RAM, xóa khỏi bộ nhớ flash, và sau đó ghi lại vào bộ nhớ flash. Các kỹ thuật để bù đắp cho điều này bao gồm **cung cấp thừa (over-provisioning)** và lệnh **TRIM**.
* **Tuổi thọ ghi có giới hạn:** Bộ nhớ flash trở nên không thể sử dụng được sau một số lần ghi nhất định. Các kỹ thuật để kéo dài tuổi thọ của ổ SSD bao gồm sử dụng bộ đệm, thuật toán **cân bằng hao mòn (wear-leveling)**, và các kỹ thuật quản lý khối lỗi tinh vi.

#### **6.4. Bộ nhớ quang (Optical Memory)**

Các hệ thống đĩa quang đã được giới thiệu đa dạng, bao gồm:

* **CD (Compact Disk):** Một đĩa không thể xóa được lưu trữ thông tin âm thanh kỹ thuật số.
* **CD-ROM (Compact Disk Read-Only Memory):** Một đĩa không thể xóa được sử dụng để lưu trữ dữ liệu máy tính. Nó chứa một rãnh xoắn ốc duy nhất và được đọc bằng cách sử dụng **tốc độ tuyến tính không đổi (Constant Linear Velocity - CLV)**. Dữ liệu được lấy ra bằng cách chiếu tia laser công suất thấp lên bề mặt đĩa và phát hiện sự thay đổi về cường độ phản xạ giữa các **lỗ (pits)** và **vùng đất (lands)**.
* **CD-R (CD Recordable):** Một đĩa ghi một lần đọc nhiều lần (write-once read-many).
* **CD-RW (CD Rewritable):** Một đĩa có thể ghi và ghi lại nhiều lần, thường sử dụng công nghệ **thay đổi pha (phase change)**.
* **DVD (Digital Versatile Disk):** Cung cấp dung lượng lớn hơn CD do các bit được đóng gói chặt chẽ hơn, sử dụng hai lớp hố và vùng đất, và có thể là hai mặt.
* **Đĩa quang độ nét cao (High-Definition Optical Disks):** Như **Blu-ray DVD**, cung cấp mật độ lưu trữ dữ liệu lớn hơn đáng kể so với DVD bằng cách sử dụng tia laser có bước sóng ngắn hơn.

#### **6.5. Băng từ (Magnetic Tape)**

* Băng từ sử dụng các kỹ thuật đọc và ghi tương tự như hệ thống đĩa.
* Dữ liệu trên băng được cấu trúc thành nhiều **rãnh song song (parallel tracks)** chạy dọc theo chiều dài của băng.
* Hầu hết các hệ thống hiện đại sử dụng **ghi tuần tự (serial recording)**, trong đó dữ liệu được bố trí thành một chuỗi các bit dọc theo mỗi rãnh. Kỹ thuật ghi điển hình được gọi là **ghi hình sin (serpentine recording)**, trong đó dữ liệu được ghi dọc theo toàn bộ chiều dài của băng, sau đó các đầu đọc/ghi được định vị lại để ghi một rãnh mới theo hướng ngược lại.
* Ổ băng là một thiết bị **truy cập tuần tự (sequential-access)**.
* Công nghệ băng từ thống trị hiện nay là hệ thống băng catridge được gọi là **Linear Tape-Open (LTO)**.

CHƯƠNG 7: NHẬP/XUẤT (INPUT/OUTPUT)

phan thanh dat, nguyen nghi phi

### **CHƯƠG 7: NHẬP/XUẤT (INPUT/OUTPUT)**

**Mục tiêu của chương này là gì?**

Sau khi học xong chương này, bạn sẽ có thể:

* Giải thích vai trò của các module I/O trong cấu trúc của một máy tính.
* Hiểu sự khác biệt giữa các kỹ thuật **I/O lập trình (programmed I/O)** và **I/O điều khiển bằng ngắt (interrupt-driven I/O)** và thảo luận về ưu điểm tương đối của chúng.
* Trình bày tổng quan về hoạt động của **truy cập bộ nhớ trực tiếp (Direct Memory Access - DMA)**.
* Giải thích chức năng và công dụng của các kênh I/O.

#### **7.1. Các Thiết Bị Ngoại Vi (External Devices)**

Hoạt động I/O được thực hiện thông qua một loạt các thiết bị ngoại vi, cung cấp phương tiện để trao đổi dữ liệu giữa môi trường bên ngoài và máy tính. Một thiết bị ngoại vi được kết nối với máy tính thông qua một liên kết đến module I/O. Liên kết này được sử dụng để trao đổi tín hiệu điều khiển, trạng thái và dữ liệu.

Các thiết bị ngoại vi có thể được phân loại thành ba nhóm chính:

1. **Có thể đọc được bởi con người (Human readable):** Thích hợp cho việc giao tiếp với người dùng máy tính, ví dụ như thiết bị đầu cuối hiển thị video (VDT) và máy in.
2. **Có thể đọc được bởi máy (Machine readable):** Thích hợp cho việc giao tiếp với các thiết bị, ví dụ như hệ thống đĩa từ và băng từ, cũng như các cảm biến và cơ cấu chấp hành trong các ứng dụng robot.
3. **Giao tiếp (Communication):** Thích hợp cho việc giao tiếp với các thiết bị từ xa, chẳng hạn như một máy tính khác.

Về tổng quan, một thiết bị ngoại vi bao gồm các thành phần sau:

* **Tín hiệu điều khiển, dữ liệu và trạng thái:** Giao diện với module I/O.
* **Logic điều khiển:** Điều khiển hoạt động của thiết bị theo chỉ dẫn từ module I/O.
* **Bộ chuyển đổi (Transducer):** Chuyển đổi dữ liệu từ dạng điện sang các dạng năng lượng khác trong quá trình xuất và ngược lại trong quá trình nhập.
* **Bộ đệm (Buffer):** Lưu trữ tạm thời dữ liệu đang được truyền giữa module I/O và môi trường bên ngoài.

#### **7.2. Các Module I/O**

Một module I/O là một thực thể trong máy tính chịu trách nhiệm điều khiển một hoặc nhiều thiết bị ngoại vi và trao đổi dữ liệu giữa các thiết bị đó với bộ nhớ chính và/hoặc các thanh ghi của CPU.

**A. Chức năng của Module I/O**

Các lý do chính tại sao các thiết bị không được kết nối trực tiếp vào bus hệ thống bao gồm:

* Sự đa dạng của các thiết bị ngoại vi về phương thức hoạt động.
* Tốc độ truyền dữ liệu của các thiết bị ngoại vi thường chậm hơn nhiều so với bộ nhớ hoặc bộ xử lý.
* Một số thiết bị ngoại vi có tốc độ truyền dữ liệu nhanh hơn bộ nhớ hoặc bộ xử lý.
* Các thiết bị ngoại vi thường sử dụng các định dạng dữ liệu và độ dài từ khác với máy tính.

Do đó, một module I/O là cần thiết. Các chức năng chính của một module I/O là:

* **Điều khiển và định thời (Control and timing):** Điều phối luồng lưu lượng giữa các tài nguyên nội bộ và các thiết bị ngoại vi.
* **Giao tiếp với bộ xử lý (Processor communication):** Bao gồm giải mã lệnh, trao đổi dữ liệu, báo cáo trạng thái và nhận dạng địa chỉ.
* **Giao tiếp với thiết bị (Device communication):** Liên quan đến các lệnh, thông tin trạng thái và dữ liệu.
* **Đệm dữ liệu (Data buffering):** Cung cấp bộ đệm để cân bằng tốc độ giữa các thiết bị và bộ nhớ.
* **Phát hiện lỗi (Error detection):** Phát hiện và báo cáo các lỗi truyền dữ liệu.

**B. Cấu trúc Module I/O**

Cấu trúc chung của một module I/O bao gồm:

* **Giao diện với bus hệ thống:** Kết nối module với phần còn lại của máy tính.
* **Các thanh ghi dữ liệu:** Đệm dữ liệu được truyền đến và đi từ module.
* **Các thanh ghi trạng thái/điều khiển:** Cung cấp thông tin trạng thái hiện tại và chấp nhận thông tin điều khiển từ bộ xử lý.
* **Logic I/O:** Tương tác với bộ xử lý thông qua các đường điều khiển.
* **Logic giao diện thiết bị ngoại vi:** Logic cụ thể cho giao diện với từng thiết bị mà nó điều khiển.

#### **7.3. I/O Lập Trình (Programmed I/O)**

Với I/O lập trình, dữ liệu được trao đổi giữa bộ xử lý và module I/O. Bộ xử lý thực thi một chương trình cho phép nó kiểm soát trực tiếp hoạt động I/O, bao gồm cảm nhận trạng thái thiết bị, gửi lệnh đọc/ghi và truyền dữ liệu. Khi bộ xử lý ra lệnh cho module I/O, nó phải đợi cho đến khi hoạt động I/O hoàn tất. Nếu bộ xử lý nhanh hơn module I/O, điều này sẽ lãng phí thời gian của bộ xử lý.

**A. Các Lệnh I/O**

Để thực thi một lệnh liên quan đến I/O, bộ xử lý sẽ đưa ra một địa chỉ và một lệnh I/O. Có bốn loại lệnh I/O mà một module I/O có thể nhận được:

1. **Điều khiển (Control):** Dùng để kích hoạt một thiết bị ngoại vi và yêu cầu nó thực hiện một hành động cụ thể.
2. **Kiểm tra (Test):** Dùng để kiểm tra các điều kiện trạng thái khác nhau của một module I/O và các thiết bị ngoại vi của nó.
3. **Đọc (Read):** Khiến module I/O lấy một mục dữ liệu từ thiết bị ngoại vi và đặt nó vào một bộ đệm nội bộ.
4. **Ghi (Write):** Khiến module I/O lấy một mục dữ liệu từ bus dữ liệu và sau đó truyền nó đến thiết bị ngoại vi.

**B. Các Lệnh I/O Máy**

Các thiết bị ngoại vi có thể được định địa chỉ bằng một trong hai cách:

* **I/O ánh xạ bộ nhớ (Memory-mapped I/O):** Có một không gian địa chỉ duy nhất cho các vị trí bộ nhớ và các thiết bị I/O. Bộ xử lý sử dụng cùng một lệnh máy để truy cập cả bộ nhớ và thiết bị I/O.
* **I/O cô lập (Isolated I/O):** Không gian địa chỉ cho I/O được cô lập khỏi không gian địa chỉ cho bộ nhớ. Các cổng I/O chỉ có thể được truy cập bằng các lệnh I/O đặc biệt.

#### **7.4. I/O Điều Khiển Bằng Ngắt (Interrupt-Driven I/O)**

Với I/O điều khiển bằng ngắt, bộ xử lý đưa ra một lệnh I/O, tiếp tục thực thi các lệnh khác, và bị module I/O ngắt khi nó hoàn thành công việc. Điều này hiệu quả hơn I/O lập trình vì nó loại bỏ thời gian chờ đợi không cần thiết.

**A. Xử lý ngắt**

Khi một thiết bị I/O hoàn thành một hoạt động I/O, một chuỗi các sự kiện phần cứng xảy ra:

1. Thiết bị đưa ra một tín hiệu ngắt đến bộ xử lý.
2. Bộ xử lý hoàn thành việc thực thi lệnh hiện tại trước khi phản hồi.
3. Bộ xử lý kiểm tra ngắt, xác định có một ngắt, và gửi tín hiệu xác nhận đến thiết bị.
4. Bộ xử lý lưu thông tin cần thiết để tiếp tục chương trình hiện tại (ví dụ: PSW và PC) vào ngăn xếp điều khiển hệ thống.
5. Bộ xử lý nạp PC với địa chỉ bắt đầu của chương trình xử lý ngắt.

Sau đó, chương trình xử lý ngắt thực hiện các hoạt động phần mềm cần thiết.

**B. Vấn đề thiết kế**

Hai vấn đề thiết kế chính phát sinh trong việc triển khai I/O điều khiển bằng ngắt là:

* **Làm thế nào bộ xử lý xác định thiết bị nào đã đưa ra ngắt?** Các kỹ thuật bao gồm nhiều đường ngắt, thăm dò phần mềm, chuỗi daisy và phân xử bus.
* **Nếu nhiều ngắt đã xảy ra, bộ xử lý quyết định xử lý cái nào trước?** Các phương pháp bao gồm vô hiệu hóa ngắt trong khi một ngắt đang được xử lý hoặc xác định các mức ưu tiên cho các ngắt.

#### **7.5. Truy Cập Bộ Nhớ Trực Tiếp (Direct Memory Access - DMA)**

I/O điều khiển bằng ngắt, mặc dù hiệu quả hơn I/O lập trình, vẫn đòi hỏi sự can thiệp tích cực của bộ xử lý để truyền dữ liệu giữa bộ nhớ và một module I/O. Với khối lượng lớn dữ liệu cần di chuyển, một kỹ thuật hiệu quả hơn là cần thiết: **truy cập bộ nhớ trực tiếp (DMA)**.

**A. Chức năng của DMA**

DMA liên quan đến một module bổ sung trên bus hệ thống có khả năng tiếp quản quyền điều khiển hệ thống từ bộ xử lý để truyền dữ liệu đến và đi từ bộ nhớ qua bus hệ thống. Hoạt động này được gọi là **đánh cắp chu kỳ (cycle stealing)**.

Khi bộ xử lý muốn đọc hoặc ghi một khối dữ liệu, nó sẽ ra lệnh cho module DMA bằng cách gửi các thông tin sau:

* Yêu cầu đọc hay ghi.
* Địa chỉ của thiết bị I/O liên quan.
* Vị trí bắt đầu trong bộ nhớ để đọc hoặc ghi vào.
* Số lượng từ cần đọc hoặc ghi.

Bộ xử lý sau đó tiếp tục với công việc khác trong khi module DMA truyền toàn bộ khối dữ liệu, mỗi lần một từ, trực tiếp đến hoặc từ bộ nhớ. Khi việc truyền hoàn tất, module DMA sẽ gửi một tín hiệu ngắt đến bộ xử lý.

**B. Cấu hình DMA**

Có nhiều cấu hình DMA khác nhau. Một số khả năng bao gồm:

* Tất cả các module chia sẻ cùng một bus hệ thống, trong đó module DMA sử dụng I/O lập trình để trao đổi dữ liệu.
* Tích hợp các chức năng DMA và I/O, tạo ra một đường dẫn giữa module DMA và các module I/O không bao gồm bus hệ thống.
* Kết nối các module I/O với module DMA bằng một bus I/O, giảm số lượng giao diện I/O trong module DMA và cung cấp một cấu hình dễ dàng mở rộng.

#### **7.6. Các Kênh và Bộ Xử Lý I/O**

Sự phát triển của chức năng I/O có thể được tóm tắt qua các bước sau:

1. CPU điều khiển trực tiếp một thiết bị ngoại vi.
2. Một bộ điều khiển hoặc module I/O được thêm vào, sử dụng I/O lập trình không có ngắt.
3. Cấu hình tương tự nhưng có sử dụng ngắt.
4. Module I/O được cấp quyền truy cập trực tiếp vào bộ nhớ thông qua DMA.
5. Module I/O được tăng cường để trở thành một bộ xử lý riêng, được gọi là **kênh I/O**, với một tập lệnh chuyên biệt cho I/O.
6. Module I/O có bộ nhớ cục bộ riêng và thực chất là một máy tính riêng, được gọi là **bộ xử lý I/O**.

**A. Đặc điểm của các Kênh I/O**

Một kênh I/O có khả năng thực thi các lệnh I/O, cho phép nó kiểm soát hoàn toàn các hoạt động I/O. Hai loại kênh I/O phổ biến là:

* **Kênh lựa chọn (Selector channel):** Điều khiển nhiều thiết bị tốc độ cao và tại một thời điểm, chỉ dành riêng cho việc truyền dữ liệu với một trong các thiết bị đó.
* **Kênh đa hợp (Multiplexor channel):** Có thể xử lý I/O với nhiều thiết bị cùng một lúc, thường là các thiết bị tốc độ thấp.

CHƯƠNG 8: HỖ TRỢ CỦA HỆ ĐIỀU HÀNH

### **CHƯƠG 8: HỖ TRỢ CỦA HỆ ĐIỀU HÀNH**

**Mục tiêu của chương này là gì?**

Sau khi học xong chương này, bạn sẽ có thể:

* Tóm tắt, ở mức độ cao, các chức năng chính của một **hệ điều hành (OS)**.
* Thảo luận về sự phát triển của hệ điều hành từ các hệ thống batch đơn giản ban đầu đến các hệ thống phức tạp hiện đại.
* Giải thích sự khác biệt giữa **lập lịch dài hạn, trung hạn và ngắn hạn**.
* Hiểu lý do của việc **phân hoạch bộ nhớ (memory partitioning)** và giải thích các kỹ thuật khác nhau được sử dụng.
* Đánh giá những ưu điểm tương đối của **phân trang (paging)** và **phân đoạn (segmentation)**.
* Định nghĩa **bộ nhớ ảo (virtual memory)**.

#### **8.1. Tổng Quan về Hệ Điều Hành**

**A. Mục Tiêu và Chức Năng của Hệ Điều Hành**

Hệ điều hành (OS) là một chương trình điều khiển việc thực thi các chương trình ứng dụng và hoạt động như một giao diện giữa ứng dụng và phần cứng máy tính. Nó có hai mục tiêu chính:

* **Tiện lợi:** Làm cho máy tính trở nên thuận tiện hơn khi sử dụng.
* **Hiệu quả:** Cho phép các tài nguyên của hệ thống máy tính được sử dụng một cách hiệu quả.

Để làm điều này, hệ điều hành cung cấp các dịch vụ trong các lĩnh vực sau:

1. **Tạo chương trình:** Cung cấp các công cụ như trình soạn thảo và trình gỡ lỗi để hỗ trợ lập trình viên.
2. **Thực thi chương trình:** Xử lý các bước cần thiết để chạy một chương trình, bao gồm nạp lệnh và dữ liệu vào bộ nhớ chính và khởi tạo các thiết bị I/O.
3. **Truy cập thiết bị I/O:** Quản lý các lệnh và tín hiệu điều khiển phức tạp cho từng thiết bị I/O, giúp lập trình viên chỉ cần nghĩ đến các thao tác đọc và ghi đơn giản.
4. **Kiểm soát truy cập tập tin:** Quản lý chi tiết về định dạng tập tin và thiết bị lưu trữ, đồng thời cung cấp các cơ chế bảo vệ trong hệ thống đa người dùng.
5. **Truy cập hệ thống:** Kiểm soát quyền truy cập vào toàn bộ hệ thống và các tài nguyên cụ thể, giải quyết xung đột và bảo vệ tài nguyên.
6. **Phát hiện và phản hồi lỗi:** Xử lý các lỗi phần cứng và phần mềm, từ việc kết thúc chương trình gây lỗi đến việc thử lại hoạt động hoặc báo cáo lỗi cho ứng dụng.
7. **Kế toán:** Thu thập thống kê sử dụng tài nguyên và theo dõi các thông số hiệu năng như thời gian phản hồi, hữu ích cho việc nâng cấp và tinh chỉnh hệ thống.

Một khía cạnh khác của hệ điều hành là vai trò **quản lý tài nguyên**. OS quản lý các tài nguyên của máy tính như bộ xử lý, bộ nhớ chính, thiết bị I/O, và tập tin. Nó hoạt động như một chương trình được thực thi bởi bộ xử lý, thường xuyên từ bỏ quyền kiểm soát để bộ xử lý thực hiện công việc hữu ích và sau đó lấy lại quyền kiểm soát để chuẩn bị cho công việc tiếp theo.

**B. Các Loại Hệ Điều Hành**

Sự phát triển của hệ điều hành có thể được tóm tắt qua các giai đoạn sau:

* **Hệ thống sơ khai (cuối 1940s - giữa 1950s):** Không có hệ điều hành; lập trình viên tương tác trực tiếp với phần cứng. Điều này gây ra hai vấn đề lớn: **lập lịch (scheduling)**, dẫn đến thời gian máy tính bị lãng phí, và **thời gian thiết lập (setup time)**, tốn nhiều công sức để nạp trình biên dịch, chương trình nguồn, và liên kết các chương trình đối tượng.
* **Hệ thống batch đơn giản:** Để cải thiện việc sử dụng, các hệ điều hành batch đơn giản, còn gọi là **monitor**, được phát triển. Người dùng nộp công việc (job) cho người vận hành, người này sẽ gộp chúng lại và đưa vào hệ thống. Phần cốt lõi của monitor, gọi là **resident monitor**, luôn nằm trong bộ nhớ chính. Monitor đọc và thực thi từng công việc một cách tuần tự, loại bỏ thời gian chết do lập lịch và tự động hóa thời gian thiết lập bằng **Ngôn ngữ Điều khiển Công việc (Job Control Language - JCL)**.
* **Hệ thống batch đa chương (Multiprogrammed Batch Systems):** Ngay cả với hệ thống batch, bộ xử lý vẫn thường xuyên rảnh rỗi vì các thiết bị I/O chậm hơn nhiều. **Đa chương (multiprogramming)** là giải pháp, trong đó nhiều chương trình được nạp vào bộ nhớ cùng một lúc. Khi một công việc phải chờ I/O, bộ xử lý có thể chuyển sang thực thi một công việc khác, giữ cho bộ xử lý luôn bận rộn.
* **Hệ thống chia sẻ thời gian (Time-Sharing Systems):** Được phát triển để hỗ trợ nhiều người dùng tương tác trực tiếp với máy tính thông qua các thiết bị đầu cuối. Hệ thống chia sẻ thời gian của bộ xử lý cho nhiều người dùng, xen kẽ việc thực thi chương trình của mỗi người dùng trong một khoảng thời gian ngắn gọi là **quantum**.

