CAE201 Notes - lý thuyết

Computer Architecture (Trường Đại học FPT)

SLOT 1: 9/9/2024

1.1 ORGANIZATION AND ARCHITECTURE (TỐI REVIEW)

- **Kiến trúc máy tính** (computer architecture) đề cập đến các thuộc tính của hệ thống mà lập trình viên có thể thấy được, tức là các yếu tố ảnh hưởng trực tiếp đến việc thực thi chương trình. Một thuật ngữ liên quan là **Instruction Set Architecture (ISA)**, nghĩa là tập hợp các lệnh, định dạng lệnh, mã lệnh, thanh ghi, bộ nhớ, và cách các lệnh tương tác với bộ nhớ và thanh ghi.

    - Ví dụ: Quyết định xem máy tính có lệnh nhân (multiply instruction) hay không là vấn đề về kiến trúc.

- **Tổ chức máy tính (computer organization)** liên quan đến cách các đơn vị hoạt động trong máy tính kết nối và thực hiện các thông số kiến trúc. Nó bao gồm các chi tiết phần cứng mà lập trình viên không thấy, như tín hiệu điều khiển, các giao diện với thiết bị ngoại vi và công nghệ bộ nhớ.

    - Ví dụ: Việc quyết định lệnh nhân được thực hiện bằng một đơn vị nhân đặc biệt hay thông qua đơn vị cộng nhiều lần là vấn đề về tổ chức.

- **Sự khác biệt giữa kiến trúc và tổ chức:** Một số hãng máy tính cung cấp các dòng sản phẩm có cùng kiến trúc nhưng khác về tổ chức, dẫn đến các mẫu có giá thành và hiệu suất khác nhau. Ví dụ nổi bật là kiến trúc IBM System/370, ra mắt năm 1970 và được nâng cấp nhiều lần nhưng vẫn giữ nguyên kiến trúc cơ bản để bảo vệ khoản đầu tư phần mềm của khách hàng.
- **Mối quan hệ giữa kiến trúc và tổ chức**: Trong các dòng máy nhỏ như microcomputers, sự thay đổi về công nghệ không chỉ ảnh hưởng đến tổ chức mà còn dẫn đến các kiến trúc phức tạp hơn, không đòi hỏi nhiều sự tương thích giữa các thế hệ.

1.2 STRUCTURE AND FUNCTION

Cấu trúc phân cấp của máy tính

- Máy tính là một hệ thống phức tạp với hàng triệu linh kiện điện tử, nhưng có thể được hiểu rõ qua cách tiếp cận phân cấp. Mỗi hệ thống lớn có thể được chia thành nhiều hệ thống con có cấu trúc phân cấp nhỏ hơn.
    - Structure: Là cách các thành phần trong máy tính liên kết với nhau.
    - Function: Là cách từng thành phần hoạt động trong cấu trúc đó.
- Have two choices: Cách tiếp cận từ trên xuống (top-down) giúp hiểu hệ thống một cách hiệu quả hơn. Cuốn sách sử dụng cách tiếp cận này, bắt đầu từ các thành phần lớn của máy tính và dần dần đi sâu vào chi tiết hơn.

Chức năng cơ bản của máy tính

- **Xử lý dữ liệu (Data processing):** Xử lý nhiều loại dữ liệu khác nhau.
- **Lưu trữ dữ liệu (Data storage):** Lưu trữ ngắn hạn (như trong bộ nhớ RAM) và dài hạn (trong ổ đĩa cứng) để sử dụng sau này.
- **Di chuyển dữ liệu (Data movement)**: Di chuyển dữ liệu giữa máy tính và các thiết bị bên ngoài (như qua input/output – I/O), hoặc giữa các thiết bị từ xa (data communications).
- **Điều khiển (Control)**: Đơn vị điều khiển quản lý các nguồn lực của máy tính và điều phối hoạt động giữa các thành phần.