#### **8.2. Lập Lịch (Scheduling)**

Chìa khóa của đa chương là lập lịch. Một **tiến trình (process)** là một chương trình đang được thực thi. OS thường liên quan đến bốn loại lập lịch:

1. **Lập lịch dài hạn (Long-Term Scheduling):** Quyết định chương trình nào được chấp nhận vào hệ thống để xử lý. Nó kiểm soát **mức độ đa chương** (số lượng tiến trình trong bộ nhớ). Trong hệ thống batch, các công việc mới được giữ trong một hàng đợi và bộ lập lịch dài hạn sẽ tạo ra các tiến trình từ hàng đợi này khi có thể.
2. **Lập lịch trung hạn (Medium-Term Scheduling):** Là một phần của chức năng **hoán đổi (swapping)**. Nó quyết định bổ sung thêm các tiến trình đã bị hoán đổi ra ngoài vào bộ nhớ chính, dựa trên nhu cầu quản lý mức độ đa chương và yêu cầu bộ nhớ của các tiến trình.
3. **Lập lịch ngắn hạn (Short-Term Scheduling):** Còn được gọi là **bộ điều phối (dispatcher)**, thực thi rất thường xuyên và đưa ra quyết định chi tiết về việc tiến trình nào sẽ được thực thi tiếp theo.
4. **Lập lịch I/O (I/O Scheduling):** Quyết định yêu cầu I/O đang chờ của tiến trình nào sẽ được xử lý bởi một thiết bị I/O có sẵn.

Để thực hiện lập lịch ngắn hạn, OS duy trì **Khối Điều khiển Tiến trình (Process Control Block - PCB)** cho mỗi tiến trình, chứa thông tin như mã định danh, trạng thái, độ ưu tiên, bộ đếm chương trình (PC), con trỏ bộ nhớ và dữ liệu ngữ cảnh. Một tiến trình có thể ở một trong năm trạng thái: **Mới, Sẵn sàng, Đang chạy, Đang chờ, hoặc Đã kết thúc**.

#### **8.3. Quản Lý Bộ Nhớ (Memory Management)**

Trong hệ thống đa chương, phần bộ nhớ của người dùng được chia nhỏ để chứa nhiều tiến trình. Nhiệm vụ này được OS thực hiện một cách linh động và được gọi là quản lý bộ nhớ.

**A. Hoán Đổi (Swapping)**

Ngay cả với đa chương, bộ xử lý vẫn có thể rảnh rỗi nếu tất cả các tiến trình trong bộ nhớ đều đang chờ I/O. Giải pháp là **hoán đổi (swapping)**: OS chuyển một tiến trình từ bộ nhớ chính ra đĩa (vào một hàng đợi trung gian) để nhường chỗ cho một tiến trình mới. Vì I/O đĩa thường nhanh hơn các loại I/O khác, việc hoán đổi thường cải thiện hiệu năng.

**B. Phân Hoạch (Partitioning)**

* **Phân hoạch cố định (Fixed-size partitions):** Bộ nhớ được chia thành các phân vùng có kích thước cố định (không nhất thiết phải bằng nhau). Khi một tiến trình được đưa vào, nó được đặt vào phân vùng nhỏ nhất có sẵn đủ để chứa nó. Nhược điểm là lãng phí bộ nhớ vì một tiến trình thường không yêu cầu chính xác kích thước của phân vùng.
* **Phân hoạch động (Variable-size partitions):** Mỗi tiến trình được cấp phát chính xác lượng bộ nhớ mà nó cần. Tuy nhiên, theo thời gian, điều này dẫn đến **phân mảnh (fragmentation)**, tạo ra nhiều lỗ hổng nhỏ trong bộ nhớ, làm giảm hiệu quả sử dụng. Một kỹ thuật để khắc phục là **dồn bộ nhớ (compaction)**, nhưng nó tốn thời gian của bộ xử lý.

Để giải quyết vấn đề địa chỉ thay đổi mỗi khi tiến trình được hoán đổi vào, OS phân biệt giữa **địa chỉ logic** (vị trí tương đối so với đầu chương trình) và **địa chỉ vật lý** (vị trí thực tế trong bộ nhớ chính). Bộ xử lý tự động chuyển đổi địa chỉ logic thành địa chỉ vật lý bằng cách cộng **địa chỉ cơ sở (base address)** của tiến trình.

**C. Phân Trang (Paging)**

Để giải quyết sự kém hiệu quả của phân hoạch, **phân trang (paging)** chia bộ nhớ thành các khối có kích thước cố định và bằng nhau gọi là **khung (frames)**, và chia mỗi tiến trình thành các khối có cùng kích thước gọi là **trang (pages)**. Các trang của một tiến trình được nạp vào các khung có sẵn, không nhất thiết phải liền kề. OS duy trì một **bảng trang (page table)** cho mỗi tiến trình để ánh xạ trang logic sang khung vật lý. Một địa chỉ logic bao gồm số trang và độ lệch trong trang, được phần cứng bộ xử lý dịch sang địa chỉ vật lý bằng cách sử dụng bảng trang.

**D. Bộ Nhớ Ảo (Virtual Memory)**

Với phân trang, ta có thể phát triển khái niệm **bộ nhớ ảo (virtual memory)**. Kỹ thuật cốt lõi là **phân trang theo yêu cầu (demand paging)**, nghĩa là mỗi trang của một tiến trình chỉ được đưa vào bộ nhớ khi nó thực sự cần thiết. Khi một chương trình tham chiếu đến một trang không có trong bộ nhớ chính, một **lỗi trang (page fault)** xảy ra, báo cho OS đưa trang đó vào.

* **Ưu điểm:** Nhiều tiến trình hơn có thể được duy trì trong bộ nhớ, và thời gian được tiết kiệm vì các trang không sử dụng không bị hoán đổi ra vào.
* **Nhược điểm:** OS phải quản lý việc **thay thế trang (page replacement)**. Nếu thay thế một trang ngay trước khi nó được sử dụng, có thể dẫn đến tình trạng **thrashing**, khi bộ xử lý dành phần lớn thời gian để hoán đổi trang thay vì thực thi lệnh.

Một hệ quả đáng chú ý của bộ nhớ ảo là một tiến trình có thể lớn hơn toàn bộ bộ nhớ chính. Bộ nhớ mà lập trình viên cảm nhận (trên đĩa) được gọi là bộ nhớ ảo, trong khi bộ nhớ chính được gọi là **bộ nhớ thực (real memory)**.

**E. Bộ Đệm Dịch Chuyển Tức Thời (Translation Lookaside Buffer - TLB)**

Mỗi tham chiếu bộ nhớ ảo có thể gây ra hai lần truy cập bộ nhớ vật lý: một để lấy mục trong bảng trang và một để lấy dữ liệu mong muốn. Để khắc phục điều này, hầu hết các hệ thống bộ nhớ ảo sử dụng một bộ đệm đặc biệt cho các mục của bảng trang, gọi là **Translation Lookaside Buffer (TLB)**. TLB hoạt động như một bộ đệm bộ nhớ và chứa các mục của bảng trang đã được sử dụng gần đây nhất, giúp cải thiện đáng kể hiệu năng.

**F. Phân Đoạn (Segmentation)**

**Phân đoạn (segmentation)** là một cách khác để chia nhỏ bộ nhớ, thường hiển thị với lập trình viên. Nó cho phép lập trình viên xem bộ nhớ như một tập hợp nhiều không gian địa chỉ hoặc **đoạn (segments)** có kích thước thay đổi.

* **Ưu điểm:** Đơn giản hóa việc xử lý các cấu trúc dữ liệu đang phát triển, cho phép các chương trình được thay đổi và biên dịch lại một cách độc lập, tạo điều kiện cho việc chia sẻ giữa các tiến trình và bảo vệ. Để kết hợp ưu điểm của cả phân trang và phân đoạn, một số hệ thống được trang bị cả phần cứng và phần mềm để cung cấp cả hai.

CHƯƠNG 9: HỆ THỐNG SỐ (NUMBER SYSTEMS)

### **CHƯƠG 9: HỆ THỐNG SỐ (NUMBER SYSTEMS)**

**Mục tiêu của chương này là gì?**

Sau khi học xong chương này, bạn sẽ có thể:

* Hiểu các khái niệm cơ bản và thuật ngữ của **hệ thống số vị trí (positional number systems)**.
* Giải thích các kỹ thuật **chuyển đổi giữa hệ thập phân và hệ nhị phân** cho cả số nguyên và phân số.
* Giải thích lý do của việc sử dụng **ký hiệu thập lục phân (hexadecimal notation)**.

#### **9.1. Hệ Thập Phân (The Decimal System)**

Trong cuộc sống hàng ngày, chúng ta sử dụng hệ thống dựa trên các chữ số thập phân (0, 1, 2, 3, 4, 5, 6, 7, 8, 9) để biểu diễn các số. Hệ thống này được gọi là hệ thập phân và có **cơ số (base hoặc radix) là 10**. Điều này có nghĩa là mỗi chữ số trong một số được nhân với 10 lũy thừa một số mũ tương ứng với vị trí của nó.

Ví dụ:

* **4728** = (4 × 10³) + (7 × 10²) + (2 × 10¹) + (8 × 10⁰).
* **0.256** = (2 × 10⁻¹) + (5 × 10⁻²) + (6 × 10⁻³).

Trong một số bất kỳ, chữ số ngoài cùng bên trái được gọi là **chữ số có nghĩa nhất (most significant digit)** và chữ số ngoài cùng bên phải là **chữ số ít có nghĩa nhất (least significant digit)**. Mỗi vị trí được gán một trọng số bằng 10 lần giá trị của vị trí bên phải nó.

#### **9.2. Hệ Thống Số Vị Trí (Positional Number Systems)**

Trong một hệ thống số vị trí, mỗi số được biểu diễn bằng một chuỗi các chữ số, trong đó mỗi vị trí *i* có một **trọng số** liên quan là **rⁱ**, với *r* là **cơ số (radix)** của hệ thống số. Dạng tổng quát của một số trong hệ thống này là:

**Giá trị** = **∑(aᵢ × rⁱ)**

Trong đó *aᵢ* là chữ số ở vị trí *i* và *0 ≤ aᵢ < r*. Dấu chấm giữa *a₀* và *a₋₁* được gọi là **dấu radix (radix point)**.

#### **9.3. Hệ Nhị Phân (The Binary System)**

Trong hệ thống nhị phân, chúng ta chỉ có hai chữ số là **1 và 0**, và các số được biểu diễn theo **cơ số 2**. Mỗi chữ số trong một số nhị phân có một giá trị phụ thuộc vào vị trí của nó.

Ví dụ:

* **11₂** = (1 × 2¹) + (1 × 2⁰) = **3₁₀**.
* **1001.101₂** = (1 × 2³) + (0 × 2²) + (0 × 2¹) + (1 × 2⁰) + (1 × 2⁻¹) + (0 × 2⁻²) + (1 × 2⁻³) = **9.625₁₀**.

Dạng tổng quát của một số nhị phân là: **Giá trị** = **∑(bᵢ × 2ⁱ)**

#### **9.4. Chuyển Đổi Giữa Hệ Nhị Phân và Hệ Thập Phân**

* **Chuyển đổi từ Nhị phân sang Thập phân:** Nhân mỗi chữ số nhị phân với lũy thừa 2 tương ứng và cộng các kết quả lại.
* **Chuyển đổi từ Thập phân sang Nhị phân:** Phần nguyên và phần thập phân được xử lý riêng biệt.  
  + **Đối với phần nguyên:** Sử dụng phương pháp **chia liên tiếp cho 2**. Số dư thu được trong mỗi lần chia (theo thứ tự ngược lại) sẽ tạo thành các chữ số nhị phân, từ ít có nghĩa nhất đến có nghĩa nhất. (Xem ví dụ trong Hình 9.1 của sách giáo khoa).
  + **Đối với phần thập phân:** Sử dụng phương pháp **nhân liên tiếp với 2**. Phần nguyên của mỗi kết quả nhân (0 hoặc 1) sẽ tạo thành các chữ số nhị phân, từ có nghĩa nhất trở đi. (Xem ví dụ trong Hình 9.2 của sách giáo khoa).

#### **9.5. Ký Hiệu Thập Lục Phân (Hexadecimal Notation)**

Do tính chất nhị phân của các thành phần máy tính, mọi dạng dữ liệu đều được biểu diễn bằng mã nhị phân. Tuy nhiên, hệ nhị phân rất cồng kềnh đối với con người. Do đó, ký hiệu **thập lục phân (hexadecimal)** đã được sử dụng.

* **Nguyên tắc:** Các chữ số nhị phân được nhóm thành các bộ 4 bit, được gọi là **nibble**. Mỗi sự kết hợp của bốn chữ số nhị phân được gán một ký hiệu.
* **Các ký hiệu:** 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F (tổng cộng 16 ký hiệu, tương ứng với cơ số 16).

**Lý do sử dụng ký hiệu thập lục phân:**

1. **Gọn gàng hơn** so với ký hiệu nhị phân.
2. Trong hầu hết các máy tính, dữ liệu nhị phân chiếm một bội số của 4 bit, do đó là bội số của một chữ số thập lục phân duy nhất.
3. **Việc chuyển đổi giữa hệ nhị phân và hệ thập lục phân cực kỳ dễ dàng**.

Ví dụ chuyển đổi:

* Chuỗi nhị phân **1101 1110 0001** tương đương với **DE1₁₆**.
* Số thập lục phân **2C₁₆** = (2 × 16¹) + (12 × 16⁰) = **44₁₀**.

CHƯƠG 10: SỐ HỌC MÁY TÍNH (COMPUTER ARITHMETIC)

### **CHƯƠG 10: SỐ HỌC MÁY TÍNH (COMPUTER ARITHMETIC)**

**Mục tiêu của chương này là gì?**

Sau khi học xong chương này, bạn sẽ có thể:

* Hiểu sự khác biệt giữa cách biểu diễn số (định dạng nhị phân) và các thuật toán được sử dụng cho các phép toán số học cơ bản.
* Giải thích cách biểu diễn số bù hai.
* Trình bày tổng quan về các kỹ thuật thực hiện các phép toán số học cơ bản trong ký hiệu bù hai.
* Hiểu việc sử dụng phần định trị (significand), cơ số (base), và số mũ (exponent) trong biểu diễn số dấu phẩy động.
* Trình bày tổng quan về tiêu chuẩn IEEE 754 cho biểu diễn dấu phẩy động.
* Hiểu một số khái niệm chính liên quan đến số học dấu phẩy động, bao gồm các bit bảo vệ (guard bits), làm tròn (rounding), số không chuẩn (subnormal numbers), tràn dưới (underflow) và tràn trên (overflow).

#### **10.1. Đơn Vị Số Học và Logic (The Arithmetic and Logic Unit - ALU)**

ALU là một phần của máy tính thực sự thực hiện các phép toán số học và logic trên dữ liệu. Tất cả các thành phần khác của hệ thống máy tính—đơn vị điều khiển, thanh ghi, bộ nhớ, I/O—chủ yếu có nhiệm vụ đưa dữ liệu vào ALU để xử lý và sau đó lấy kết quả ra. Một ALU và tất cả các thành phần điện tử trong máy tính đều dựa trên việc sử dụng các thiết bị logic số đơn giản có thể lưu trữ các chữ số nhị phân và thực hiện các phép toán logic Boole đơn giản.

Hình 10.1 trong sách giáo khoa cho thấy cách ALU được kết nối với phần còn lại của bộ xử lý. Các toán hạng cho các phép toán số học và logic được đưa vào ALU trong các thanh ghi, và kết quả của một phép toán được lưu trữ trong các thanh ghi. Các thanh ghi này là các vị trí lưu trữ tạm thời trong bộ xử lý được kết nối bằng các đường tín hiệu đến ALU. ALU cũng có thể thiết lập các cờ (flags) là kết quả của một phép toán. Ví dụ, một cờ tràn (overflow flag) được đặt thành 1 nếu kết quả của một phép tính vượt quá độ dài của thanh ghi nơi nó sẽ được lưu trữ. Các giá trị cờ cũng được lưu trữ trong các thanh ghi trong bộ xử lý.

#### **10.2. Biểu Diễn Số Nguyên (Integer Representation)**

Trong hệ thống số nhị phân, các số tùy ý có thể được biểu diễn chỉ bằng các chữ số không và một, dấu trừ, và dấu chấm cơ số. Tuy nhiên, đối với mục đích lưu trữ và xử lý máy tính, chúng ta chỉ có thể sử dụng các chữ số nhị phân (0 và 1). Nếu chúng ta giới hạn ở các số nguyên không âm, việc biểu diễn rất đơn giản. Một từ 8-bit có thể biểu diễn các số từ 0 đến 255.

**A. Biểu diễn Dấu-Độ lớn (Sign-Magnitude Representation)**

Đây là hình thức biểu diễn đơn giản nhất sử dụng một bit dấu (sign bit). Bit ngoài cùng bên trái trong một từ được coi là bit dấu. Nếu bit dấu là 0, số đó là dương; nếu là 1, số đó là âm. Các bit còn lại (n-1 bit) chứa độ lớn của số nguyên.

* **Ví dụ:** +18 = **0**0010010; -18 = **1**0010010.
* **Nhược điểm:**
  1. Phép cộng và trừ đòi hỏi phải xem xét cả dấu và độ lớn tương đối của các số.
  2. Có hai cách biểu diễn cho số 0 (+0 và -0), điều này gây bất tiện.

Do những nhược điểm này, biểu diễn dấu-độ lớn hiếm khi được sử dụng.

**B. Biểu diễn Bù Hai (Twos Complement Representation)**

Đây là lược đồ phổ biến nhất cho biểu diễn số nguyên trong bộ xử lý. Nó cũng sử dụng bit ngoài cùng bên trái làm bit dấu, giúp dễ dàng kiểm tra xem một số nguyên là dương hay âm.

* **Đối với số dương (A ≥ 0):** bit dấu là 0, và các bit còn lại biểu diễn độ lớn giống như trong biểu diễn dấu-độ lớn. Phạm vi của các số nguyên dương có thể được biểu diễn là từ 0 đến 2^(n-1) - 1.
* **Đối với số âm (A < 0):** bit dấu là 1. Trọng số của bit có nghĩa nhất là -2^(n-1).
* **Công thức tổng quát** cho một số nguyên n-bit A trong biểu diễn bù hai là: **A = -2^(n-1) \* a\_(n-1) + ∑\_(i=0)^(n-2) a\_i \* 2^i**

Biểu diễn bù hai giải quyết các vấn đề của biểu diễn dấu-độ lớn: nó chỉ có một cách biểu diễn cho số 0 và tạo điều kiện cho các phép toán số học đơn giản.

**C. Mở Rộng Dải (Range Extension)**

Đôi khi cần phải lấy một số nguyên n-bit và lưu trữ nó trong m bit, trong đó m > n.

* **Đối với số dấu-độ lớn:** chỉ cần di chuyển bit dấu đến vị trí ngoài cùng bên trái mới và điền vào các bit trống bằng số không.
* **Đối với số bù hai:** di chuyển bit dấu đến vị trí ngoài cùng bên trái mới và điền vào các bit trống bằng các bản sao của bit dấu (0 cho số dương, 1 cho số âm). Quá trình này được gọi là **mở rộng dấu (sign extension)**.

**D. Biểu diễn Điểm Cố Định (Fixed-Point Representation)**

Các biểu diễn được thảo luận trong phần này đôi khi được gọi là điểm cố định vì dấu chấm cơ số (dấu chấm nhị phân) là cố định và được giả định là ở bên phải của chữ số ngoài cùng bên phải.

#### **10.3. Số Học Số Nguyên (Integer Arithmetic)**

**A. Phủ định (Negation)**

Để tạo ra phủ định của một số nguyên trong ký hiệu bù hai:

1. Lấy phần bù Boole của mỗi bit của số nguyên (bao gồm cả bit dấu).
2. Coi kết quả là một số nguyên nhị phân không dấu và cộng thêm 1.

Quá trình hai bước này được gọi là **phép toán bù hai (twos complement operation)**.

* **Ví dụ:** +18 = 00010010. Bù bit: 11101101. Cộng 1: 11101110 = -18.
* **Trường hợp đặc biệt:** Phủ định của 0 là 0. Tuy nhiên, phủ định của -2^(n-1) (ví dụ: -128 cho 8-bit) sẽ cho lại chính nó, một sự bất thường không thể tránh khỏi vì có một số lượng lẻ các số nguyên khác không để biểu diễn.

**B. Phép Cộng và Phép Trừ**

* **Phép cộng:** Các số bù hai được cộng như thể chúng là các số nguyên không dấu. Bất kỳ bit nhớ nào vượt ra ngoài từ đều bị bỏ qua.
  + **Quy tắc tràn (Overflow Rule):** Nếu hai số được cộng và chúng cùng dấu (cả hai đều dương hoặc cả hai đều âm), thì tràn xảy ra khi và chỉ khi kết quả có dấu ngược lại.
* **Phép trừ:**
  + **Quy tắc trừ (Subtraction Rule):** Để trừ một số (số trừ) khỏi một số khác (số bị trừ), lấy bù hai (phủ định) của số trừ và cộng nó vào số bị trừ. Do đó, phép trừ được thực hiện bằng cách sử dụng phép cộng, và quy tắc tràn vẫn áp dụng.