1.3 CHƯA HỌC LẠI

SLOT 2: 11/9/2024.

2.1 DESIGNING FOR PERFORMANCE (tìm slide gốc học lại

Sự phát triển của hệ thống máy tính:

- Chi phí của hệ thống máy tính đang giảm mạnh theo thời gian, trong khi hiệu năng và dung lượng lại tăng mạnh. Máy tính hiện nay mạnh mẽ hơn nhiều so với những hệ thống mainframe của IBM từ 10-15 năm trước. Điều này cho thấy sức mạnh tính toán ngày càng rẻ và phổ biến hơn, dẫn đến sự phát triển của nhiều ứng dụng phức tạp như xử lý hình ảnh, nhận diện giọng nói, hội nghị video, và mô hình hóa mô phỏng.

Tốc độ của bộ vi xử lý: (Microprocessor speed)

- Bộ vi xử lý ngày nay mạnh mẽ nhờ các nhà sản xuất chip không ngừng tăng tốc độ xử lý. Điều này dựa trên định luật Moore, nơi số lượng bóng bán dẫn tăng gấp đôi mỗi ba năm. Tuy nhiên, để phát huy hết tiềm năng của vi xử lý, cần một dòng lệnh liên tục, và các kỹ thuật tiên tiến đã được áp dụng để đảm bảo vi xử lý luôn bận rộn và không bị gián đoạn.

Các kỹ thuật giúp cải thiện hiệu năng vi xử lý:

- **Pipeline:** Phân chia quá trình thực hiện lệnh thành nhiều giai đoạn và xử lý đồng thời nhiều lệnh.
- **Dự đoán nhánh:** Bộ xử lý dự đoán trước lệnh nào sẽ được thực hiện để lấy trước lệnh và giữ cho quá trình xử lý không bị gián đoạn.
- **Thực thi superscalar:** Cho phép thực hiện nhiều lệnh trong một chu kỳ đồng hồ, tăng tốc độ xử lý.
- **Phân tích luồng dữ liệu:** Tối ưu hóa lịch trình thực hiện lệnh dựa trên sự phụ thuộc dữ liệu giữa các lệnh.
- **Thực thi suy đoán:** Thực hiện các lệnh có khả năng sẽ cần trước khi chúng thực sự được yêu cầu, giúp tối ưu hóa tốc độ xử lý.

Cân bằng hiệu năng: balance performance

- Dù vi xử lý phát triển rất nhanh, các thành phần khác như bộ nhớ chính không theo kịp, dẫn đến vấn đề tắc nghẽn khi chuyển dữ liệu giữa bộ nhớ và vi xử lý. Điều này có thể được giải quyết bằng cách mở rộng băng thông, sử dụng các cache, hoặc cải tiến các giao diện bộ nhớ.

Cải tiến trong tổ chức chip và kiến trúc:improvement in chip Organization and Architacture.

- Tăng tốc độ xử lý không chỉ đến từ việc tăng tốc độ đồng hồ (clock speed) mà còn đến từ các thay đổi kiến trúc như tăng kích thước cache và sử dụng kỹ thuật xử lý song song (pipeline và superscalar).
- Tuy nhiên, việc tiếp tục tăng tốc độ đồng hồ gặp giới hạn vật lý như vấn đề về tiêu thụ năng lượng và nhiệt tỏa ra, do đó việc cải tiến sẽ ngày càng tập trung vào kiến trúc và tổ chức chip hơn là chỉ tăng tốc độ đồng hồ.

Tăng cường bộ nhớ đệm (cache) và xử lý song song:

- Ngày càng nhiều bộ nhớ đệm được tích hợp vào chip vi xử lý để giảm thời gian truy cập dữ liệu.
- Các chiến lược như pipelining và superscalar giúp tăng cường khả năng xử lý nhiều lệnh cùng lúc.

Giới hạn vật lý:

- Việc tiếp tục tăng tốc độ đồng hồ và mật độ bóng bán dẫn gặp các thách thức như tiêu tốn năng lượng, độ trễ và khả năng quản lý nhiệt. Do đó, hiệu năng sẽ dựa vào các cải tiến kiến trúc hơn là chỉ dựa vào tốc độ đồng hồ.

Phần này trình bày về sự **cân bằng hiệu năng** giữa bộ vi xử lý, bộ nhớ và các thành phần khác trong hệ thống máy tính. Trong khi tốc độ bộ vi xử lý tăng nhanh, các thành phần khác như bộ nhớ và thiết bị ngoại vi không phát triển đồng đều, dẫn đến việc cần tối ưu hóa kiến trúc và tổ chức hệ thống. Đặc biệt, **giao diện giữa bộ xử lý và bộ nhớ** là vấn đề quan trọng, vì nếu bộ nhớ không theo kịp tốc độ của vi xử lý, hiệu suất của hệ thống sẽ bị ảnh hưởng.

Các giải pháp cải thiện hiệu suất bao gồm:

- Tăng số bit lấy cùng lúc từ bộ nhớ bằng cách làm **DRAM rộng hơn** và sử dụng đường dẫn dữ liệu rộng.
- Thay đổi giao diện **DRAM** để tăng hiệu quả, như tích hợp cache hoặc đệm trên chip DRAM.
- Giảm tần suất truy cập bộ nhớ bằng việc sử dụng **cache** phức tạp và hiệu quả hơn giữa vi xử lý và bộ nhớ.
- Tăng băng thông kết nối giữa **vi xử lý và bộ nhớ** bằng cách sử dụng bus tốc độ cao hơn.

Để đáp ứng nhu cầu của các thiết bị I/O, các chiến lược bao gồm sử dụng **cache**, **đệm**, và các cấu trúc kết nối tốc độ cao hơn.

Tăng tốc độ bộ vi xử lý vẫn là mục tiêu chính, với các phương pháp:

1. Tăng **tốc độ phần cứng** của vi xử lý bằng cách thu nhỏ các cổng logic và tăng tốc độ đồng hồ.
2. Tăng kích thước và tốc độ của **cache** nằm giữa vi xử lý và bộ nhớ.
3. Thay đổi tổ chức và kiến trúc để tận dụng **xử lý song song**.

Những thách thức trong việc tiếp tục tăng tốc độ bao gồm vấn đề về **năng lượng**, **độ trễ RC**, và sự chênh lệch về tốc độ giữa bộ nhớ và vi xử lý. Các chiến lược tăng hiệu suất đã đạt đến điểm bão hòa, dẫn đến việc các nhà thiết kế chuyển sang sử dụng chip **đa lõi** nhằm khai thác số lượng bóng bán dẫn ngày càng tăng thay vì chỉ dựa vào tăng tốc độ đồng hồ.

1.3 Multicore, Mics and GPGPUS.

A. **Chiến lược đa lõi (Multicore)**:
   - Sử dụng nhiều bộ xử lý trên cùng một chip giúp tăng hiệu suất mà không cần tăng tốc độ xung nhịp.
   - Nghiên cứu cho thấy tăng số lượng lõi đơn giản sẽ gần như tăng gấp đôi hiệu suất so với chỉ dùng một bộ xử lý phức tạp.
   - Với nhiều lõi, việc mở rộng kích thước bộ nhớ đệm (cache) lớn hơn là hợp lý vì bộ nhớ tiêu thụ ít điện năng hơn logic xử lý.

B. **Sự phát triển của đa lõi**:
   - Xu hướng tăng số lõi và kích thước bộ nhớ đệm trên chip: từ 2 lõi, 4 lõi, 8 lõi, 16 lõi, lên tới hơn 50 lõi trên mỗi chip.
   - Bộ nhớ đệm cấp 1 thường dành riêng cho mỗi lõi, trong khi các cấp 2 và 3 ban đầu dùng chung, nhưng hiện tại cấp 2 cũng có thể là riêng biệt cho từng lõi.