**C. Phép Nhân**

* **Số nguyên không dấu:** Phép nhân liên quan đến việc tạo ra các **tích riêng (partial products)**, một cho mỗi chữ số trong số nhân. Các tích riêng này sau đó được cộng lại để tạo ra tích cuối cùng. Mỗi tích riêng kế tiếp được dịch sang trái một vị trí so với tích riêng trước đó.
* **Số nguyên bù hai:** Phép nhân đơn giản không hoạt động nếu một trong hai toán hạng là âm. Một trong những kỹ thuật phổ biến nhất là **thuật toán của Booth (Booth's algorithm)**, thuật toán này cũng có lợi ích là tăng tốc quá trình nhân. Thuật toán của Booth kiểm tra các bit của số nhân từng cặp và thực hiện phép cộng, trừ, hoặc chỉ dịch chuyển, dựa trên các mẫu bit (00, 01, 10, 11).

**D. Phép Chia**

Phép chia phức tạp hơn phép nhân nhưng dựa trên các nguyên tắc chung tương tự, liên quan đến việc dịch chuyển và cộng hoặc trừ lặp đi lặp lại.

* Quá trình này tương tự như phép chia dài trên giấy, trong đó các bit của **số bị chia (dividend)** được kiểm tra từ trái sang phải cho đến khi tập hợp các bit được kiểm tra lớn hơn hoặc bằng **số chia (divisor)**.
* Đối với các số bù hai, một cách tiếp cận là chuyển đổi các toán hạng thành các giá trị không dấu, thực hiện phép chia, và sau đó điều chỉnh dấu của **thương (quotient)** và **số dư (remainder)** khi cần.

#### **10.4. Biểu Diễn Dấu Phẩy Động (Floating-Point Representation)**

**A. Các Nguyên Tắc**

Để biểu diễn một dải số rất lớn và các phân số rất nhỏ, chúng ta sử dụng một dạng tương tự như ký hiệu khoa học, được gọi là ký hiệu dấu phẩy động. Một số được biểu diễn dưới dạng: **±S × B^±E** Nó được lưu trữ trong một từ nhị phân với ba trường:

* **Dấu (Sign):** cộng hoặc trừ.
* **Phần định trị (Significand) S:** phần có nghĩa của số.
* **Số mũ (Exponent) E:** giữ giá trị của số mũ.
* **Cơ số (Base) B:** là ẩn và không cần lưu trữ.

Thông thường, số mũ được lưu trữ dưới dạng **biểu diễn thiên lệch (biased representation)**. Một giá trị cố định, gọi là thiên lệch, được trừ khỏi trường để có được giá trị số mũ thực. Điều này cho phép các số dấu phẩy động không âm được coi là số nguyên cho mục đích so sánh.

Các số dấu phẩy động thường được **chuẩn hóa (normalized)**, nghĩa là chữ số có nghĩa nhất của phần định trị là khác không. Đối với biểu diễn cơ số 2, điều này có nghĩa là bit có nghĩa nhất là 1, và bit này có thể được ẩn đi để tiết kiệm không gian, mang lại một bit chính xác hiệu quả.

**B. Tiêu chuẩn IEEE 754 cho Biểu diễn Dấu phẩy động Nhị phân**

Đây là tiêu chuẩn được sử dụng rộng rãi nhất cho số học dấu phẩy động.

* Nó xác định các định dạng có độ dài bit khác nhau: **nhị phân32 (độ chính xác đơn)**, **nhị phân64 (độ chính xác kép)**, và **nhị phân128 (độ chính xác bốn)**, với các số mũ lần lượt là 8, 11, và 15 bit.
* Tiêu chuẩn này cũng xác định các giá trị đặc biệt:
  + **Số không dương và âm:** Khi số mũ và phần phân số đều bằng không.
  + **Vô cùng dương và âm:** Khi số mũ toàn là 1 và phần phân số bằng không.
  + **Số không chuẩn (Subnormal Number):** Khi số mũ bằng không và phần phân số khác không. Chúng được sử dụng để xử lý các trường hợp tràn dưới số mũ (exponent underflow).
  + **NaN (Not a Number):** Khi số mũ toàn là 1 và phần phân số khác không, được sử dụng để báo hiệu các điều kiện ngoại lệ khác nhau.

#### **10.5. Số Học Dấu Phẩy Động (Floating-Point Arithmetic)**

* **Phép cộng và trừ:** Phức tạp hơn phép nhân và chia vì chúng đòi hỏi phải **căn chỉnh phần định trị (aligning the significands)** để các số mũ bằng nhau. Quá trình này bao gồm bốn giai đoạn cơ bản: kiểm tra số không, căn chỉnh phần định trị, cộng hoặc trừ phần định trị, và chuẩn hóa kết quả.
* **Phép nhân và chia:** Đơn giản hơn. Đối với phép nhân, các số mũ được cộng lại và các phần định trị được nhân lên. Đối với phép chia, số mũ của số chia được trừ khỏi số mũ của số bị chia và các phần định trị được chia.

**A. Các Vấn đề về Độ Chính Xác**

* **Các bit bảo vệ (Guard Bits):** Các thanh ghi ALU thường rộng hơn độ dài của phần định trị để chứa các bit bổ sung, được gọi là các bit bảo vệ. Các bit này được sử dụng để đệm vào phần cuối bên phải của phần định trị bằng các số không trong quá trình căn chỉnh, ngăn ngừa việc mất các bit có nghĩa.
* **Làm tròn (Rounding):** Kết quả của một phép toán thường được lưu trữ trong một thanh ghi dài hơn và phải được làm tròn để vừa với định dạng dấu phẩy động. Tiêu chuẩn IEEE 754 liệt kê bốn phương pháp làm tròn: **làm tròn đến số gần nhất (round to nearest)**, **làm tròn về phía +∞ (round toward +∞)**, **làm tròn về phía -∞ (round toward -∞)**, và **làm tròn về phía 0 (round toward 0)**.

CHƯƠNG 11: LOGIC SỐ (DIGITAL LOGIC)

### **CHƯƠNG 11: LOGIC SỐ (DIGITAL LOGIC)**

**Mục tiêu của chương này là gì?**

Sau khi học xong chương này, bạn sẽ có thể:

* Hiểu các phép toán cơ bản của **đại số Boole (Boolean algebra)**.
* Phân biệt được các loại **flip-flop** khác nhau.
* Sử dụng **bản đồ Karnaugh** để đơn giản hóa một biểu thức Boole.
* Trình bày tổng quan về **các thiết bị logic khả trình (programmable logic devices)**.

#### **11.1. Đại Số Boole (Boolean Algebra)**

Hoạt động của máy tính số dựa trên việc lưu trữ và xử lý dữ liệu nhị phân. **Đại số Boole**, được đặt theo tên nhà toán học người Anh George Boole, là một môn toán học được sử dụng để thiết kế và phân tích mạch logic số. Nó là một công cụ tiện lợi cho hai mục đích chính:

* **Phân tích:** Là một cách kinh tế để mô tả chức năng của mạch logic số.
* **Thiết kế:** Cho một chức năng mong muốn, đại số Boole có thể được áp dụng để phát triển một cách triển khai đơn giản hóa cho chức năng đó.

Đại số Boole sử dụng các biến và các phép toán logic. Một biến chỉ có thể nhận một trong hai giá trị: 1 (TRUE) hoặc 0 (FALSE). Các phép toán logic cơ bản là **AND, OR, và NOT**.

* **AND** (biểu thị bằng dấu chấm . hoặc viết liền): Cho kết quả là 1 (TRUE) khi và chỉ khi cả hai toán hạng đều là 1.
* **OR** (biểu thị bằng dấu cộng +): Cho kết quả là 1 (TRUE) nếu một trong hai hoặc cả hai toán hạng là 1.
* **NOT** (biểu thị bằng gạch ngang trên biến, ví dụ Ā): Đảo ngược giá trị của toán hạng (1 thành 0, 0 thành 1).

Ngoài ra, còn có các phép toán hữu ích khác như **XOR, NAND, và NOR**.

* **Exclusive-OR (XOR):** Cho kết quả là 1 khi và chỉ khi có đúng một trong hai toán hạng là 1.
* **NAND:** Là phép phủ định (NOT) của AND.
* **NOR:** Là phép phủ định (NOT) của OR.

Các phép toán này có thể được định nghĩa bằng **bảng chân lý (truth table)**, liệt kê giá trị của một phép toán cho mọi tổ hợp giá trị có thể có của các toán hạng. Bảng 11.2 trong sách giáo khoa tóm tắt các đồng nhất thức quan trọng của đại số Boole, bao gồm các định lý DeMorgan.

#### **11.2. Cổng Logic (Gates)**

Khối xây dựng cơ bản của tất cả các mạch logic số là **cổng logic (gate)**. Một cổng là một mạch điện tử tạo ra một tín hiệu đầu ra là một phép toán Boole đơn giản trên các tín hiệu đầu vào của nó. Các cổng logic cơ bản là AND, OR, NOT, NAND, NOR, và XOR.

Một tập hợp các cổng được gọi là **hoàn chỉnh về mặt chức năng (functionally complete)** nếu bất kỳ hàm Boole nào cũng có thể được triển khai chỉ bằng cách sử dụng các cổng trong tập hợp đó. Các tập hợp hoàn chỉnh về mặt chức năng bao gồm:

* {AND, OR, NOT}
* {AND, NOT}
* {OR, NOT}
* {NAND}
* {NOR}

Do đó, các mạch logic số thường được triển khai chỉ bằng cổng NAND hoặc chỉ bằng cổng NOR để đơn giản hóa việc thiết kế và chế tạo.

#### **11.3. Mạch Tổ Hợp (Combinational Circuits)**

Một **mạch tổ hợp (combinational circuit)** là một tập hợp các cổng được kết nối với nhau mà đầu ra tại bất kỳ thời điểm nào chỉ là một hàm của đầu vào tại thời điểm đó. Mạch tổ hợp không có bộ nhớ; chúng không lưu giữ thông tin về các đầu vào trước đó.

**A. Triển khai các Hàm Boole**

Bất kỳ hàm Boole nào cũng có thể được triển khai dưới dạng một mạng lưới các cổng. Thông thường, một hàm Boole được biểu diễn dưới hai dạng chuẩn:

1. **Dạng Tổng của các Tích (Sum of Products - SOP):** Biểu thức được tạo thành từ việc OR các tích (AND) của các biến. Mỗi tích tương ứng với một hàng trong bảng chân lý có đầu ra là 1.
2. **Dạng Tích của các Tổng (Product of Sums - POS):** Biểu thức được tạo thành từ việc AND các tổng (OR) của các biến. Mỗi tổng tương ứng với một hàng trong bảng chân lý có đầu ra là 0.

**B. Đơn giản hóa Hàm Boole**

Một hàm Boole càng đơn giản thì càng cần ít cổng logic để triển khai nó. Các phương pháp để đơn giản hóa một hàm Boole bao gồm:

* **Đơn giản hóa đại số (Algebraic simplification):** Sử dụng các đồng nhất thức của đại số Boole để rút gọn biểu thức.
* **Bản đồ Karnaugh (Karnaugh maps):** Là một cách biểu diễn trực quan một hàm Boole của một số lượng nhỏ biến (thường là 2 đến 4 biến). Bản đồ là một mảng các ô vuông, mỗi ô tương ứng với một tổ hợp đầu vào. Bằng cách nhóm các ô liền kề có giá trị 1, ta có thể rút ra một biểu thức Boole đơn giản hơn.
* **Phương pháp Quine–McCluskey:** Là một kỹ thuật dạng bảng phù hợp cho việc lập trình trên máy tính để tự động tối thiểu hóa các biểu thức Boole, đặc biệt hữu ích khi có nhiều hơn bốn biến.

**C. Một số Mạch Tổ hợp Phổ biến**

* **Bộ ghép kênh (Multiplexer - MUX):** Là một mạch kết nối nhiều đầu vào với một đầu ra duy nhất. Tại một thời điểm, chỉ một trong các đầu vào được chọn để truyền đến đầu ra, dựa trên một mã chọn (select code).
* **Bộ giải mã (Decoder):** Là một mạch có n đầu vào và 2^n đầu ra. Tại một thời điểm, chỉ một trong các đầu ra được khẳng định (asserted), tùy thuộc vào mẫu của các đầu vào.
* **Bộ nhớ chỉ đọc (Read-Only Memory - ROM):** Về bản chất, ROM là một mạch tổ hợp. Nó có thể được xem như một bảng chân lý, trong đó các đầu vào là các đường địa chỉ và các đầu ra là các đường dữ liệu.
* **Bộ cộng (Adder):** Là mạch thực hiện phép cộng nhị phân. Một bộ cộng đầy đủ (full adder) cho một bit sẽ có ba đầu vào (hai bit cần cộng và một bit nhớ từ vị trí trước) và hai đầu ra (bit tổng và bit nhớ cho vị trí sau). Các bộ cộng nhiều bit có thể được xây dựng bằng cách kết nối các bộ cộng một bit. Để tăng tốc, kỹ thuật **nhớ trước (carry lookahead)** có thể được sử dụng để tính toán các bit nhớ mà không cần phải chờ chúng lan truyền qua từng giai đoạn.

*Lưu ý: Các phần về Mạch Tuần tự (Sequential Circuits) và Thiết bị Logic Khả trình (Programmable Logic Devices) không được yêu cầu trong câu hỏi của bạn. Nếu bạn muốn tìm hiểu về chúng, xin vui lòng cho tôi biết.*

**CHƯƠNG 11: LOGIC SỐ (Tiếp theo)**

**Mạch Tuần tự (Sequential Circuits)**

Phần này bổ sung cho các khái niệm về Flip-Flop đã có.

• **Định nghĩa:** Mạch tuần tự là mạch mà đầu ra hiện tại không chỉ phụ thuộc vào đầu vào hiện tại mà còn phụ thuộc vào lịch sử các đầu vào trước đó. Nói cách khác, đầu ra hiện tại phụ thuộc vào đầu vào hiện tại và trạng thái hiện tại của mạch. Mạch tuần tự có bộ nhớ để lưu trữ trạng thái.

• **Flip-Flop:** Là dạng mạch tuần tự đơn giản nhất và là khối xây dựng cơ bản cho các mạch phức tạp hơn như thanh ghi và bộ đếm.

◦ **Tính chất:** Flip-flop là một thiết bị lưỡng ổn (bistable), nghĩa là nó có hai trạng thái ổn định (biểu diễn bit 0 và 1) và sẽ duy trì trạng thái đó nếu không có tín hiệu đầu vào. Nó có hai đầu ra, Q và Q̅, luôn ở trạng thái đối lập nhau. Vì vậy, nó có thể hoạt động như một bộ nhớ 1-bit.

◦ **Hoạt động không đồng bộ (Asynchronous):** Đầu ra của S-R Latch thay đổi gần như ngay lập tức sau khi đầu vào thay đổi (chỉ trễ một khoảng thời gian nhỏ).

◦ **Hoạt động đồng bộ (Synchronous):** Hầu hết các sự kiện trong máy tính số được đồng bộ hóa với một xung nhịp (clock pulse). Các flip-flop có xung nhịp (clocked flip-flops) chỉ thay đổi trạng thái khi có xung nhịp, giúp kiểm soát thời gian tốt hơn trong các mạch phức tạp.

• **Thanh ghi (Registers):**

◦ **Thanh ghi song song (Parallel Register):** Gồm một tập hợp các bộ nhớ 1-bit (flip-flop) có thể đọc hoặc ghi đồng thời, dùng để lưu trữ dữ liệu. Các thanh ghi đa năng trong CPU là loại này.

◦ **Thanh ghi dịch (Shift Register):** Chấp nhận và/hoặc truyền thông tin một cách tuần tự, từng bit một. Chúng được dùng để giao tiếp với các thiết bị I/O tuần tự hoặc thực hiện các phép dịch logic và quay vòng trong ALU.

• **Bộ đếm (Counters):**

◦ Là một thanh ghi mà giá trị của nó có thể được tăng lên 1 một cách dễ dàng. Một thanh ghi n-bit có thể đếm đến 2ⁿ-1.

◦ **Bộ đếm gợn sóng (Ripple Counter):** Là bộ đếm không đồng bộ, sự thay đổi trạng thái bắt đầu từ một đầu và "lan truyền" (ripple) đến đầu kia. Nó chậm vì độ trễ tỷ lệ với độ dài bộ đếm.

◦ **Bộ đếm đồng bộ (Synchronous Counter):** Tất cả các flip-flop thay đổi trạng thái cùng lúc, nhanh hơn nhiều và được sử dụng trong CPU.

• **Thiết bị Logic Khả trình (Programmable Logic Devices - PLD):**

◦ Thay vì xây dựng mạch tùy chỉnh cho mỗi chức năng logic, PLD là các chip đa năng có thể được cấu hình để thực hiện các chức năng cụ thể.

◦ **PLA (Programmable Logic Array):** Gồm một mảng AND và một mảng OR, cả hai đều có thể lập trình được. Bất kỳ hàm Boole nào cũng có thể được biểu diễn dưới dạng Tổng của các Tích (SOP) và triển khai trên PLA.

◦ **FPGA (Field-Programmable Gate Array):** Là một loại CPLD (Complex PLD) rất phổ biến, chứa một mảng các khối logic (logic blocks) và các tài nguyên kết nối có thể lập trình được. FPGA cho phép triển khai các logic rất phức tạp.

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

Huong dan giai bai tap:

## **Phần 1: Lý Thuyết Nền Tảng**

### **1. Đại số Boolean (Boolean Algebra) là gì?**

💡 Đại số Boolean là một nhánh toán học dùng để phân tích các mạch logic (digital logic circuits). Nó chỉ làm việc với hai giá trị: **Đúng (True)** và **Sai (False)**, thường được biểu diễn bằng **1** và **0**.

### **2. Các Phép Toán Cơ Bản**

Có 3 phép toán cốt lõi bạn cần nắm:

* **Phép AND (Nhân Logic):**
  + Ký hiệu: Dấu chấm (.), dấu ^, hoặc viết liền nhau (ví dụ: A.B hoặc AB).
  + Ý nghĩa: Cho kết quả là **1** chỉ khi **TẤT CẢ** các đầu vào đều là **1**.
  + Cổng logic: **AND gate**.

| A | B | A.B |
| --- | --- | --- |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

* **Phép OR (Cộng Logic):**
  + Ký hiệu: Dấu cộng (+), dấu v.
  + Ý nghĩa: Cho kết quả là **1** nếu có **BẤT KỲ** đầu vào nào là **1**.
  + Cổng logic: **OR gate**.

| A | B | A+B |
| --- | --- | --- |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

* **Phép NOT (Phủ Định):**
  + Ký hiệu: Gạch ngang trên đầu (ví dụ: Ā), dấu phẩy (A'), hoặc dấu ~.
  + Ý nghĩa: Đảo ngược giá trị đầu vào (0 thành 1, 1 thành 0).
  + Cổng logic: **NOT gate** (hay Inverter).

| A | Ā |
| --- | --- |
| 0 | 1 |
| 1 | 0 |

### **3. Các Dạng Biểu Diễn Một Hàm Boolean**

Một hàm logic có thể được biểu diễn theo 3 cách tương đương nhau:

1. **Phương trình Boolean (Boolean Equation):** Công thức toán học sử dụng các biến và phép toán logic.
2. **Bảng sự thật (Truth Table):** Một bảng liệt kê tất cả các tổ hợp đầu vào có thể có và kết quả đầu ra tương ứng.
3. **Sơ đồ cổng logic (Graphical Symbol/Logic Diagram):** Một sơ đồ trực quan sử dụng các ký hiệu cổng logic để thể hiện phương trình.

## **Phần 2: Hướng Dẫn Cụ Thể Từng Dạng Bài**

Chúng ta sẽ phân tích hai trường hợp tiêu biểu từ bài tập của bạn: dạng **SOP** (Tổng của các Tích) và dạng **POS** (Tích của các Tổng).

### **Trường hợp 1: Biểu thức dạng Tổng của các Tích (Sum of Products - SOP)**

Đây là dạng mà nhiều "cụm nhân" (tích) được "cộng" (tổng) lại với nhau.

Ví dụ: F = ABC + A̅B̅C̅ (Câu a)

#### **Cách xây dựng Bảng sự thật (Truth Table)**

1. **Xác định số biến:** Có 3 biến (A, B, C). Vậy bảng sẽ có hàng, tương ứng với các tổ hợp từ 000 đến 111.
2. **Phân tích từng Tích (Product term):**
   * Cụm ABC: Cụm này sẽ bằng 1 khi nào? Khi A=1 VÀ B=1 VÀ C=1. Bạn hãy tìm đến hàng 111 trong bảng và điền 1 vào cột F.
   * Cụm A̅B̅C̅: Cụm này sẽ bằng 1 khi nào? Khi A̅=1 VÀ B̅=1 VÀ C̅=1, tức là khi A=0 VÀ B=0 VÀ C=0. Bạn tìm đến hàng 000 và điền 1 vào cột F.
3. **Xác định giá trị cuối cùng của F:**
   * Vì phương trình là phép cộng (OR) của hai cụm trên, F sẽ bằng 1 nếu **BẤT KỲ** cụm nào bằng 1.
   * Do đó, F = 1 tại hàng 000 và hàng 111.
   * Tất cả các hàng còn lại, F sẽ bằng 0.

✅ **Kết quả Bảng sự thật:**

| A | B | C | F | *Ghi chú* |
| --- | --- | --- | --- | --- |
| 0 | 0 | 0 | **1** | Vì A̅B̅C̅ = 1 |
| 0 | 0 | 1 | **0** |  |
| 0 | 1 | 0 | **0** |  |
| 0 | 1 | 1 | **0** |  |
| 1 | 0 | 0 | **0** |  |
| 1 | 0 | 1 | **0** |  |
| 1 | 1 | 0 | **0** |  |
| 1 | 1 | 1 | **1** | Vì ABC = 1 |

#### **Cách vẽ Sơ đồ cổng logic (Logic Diagram)**

Vẽ theo cấu trúc của phương trình:

1. **Lớp đầu vào và phủ định:** Vẽ 3 đường tín hiệu đầu vào cho A, B, C. Dùng các cổng NOT để tạo ra A̅, B̅, C̅ nếu cần.
2. **Lớp Tích (AND):**
   * Vẽ một cổng AND 3 đầu vào cho cụm ABC. Nối các đầu vào A, B, C vào cổng này.
   * Vẽ một cổng AND 3 đầu vào khác cho cụm A̅B̅C̅. Nối các đầu vào A̅, B̅, C̅ vào cổng này.
3. **Lớp Tổng (OR):**
   * Lấy đầu ra của 2 cổng AND ở trên, nối chúng vào một cổng OR 2 đầu vào.
   * Đầu ra của cổng OR này chính là F.

### **Trường hợp 2: Biểu thức dạng Tích của các Tổng (Product of Sums - POS)**

Đây là dạng mà nhiều "cụm cộng" (tổng) được "nhân" (tích) lại với nhau.

Ví dụ: F = (A + B)(A + C)(A̅ + B̅) (Câu d)

#### **Cách xây dựng Bảng sự thật (Truth Table)**

Logic ở đây hơi ngược lại với SOP. F sẽ bằng 0 nếu **BẤT KỲ** cụm tổng nào bằng 0. Vì vậy, ta đi tìm các trường hợp làm cho F = 0.

1. **Lập bảng 8 hàng** như trên.
2. **Phân tích từng Tổng (Sum term):**
   * Cụm (A + B): Cụm này bằng 0 khi nào? Khi A=0 VÀ B=0. Điều này xảy ra ở hàng 000 và 001. Vậy F chắc chắn bằng 0 ở hai hàng này.
   * Cụm (A + C): Cụm này bằng 0 khi nào? Khi A=0 VÀ C=0. Điều này xảy ra ở hàng 000 (đã xét) và 010. Vậy F cũng bằng 0 ở hàng 010.
   * Cụm (A̅ + B̅): Cụm này bằng 0 khi nào? Khi A̅=0 VÀ B̅=0, tức là A=1 VÀ B=1. Điều này xảy ra ở hàng 110 và 111. Vậy F cũng bằng 0 ở hai hàng này.
3. **Xác định giá trị cuối cùng của F:**
   * Tóm lại, F = 0 ở các hàng: 000, 001, 010, 110, 111.
   * Tất cả các hàng còn lại, F sẽ bằng 1.

✅ **Kết quả Bảng sự thật:**

| A | B | C | F | *Ghi chú (Các trường hợp làm F=0)* |
| --- | --- | --- | --- | --- |
| 0 | 0 | 0 | **0** | Vì A+B=0 và A+C=0 |
| 0 | 0 | 1 | **0** | Vì A+B=0 |
| 0 | 1 | 0 | **0** | Vì A+C=0 |
| 0 | 1 | 1 | **1** |  |
| 1 | 0 | 0 | **1** |  |
| 1 | 0 | 1 | **1** |  |
| 1 | 1 | 0 | **0** | Vì A̅+B̅=0 |
| 1 | 1 | 1 | **0** | Vì A̅+B̅=0 |

#### **Cách vẽ Sơ đồ cổng logic (Logic Diagram)**

Vẽ theo cấu trúc của phương trình:

1. **Lớp đầu vào và phủ định:** Tương tự, vẽ A, B, C và tạo A̅, B̅ nếu cần.
2. **Lớp Tổng (OR):**
   * Vẽ một cổng OR 2 đầu vào cho cụm (A + B). Nối A, B vào cổng này.
   * Vẽ một cổng OR 2 đầu vào cho cụm (A + C). Nối A, C vào cổng này.
   * Vẽ một cổng OR 2 đầu vào cho cụm (A̅ + B̅). Nối A̅, B̅ vào cổng này.
3. **Lớp Tích (AND):**
   * Lấy đầu ra của 3 cổng OR ở trên, nối chúng vào một cổng AND 3 đầu vào.
   * Đầu ra của cổng AND này chính là F.

Hy vọng với phần giải thích chi tiết này, bạn đã hiểu rõ hơn về bản chất và cách giải quyết các dạng bài tập này. Chúc bạn học tập tốt!

CHƯƠNG 12: TẬP LỆNH: CÁC ĐẶC ĐIỂM VÀ CHỨC NĂNG

### **CHƯƠNG 12: TẬP LỆNH: CÁC ĐẶC ĐIỂM VÀ CHỨC NĂNG**

**Mục tiêu của chương này là gì?**

Sau khi học xong chương này, bạn sẽ có thể:

* Trình bày tổng quan về các đặc điểm thiết yếu của lệnh máy.
* Mô tả các loại toán hạng được sử dụng trong các tập lệnh máy điển hình.
* Trình bày tổng quan về các kiểu dữ liệu của x86 và ARM.
* Mô tả các loại toán tử được hỗ trợ bởi các tập lệnh máy điển hình.
* Trình bày tổng quan về các loại toán tử của x86 và ARM.

#### **12.1. Các Đặc Điểm của Lệnh Máy (Machine Instruction Characteristics)**

Hoạt động của bộ xử lý được quyết định bởi các lệnh mà nó thực thi, được gọi là **lệnh máy (machine instructions)** hoặc **lệnh máy tính (computer instructions)**. Tập hợp các lệnh khác nhau mà bộ xử lý có thể thực thi được gọi là **tập lệnh (instruction set)** của bộ xử lý.

**A. Các Thành Phần của một Lệnh Máy**

Mỗi lệnh phải chứa thông tin cần thiết để bộ xử lý thực thi nó. Các thành phần này bao gồm:

1. **Mã toán tử (Operation code - opcode):** Chỉ định phép toán sẽ được thực hiện (ví dụ: ADD, I/O).
2. **Tham chiếu toán hạng nguồn (Source operand reference):** Phép toán có thể liên quan đến một hoặc nhiều toán hạng nguồn, là các đầu vào cho phép toán.
3. **Tham chiếu toán hạng kết quả (Result operand reference):** Phép toán có thể tạo ra một kết quả.
4. **Tham chiếu lệnh tiếp theo (Next instruction reference):** Cho bộ xử lý biết nơi để tìm nạp lệnh tiếp theo sau khi hoàn thành việc thực thi lệnh hiện tại.

Các toán hạng nguồn và kết quả có thể nằm ở một trong bốn khu vực sau:

* **Bộ nhớ chính hoặc bộ nhớ ảo:** Phải cung cấp địa chỉ bộ nhớ chính hoặc ảo.
* **Thanh ghi bộ xử lý:** Một lệnh phải chứa số hiệu của thanh ghi mong muốn.
* **Tức thời (Immediate):** Giá trị của toán hạng được chứa trong một trường trong chính lệnh đang được thực thi.
* **Thiết bị I/O:** Lệnh phải chỉ định module I/O và thiết bị cho hoạt động.

**B. Biểu Diễn Lệnh**

Bên trong máy tính, mỗi lệnh được biểu diễn bằng một chuỗi bit, được chia thành các **trường (fields)** tương ứng với các thành phần của lệnh. Để dễ đọc hơn cho con người, người ta thường sử dụng **biểu diễn tượng trưng (symbolic representation)**.

* **Mã toán tử (Opcodes)** được biểu diễn bằng các chữ viết tắt, gọi là **mnemonics**, để chỉ ra hoạt động. Ví dụ: ADD (cộng), SUB (trừ), LOAD (tải dữ liệu từ bộ nhớ), STOR (lưu dữ liệu vào bộ nhớ).
* **Toán hạng (Operands)** cũng được biểu diễn bằng ký hiệu. Ví dụ, lệnh ADD R, Y có thể có nghĩa là cộng giá trị tại vị trí dữ liệu Y vào nội dung của thanh ghi R.

**C. Các Loại Lệnh**

Một máy tính nên có một tập hợp lệnh cho phép người dùng xây dựng bất kỳ tác vụ xử lý dữ liệu nào. Các loại lệnh có thể được phân loại như sau:

* **Xử lý dữ liệu (Data processing):** Các lệnh số học và logic.
* **Lưu trữ dữ liệu (Data storage):** Các lệnh di chuyển dữ liệu vào hoặc ra khỏi các thanh ghi và/hoặc các vị trí bộ nhớ.
* **Di chuyển dữ liệu (Data movement):** Các lệnh I/O.
* **Điều khiển (Control):** Các lệnh kiểm tra và rẽ nhánh.

**D. Số Lượng Địa Chỉ**

Một trong những cách truyền thống để mô tả kiến trúc bộ xử lý là dựa trên số lượng địa chỉ có trong mỗi lệnh.

* **Hướng dẫn ba địa chỉ:** Mỗi lệnh chỉ định hai vị trí toán hạng nguồn và một vị trí toán hạng đích. Dạng này không phổ biến vì yêu cầu định dạng lệnh dài.
* **Hướng dẫn hai địa chỉ:** Một địa chỉ vừa là toán hạng vừa là nơi lưu kết quả.
* **Hướng dẫn một địa chỉ:** Một địa chỉ thứ hai phải được ngụ ý. Điều này thường là một thanh ghi bộ xử lý được gọi là **bộ tích lũy (accumulator - AC)**.
* **Hướng dẫn không địa chỉ:** Áp dụng cho một tổ chức bộ nhớ đặc biệt gọi là **ngăn xếp (stack)**.

Ít địa chỉ hơn mỗi lệnh dẫn đến các lệnh nguyên thủy hơn, yêu cầu bộ xử lý ít phức tạp hơn và có độ dài lệnh ngắn hơn. Mặt khác, các chương trình chứa nhiều lệnh hơn, dẫn đến thời gian thực thi dài hơn. Hầu hết các máy tính hiện đại sử dụng sự kết hợp giữa các lệnh hai và ba địa chỉ.

**E. Thiết Kế Tập Lệnh**

Thiết kế tập lệnh là một trong những khía cạnh thú vị và được phân tích nhiều nhất của thiết kế máy tính. Các vấn đề thiết kế cơ bản bao gồm:

* **Danh mục hoạt động (Operation repertoire):** Cung cấp bao nhiêu và những hoạt động nào, và các hoạt động nên phức tạp đến mức nào.
* **Kiểu dữ liệu (Data types):** Các loại dữ liệu khác nhau mà các hoạt động được thực hiện trên đó.
* **Định dạng lệnh (Instruction format):** Độ dài lệnh (tính bằng bit), số lượng địa chỉ, kích thước của các trường khác nhau, v.v..
* **Thanh ghi (Registers):** Số lượng thanh ghi bộ xử lý có thể được tham chiếu bởi các lệnh và cách sử dụng của chúng.
* **Địa chỉ hóa (Addressing):** Chế độ hoặc các chế độ mà địa chỉ của một toán hạng được chỉ định.

#### **12.2. Các Loại Toán Hạng (Types of Operands)**

Các lệnh máy hoạt động trên dữ liệu. Các loại dữ liệu chung quan trọng nhất là:

* **Địa chỉ (Addresses):** Có thể được coi là số nguyên không dấu.
* **Số (Numbers):** Tất cả các ngôn ngữ máy đều bao gồm các kiểu dữ liệu số. Có ba loại phổ biến:
  + **Số nguyên nhị phân hoặc điểm cố định nhị phân**
  + **Số dấu phẩy động nhị phân**
  + **Số thập phân:** Thường được biểu diễn dưới dạng **số thập phân được đóng gói (packed decimal)**, trong đó mỗi chữ số thập phân được biểu diễn bằng mã 4 bit, với hai chữ số được lưu trữ mỗi byte.
* **Ký tự (Characters):** Một dạng dữ liệu phổ biến là văn bản hoặc chuỗi ký tự. Mã ký tự được sử dụng phổ biến nhất là **Bảng chữ cái tham chiếu quốc tế (International Reference Alphabet - IRA)**, hay còn gọi là **ASCII**. Một mã khác được sử dụng trên các máy mainframe của IBM là **EBCDIC**.
* **Dữ liệu logic (Logical Data):** Một đơn vị n-bit có thể được coi là bao gồm n mục dữ liệu 1-bit, mỗi mục có giá trị 0 hoặc 1. Dữ liệu được xem theo cách này được coi là dữ liệu logic.

#### **12.4. Các Loại Hoạt Động (Types of Operations)**

Các loại hoạt động chung được tìm thấy trên tất cả các máy tính bao gồm:

* **Truyền dữ liệu (Data transfer):** Di chuyển dữ liệu từ vị trí này sang vị trí khác. Nếu có liên quan đến bộ nhớ, bộ xử lý phải tính toán địa chỉ bộ nhớ, dịch địa chỉ ảo sang địa chỉ thực, kiểm tra bộ đệm và khởi tạo đọc/ghi bộ nhớ.
* **Số học (Arithmetic):** Cung cấp khả năng tính toán cho việc xử lý dữ liệu số. Hầu hết các máy đều cung cấp các phép toán cộng, trừ, nhân và chia.
* **Logic (Logical):** Hoạt động trên các bit của một từ như các bit thay vì số. Bao gồm các phép toán như AND, OR, XOR, và NOT. Các lệnh dịch chuyển và quay cũng được bao gồm trong loại này.
* **Chuyển đổi (Conversion):** Các lệnh thay đổi định dạng hoặc hoạt động trên định dạng của dữ liệu (ví dụ: chuyển đổi từ thập phân sang nhị phân).
* **I/O:** Di chuyển dữ liệu giữa máy tính và các thiết bị bên ngoài.
* **Điều khiển hệ thống (System control):** Các lệnh đặc quyền chỉ có thể được thực thi khi bộ xử lý ở trạng thái đặc quyền, thường dành cho hệ điều hành sử dụng.
* **Chuyển giao điều khiển (Transfer of control):** Thay đổi trình tự thực thi lệnh. Điều này rất cần thiết cho các vòng lặp, ra quyết định và các thủ tục. Các loại phổ biến bao gồm:
  + **Rẽ nhánh (Branch):** Còn được gọi là nhảy (jump), có thể là có điều kiện hoặc không điều kiện.
  + **Bỏ qua (Skip):** Bao gồm một địa chỉ ngụ ý, thường là bỏ qua một lệnh.
  + **Gọi thủ tục (Procedure call):** Nhánh đến một thủ tục, lưu lại địa chỉ trả về. Việc trả về được thực hiện bằng một lệnh trả về. Ngăn xếp thường được sử dụng để quản lý các lệnh gọi thủ tục và truyền tham số. Toàn bộ tập hợp các tham số và địa chỉ trả về được lưu trữ cho một lần gọi thủ tục được gọi là **khung ngăn xếp (stack frame)**.

CHƯƠNG 13: TẬP LỆNH: CHẾ ĐỘ ĐỊA CHỈ VÀ ĐỊNH DẠNG

### **CHƯƠNG 13: TẬP LỆNH: CHẾ ĐỘ ĐỊA CHỈ VÀ ĐỊNH DẠNG**

**Mục tiêu của chương này là gì?**

Sau khi học xong chương này, bạn sẽ có thể:

* Mô tả các loại **chế độ địa chỉ (addressing modes)** phổ biến trong các tập lệnh.
* Trình bày tổng quan về các chế độ địa chỉ của x86 và ARM.
* Tóm tắt các vấn đề và sự đánh đổi liên quan đến việc thiết kế **định dạng lệnh (instruction format)**.
* Trình bày tổng quan về các định dạng lệnh của x86 và ARM.
* Hiểu sự khác biệt giữa hợp ngữ (assembly language) và ngôn ngữ máy (machine language).

#### **13.1. Chế độ địa chỉ (Addressing Modes)**

Trường địa chỉ trong định dạng lệnh thường tương đối nhỏ. Do đó, một loạt các kỹ thuật định địa chỉ, hay còn gọi là **chế độ địa chỉ**, đã được phát triển để mở rộng phạm vi tham chiếu đến các vị trí trong bộ nhớ chính hoặc bộ nhớ ảo. Các chế độ này là sự đánh đổi giữa dải địa chỉ, tính linh hoạt, số lần tham chiếu bộ nhớ và độ phức tạp của việc tính toán địa chỉ.

Các chế độ địa chỉ phổ biến nhất bao gồm:

* **Tức thời (Immediate):** Toán hạng nằm ngay trong lệnh.
  + **Ưu điểm:** Không cần tham chiếu bộ nhớ để lấy toán hạng, giúp tiết kiệm một chu kỳ bộ nhớ.
  + **Nhược điểm:** Kích thước của toán hạng bị giới hạn bởi kích thước của trường địa chỉ.
* **Trực tiếp (Direct):** Trường địa chỉ chứa địa chỉ hiệu dụng (EA - effective address) của toán hạng.
  + **Công thức:** EA = A.
  + **Ưu điểm:** Đơn giản, chỉ cần một lần tham chiếu bộ nhớ.
  + **Nhược điểm:** Không gian địa chỉ bị giới hạn.
* **Gián tiếp (Indirect):** Trường địa chỉ trong lệnh trỏ đến một địa chỉ trong bộ nhớ, và vị trí đó chứa địa chỉ đầy đủ của toán hạng.
  + **Công thức:** EA = (A).
  + **Ưu điểm:** Cung cấp không gian địa chỉ lớn (2^N với N là độ dài từ).
  + **Nhược điểm:** Yêu cầu hai lần tham chiếu bộ nhớ để lấy toán hạng.
* **Thanh ghi (Register):** Trường địa chỉ tham chiếu đến một thanh ghi thay vì một địa chỉ bộ nhớ chính.
  + **Công thức:** EA = R.
  + **Ưu điểm:** Cần trường địa chỉ nhỏ, không cần tham chiếu bộ nhớ tốn thời gian.
  + **Nhược điểm:** Không gian địa chỉ rất hạn chế (số lượng thanh ghi có hạn).
* **Thanh ghi gián tiếp (Register Indirect):** Tương tự như định địa chỉ gián tiếp, nhưng trường địa chỉ tham chiếu đến một thanh ghi chứa địa chỉ của toán hạng.
  + **Công thức:** EA = (R).
  + **Ưu điểm:** Không gian địa chỉ lớn, dùng ít hơn một lần tham chiếu bộ nhớ so với định địa chỉ gián tiếp.
* **Dịch chuyển (Displacement):** Kết hợp khả năng của định địa chỉ trực tiếp và thanh ghi gián tiếp.
  + **Công thức:** EA = A + (R).
  + Lệnh có hai trường địa chỉ: một giá trị (A) được sử dụng trực tiếp và một trường khác tham chiếu đến thanh ghi. Nội dung của thanh ghi được cộng với A để tạo ra địa chỉ hiệu dụng.
  + Ba cách sử dụng phổ biến của chế độ này là:
    1. **Định địa chỉ tương đối (Relative Addressing):** Thanh ghi được tham chiếu ngầm là bộ đếm chương trình (PC). Địa chỉ hiệu dụng là một sự dịch chuyển so với địa chỉ của lệnh hiện tại. Chế độ này khai thác **tính cục bộ (locality)** của tham chiếu bộ nhớ.
    2. **Định địa chỉ thanh ghi cơ sở (Base-Register Addressing):** Thanh ghi được tham chiếu chứa một địa chỉ bộ nhớ chính, và trường địa chỉ chứa một giá trị dịch chuyển từ địa chỉ đó. Nó rất tiện lợi để triển khai **phân đoạn bộ nhớ (segmentation)**.
    3. **Lập chỉ mục (Indexing):** Trường địa chỉ tham chiếu đến một địa chỉ bộ nhớ chính, và thanh ghi được tham chiếu chứa một giá trị dịch chuyển dương từ địa chỉ đó. Nó rất hiệu quả cho các hoạt động lặp đi lặp lại trên các mảng dữ liệu. Một số hệ thống còn hỗ trợ **tự động lập chỉ mục (autoindexing)**, trong đó thanh ghi chỉ mục được tự động tăng hoặc giảm sau mỗi lần tham chiếu.
* **Ngăn xếp (Stack):** Là một dạng định địa chỉ ngụ ý. Các lệnh máy không cần chứa tham chiếu bộ nhớ mà hoạt động ngầm định trên đỉnh của ngăn xếp. Một con trỏ ngăn xếp (stack pointer) được duy trì trong một thanh ghi.

#### **13.3. Định dạng Lệnh (Instruction Formats)**

Định dạng lệnh xác định cách bố trí các bit của một lệnh, bao gồm mã toán tử (opcode) và các toán hạng.

**A. Độ dài Lệnh (Instruction Length)**

Đây là vấn đề thiết kế cơ bản nhất. Có một sự đánh đổi giữa mong muốn có một bộ lệnh mạnh mẽ (nhiều opcode, nhiều toán hạng, nhiều chế độ địa chỉ) và nhu cầu tiết kiệm không gian.

* Lệnh dài hơn cho phép có nhiều opcode, nhiều toán hạng và chế độ địa chỉ hơn, giúp lập trình viên viết chương trình ngắn hơn.
* Tuy nhiên, lệnh dài hơn có thể lãng phí không gian.
* Độ dài lệnh nên bằng hoặc là bội số của độ dài truyền tải bộ nhớ (memory-transfer length) để tìm nạp lệnh hiệu quả.
* Lệnh ngắn hơn có thể được tìm nạp nhanh hơn, giúp giảm bớt tình trạng tắc nghẽn cổ chai của bộ nhớ.