C. **Vấn đề phần mềm**:
   - Số lượng lõi tăng đặt ra thách thức lớn cho phát triển phần mềm, cần khai thác tối đa hiệu suất của các lõi này.

D. **Chiến lược lõi tích hợp nhiều (MIC)**:

- Thuật ngữ MIC (Many Integrated Core) xuất hiện để mô tả việc tích hợp nhiều hơn 50 lõi trên một chip.
- MIC có tiềm năng tăng hiệu suất, nhưng đòi hỏi phần mềm hỗ trợ mạnh mẽ để khai thác hết lợi ích.

E. **Chip với các lõi chuyên dụng**:
- Một xu hướng thiết kế khác là chip kết hợp nhiều bộ xử lý đa dụng với GPU (đơn vị xử lý đồ họa) và các lõi chuyên biệt cho xử lý video hoặc tác vụ khác.
- GPU được thiết kế để thực hiện các phép toán song song trên dữ liệu đồ họa, thường dùng cho mã hóa và hiển thị đồ họa 2D, 3D và xử lý video.

F. **GPU và GPGPU (General-purpose computing on GPUs)**:
- GPU ngày càng được sử dụng như các bộ xử lý vector cho các ứng dụng tính toán lặp lại, làm mờ ranh giới giữa GPU và CPU.
- Khi GPU được dùng cho nhiều ứng dụng ngoài đồ họa, thuật ngữ GPGPU (tính toán đa dụng trên GPU) xuất hiện.

Các khái niệm đa lõi, MIC và GPGPU sẽ được khám phá chi tiết hơn trong các chương tiếp theo của tài liệu.

2.4 BASIC MEASURE OF COMPUTER PERFORMANCE

Hiệu suất máy tính:

- Khi đánh giá phần cứng bộ xử lý và đặt yêu cầu cho các hệ thống mới, hiệu suất là một yếu tố quan trọng, cùng với chi phí, kích thước, bảo mật, độ tin cậy và tiêu thụ điện năng.
- So sánh hiệu suất giữa các bộ xử lý rất phức tạp và không chỉ dựa vào tốc độ xung nhịp mà còn phụ thuộc vào bộ lệnh, ngôn ngữ lập trình, trình biên dịch và kỹ năng lập trình.

Tốc độ xung nhịp (Clock Speed):đơn vị Hz ví dụ nó đổi 50 lần thì sẽ = 50Hz

- Tốc độ xử lý của bộ xử lý được quyết định bởi tần số xung của hệ thống, đo bằng Hertz (Hz), với mỗi nhịp xung gọi là một chu kỳ xung.
- Tốc độ xung (clock rate) phải phù hợp với cấu trúc vật lý của bộ xử lý, vì tín hiệu cần thời gian di chuyển giữa các phần tử trong bộ xử lý. Điều này làm cho các hoạt động cần được đồng bộ hóa và thực hiện trong nhiều chu kỳ xung khác nhau.

Tốc độ thực thi lệnh (Instruction Execution Rate):

- Bộ xử lý hoạt động với tần số xung cố định và thời gian chu kỳ, với chỉ số quan trọng là **CPI (Cycles Per Instruction)**, là số chu kỳ cần thiết để hoàn thành một lệnh.
- CPI phụ thuộc vào từng loại lệnh (load, store, branch, v.v.). Tổng CPI được tính dựa trên trung bình số chu kỳ cho mỗi loại lệnh.
- Công thức tính thời gian thực thi chương trình:

$$CPI = \frac{\sum_{i=1}^{n}(CPI_i \times I_i)}{I_c}$$

Trong đó:
- **Ic**: số lượng lệnh trong chương trình mà bộ xử lí thực hiện
- **CPI**: chu kỳ trung bình mỗi lệnh
- **t**: thời gian chu kỳ(tốn bao nhiêu thời gian để xử lí vấn đề đó)