**B. Phân bổ Bit (Allocation of Bits)**

Đối với một độ dài lệnh nhất định, có sự đánh đổi giữa số lượng opcode và sức mạnh của khả năng định địa chỉ. Các yếu tố cần cân nhắc bao gồm:

* **Số lượng chế độ địa chỉ:** Chế độ địa chỉ có thể được chỉ định ngầm định hoặc tường minh (cần thêm bit).
* **Số lượng toán hạng:** Lệnh có ít địa chỉ hơn sẽ nguyên thủy hơn, yêu cầu bộ xử lý ít phức tạp hơn và có độ dài lệnh ngắn hơn. Ngược lại, chương trình sẽ chứa nhiều lệnh hơn, dẫn đến thời gian thực thi dài hơn. Hầu hết các máy tính hiện đại sử dụng kết hợp các lệnh hai và ba địa chỉ.
* **Thanh ghi so với bộ nhớ:** Tham chiếu thanh ghi cần ít bit hơn và nhanh hơn tham chiếu bộ nhớ. Việc sử dụng nhiều thanh ghi giúp tăng tốc độ thực thi. Hầu hết các kiến trúc đương đại có ít nhất 32 thanh ghi.
* **Dải địa chỉ:** Với định địa chỉ dịch chuyển, dải địa chỉ được mở rộng bằng độ dài của thanh ghi địa chỉ. Việc cho phép các giá trị dịch chuyển lớn sẽ yêu cầu nhiều bit địa chỉ hơn trong lệnh.
* **Độ chi tiết địa chỉ (Address Granularity):** Định địa chỉ theo byte rất tiện lợi cho thao tác ký tự nhưng yêu cầu nhiều bit địa chỉ hơn so với định địa chỉ theo từ (word).

**C. Lệnh có độ dài thay đổi (Variable-Length Instructions)**

Thay vì một độ dài lệnh cố định, nhà thiết kế có thể cung cấp nhiều định dạng lệnh với độ dài khác nhau.

* **Ưu điểm:** Dễ dàng cung cấp một bộ lệnh lớn với các độ dài opcode khác nhau. Việc định địa chỉ có thể linh hoạt hơn. Các biến thể này có thể được cung cấp một cách hiệu quả và nhỏ gọn.
* **Nhược điểm:** Tăng độ phức tạp của bộ xử lý. Tuy nhiên, với sự phát triển của công nghệ, đây là một cái giá nhỏ phải trả. Các máy RISC và siêu vô hướng có thể khai thác các lệnh có độ dài cố định để cải thiện hiệu năng.

#### **13.5. Hợp ngữ (Assembly Language)**

Một bộ xử lý có thể hiểu và thực thi các lệnh máy, là các số nhị phân được lưu trữ trong máy tính. Việc lập trình trực tiếp bằng ngôn ngữ máy rất tẻ nhạt và dễ gây lỗi.

* **Chương trình tượng trưng (Symbolic program):** Một cải tiến là sử dụng tên ghi nhớ (mnemonic) cho mỗi lệnh. Điều này giúp chương trình dễ đọc hơn.
* **Hợp ngữ (Assembly Language):** Một cải tiến hơn nữa là sử dụng địa chỉ tượng trưng (symbolic addresses) thay vì địa chỉ số tuyệt đối.
  + Các chương trình viết bằng hợp ngữ được dịch sang ngôn ngữ máy bởi một **trình hợp dịch (assembler)**.
  + Trình hợp dịch không chỉ thực hiện việc dịch tượng trưng mà còn gán các địa chỉ bộ nhớ cho các địa chỉ tượng trưng.
  + Sự phát triển của hợp ngữ là một cột mốc quan trọng trong sự tiến hóa của công nghệ máy tính và là bước đầu tiên hướng tới các ngôn ngữ lập trình bậc cao hiện nay.

Hợp ngữ là một ngôn ngữ lập trình cấp thấp, thường có mối quan hệ một-một giữa lệnh hợp ngữ và lệnh máy. Nó phụ thuộc vào phần cứng và đòi hỏi lập trình viên phải hiểu kiến trúc của máy tính. Mặc dù ngày nay ít được sử dụng cho các ứng dụng, nó vẫn là một công cụ hữu ích cho các chương trình hệ thống như trình biên dịch và các thủ tục I/O.

CHƯƠNG 14: CẤU TRÚC VÀ CHỨC NĂNG CỦA BỘ XỬ LÝ

### **CHƯƠNG 14: CẤU TRÚC VÀ CHỨC NĂNG CỦA BỘ XỬ LÝ**

**Mục tiêu của chương này là gì?**

Sau khi học xong chương này, bạn sẽ có thể:

* Phân biệt giữa các loại thanh ghi **user-visible (người dùng thấy được)** và **control/status (điều khiển/trạng thái)**, đồng thời thảo luận về mục đích của từng loại.
* Tóm tắt về **chu kỳ lệnh (instruction cycle)**.
* Thảo luận về nguyên tắc đằng sau **kỹ thuật đường ống lệnh (instruction pipelining)** và cách nó hoạt động trên thực tế.
* So sánh và đối chiếu các dạng khác nhau của **rủi ro đường ống (pipeline hazards)**.

#### **14.1. Tổ chức Bộ xử lý (Processor Organization)**

Để hiểu được tổ chức của bộ xử lý, chúng ta cần xem xét các yêu cầu đặt ra cho nó, tức là những việc nó phải làm:

* **Tìm nạp lệnh (Fetch instruction):** Bộ xử lý đọc một lệnh từ bộ nhớ (thanh ghi, bộ đệm, hoặc bộ nhớ chính).
* **Diễn giải lệnh (Interpret instruction):** Lệnh được giải mã để xác định hành động cần thực hiện.
* **Tìm nạp dữ liệu (Fetch data):** Việc thực thi một lệnh có thể yêu cầu đọc dữ liệu từ bộ nhớ hoặc một module I/O.
* **Xử lý dữ liệu (Process data):** Việc thực thi một lệnh có thể yêu cầu thực hiện một phép toán số học hoặc logic trên dữ liệu.
* **Ghi dữ liệu (Write data):** Kết quả của một lần thực thi có thể yêu cầu ghi dữ liệu vào bộ nhớ hoặc một module I/O.

Để thực hiện những nhiệm vụ này, bộ xử lý cần một bộ nhớ trong nhỏ để lưu trữ dữ liệu tạm thời. Các thành phần chính của bộ xử lý bao gồm **ALU (Đơn vị Số học và Logic)**, **CU (Đơn vị Điều khiển)** và một tập hợp các vị trí lưu trữ được gọi là **thanh ghi (registers)**. Đơn vị điều khiển chịu trách nhiệm điều khiển sự di chuyển của dữ liệu và lệnh vào ra bộ xử lý, cũng như điều khiển hoạt động của ALU.

#### **14.2. Tổ chức Thanh ghi (Register Organization)**

Các thanh ghi trong bộ xử lý có hai vai trò chính:

1. **Thanh ghi người dùng thấy được (User-Visible Registers):** Cho phép người lập trình hợp ngữ tối ưu hóa việc sử dụng thanh ghi để giảm thiểu các tham chiếu đến bộ nhớ chính.
2. **Thanh ghi điều khiển và trạng thái (Control and Status Registers):** Được đơn vị điều khiển sử dụng để điều khiển hoạt động của bộ xử lý và được các chương trình hệ điều hành ở chế độ đặc quyền sử dụng để kiểm soát việc thực thi chương trình.

**A. Thanh ghi người dùng thấy được (User-Visible Registers)**

Các thanh ghi này có thể được tham chiếu bằng ngôn ngữ máy và được phân thành các loại sau:

* **Đa năng (General-purpose):** Có thể được lập trình viên gán cho nhiều chức năng khác nhau.
* **Dữ liệu (Data):** Chỉ được dùng để chứa dữ liệu và không thể dùng trong tính toán địa chỉ toán hạng.
* **Địa chỉ (Address):** Dùng cho các chế độ địa chỉ cụ thể như con trỏ đoạn (segment pointers), thanh ghi chỉ mục (index registers), và con trỏ ngăn xếp (stack pointer).
* **Mã điều kiện (Condition Codes) hay cờ (Flags):** Là các bit được phần cứng của bộ xử lý thiết lập như là kết quả của các phép toán. Chúng có thể được kiểm tra bởi các lệnh rẽ nhánh có điều kiện.

**B. Thanh ghi Điều khiển và Trạng thái (Control and Status Registers)**

Các thanh ghi này được dùng để điều khiển hoạt động của bộ xử lý và hầu hết không thể thấy được bởi người dùng. Bốn thanh ghi cần thiết cho việc thực thi lệnh là:

* **PC (Program Counter - Bộ đếm chương trình):** Chứa địa chỉ của lệnh tiếp theo sẽ được tìm nạp.
* **IR (Instruction Register - Thanh ghi lệnh):** Chứa lệnh được tìm nạp gần đây nhất.
* **MAR (Memory Address Register - Thanh ghi địa chỉ bộ nhớ):** Chứa địa chỉ của một vị trí trong bộ nhớ.
* **MBR (Memory Buffer Register - Thanh ghi đệm bộ nhớ):** Chứa một từ dữ liệu sẽ được ghi vào bộ nhớ hoặc từ được đọc gần đây nhất.

Nhiều bộ xử lý còn có một thanh ghi (hoặc tập hợp thanh ghi) được gọi là **Từ Trạng thái Chương trình (Program Status Word - PSW)**. PSW thường chứa mã điều kiện và các thông tin trạng thái khác như: dấu (Sign), không (Zero), nhớ (Carry), tràn (Overflow), cho phép/vô hiệu hóa ngắt (Interrupt Enable/Disable), và chế độ giám sát/người dùng (Supervisor/User).

#### **14.3. Chu kỳ Lệnh (Instruction Cycle)**

Một chu kỳ lệnh bao gồm các giai đoạn sau:

* **Tìm nạp (Fetch):** Đọc lệnh tiếp theo từ bộ nhớ vào bộ xử lý.
* **Thực thi (Execute):** Diễn giải mã toán tử (opcode) và thực hiện hoạt động được chỉ định.
* **Ngắt (Interrupt):** Nếu ngắt được cho phép và có một ngắt xảy ra, lưu lại trạng thái tiến trình hiện tại và phục vụ ngắt.

Một giai đoạn bổ sung, được gọi là **chu kỳ gián tiếp (indirect cycle)**, là cần thiết nếu lệnh sử dụng địa chỉ gián tiếp.

**Luồng dữ liệu (Data Flow):**

* **Chu kỳ Tìm nạp:** PC cung cấp địa chỉ lệnh, địa chỉ này được chuyển đến MAR. Lệnh được đọc từ bộ nhớ và đưa vào MBR, sau đó chuyển đến IR. Đồng thời, PC được tăng lên.
* **Chu kỳ Gián tiếp:** Các bit địa chỉ từ IR được chuyển đến MAR. Địa chỉ của toán hạng được tìm nạp từ bộ nhớ vào MBR.
* **Chu kỳ Ngắt:** PC hiện tại được lưu vào MBR để ghi vào bộ nhớ. Địa chỉ của chương trình xử lý ngắt được nạp vào PC.

#### **14.4. Kỹ thuật Đường ống Lệnh (Instruction Pipelining)**

Kỹ thuật đường ống lệnh tương tự như một dây chuyền lắp ráp, cho phép xử lý đồng thời nhiều lệnh ở các giai đoạn khác nhau.

**A. Chiến lược Đường ống (Pipelining Strategy)**

* **Đường ống hai giai đoạn:** Chia quá trình xử lý lệnh thành hai giai đoạn: **tìm nạp lệnh (fetch)** và **thực thi lệnh (execute)**. Kỹ thuật này được gọi là **tìm nạp trước lệnh (instruction prefetch)** hay **trùng lặp tìm nạp (fetch overlap)**. Tuy nhiên, hiệu quả bị giới hạn vì thời gian thực thi thường dài hơn thời gian tìm nạp và các lệnh rẽ nhánh có điều kiện làm cho địa chỉ lệnh tiếp theo không chắc chắn.
* **Đường ống nhiều giai đoạn:** Để tăng tốc hơn nữa, đường ống được chia thành nhiều giai đoạn hơn, ví dụ như sáu giai đoạn:
  1. **Tìm nạp lệnh (Fetch Instruction - FI)**
  2. **Giải mã lệnh (Decode Instruction - DI)**
  3. **Tính toán toán hạng (Calculate Operands - CO)**
  4. **Tìm nạp toán hạng (Fetch Operands - FO)**
  5. **Thực thi lệnh (Execute Instruction - EI)**
  6. **Ghi toán hạng (Write Operand - WO)**

Một đường ống sáu giai đoạn có thể giảm đáng kể thời gian thực thi tổng thể cho một chuỗi lệnh.

**B. Hiệu năng Đường ống (Pipeline Performance)**

Tốc độ tăng tốc (speedup) của một đường ống k giai đoạn so với một bộ xử lý không có đường ống có thể được tính bằng công thức: **Sₖ = nk / [k + (n - 1)]** Trong đó *n* là số lệnh và *k* là số giai đoạn. Khi *n* rất lớn, tốc độ tăng tốc tiến gần đến *k*. Tuy nhiên, trên thực tế, các yếu tố như chi phí, độ trễ giữa các giai đoạn và các lệnh rẽ nhánh sẽ hạn chế lợi ích của việc thêm nhiều giai đoạn hơn.

**C. Rủi ro Đường ống (Pipeline Hazards)**

Một rủi ro đường ống xảy ra khi đường ống phải dừng lại (stall), còn được gọi là **bong bóng đường ống (pipeline bubble)**. Có ba loại rủi ro:

1. **Rủi ro tài nguyên (Resource Hazard):** Xảy ra khi hai hoặc nhiều lệnh trong đường ống cần cùng một tài nguyên (ví dụ: ALU, cổng bộ nhớ).
2. **Rủi ro dữ liệu (Data Hazard):** Xảy ra khi có xung đột trong việc truy cập một vị trí toán hạng. Có ba loại:
   * **Đọc sau khi ghi (Read After Write - RAW) hay phụ thuộc thực sự:** Một lệnh đọc một toán hạng trước khi lệnh trước đó ghi xong giá trị vào

CHƯƠNG 15: MÁY TÍNH VỚI TẬP LỆNH RÚT GỌN (RISC)

### **CHƯƠNG 15: MÁY TÍNH VỚI TẬP LỆNH RÚT GỌN (RISC)**

**Mục tiêu của chương này là gì?**

Sau khi học xong chương này, bạn sẽ có thể:

* Trình bày tổng quan về các kết quả nghiên cứu về đặc điểm thực thi lệnh đã thúc đẩy sự phát triển của phương pháp RISC.
* Tóm tắt các đặc điểm chính của máy RISC.
* Hiểu các hàm ý về thiết kế và hiệu năng của việc sử dụng một tệp thanh ghi lớn.
* Hiểu việc sử dụng tối ưu hóa thanh ghi dựa trên trình biên dịch để cải thiện hiệu năng.
* Thảo luận về hàm ý của kiến trúc RISC đối với thiết kế và hiệu năng đường ống lệnh.
* Liệt kê và giải thích các phương pháp chính để tối ưu hóa đường ống lệnh trên máy RISC.

#### **15.1. Đặc Điểm Thực Thi Lệnh (Instruction Execution Characteristics)**

Sự phát triển của các ngôn ngữ lập trình bậc cao (HLLs) đã dẫn đến một "khoảng cách ngữ nghĩa" (semantic gap) giữa các phép toán trong HLL và các phép toán do kiến trúc máy tính cung cấp. Để thu hẹp khoảng cách này, các nhà thiết kế đã tạo ra các **Kiến trúc Tập lệnh Phức tạp (CISC - Complex Instruction Set Computer)** với các đặc điểm như tập lệnh lớn, nhiều chế độ địa chỉ, và các lệnh phần cứng phức tạp. Mục đích là để đơn giản hóa công việc của trình biên dịch và cải thiện hiệu năng thực thi.

Tuy nhiên, các nghiên cứu về cách các chương trình HLL thực sự được thực thi đã đưa ra một góc nhìn khác, thúc đẩy sự ra đời của **Kiến trúc Tập lệnh Rút gọn (RISC - Reduced Instruction Set Computer)**.

**A. Các phép toán (Operations)**

* Các nghiên cứu cho thấy các câu lệnh **gán (assignment)**, vốn là các thao tác di chuyển dữ liệu đơn giản, chiếm ưu thế trong các chương trình HLL.
* Các câu lệnh **điều kiện (IF, LOOP)** cũng rất phổ biến, cho thấy tầm quan trọng của cơ chế điều khiển chuỗi lệnh (so sánh và rẽ nhánh).
* Quan trọng hơn, khi phân tích thời gian thực thi, các lệnh **gọi/trở về thủ tục (procedure call/return)** là các phép toán tốn nhiều thời gian nhất.

**B. Các toán hạng (Operands)**

* Hầu hết các tham chiếu đều đến các **biến vô hướng đơn giản (simple scalar variables)**.
* Phần lớn các biến vô hướng này là **biến cục bộ (local variables)** của một thủ tục.
* Điều này cho thấy việc tối ưu hóa cơ chế lưu trữ và truy cập các biến vô hướng cục bộ là rất quan trọng.

**C. Các hàm ý (Implications)**

Các kết quả nghiên cứu này ngụ ý rằng chiến lược thiết kế hiệu quả nhất không phải là làm cho tập lệnh trở nên phức tạp hơn, mà là **tối ưu hóa hiệu năng của các tính năng tốn nhiều thời gian nhất** trong các chương trình HLL điển hình. Ba yếu tố chính đặc trưng cho kiến trúc RISC đã ra đời từ nhận định này:

1. **Sử dụng một số lượng lớn thanh ghi** hoặc sử dụng trình biên dịch để tối ưu hóa việc sử dụng thanh ghi nhằm mục đích tối ưu hóa việc tham chiếu toán hạng.
2. **Chú trọng đến việc thiết kế đường ống lệnh (instruction pipeline)** hiệu quả, vì các lệnh rẽ nhánh và gọi thủ tục chiếm tỷ lệ cao.
3. Sử dụng một **tập lệnh bao gồm các lệnh nguyên thủy hiệu năng cao**.

#### **15.2. Việc sử dụng một Tệp Thanh ghi Lớn (The Use of a Large Register File)**

Do hầu hết các tham chiếu toán hạng là đến các biến vô hướng cục bộ, một phương pháp hiệu quả là lưu trữ chúng trong các thanh ghi. Thanh ghi là bộ nhớ nhanh nhất có sẵn, nhanh hơn cả bộ nhớ chính và bộ đệm.

* **Vấn đề:** Định nghĩa về "cục bộ" thay đổi theo mỗi lần gọi và trở về thủ tục, là những hoạt động xảy ra thường xuyên. Mỗi lần gọi, các biến cục bộ phải được lưu từ thanh ghi vào bộ nhớ để các thanh ghi có thể được tái sử dụng bởi thủ tục được gọi.
* **Giải pháp phần cứng:** Sử dụng nhiều bộ thanh ghi nhỏ, được gọi là **cửa sổ thanh ghi (register windows)**. Mỗi cửa sổ được gán cho một thủ tục khác nhau.
  + Một lệnh gọi thủ tục sẽ tự động chuyển bộ xử lý sang sử dụng một cửa sổ thanh ghi khác, thay vì phải lưu các thanh ghi vào bộ nhớ.
  + Các cửa sổ cho các thủ tục liền kề được **xếp chồng (overlapped)** để cho phép truyền tham số mà không cần di chuyển dữ liệu thực tế.
  + Tệp thanh ghi được tổ chức như một **bộ đệm vòng (circular buffer)** của các cửa sổ xếp chồng. Nếu số lần gọi thủ tục lồng nhau vượt quá số lượng cửa sổ, một ngắt sẽ xảy ra để lưu cửa sổ cũ nhất vào bộ nhớ.
* **Biến toàn cục (Global Variables):** Các biến được truy cập bởi nhiều hơn một thủ tục có thể được gán vào các vị trí bộ nhớ hoặc một tập hợp các **thanh ghi toàn cục (global registers)** riêng biệt.

#### **15.3. Tối ưu hóa Thanh ghi dựa trên Trình biên dịch (Compiler-Based Register Optimization)**

Một cách tiếp cận khác, dựa trên phần mềm, là dựa vào trình biên dịch để tối ưu hóa việc sử dụng một số lượng nhỏ các thanh ghi (ví dụ: 16–32).

* **Nguyên tắc:** Trình biên dịch gán các biến chương trình cho các **thanh ghi tượng trưng (symbolic registers)** không giới hạn, sau đó ánh xạ các thanh ghi tượng trưng này vào một số lượng thanh ghi vật lý cố định.
* **Kỹ thuật tô màu đồ thị (Graph Coloring):**
  + Trình biên dịch xây dựng một **đồ thị nhiễu thanh ghi (register interference graph)**, trong đó các nút là các thanh ghi tượng trưng.
  + Một cạnh nối hai nút nếu các thanh ghi tượng trưng tương ứng "sống" (live) trong cùng một đoạn chương trình.
  + Sau đó, đồ thị được "tô màu" với *n* màu, trong đó *n* là số lượng thanh ghi vật lý. Các nút có cùng màu có thể chia sẻ cùng một thanh ghi vật lý.
  + Các nút không thể tô màu phải được đặt trong bộ nhớ và các lệnh tải/lưu (load/store) được sử dụng khi cần.

Các nghiên cứu cho thấy với các kỹ thuật tối ưu hóa thanh ghi, việc sử dụng nhiều hơn 32 hoặc 64 thanh ghi chỉ mang lại lợi ích hiệu năng không đáng kể.

#### **15.4. Kiến trúc Tập lệnh Rút gọn (Reduced Instruction Set Architecture)**

Các đặc điểm chung của kiến trúc RISC bao gồm:

* **Một lệnh mỗi chu kỳ (One instruction per cycle):** Các lệnh máy của RISC nên đơn giản và thực thi nhanh như các vi lệnh trên máy CISC. Điều này cho phép thực thi lệnh bằng phần cứng nối cứng (hardwired) thay vì vi mã (microcode), giúp tăng tốc độ.
* **Các phép toán thanh ghi-thanh ghi (Register-to-register operations):** Hầu hết các phép toán được thực hiện trên các toán hạng trong thanh ghi. Chỉ có các lệnh LOAD và STORE đơn giản mới truy cập bộ nhớ. Điều này đơn giản hóa tập lệnh và bộ điều khiển.
* **Các chế độ địa chỉ đơn giản (Simple addressing modes):** Hầu hết các lệnh RISC sử dụng định địa chỉ thanh ghi đơn giản. Các chế độ phức tạp hơn có thể được tổng hợp trong phần mềm từ các chế độ đơn giản.
* **Các định dạng lệnh đơn giản (Simple instruction formats):** Thường chỉ sử dụng một hoặc một vài định dạng lệnh có độ dài cố định và các trường được căn chỉnh cố định, giúp đơn giản hóa việc giải mã lệnh và tối ưu hóa việc tìm nạp lệnh.

#### **15.5. Đường ống lệnh RISC (RISC Pipelining)**

Kiến trúc RISC đặc biệt phù hợp với kỹ thuật đường ống lệnh. Hầu hết các lệnh RISC là thanh ghi-thanh ghi, và chu kỳ lệnh có thể được chia thành các giai đoạn như sau:

* **I:** Tìm nạp lệnh (Instruction fetch).
* **E:** Thực thi (Execute), thực hiện một phép toán ALU. Đối với các lệnh tải và lưu (load/store), cần thêm một giai đoạn:
* **D:** Bộ nhớ (Memory), thực hiện thao tác thanh ghi-bộ nhớ hoặc bộ nhớ-thanh ghi.

**Tối ưu hóa đường ống lệnh (Optimization of Pipelining)**

* **Rẽ nhánh trễ (Delayed Branch):** Lệnh rẽ nhánh không có hiệu lực cho đến sau khi thực thi lệnh ngay sau nó. Vị trí lệnh ngay sau lệnh rẽ nhánh được gọi là **khe trễ (delay slot)**.
  + Trình biên dịch cố gắng đặt một lệnh hữu ích vào khe trễ. Nếu không thể, một lệnh **NOOP (không hoạt động)** sẽ được chèn vào.
  + Kỹ thuật này giúp giữ cho đường ống lệnh luôn đầy trong khi bộ xử lý tìm nạp một luồng lệnh mới, giúp cải thiện hiệu năng.
* **Tải trễ (Delayed Load):** Trên các lệnh LOAD, thanh ghi đích sẽ bị khóa. Bộ xử lý tiếp tục thực thi các lệnh tiếp theo cho đến khi gặp một lệnh yêu cầu thanh ghi đó, lúc đó nó sẽ tạm dừng cho đến khi việc tải hoàn tất. Trình biên dịch có thể sắp xếp lại các lệnh để thực hiện công việc hữu ích trong khi việc tải đang diễn ra trong đường ống.
* **Mở rộng vòng lặp (Loop Unrolling):** Trình biên dịch sao chép thân vòng lặp một số lần để giảm chi phí vòng lặp, tăng song song hóa lệnh, và cải thiện tính cục bộ của thanh ghi và bộ đệm.

CHƯƠNG 16: SONG SONGLỆNH VÀ BỘ XỬ LÝ SIÊU VÔ HƯỚNG

### **CHƯƠNG 16: SONG SONG MỨC LỆNH VÀ BỘ XỬ LÝ SIÊU VÔ HƯỚNG**

**Mục tiêu của chương này là gì?**

Sau khi học xong chương này, bạn sẽ có thể:

* Giải thích sự khác biệt giữa các phương pháp **siêu vô hướng (superscalar)** và **siêu đường ống (superpipelined)**.
* Định nghĩa **song song mức lệnh (instruction-level parallelism)**.
* Thảo luận về các loại phụ thuộc và xung đột tài nguyên như là những giới hạn đối với song song mức lệnh.
* Trình bày tổng quan về các vấn đề thiết kế liên quan đến song song mức lệnh.
* So sánh và đối chiếu các kỹ thuật cải thiện hiệu năng đường ống lệnh trong máy RISC và máy siêu vô hướng.

#### **16.1. Tổng Quan (Overview)**

**Bộ xử lý siêu vô hướng (superscalar processor)** là một bộ xử lý mà trong đó các lệnh thông thường—như số học nguyên và dấu phẩy động, tải, lưu, và rẽ nhánh có điều kiện—có thể được khởi tạo đồng thời và thực thi một cách độc lập.

Bản chất của phương pháp siêu vô hướng là khả năng **thực thi các lệnh một cách độc lập và đồng thời trong các đường ống khác nhau**. Bộ xử lý có nhiều đơn vị chức năng, mỗi đơn vị được triển khai dưới dạng một đường ống. Việc sử dụng nhiều đơn vị chức năng cho phép bộ xử lý thực thi các luồng lệnh song song, mỗi luồng cho một đường ống.

**A. So sánh Siêu vô hướng và Siêu đường ống (Superscalar versus Superpipelined)**

Một phương pháp khác để đạt được hiệu năng cao hơn được gọi là **siêu đường ống (superpipelining)**. Phương pháp này khai thác thực tế là nhiều giai đoạn của đường ống đòi hỏi ít hơn một nửa chu kỳ xung nhịp. Do đó, việc nhân đôi tần số xung nhịp nội bộ cho phép thực hiện hai tác vụ trong một chu kỳ xung nhịp bên ngoài.

* Một **đường ống cơ sở (base pipeline)** phát hành một lệnh mỗi chu kỳ xung nhịp và có thể thực hiện một giai đoạn đường ống mỗi chu kỳ.
* Một **bộ xử lý siêu đường ống (superpipelined processor)** có khả năng thực hiện hai giai đoạn đường ống mỗi chu kỳ xung nhịp, do đó có thể nhân đôi thông lượng.
* Một **bộ xử lý siêu vô hướng (superscalar processor)** có khả năng thực thi hai thực thể của mỗi giai đoạn song song.

Cả hai cách tiếp cận siêu đường ống và siêu vô hướng đều có cùng số lượng lệnh được thực thi cùng lúc trong trạng thái ổn định. Bộ xử lý siêu đường ống sẽ bị tụt hậu so với bộ xử lý siêu vô hướng ở đầu chương trình và tại mỗi mục tiêu rẽ nhánh.

**B. Các Hạn Chế (Constraints)**

Phương pháp siêu vô hướng phụ thuộc vào khả năng thực thi nhiều lệnh song song. Mức độ mà các lệnh của một chương trình có thể được thực thi song song bằng cách chồng chéo được gọi là **song song mức lệnh (instruction-level parallelism)**. Các hạn chế cơ bản đối với khả năng song song hóa bao gồm:

1. **Phụ thuộc dữ liệu thực sự (True Data Dependency):** Xảy ra khi một lệnh cần dữ liệu được tạo ra bởi một lệnh trước đó. Lệnh thứ hai không thể thực thi cho đến khi lệnh đầu tiên hoàn tất. Đây còn được gọi là phụ thuộc luồng (flow dependency) hoặc đọc sau khi ghi (Read After Write - RAW).
2. **Phụ thuộc thủ tục (Procedural Dependency):** Sự hiện diện của các lệnh rẽ nhánh trong một chuỗi lệnh làm phức tạp hoạt động của đường ống. Các lệnh theo sau một lệnh rẽ nhánh (dù rẽ nhánh hay không) có phụ thuộc thủ tục vào lệnh rẽ nhánh và không thể được thực thi cho đến khi lệnh rẽ nhánh được thực thi.
3. **Xung đột tài nguyên (Resource Conflict):** Là sự cạnh tranh của hai hoặc nhiều lệnh cho cùng một tài nguyên tại cùng một thời điểm, chẳng hạn như bộ nhớ, bộ đệm, bus, cổng tệp thanh ghi và các đơn vị chức năng. Điều này buộc các lệnh phải được thực thi tuần tự thay vì song song trong một phần của đường ống.
4. **Phụ thuộc đầu ra (Output Dependency):** Xảy ra khi hai lệnh cùng ghi vào cùng một vị trí đầu ra. Đây còn được gọi là ghi sau khi ghi (Write After Write - WAW).
5. **Phản phụ thuộc (Antidependency):** Xảy ra khi một lệnh đọc một vị trí và một lệnh kế tiếp ghi vào vị trí đó. Đây còn được gọi là ghi sau khi đọc (Write After Read - WAR).

#### **16.2. Các Vấn Đề Thiết Kế (Design Issues)**

**A. Song song mức lệnh và Song song máy (Instruction-Level Parallelism and Machine Parallelism)**

* **Song song mức lệnh** tồn tại khi các lệnh trong một chuỗi là độc lập và do đó có thể được thực thi song song bằng cách chồng chéo. Mức độ này được quyết định bởi tần suất của các phụ thuộc dữ liệu thực sự và phụ thuộc thủ tục trong mã.
* **Song song máy** là thước đo khả năng của bộ xử lý trong việc tận dụng song song mức lệnh. Nó được xác định bởi số lượng lệnh có thể được tìm nạp và thực thi cùng một lúc (số lượng đường ống song song) và bởi tốc độ và sự tinh vi của các cơ chế mà bộ xử lý sử dụng để tìm các lệnh độc lập.

Cả hai yếu tố này đều quan trọng. Một chương trình có thể không có đủ song song mức lệnh để tận dụng tối đa song song máy. Ngược lại, song song máy hạn chế sẽ giới hạn hiệu năng bất kể bản chất của chương trình là gì.

**B. Chính sách phát hành lệnh (Instruction Issue Policy)**

**Phát hành lệnh (Instruction issue)** là quá trình khởi tạo việc thực thi lệnh trong các đơn vị chức năng của bộ xử lý, khi lệnh di chuyển từ giai đoạn giải mã đến giai đoạn thực thi đầu tiên của đường ống. Các chính sách phát hành lệnh siêu vô hướng có thể được phân loại thành:

* **Phát hành theo thứ tự, hoàn thành theo thứ tự (In-order issue with in-order completion):** Chính sách đơn giản nhất, trong đó các lệnh được phát hành theo thứ tự tuần tự nghiêm ngặt và việc ghi kết quả cũng theo thứ tự đó. Việc phát hành lệnh tạm thời bị dừng lại khi có xung đột tài nguyên hoặc phụ thuộc dữ liệu.
* **Phát hành theo thứ tự, hoàn thành không theo thứ tự (In-order issue with out-of-order completion):** Cho phép các lệnh hoàn thành không theo thứ tự để cải thiện hiệu năng của các lệnh cần nhiều chu kỳ. Chính sách này gặp phải các phụ thuộc đầu ra (WAW).
* **Phát hành không theo thứ tự, hoàn thành không theo thứ tự (Out-of-order issue with out-of-order completion):** Để cho phép phát hành không theo thứ tự, giai đoạn giải mã và thực thi của đường ống được tách rời bằng một bộ đệm gọi là **cửa sổ lệnh (instruction window)**. Sau khi được giải mã, các lệnh được đặt vào cửa sổ này và có thể được phát hành đến một đơn vị thực thi có sẵn miễn là không vi phạm các xung đột hoặc phụ thuộc. Điều này mang lại cho bộ xử lý khả năng "nhìn trước" để tìm các lệnh độc lập. Một kỹ thuật phổ biến để hỗ trợ hoàn thành không theo thứ tự là **bộ đệm sắp xếp lại (reorder buffer)**. Chính sách này gặp phải cả phản phụ thuộc (WAR) và phụ thuộc đầu ra (WAW).

**C. Đổi tên thanh ghi (Register Renaming)**

Phụ thuộc WAR và WAW là các xung đột lưu trữ phát sinh do nhiều lệnh cạnh tranh để sử dụng cùng một vị trí thanh ghi. **Đổi tên thanh ghi** là một kỹ thuật trong đó các thanh ghi được cấp phát động bởi phần cứng của bộ xử lý.

* Khi một lệnh thực thi tạo ra một giá trị thanh ghi mới, một thanh ghi vật lý mới sẽ được cấp phát cho giá trị đó.
* Các lệnh sau đó truy cập giá trị đó phải được đổi tên để tham chiếu đến thanh ghi vật lý mới được cấp phát gần nhất (theo thứ tự chương trình).
* Kỹ thuật này loại bỏ các phụ thuộc WAR và WAW, cho phép nhiều lệnh được thực thi song song hơn.

**D. Dự đoán rẽ nhánh (Branch Prediction)**

Các máy siêu vô hướng đã quay trở lại các kỹ thuật dự đoán rẽ nhánh trước thời RISC. Trong khi các máy RISC ban đầu sử dụng kỹ thuật **rẽ nhánh trễ (delayed branch)**, các bộ xử lý siêu vô hướng tinh vi hơn, như PowerPC 620 và Pentium 4, sử dụng **dự đoán rẽ nhánh động (dynamic branch prediction)** dựa trên phân tích lịch sử rẽ nhánh.

**E. Thực thi và Triển khai Siêu vô hướng (Superscalar Execution and Implementation)**

* **Thực thi:** Dòng lệnh tĩnh của chương trình được chuyển thành một dòng lệnh động thông qua việc tìm nạp lệnh và dự đoán rẽ nhánh. Bộ xử lý sau đó gửi các lệnh vào một cửa sổ thực thi, nơi chúng được cấu trúc lại theo các phụ thuộc dữ liệu thực sự. Các lệnh được thực thi theo thứ tự được quyết định bởi các phụ thuộc dữ liệu thực sự và sự sẵn có của tài nguyên phần cứng.
* **Cam kết (Committing):** Các kết quả được đưa trở lại theo thứ tự tuần tự để ghi vào bộ nhớ hoặc thanh ghi. Bước này, còn gọi là **về hưu (retiring)**, là cần thiết vì các lệnh có thể hoàn thành không theo thứ tự và một số lệnh có thể bị hủy bỏ do dự đoán rẽ nhánh sai.
* **Triển khai:** Các yếu tố phần cứng chính bao gồm:
  + Các chiến lược tìm nạp lệnh có thể tìm nạp nhiều lệnh cùng lúc, thường bằng cách tìm nạp vượt ra ngoài các lệnh rẽ nhánh có điều kiện.
  + Logic để xác định các phụ thuộc thực sự liên quan đến các giá trị thanh ghi.
  + Các cơ chế để phát hành nhiều lệnh song song.
  + Các tài nguyên để thực thi song song nhiều lệnh, bao gồm nhiều đơn vị chức năng có đường ống và các hệ thống phân cấp bộ nhớ có khả năng phục vụ nhiều tham chiếu bộ nhớ cùng lúc.
  + Các cơ chế để cam kết trạng thái của tiến trình theo đúng thứ tự.

CHƯƠG 17: XỬ LÝ SONG SONG (PARALLEL PROCESSING)

### **CHƯƠG 17: XỬ LÝ SONG SONG (PARALLEL PROCESSING)**

**Mục tiêu của chương này là gì?**

Sau khi học xong chương này, bạn sẽ có thể:

* Tóm tắt các loại tổ chức bộ xử lý song song.
* Trình bày tổng quan về các đặc điểm thiết kế của **bộ đa xử lý đối xứng (symmetric multiprocessors - SMP)**.
* Hiểu vấn đề **nhất quán bộ đệm (cache coherence)** trong một hệ thống đa bộ xử lý.
* Giải thích các đặc điểm chính của giao thức **MESI (Modified/Exclusive/Shared/Invalid)**.
* Giải thích sự khác biệt giữa **đa luồng ẩn (implicit multithreading)** và **đa luồng tường minh (explicit multithreading)**.
* Tóm tắt các vấn đề thiết kế chính cho các **cụm máy tính (clusters)**.

#### **17.1. Các Tổ chức Đa Bộ xử lý (Multiple Processor Organizations)**

Theo truyền thống, máy tính được xem là một cỗ máy tuần tự. Tuy nhiên, để tăng cường hiệu năng, các nhà thiết kế đã tìm kiếm cơ hội cho xử lý song song.

**A. Phân loại các Hệ thống Bộ xử lý Song song**

Một hệ thống phân loại do Flynn đề xuất vẫn là cách phổ biến nhất để phân loại các hệ thống có khả năng xử lý song song:

* **SISD (Single Instruction, Single Data stream - Luồng lệnh đơn, luồng dữ liệu đơn):** Một bộ xử lý đơn thực thi một luồng lệnh duy nhất để hoạt động trên dữ liệu được lưu trữ trong một bộ nhớ duy nhất. Các bộ xử lý đơn (uniprocessors) thuộc loại này.
* **SIMD (Single Instruction, Multiple Data stream - Luồng lệnh đơn, luồng dữ liệu đa):** Một lệnh máy duy nhất điều khiển việc thực thi đồng thời của một số phần tử xử lý trên cơ sở đồng bộ (lockstep basis). Mỗi phần tử xử lý có bộ nhớ dữ liệu riêng, do đó các lệnh được thực thi trên các tập dữ liệu khác nhau bởi các bộ xử lý khác nhau. Các bộ xử lý vector và mảng thuộc loại này.
* **MISD (Multiple Instruction, Single Data stream - Luồng lệnh đa, luồng dữ liệu đơn):** Một chuỗi dữ liệu được truyền đến một tập hợp các bộ xử lý, mỗi bộ xử lý thực thi một chuỗi lệnh khác nhau. Cấu trúc này không được triển khai thương mại.
* **MIMD (Multiple Instruction, Multiple Data stream - Luồng lệnh đa, luồng dữ liệu đa):** Một tập hợp các bộ xử lý đồng thời thực thi các chuỗi lệnh khác nhau trên các tập dữ liệu khác nhau. **SMP, clusters, và hệ thống NUMA** thuộc loại này.

**B. Các Tổ chức Song song**

Các hệ thống MIMD có thể được chia nhỏ hơn dựa trên cách các bộ xử lý giao tiếp với nhau:

1. **Bộ nhớ chia sẻ (Shared Memory) hay kết nối chặt (Tightly Coupled):**
   * Các bộ xử lý chia sẻ một bộ nhớ chung và giao tiếp với nhau thông qua bộ nhớ đó.
   * **Bộ đa xử lý đối xứng (Symmetric Multiprocessor - SMP):** Dạng phổ biến nhất của hệ thống này, trong đó thời gian truy cập bộ nhớ đến bất kỳ vùng nào của bộ nhớ là gần như bằng nhau đối với mỗi bộ xử lý.
   * **Truy cập bộ nhớ không đồng nhất (Nonuniform Memory Access - NUMA):** Thời gian truy cập bộ nhớ đến các vùng khác nhau của bộ nhớ có thể khác nhau đối với một bộ xử lý.
2. **Bộ nhớ phân tán (Distributed Memory) hay kết nối lỏng (Loosely Coupled):**
   * Bao gồm một tập hợp các máy tính độc lập (uniprocessors hoặc SMPs) được kết nối với nhau để tạo thành một **cụm máy tính (cluster)**. Giao tiếp giữa các máy tính thường thông qua các đường dẫn cố định hoặc một cơ sở mạng.

#### **17.2. Bộ đa xử lý đối xứng (Symmetric Multiprocessors - SMP)**

SMP là một kiến trúc máy tính trong đó hai hoặc nhiều bộ xử lý tương tự có cùng khả năng chia sẻ chung một bộ nhớ chính và các thiết bị I/O, và được kết nối với nhau bằng một bus hoặc một cơ chế kết nối nội bộ khác.

**A. Đặc điểm và Ưu điểm của SMP**

* **Đặc điểm chính:**
  + Có hai hoặc nhiều bộ xử lý tương tự.
  + Chia sẻ cùng một bộ nhớ chính và các thiết bị I/O.
  + Tất cả các bộ xử lý chia sẻ quyền truy cập vào các thiết bị I/O.
  + Tất cả các bộ xử lý có thể thực hiện các chức năng như nhau.
  + Hệ thống được điều khiển bởi một hệ điều hành tích hợp.
* **Ưu điểm tiềm năng:**
  + **Hiệu năng (Performance):** Nếu công việc có thể được tổ chức song song, một hệ thống đa bộ xử lý sẽ mang lại hiệu năng cao hơn so với một bộ xử lý đơn cùng loại.
  + **Tính sẵn sàng (Availability):** Sự cố của một bộ xử lý không làm dừng toàn bộ hệ thống; nó có thể tiếp tục hoạt động với hiệu năng giảm đi.
  + **Tăng trưởng theo từng bước (Incremental Growth):** Người dùng có thể nâng cao hiệu năng bằng cách thêm một bộ xử lý mới.
  + **Khả năng mở rộng (Scaling):** Các nhà cung cấp có thể cung cấp một loạt sản phẩm với các đặc điểm về giá cả và hiệu năng khác nhau dựa trên số lượng bộ xử lý.