- Công thức được cải thiện khi tính đến số chu kỳ cần thiết cho bộ nhớ:
  $T = Ic * [p + (m * k)] * t$
  Với:
  - **p**: số chu kỳ xử lý
  - **m**: số lần tham chiếu bộ nhớ
  - **k**: tỷ lệ giữa thời gian chu kỳ bộ nhớ và bộ xử lý

Đo lường hiệu suất MIPS:

- **MIPS (Millions of Instructions Per Second)**: là thước đo hiệu suất phổ biến, tính theo tốc độ xung và CPI:
  $MIPSrate = \frac{f}{CPI * 10^6}$
- Ví dụ 2.2: Một chương trình chạy trên bộ xử lý 400 MHz, với 4 loại lệnh chính và các giá trị CPI khác nhau, có thể đạt được tốc độ MIPS khoảng **178**.

Hiệu suất MFLOPS:

- **MFLOPS (Millions of Floating-Point Operations Per Second)**: là thước đo hiệu suất cho các lệnh dấu phẩy động, phổ biến trong các ứng dụng khoa học và trò chơi.

Chapter 4

4.1 Computer Memory System Overview

Cache Memory Design Elements (Chapter 4.3):

1. **Cache Basics**:
   - **Cache memory** is a smaller, faster memory that stores copies of data from the most frequently accessed main memory locations.
   - Its main goal is to improve data retrieval speed for the CPU by reducing the time spent accessing the slower main memory.
2. **Cache Mapping Techniques**:
   - **Direct Mapping**: Each block in the main memory is mapped to a specific line in the cache. It's simple but can lead to frequent conflicts (called **thrashing**) when multiple memory blocks compete for the same cache line.
   - **Associative Mapping**: A memory block can be stored in any cache line. This flexibility reduces conflicts but increases the complexity of finding where data is stored in the cache.
   - **Set-Associative Mapping**: A combination of the two methods above, where the cache is divided into sets, and each memory block maps to a set, but within that set, it can be stored in any line. This balances flexibility and simplicity.
3. **Cache Size and Block Size**:
   - Cache size is a crucial factor in performance. Larger caches hold more data but may be slower due to more complex circuits.
   - **Block size** refers to the amount of data fetched from memory at once. A larger block size means fewer fetches, but this can also cause more data to be fetched than needed, wasting space.
4. **Replacement Policies**:
   - When the cache is full, a block must be replaced. Common replacement strategies include:
     - **Least Recently Used (LRU)**: Replaces the block that hasn't been accessed in the longest time.
     - **First In, First Out (FIFO)**: Replaces the oldest block in the cache.

        - **Random Replacement**: Replaces a random block, which can sometimes perform surprisingly well.
5. **Write Policies**:
    - **Write-Through**: Any write to the cache is also immediately written to main memory. It ensures data consistency but can be slow.
    - **Write-Back**: Data is written to main memory only when it's replaced in the cache, reducing memory write operations but requiring extra checks for data consistency.
6. **Cache Levels**:
    - Modern systems often use multiple levels of cache:
        - **L1 (Level 1)**: The smallest and fastest cache, closest to the CPU.
        - **L2 (Level 2)**: Larger and slower than L1, but still faster than main memory.
    - Some systems also have **L3** caches, which are even larger and shared between multiple CPU cores.
7. **Cache Performance**:
    - The document discusses how the **hit rate** (percentage of data requests the cache can serve) varies with cache size and mapping techniques.
    - **Set-Associative Caches** generally have higher hit rates than direct-mapped caches, with more flexible block storage.
    - There are diminishing returns in increasing cache size after a certain point, as larger caches provide smaller performance improvements.
8. **Cache Coherence** (for multi-processor systems):
    - In multi-core systems, each core might have its own cache. **Cache coherence** mechanisms ensure that changes in one cache are reflected in others to maintain data consistency.
    - Common strategies include **bus snooping** (where caches monitor memory operations) and **directory-based systems** (where a central directory tracks which caches store which data).