**B. Tổ chức SMP**

* **Tổ chức chung:** Mỗi bộ xử lý bao gồm một đơn vị điều khiển, ALU, thanh ghi và một hoặc nhiều cấp bộ đệm. Các bộ xử lý giao tiếp với nhau thông qua bộ nhớ chung.
* **Tổ chức dựa trên Bus:** Cấu trúc đơn giản nhất là **bus chia sẻ thời gian (time-shared bus)**.
  + **Ưu điểm:** Đơn giản, linh hoạt và đáng tin cậy.
  + **Nhược điểm:** Hiệu năng bị giới hạn bởi chu kỳ của bus vì tất cả các tham chiếu bộ nhớ đều phải đi qua bus chung. Việc trang bị cho mỗi bộ xử lý một bộ đệm cục bộ (local cache) là cần thiết để giảm số lần truy cập bus.

**C. Các Vấn đề Thiết kế Hệ điều hành cho SMP**

Hệ điều hành SMP phải cung cấp tất cả các chức năng của một hệ thống đa chương và thêm các tính năng để hỗ trợ đa bộ xử lý. Các vấn đề thiết kế chính bao gồm:

* **Các tiến trình đồng thời:** Các thường trình của hệ điều hành cần phải là **reentrant** (tái nhập) để cho phép nhiều bộ xử lý thực thi cùng một mã hệ điều hành một cách đồng thời.
* **Lập lịch (Scheduling):** Bất kỳ bộ xử lý nào cũng có thể thực hiện lập lịch, do đó cần phải tránh xung đột.
* **Đồng bộ hóa (Synchronization):** Cần có cơ chế đồng bộ hóa hiệu quả để thực thi loại trừ tương hỗ và sắp xếp thứ tự sự kiện khi nhiều tiến trình truy cập vào các không gian địa chỉ hoặc tài nguyên I/O được chia sẻ.
* **Quản lý bộ nhớ (Memory Management):** Ngoài các vấn đề của máy tính đơn bộ xử lý, hệ điều hành cần khai thác song song phần cứng có sẵn để đạt hiệu năng tốt nhất.
* **Độ tin cậy và Khả năng chịu lỗi (Reliability and Fault Tolerance):** Hệ điều hành nên cung cấp sự suy giảm hiệu năng một cách duyên dáng (graceful degradation) khi có lỗi bộ xử lý.

#### **17.3. Nhất quán Bộ đệm và Giao thức MESI**

Khi mỗi bộ xử lý có bộ đệm riêng, vấn đề **nhất quán bộ đệm (cache coherence problem)** phát sinh: nhiều bản sao của cùng một dữ liệu có thể tồn tại trong các bộ đệm khác nhau, và nếu các bộ xử lý được phép cập nhật bản sao của mình một cách tự do, có thể dẫn đến một cái nhìn không nhất quán về bộ nhớ.

**A. Các Giải pháp cho Vấn đề Nhất quán Bộ đệm**

* **Giải pháp Phần mềm (Software Solutions):** Dựa vào trình biên dịch và hệ điều hành để xử lý vấn đề, tránh cần thêm phần cứng. Trình biên dịch phân tích mã để xác định các mục dữ liệu nào có thể không an toàn để lưu vào bộ đệm và đánh dấu chúng.
* **Giải pháp Phần cứng (Hardware Solutions) / Giao thức Nhất quán Bộ đệm:** Cung cấp sự nhận dạng động tại thời gian chạy về các điều kiện không nhất quán tiềm ẩn. Các giải pháp phần cứng có thể được chia thành hai loại:
  1. **Giao thức Thư mục (Directory Protocols):** Thu thập và duy trì thông tin về vị trí của các bản sao của các dòng dữ liệu, thường có một bộ điều khiển tập trung.
  2. **Giao thức Nghe trộm (Snoopy Protocols):** Phân tán trách nhiệm duy trì tính nhất quán cho tất cả các bộ điều khiển bộ đệm. Mỗi bộ điều khiển bộ đệm "nghe trộm" (snoop) trên bus để quan sát các thông báo quảng bá (broadcast notifications) và phản ứng tương ứng. Có hai cách tiếp cận chính:
     + **Vô hiệu hóa khi ghi (Write-Invalidate):** Khi một bộ xử lý muốn ghi vào một dòng được chia sẻ, nó sẽ phát một thông báo làm vô hiệu hóa dòng đó trong tất cả các bộ đệm khác.
     + **Cập nhật khi ghi (Write-Update):** Khi một bộ xử lý cập nhật một dòng được chia sẻ, từ được cập nhật sẽ được phân phối đến tất cả các bộ đệm khác chứa dòng đó.

**B. Giao thức MESI (Modified/Exclusive/Shared/Invalid)**

MESI là một giao thức vô hiệu hóa khi ghi (write-invalidate) được sử dụng rộng rãi, trong đó mỗi dòng trong bộ đệm có thể ở một trong bốn trạng thái sau:

* **Modified (Đã sửa đổi):** Dòng trong bộ đệm đã được sửa đổi (khác với bộ nhớ chính) và chỉ có trong bộ đệm này.
* **Exclusive (Độc quyền):** Dòng trong bộ đệm giống với bộ nhớ chính và không có trong bất kỳ bộ đệm nào khác.
* **Shared (Được chia sẻ):** Dòng trong bộ đệm giống với bộ nhớ chính và có thể có trong một bộ đệm khác.
* **Invalid (Không hợp lệ):** Dòng trong bộ đệm không chứa dữ liệu hợp lệ.

**Hoạt động của Giao thức MESI:**

* **Read Miss (Đọc trượt):** Khi một bộ xử lý đọc một dòng không có trong bộ đệm của nó, nó sẽ phát một tín hiệu đọc trên bus.
  + Nếu một bộ đệm khác có một bản sao đã sửa đổi (Modified), nó sẽ chặn việc đọc từ bộ nhớ, cung cấp dòng cho bộ đệm yêu cầu, và thay đổi trạng thái của nó thành Shared.
  + Nếu một hoặc nhiều bộ đệm có bản sao sạch (Shared hoặc Exclusive), chúng sẽ báo hiệu rằng chúng chia sẻ dòng đó, và bộ đệm yêu cầu sẽ đọc dòng từ bộ nhớ và đặt nó ở trạng thái Shared.
  + Nếu không có bộ đệm nào khác có bản sao, bộ đệm yêu cầu sẽ đọc dòng từ bộ nhớ và đặt nó ở trạng thái Exclusive.
* **Write Miss (Ghi trượt):** Bộ xử lý phát một tín hiệu đọc với ý định sửa đổi (read-with-intent-to-modify - RWITM). Các bộ đệm khác chứa dòng đó sẽ làm vô hiệu hóa bản sao của chúng.
* **Write Hit (Ghi trúng):**
  + Nếu dòng ở trạng thái Modified hoặc Exclusive, bộ xử lý chỉ cần cập nhật cục bộ.
  + Nếu dòng ở trạng thái Shared, bộ xử lý sẽ phát một tín hiệu vô hiệu hóa trên bus trước khi cập nhật dòng của mình và chuyển nó sang trạng thái Modified.

#### **17.4. Đa luồng và Chip Đa bộ xử lý (Multithreading and Chip Multiprocessors)**

**Đa luồng (multithreading)** cho phép một bộ xử lý thực thi các luồng lệnh khác nhau một cách đồng thời, làm tăng **tính song song mức lệnh (instruction-level parallelism)**.

**A. Đa luồng Ẩn và Tường minh**

* **Tiến trình (Process):** Một thực thể sở hữu tài nguyên (không gian địa chỉ ảo) và là một đơn vị lập lịch/thực thi.
* **Luồng (Thread):** Một đơn vị công việc có thể điều phối trong một tiến trình. Các luồng trong cùng một tiến trình chia sẻ tài nguyên.
* **Đa luồng tường minh (Explicit Multithreading):** Thực thi đồng thời các lệnh từ các luồng tường minh khác nhau được định nghĩa trong phần mềm.
* **Đa luồng ẩn (Implicit Multithreading):** Thực thi đồng thời nhiều luồng được trích xuất từ một chương trình tuần tự duy nhất.

**B. Các Cách tiếp cận Đa luồng Tường minh**

* **Đa luồng xen kẽ (Interleaved Multithreading / Fine-grained):** Bộ xử lý chuyển đổi giữa các luồng ở mỗi chu kỳ xung nhịp.
* **Đa luồng bị chặn (Blocked Multithreading / Coarse-grained):** Các lệnh của một luồng được thực thi liên tiếp cho đến khi một sự kiện gây trễ (như cache miss) xảy ra, lúc đó bộ xử lý chuyển sang một luồng khác.
* **Đa luồng đồng thời (Simultaneous Multithreading - SMT):** Các lệnh được phát hành đồng thời từ nhiều luồng đến các đơn vị thực thi của một bộ xử lý siêu vô hướng.
* **Đa xử lý trên chip (Chip Multiprocessing / Multicore):** Nhiều lõi (cores) được triển khai trên một chip duy nhất và mỗi lõi xử lý các luồng riêng biệt.

CHƯƠNG 18: MÁY TÍNH ĐA LÕI (MULTICORE COMPUTERS)

### **CHƯƠNG 18: MÁY TÍNH ĐA LÕI (MULTICORE COMPUTERS)**

**Mục tiêu của chương này là gì?**

Sau khi học xong chương này, bạn sẽ có thể:

* Hiểu các vấn đề về hiệu năng phần cứng đã thúc đẩy sự chuyển dịch sang máy tính đa lõi.
* Hiểu các thách thức về hiệu năng phần mềm do việc sử dụng máy tính đa lõi đa luồng đặt ra.
* Trình bày tổng quan về các cách tiếp cận chính đối với tổ chức đa lõi không đồng nhất (heterogeneous).
* Nắm bắt được việc sử dụng tổ chức đa lõi trên các hệ thống nhúng, PC, máy chủ và máy tính lớn (mainframe).

#### **18.1. Các Vấn đề về Hiệu năng Phần cứng (Hardware Performance Issues)**

Trong nhiều thập kỷ, hiệu năng của vi xử lý đã tăng trưởng đều đặn nhờ vào các yếu tố như tăng tần số xung nhịp, tăng mật độ transistor và các cải tiến trong tổ chức bộ xử lý. Tuy nhiên, các kỹ thuật truyền thống để tăng hiệu năng đã đạt đến giới hạn, thúc đẩy sự ra đời của kiến trúc đa lõi.

**A. Gia tăng Tính song song và Độ phức tạp**

Các thay đổi về tổ chức trong thiết kế bộ xử lý chủ yếu tập trung vào việc khai thác **Tính song song mức lệnh (Instruction-Level Parallelism - ILP)** để thực hiện nhiều công việc hơn trong mỗi chu kỳ xung nhịp. Các thay đổi này bao gồm:

* **Đường ống lệnh (Pipelining):** Các lệnh riêng lẻ được thực thi qua nhiều giai đoạn của một đường ống, cho phép nhiều lệnh ở các giai đoạn khác nhau được xử lý đồng thời.
* **Siêu vô hướng (Superscalar):** Nhiều đường ống được xây dựng bằng cách nhân bản các tài nguyên thực thi, cho phép thực thi song song các lệnh miễn là tránh được các rủi ro.
* **Đa luồng đồng thời (Simultaneous Multithreading - SMT):** Các khối thanh ghi được nhân bản để nhiều luồng có thể chia sẻ việc sử dụng các tài nguyên đường ống.

Với mỗi sự đổi mới này, các nhà thiết kế đã cố gắng tăng hiệu năng bằng cách bổ sung thêm sự phức tạp. Tuy nhiên, việc tăng số giai đoạn đường ống, số lượng đường ống song song, hay số lượng luồng được quản lý đều có giới hạn thực tế. Logic điều phối và truyền tín hiệu trở nên phức tạp hơn, chiếm nhiều diện tích chip hơn và gây khó khăn cho việc thiết kế, chế tạo và gỡ lỗi.

**B. Tiêu thụ Điện năng (Power Consumption)**

Để duy trì xu hướng hiệu năng cao hơn, các nhà thiết kế đã sử dụng các thiết kế bộ xử lý phức tạp và tần số xung nhịp cao. Tuy nhiên, yêu cầu về điện năng đã tăng theo cấp số nhân khi mật độ chip và tần số xung nhịp tăng lên.

* Mật độ năng lượng (Watts/cm²) tăng lên, gây khó khăn trong việc tản nhiệt.
* Để kiểm soát mật độ năng lượng, một cách tiếp cận là sử dụng nhiều diện tích chip hơn cho bộ nhớ đệm (cache). Các transistor bộ nhớ nhỏ hơn và có mật độ năng lượng thấp hơn logic xử lý khoảng một bậc.

Những thách thức này đã dẫn đến một quy tắc kinh nghiệm được gọi là **Quy tắc Pollack**, phát biểu rằng sự gia tăng hiệu năng gần như tỷ lệ thuận với căn bậc hai của sự gia tăng độ phức tạp. Nói cách khác, việc tăng gấp đôi logic trong một lõi xử lý chỉ mang lại hiệu suất cao hơn khoảng 40%. Ngược lại, việc sử dụng nhiều lõi có khả năng mang lại sự cải thiện hiệu năng gần như tuyến tính với số lượng lõi, miễn là phần mềm có thể tận dụng được nó.

#### **18.2. Các Vấn đề về Hiệu năng Phần mềm (Software Performance Issues)**

Lợi ích hiệu năng tiềm năng của một tổ chức đa lõi phụ thuộc vào khả năng khai thác hiệu quả các tài nguyên song song có sẵn cho ứng dụng.

**A. Phần mềm trên Đa lõi**

**Định luật Amdahl** cho thấy ngay cả một phần nhỏ mã tuần tự cũng có tác động đáng kể đến hiệu năng. Nếu chỉ 10% mã là tuần tự, việc chạy chương trình trên một hệ thống đa lõi với tám bộ xử lý chỉ mang lại mức tăng hiệu suất gấp 4,7 lần. Hơn nữa, phần mềm thường phải chịu chi phí do giao tiếp và phân phối công việc giữa các bộ xử lý và chi phí cho tính nhất quán của bộ đệm.

Tuy nhiên, các kỹ sư phần mềm đã giải quyết vấn đề này và có nhiều ứng dụng có thể khai thác hiệu quả hệ thống đa lõi. Các ví dụ bao gồm:

* **Ứng dụng gốc đa luồng:** Các ứng dụng có một số lượng nhỏ các tiến trình được phân luồng cao độ (ví dụ: Lotus Domino).
* **Ứng dụng đa tiến trình:** Các ứng dụng có nhiều tiến trình đơn luồng (ví dụ: Oracle, SAP).
* **Ứng dụng Java:** Máy ảo Java (JVM) là một tiến trình đa luồng cung cấp lập lịch và quản lý bộ nhớ cho các ứng dụng Java.
* **Ứng dụng đa phiên bản:** Chạy nhiều phiên bản của cùng một ứng dụng song song. Công nghệ ảo hóa có thể được sử dụng để cung cấp cho mỗi phiên bản một miền riêng biệt và an toàn.

**B. Ví dụ Ứng dụng: Phần mềm Trò chơi Valve**

Valve, một công ty giải trí và công nghệ, đã lập trình lại phần mềm engine Source của mình để sử dụng đa luồng nhằm khai thác khả năng mở rộng của các bộ xử lý đa lõi. Valve đã khám phá ra các tùy chọn **mức độ chi tiết của luồng (threading granularity)** sau:

* **Luồng hạt thô (Coarse-grained threading):** Các module riêng lẻ (ví dụ: rendering, AI, vật lý) được gán cho các bộ xử lý riêng lẻ.
* **Luồng hạt mịn (Fine-grained threading):** Nhiều tác vụ giống nhau hoặc tương tự được phân tán trên nhiều bộ xử lý.
* **Luồng lai (Hybrid threading):** Sử dụng có chọn lọc luồng hạt mịn cho một số hệ thống và luồng đơn cho các hệ thống khác.

Valve nhận thấy rằng **luồng lai** là phương pháp hứa hẹn nhất và có khả năng mở rộng tốt nhất khi các hệ thống đa lõi với tám hoặc mười sáu bộ xử lý trở nên phổ biến. Các module như kết xuất cảnh (scene rendering) có thể được tổ chức thành nhiều luồng để đạt hiệu suất cao hơn khi được phân tán trên nhiều bộ xử lý hơn.

#### **18.3. Tổ chức Đa lõi (Multicore Organization)**

Các biến thiết kế chính trong một tổ chức đa lõi bao gồm:

* Số lượng lõi xử lý trên chip.
* Số lượng cấp bộ nhớ đệm.
* Cách bộ nhớ đệm được chia sẻ giữa các lõi.

**A. Các cấp Bộ đệm (Levels of Cache)**

Các tổ chức đa lõi phổ biến bao gồm:

* **L1 chuyên dụng:** Mỗi lõi có L1 chuyên dụng riêng; không có các cấp cache cao hơn trên chip (ví dụ: ARM11 MPCore).
* **L1 và L2 chuyên dụng:** Mỗi lõi có L1 và L2 chuyên dụng (ví dụ: AMD Opteron).
* **L2 chia sẻ:** Mỗi lõi có L1 chuyên dụng và chia sẻ một L2 chung (ví dụ: Intel Core Duo).
* **L3 chia sẻ:** Mỗi lõi có L1 và L2 chuyên dụng và chia sẻ một L3 chung (ví dụ: Intel Core i7).

Việc sử dụng bộ đệm cấp cao hơn được chia sẻ trên chip có một số ưu điểm:

1. **Sự giao thoa mang tính xây dựng** có thể làm giảm tỷ lệ trượt tổng thể.
2. Dữ liệu được chia sẻ bởi nhiều lõi không bị sao chép ở cấp bộ đệm chia sẻ.
3. Lượng bộ đệm chia sẻ được phân bổ cho mỗi lõi là **động**.
4. Giao tiếp giữa các lõi dễ dàng thực hiện thông qua các vị trí bộ nhớ được chia sẻ.
5. Vấn đề **nhất quán bộ đệm** được giới hạn ở các cấp bộ đệm thấp hơn.

**B. Đa luồng Đồng thời (Simultaneous Multithreading - SMT)**

Việc các lõi riêng lẻ có thực hiện SMT hay không là một quyết định thiết kế tổ chức khác. SMT có tác dụng mở rộng số lượng luồng cấp phần cứng mà hệ thống đa lõi hỗ trợ. Một hệ thống đa lõi với bốn lõi và SMT hỗ trợ bốn luồng đồng thời trong mỗi lõi sẽ có vẻ giống như một hệ thống đa lõi với 16 lõi đối với cấp ứng dụng.

*Lưu ý: Các phần về tổ chức đa lõi không đồng nhất và các ví dụ cụ thể như Intel Core i7, ARM Cortex-A15, và IBM zEnterprise EC12 được đề cập trong sách giáo khoa nhưng không được yêu cầu chi tiết trong truy vấn của bạn. Nếu bạn muốn tìm hiểu về chúng, xin vui lòng cho tôi biết*.

Chắc chắn rồi. Rất vui khi bạn muốn tìm hiểu sâu hơn về các kiến trúc đa lõi hiện đại. Dưới đây là phần trình bày chi tiết và kỹ lưỡng về các chủ đề đó, được tổng hợp từ sách giáo khoa *Computer Organization and Architecture* của William Stallings.

### **CHƯƠNG 18: MÁY TÍNH ĐA LÕI (Bổ sung chi tiết)**

Phần này sẽ đi sâu vào các chủ đề bạn quan tâm: tổ chức đa lõi không đồng nhất và các ví dụ kiến trúc cụ thể từ Intel, ARM, và IBM.

#### **18.4. Tổ chức Đa lõi Không đồng nhất (Heterogeneous Multicore Organization)**

Trong khi tổ chức đa lõi **đồng nhất (homogeneous)**—với các lõi giống hệt nhau—rất phổ biến, một xu hướng ngày càng tăng là tổ chức đa lõi **không đồng nhất (heterogeneous)**, tức là một chip chứa nhiều hơn một loại lõi. Cách tiếp cận này nhằm tối ưu hóa tốt hơn về hiệu năng và/hoặc mức tiêu thụ điện năng. Có hai phương pháp chính cho thiết kế này.

**A. Các Kiến trúc Tập lệnh (ISA) Khác nhau**

Cách tiếp cận này kết hợp các lõi có kiến trúc tập lệnh khác nhau, thường là pha trộn giữa các lõi xử lý trung tâm (CPU) thông thường với các lõi chuyên dụng được tối ưu hóa cho các loại dữ liệu hoặc ứng dụng cụ thể.