Chapter 5

5.1 SEMICONDUCTOR MAIN MEMORY (5.1 BỘ NHỚ CHÍNH BÁN DẪN)

Semiconductor Memory Overview:

- **Memory Cell**: The fundamental element of semiconductor memory. It has two stable states// **trạng thái ổn định//,** which represent binary 1 and 0. It has **terminals// có thiết bị đầu cuối//** for selecting, reading, and writing operations.

Types of RAM (Random Access Memory):

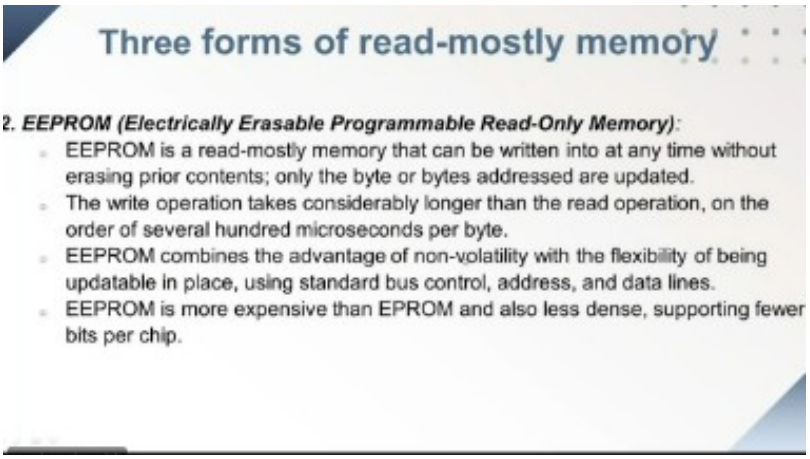
- **DRAM (Dynamic RAM)**:
    - Stores data as electrical charges on capacitors.// **điện tích trên tụ điện.//**
    - Requires periodic refreshing since capacitors discharge over time.
    - Used for storing a single bit, but behaves analogically by storing charge within a range, with a threshold for binary determination.
- **SRAM (Static RAM)**:
    - Uses flip-flop logic to store binary data and is a purely digital device.
    - Does not need refreshing and retains data as long as power is supplied.

SRAM vs DRAM

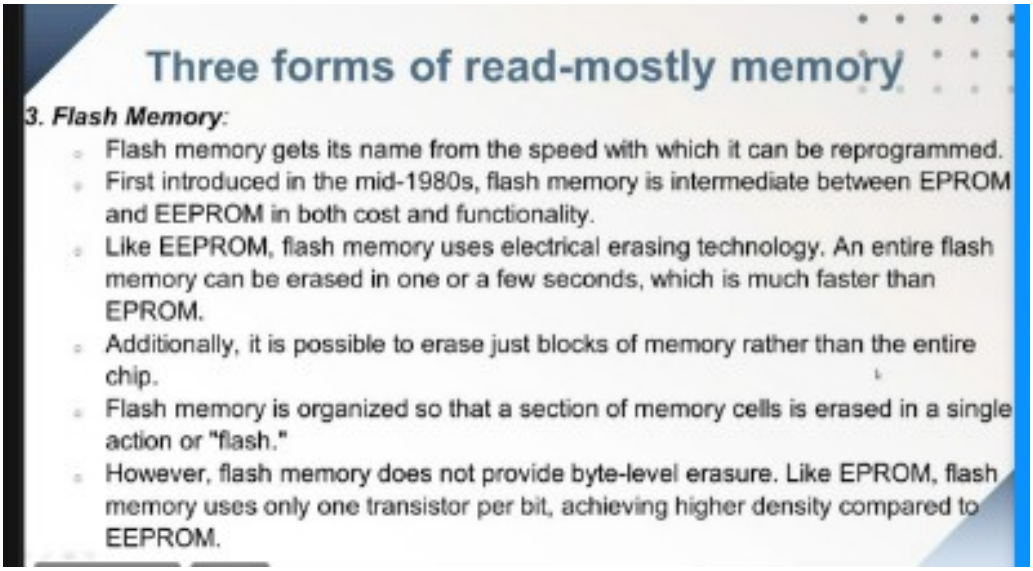
	Characteristic	Cost	Application
DRAM	Volatile	Less expensive	Large memory main memory
SRAM	Votalite	expensive	Cache memory