* **Multicore CPU/GPU:** Đây là xu hướng nổi bật nhất, kết hợp CPU và các **đơn vị xử lý đồ họa (GPU)** trên cùng một chip.  
  + **Đặc điểm GPU:** GPU được thiết kế để thực hiện các phép toán song song trên dữ liệu đồ họa, với khả năng hỗ trợ hàng nghìn luồng thực thi song song. Chúng rất phù hợp cho các ứng dụng xử lý lượng lớn dữ liệu vector và ma trận. Nhờ các mô hình lập trình dễ tiếp cận như CUDA, GPU ngày càng được sử dụng như các bộ xử lý vector cho nhiều ứng dụng tính toán lặp đi lặp lại (GPGPU).
  + **Thách thức về bộ nhớ đệm:** Trong một tổ chức CPU/GPU điển hình, các CPU và GPU chia sẻ các tài nguyên trên chip như bộ đệm cấp cuối cùng (LLC), mạng kết nối và bộ điều khiển bộ nhớ. Tuy nhiên, sự khác biệt về độ nhạy của bộ đệm và tốc độ truy cập bộ nhớ giữa CPU và GPU tạo ra những thách thức đáng kể cho việc chia sẻ LLC một cách hiệu quả.
  + **Vấn đề trao đổi dữ liệu:** Trong các triển khai ban đầu, bộ nhớ vật lý được phân chia giữa CPU và GPU. Khi một ứng dụng cần xử lý GPU, CPU phải sao chép rõ ràng dữ liệu vào bộ nhớ GPU, và GPU sau đó sao chép kết quả trở lại bộ nhớ CPU. Việc trao đổi dữ liệu vật lý này gây ra một chi phí hiệu năng đáng kể.
  + **Giải pháp (HSA):** Các nỗ lực như nền tảng **Kiến trúc Hệ thống Không đồng nhất (HSA)** đang được phát triển để giải quyết vấn đề này. Các tính năng chính bao gồm: không gian bộ nhớ ảo hợp nhất cho cả CPU và GPU; chính sách bộ nhớ nhất quán để đảm bảo cả hai bộ đệm CPU và GPU đều thấy dữ liệu được cập nhật; và một giao diện lập trình hợp nhất.
* **Multicore CPU/DSP:** Một ví dụ phổ biến khác là sự kết hợp giữa CPU và **bộ xử lý tín hiệu số (DSP)**. DSP cung cấp các chuỗi lệnh cực nhanh (như dịch chuyển-và-cộng, nhân-và-cộng) thường được sử dụng trong các ứng dụng xử lý tín hiệu số chuyên sâu về toán học. Chúng được sử dụng rộng rãi trong điện thoại di động, card âm thanh, máy fax, modem, và TV kỹ thuật số.

**B. Các Kiến trúc Tập lệnh (ISA) Tương đương**

Cách tiếp cận này sử dụng nhiều lõi có ISA tương đương nhưng khác nhau về hiệu năng hoặc hiệu quả năng lượng.

* **Kiến trúc big.LITTLE của ARM:** Đây là ví dụ hàng đầu, nhắm vào thị trường điện thoại thông minh và máy tính bảng, nơi nhu cầu về hiệu năng tăng nhanh hơn dung lượng pin.
  + **Nguyên tắc:** Kiến trúc này kết hợp các lõi "LITTLE" (ví dụ: Cortex-A7) tiêu thụ điện năng thấp, hiệu quả cao với các lõi "big" (ví dụ: Cortex-A15) hiệu năng cao. Lõi A7 xử lý các tác vụ ít tính toán hơn (như xử lý nền, nghe nhạc, nhắn tin), trong khi lõi A15 được kích hoạt cho các tác vụ cường độ cao (như video, chơi game).
  + **Sự khác biệt về đường ống lệnh:** Lõi A7 là một CPU thực thi theo thứ tự (in-order) với đường ống dài 8-10 giai đoạn, trong khi A15 là một bộ xử lý thực thi không theo thứ tự (out-of-order) với đường ống dài 15-24 giai đoạn. Điều này dẫn đến việc A15 mang lại hiệu năng gần gấp đôi A7 trên mỗi MHz, nhưng A7 lại hiệu quả về năng lượng hơn khoảng ba lần.
  + **Mô hình xử lý phần mềm:**
    1. **Di chuyển (Migration):** Các lõi big và LITTLE được ghép cặp. Với bộ lập lịch của HĐH, mỗi cặp được xem như một lõi duy nhất. Phần mềm quản lý năng lượng di chuyển các ngữ cảnh phần mềm giữa hai lõi. Khi hiệu năng yêu cầu tăng lên, tác vụ sẽ được di chuyển từ lõi LITTLE sang lõi big.
    2. **Đa xử lý (MP):** Cho phép bất kỳ sự kết hợp nào của các lõi A15 và A7 được bật nguồn và thực thi đồng thời. Các tác vụ đòi hỏi hiệu năng cao được gán cho lõi big, trong khi các tác vụ yêu cầu thấp chạy trên lõi LITTLE. Mô hình này phức tạp hơn nhưng hiệu quả hơn về tài nguyên.
  + **Tính nhất quán của bộ đệm (Cache Coherence):** ARM sử dụng một giao thức phần cứng gọi là **ACE (Advanced Extensible Interface Coherence Extensions)**. ACE dựa trên một mô hình 5 trạng thái gọi là **MOESI** (Modified, Owned, Exclusive, Shared, Invalid) để duy trì tính nhất quán giữa các bộ đệm L1 của các lõi và bộ đệm L2 được chia sẻ.

#### **Các Ví dụ về Kiến trúc Đa lõi**

**1. Intel Core i7-990X**

Đây là một ví dụ về tổ chức đa lõi đồng nhất (homogeneous) được sử dụng trong các máy trạm và máy chủ hiệu năng cao.

* **Cấu trúc chung:**
  + Chip này có **sáu lõi (six cores)**, mỗi lõi có bộ đệm L2 chuyên dụng riêng.
  + Tất cả sáu lõi chia sẻ chung một bộ đệm **L3 12 MB**. Intel sử dụng cơ chế **tìm nạp trước (prefetching)**, trong đó phần cứng kiểm tra các mẫu truy cập bộ nhớ và cố gắng điền trước vào bộ đệm các dữ liệu có khả năng sẽ được yêu cầu sớm.
* **Kết nối ngoại vi:**
  + **Bộ điều khiển bộ nhớ DDR3:** Bộ điều khiển bộ nhớ chính DDR được tích hợp ngay trên chip, giúp loại bỏ Front Side Bus và tăng tốc độ truy cập bộ nhớ. Giao diện hỗ trợ ba kênh rộng 8 byte (tổng cộng 192 bit), cho tốc độ dữ liệu tổng hợp lên tới 32 GB/s.
  + **QuickPath Interconnect (QPI):** Đây là một kết nối điểm-tới-điểm (point-to-point) nhất quán bộ đệm, cho phép giao tiếp tốc độ cao giữa các chip xử lý được kết nối. Mỗi liên kết QPI hoạt động ở tốc độ 6.4 GT/s (gigatransfers per second) với tổng băng thông hai chiều là 25.6 GB/s.

**2. ARM Cortex-A15 MPCore**

Đây là một chip đa lõi đồng nhất hiệu năng cao nhắm vào các ứng dụng như điện toán di động, máy chủ gia đình kỹ thuật số cao cấp và cơ sở hạ tầng không dây.

* **Các thành phần chính:**
  + **Lõi (Core):** Có thể cấu hình với tối đa 4 lõi Cortex-A15.
  + **Bộ đệm (Cache):** Mỗi lõi có bộ đệm L1 chỉ lệnh và L1 dữ liệu chuyên dụng. Có một bộ đệm L2 được chia sẻ giữa tất cả các lõi.
  + **Bộ điều khiển ngắt chung (GIC):** Xử lý việc phát hiện, ưu tiên và phân phối ngắt đến các lõi riêng lẻ. Nó cũng cung cấp một phương tiện cho giao tiếp giữa các bộ xử lý (interprocessor communication).
  + **Đơn vị điều khiển Snoop (SCU):** Chịu trách nhiệm duy trì tính nhất quán của bộ đệm L1/L2.
* **Tính nhất quán của bộ đệm L1:**
  + SCU sử dụng giao thức MESI để duy trì tính nhất quán của bộ đệm L1. Nó giới thiệu ba loại tối ưu hóa:
    1. **Can thiệp dữ liệu trực tiếp (Direct data intervention):** Cho phép sao chép dữ liệu sạch từ bộ đệm L1 của một CPU sang bộ đệm L1 của CPU khác mà không cần truy cập bộ nhớ ngoài, giúp giải quyết một lần trượt L1 cục bộ tại một bộ đệm L1 từ xa.
    2. **RAM thẻ nhân bản (Duplicated tag RAMs):** SCU có các bản sao của các RAM thẻ L1 để kiểm tra sự sẵn có của dữ liệu trước khi gửi các lệnh nhất quán, giúp giảm lưu lượng snoop không cần thiết.
    3. **Các dòng di chuyển (Migratory lines):** Cho phép di chuyển dữ liệu bẩn từ CPU này sang CPU khác mà không cần ghi vào L2 rồi đọc lại, giúp tối ưu hóa việc di chuyển dữ liệu được chia sẻ giữa các lõi.

**3. IBM zEnterprise EC12 Mainframe**

Đây là một ví dụ về kiến trúc đa lõi được sử dụng trong các hệ thống máy tính lớn (mainframe) hiệu năng cực cao.

* **Tổ chức:**
  + Khối xây dựng chính là **mô-đun đa chip (MCM)**, chứa sáu chip **đơn vị xử lý (PU)** và hai chip **điều khiển lưu trữ (SC)**.
  + Mỗi chip PU chứa **sáu lõi 5.5 GHz**, mỗi lõi có một đường ống siêu vô hướng, thực thi không theo thứ tự, có khả năng giải mã ba lệnh CISC mỗi chu kỳ xung nhịp và thực thi tới bảy phép toán mỗi chu kỳ.
* **Cấu trúc bộ đệm bốn cấp (Four-Level Cache):**
  + **L1:** Mỗi lõi có L1 chuyên dụng 160 KB (96 KB dữ liệu, 64 KB chỉ lệnh). L1 là bộ đệm **ghi thẳng (write-through)** vào L2.
  + **L2:** Mỗi lõi có L2 chuyên dụng 2 MB (1 MB dữ liệu, 1 MB chỉ lệnh). L2 cũng là bộ đệm **ghi thẳng (write-through)** vào L3.
  + **L3:** Mỗi chip PU có một bộ đệm L3 24 MB được chia sẻ bởi tất cả sáu lõi. L3 được triển khai bằng **eDRAM (embedded DRAM)**, cho phép dung lượng lớn hơn so với SRAM trên cùng một diện tích, điều này quan trọng đối với các khối lượng công việc xử lý giao dịch.
  + **L4:** Tất cả 6 PU trên một MCM chia sẻ một bộ đệm L4 192 MB (được chia trên hai chip SC). L4 đóng vai trò là **trình quản lý nhất quán (coherence manager)**, có nghĩa là tất cả các lần tìm nạp bộ nhớ phải ở trong L4 trước khi dữ liệu có thể được sử dụng bởi bộ xử lý. Cấu trúc này nhằm giảm thiểu nhu cầu truy cập bộ nhớ chính (DRAM) do sự chênh lệch tốc độ lớn với các lõi xử lý tốc độ cao.
* **Tối ưu hóa hiệu năng:** Hệ điều hành z/OS và Trình quản lý tài nguyên hệ thống (SRM) phối hợp với nhau để giữ cho càng nhiều công việc càng tốt trong ranh giới của càng ít lõi và không gian bộ đệm L4 (tốt nhất là trong cùng một "book"), nhằm khắc phục độ trễ vốn có trong thiết kế đa chip và đa book.

Hy vọng những giải thích chi tiết này giúp bạn hiểu sâu hơn về các kiến trúc phức tạp và thú vị này. Nếu có bất kỳ câu hỏi nào khác, xin vui lòng cho tôi biết!

ASSEMBLY

### **Tổng Hợp Kiến Thức Cần Thiết Để Thông Thạo Assembly**

Để lập trình Assembly hiệu quả, bạn cần nắm vững kiến thức ở 4 mảng chính: **1) Kiến trúc tập lệnh (ISA)**, **2) Cấu trúc và chức năng của CPU**, **3) Cách máy tính quản lý bộ nhớ**, và **4) Ngôn ngữ Assembly và công cụ liên quan**.

#### **Phần 1: Hiểu Rõ "Luật Chơi" – Kiến Trúc Tập Lệnh (ISA)**

Đây là nền tảng cơ bản nhất. ISA chính là giao diện giữa phần cứng và phần mềm, định nghĩa những gì bạn có thể ra lệnh cho CPU làm. Bạn cần nắm vững các khái niệm sau:

* **Các Thành Phần Của Một Lệnh Máy (Chương 12):**
  + **Opcode (Mã toán tử):** Xác định hành động cần thực hiện (ví dụ: ADD, SUB, MOV).
  + **Toán Hạng (Operands):** Dữ liệu mà lệnh sẽ tác động lên. Bạn phải biết các loại toán hạng:
    - **Tức thời (Immediate):** Giá trị nằm ngay trong lệnh (ví dụ: MOV EAX, 100).
    - **Thanh ghi (Register):** Dữ liệu nằm trong thanh ghi của CPU (ví dụ: MOV EAX, EBX).
    - **Bộ nhớ (Memory):** Dữ liệu nằm ở một địa chỉ trong bộ nhớ chính.
* **Các Chế Độ Địa Chỉ (Addressing Modes - Chương 13):** Đây là phần **cực kỳ quan trọng** trong Assembly. Nó quy định cách CPU tìm ra địa chỉ chính xác của toán hạng. Các chế độ chính bạn phải thông thạo bao gồm:  
  + **Immediate, Direct, Register:** Các chế độ cơ bản nhất.
  + **Register Indirect:** Dùng một thanh ghi để chứa địa chỉ của dữ liệu, rất linh hoạt.
  + **Displacement (Dịch chuyển):** Chế độ mạnh mẽ nhất, kết hợp một thanh ghi với một giá trị dịch chuyển (offset) để tính ra địa chỉ cuối cùng. Nó là cơ sở cho các kỹ thuật:
    - **Base-Register Addressing:** Dùng để truy cập các biến cục bộ trong stack frame.
    - **Indexing:** Dùng để truy cập các phần tử trong mảng một cách hiệu quả.
    - **Relative Addressing:** Dùng trong các lệnh nhảy, tính toán địa chỉ đích dựa trên vị trí lệnh hiện tại (PC).
* **Các Loại Lệnh Chính (Chương 12):** Bạn cần hiểu rõ chức năng của các nhóm lệnh sau:  
  + **Truyền dữ liệu (Data Transfer):** MOV, PUSH, POP. Đây là nhóm lệnh phổ biến nhất.
  + **Số học (Arithmetic):** ADD, SUB, MUL, DIV. Các lệnh này thường thay đổi các cờ trạng thái (status flags).
  + **Logic và Dịch chuyển (Logical & Shift):** AND, OR, XOR, SHL, SHR. Rất hữu ích để thao tác bit.
  + **Chuyển giao điều khiển (Transfer of Control):** JMP, JE, JG (lệnh nhảy có điều kiện và không điều kiện), và đặc biệt là CALL/RET để gọi và trở về từ thủ tục (hàm).

#### **Phần 2: Hiểu Rõ "Người Thực Thi" – Cấu Trúc và Chức Năng CPU**

Viết code Assembly là bạn đang ra lệnh trực tiếp cho CPU. Vì vậy, hiểu rõ cấu trúc bên trong của nó là điều bắt buộc.

* **Tổ Chức Thanh Ghi (Register Organization - Chương 14):**
  + **Thanh ghi đa năng (General-Purpose Registers):** EAX, EBX, ECX, EDX là những thanh ghi chính bạn sẽ dùng để tính toán và lưu trữ dữ liệu tạm thời.
  + **Thanh ghi con trỏ và chỉ mục (Pointer & Index Registers):** ESP (con trỏ đỉnh stack), EBP (con trỏ nền stack), ESI, EDI rất quan trọng trong việc truy cập bộ nhớ và quản lý stack frame .
  + **Thanh ghi cờ (FLAGS/EFLAGS Register):** Chứa các mã điều kiện (condition codes) như Zero Flag (ZF), Sign Flag (SF), Carry Flag (CF), Overflow Flag (OF). Các lệnh nhảy có điều kiện (JE, JG,...) hoạt động dựa trên trạng thái của các cờ này. Bạn **bắt buộc** phải hiểu lệnh nào ảnh hưởng đến cờ nào để viết logic điều khiển chính xác.
  + **Thanh ghi con trỏ lệnh (Instruction Pointer - EIP/PC):** Chứa địa chỉ của lệnh tiếp theo sẽ thực thi. Các lệnh nhảy và gọi hàm thực chất là thay đổi giá trị của thanh ghi này.
* **Chu Trình Lệnh và Tương Tác Phần Cứng (Chương 3 & 14):**
  + Hiểu rõ các bước **Fetch-Decode-Execute** giúp bạn hình dung được CPU xử lý từng lệnh của bạn như thế nào.
  + **Ngắt (Interrupts):** Biết rằng các sự kiện như I/O hoàn tất sẽ tạm dừng chương trình của bạn, lưu trạng thái (context) vào stack, và nhảy đến một trình xử lý ngắt (thường là của hệ điều hành).

#### **Phần 3: Hiểu Rõ "Không Gian Làm Việc" – Quản Lý Bộ Nhớ**

Chương trình và dữ liệu của bạn nằm trong bộ nhớ. Cách hệ điều hành và CPU quản lý bộ nhớ ảnh hưởng trực tiếp đến cách bạn viết code.

* **Ngăn Xếp (Stack - Chương 12 & Phụ lục B):** Đây là khái niệm **sống còn** trong Assembly.  
  + **Nguyên tắc LIFO (Vào sau, ra trước):** Dữ liệu được thêm vào (PUSH) và lấy ra (POP) từ đỉnh stack.
  + **Quản lý Thủ tục (Procedure Management):** Lệnh CALL tự động PUSH địa chỉ trả về vào stack. Lệnh RET sẽ POP địa chỉ này ra để quay về nơi đã gọi.
  + **Truyền tham số và Biến cục bộ:** Stack được dùng để truyền tham số cho hàm và cấp phát không gian cho các biến cục bộ. EBP và ESP là hai thanh ghi chính để quản lý **khung stack (stack frame)**.
* **Địa Chỉ Logic và Vật Lý (Chương 8):** Hiểu rằng địa chỉ bạn dùng trong code (OFFSET var1, [EBP-4]) là địa chỉ logic (hoặc ảo). Hệ điều hành và MMU (Memory Management Unit) sẽ chuyển đổi nó thành địa chỉ vật lý trong RAM.

#### **Phần 4: Công Cụ và Ngôn Ngữ**

* **Ngôn Ngữ Assembly và Assembler (Chương 13 & Phụ lục B):**
  + **Cú pháp:** Nắm vững cấu trúc một dòng lệnh: Label Mnemonic Operand(s) ;Comment.
  + **Chỉ thị (Directives):** Các lệnh cho Assembler (không phải CPU) như .data để khai báo biến toàn cục, .code để bắt đầu vùng mã, EQU để định nghĩa hằng số, PROTO để khai báo prototype cho hàm.
  + **Trình hợp dịch (Assembler):** Hiểu rằng Assembler là chương trình dịch mã Assembly của bạn thành mã máy nhị phân (object code). Assembler hai lượt (two-pass) là phổ biến nhất: lượt đầu xây dựng bảng ký hiệu (symbol table), lượt hai dịch lệnh và phân giải địa chỉ.
  + **Linker và Loader:** Biết rằng Linker sẽ kết hợp các file object code và thư viện lại thành một file thực thi duy nhất (.exe). Loader của hệ điều hành sẽ nạp file này vào bộ nhớ để chạy.

### **Lộ Trình Học Tập Đề Xuất:**

1. **Nền tảng (Chương 9, 10, 11):** Bắt đầu bằng cách đảm bảo bạn hiểu rõ hệ đếm (nhị phân, thập lục phân), cách biểu diễn số nguyên (bù hai) và các phép toán số học cơ bản ở mức bit.
2. **Làm quen với Assembly và công cụ (MASM):**
   * Cài đặt MASM và làm quen với IDE.
   * Viết các chương trình đơn giản như "Hello World", sử dụng các hàm I/O có sẵn trong thư viện (print, input, str$, sval) để tương tác. Mục tiêu là hiểu cấu trúc một chương trình MASM, cách biên dịch và chạy.
3. **Tập trung vào Tập lệnh và CPU (Chương 12, 13, 14):**
   * Học các lệnh cơ bản: MOV, ADD, SUB, CMP, các lệnh nhảy JMP, JE, JG,....
   * **Thực hành với thanh ghi:** Viết các chương trình tính toán đơn giản chỉ sử dụng thanh ghi.
   * **Thực hành các chế độ địa chỉ:** Viết code truy cập biến toàn cục (.data) và biến cục bộ (LOCAL). Đặc biệt chú trọng vào lệnh LEA để lấy địa chỉ biến cục bộ.
4. **Làm chủ Thủ tục và Stack (Chương 12 & Assembly 02):**
   * Học cách định nghĩa và gọi thủ tục (PROC, ENDP, CALL, RET).
   * **Vẽ sơ đồ bộ nhớ (memory map):** Thực hành vẽ ra trạng thái của stack trước và sau khi gọi hàm. Theo dõi giá trị của ESP, EBP, địa chỉ trả về, các tham số. Đây là cách tốt nhất để hiểu sâu về stack.
   * Thực hành truyền tham số theo cả hai cách: **truyền trị** (pass-by-value) và **truyền tham chiếu** (pass-by-reference) bằng cách truyền con trỏ.
5. **Cấu trúc điều khiển nâng cao:**
   * Học cách triển khai các vòng lặp (LOOP hoặc dùng CMP/JMP).
   * Thực hành dịch các cấu trúc if-else và switch-case từ ngôn ngữ bậc cao sang Assembly.

Việc nắm vững các khái niệm nền tảng về tổ chức máy tính sẽ giúp bạn không chỉ vượt qua môn học mà còn trở thành một lập trình viên giỏi hơn, có khả năng viết mã hiệu quả và gỡ lỗi ở mức độ sâu. Chúc bạn học tốt!