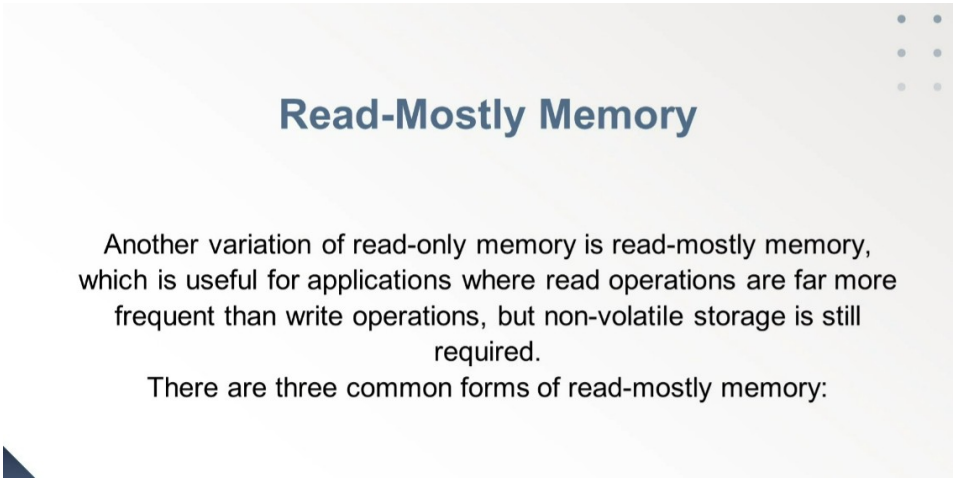
Types of Read-Only Memory (ROM):

- **ROM (Read-Only Memory)**: Data is written during manufacturing and cannot be altered**.// bị thay đổi/**
- **PROM (Programmable ROM)**: Can be written electrically once but not modified after.
- **EPROM (Erasable Programmable ROM)**: Can be erased using UV light and rewritten multiple times.
- **EEPROM (Electrically Erasable Programmable ROM)**: Can be erased electrically without needing UV light and modified at the byte level.



- **Flash Memory**: Similar to EEPROM but faster and more efficient for block-level erasure. Widely used for storage devices.





dạng biến dạng.

Chip Logic and Packaging:

- Semiconductor memory is organized in chips, with trade-offs between speed, density, and cost.
- DRAM chips require multiplexing to manage addressing and include refresh circuitry to maintain data.
- ROM and DRAM chips use specific pins for address lines, data, and control signals (e.g., chip enable, write enable).

5.2 Error Correction (English) nhóm 5

Error Types:

- **Hard Errors**: Permanent physical defects in memory cells, causing them to be stuck at 0 or 1 or switch erratically. Causes include environmental damage, manufacturing defects, and wear.
- **Soft Errors**: sự kiện Random, non-destructive errors that alter memory contents without damaging the memory, often caused by power issues or alpha particles from radioactive decay.

Error-Correcting Process:

- A function is applied to the data to generate a code, which is stored alongside the data.
- When data is read from memory, a new code is generated and compared to the stored code. Based on the comparison:

- **No error**: The data is sent as-is.
- **Correctable error**: Data is corrected and sent out.
- **Uncorrectable error**: The error is reported.

Error-Correcting Codes (ECC):

- These codes detect and correct errors. The Hamming code is a well-known method that can correct single-bit errors in a word.
- The concept is based on parity bits and syndrome words, where exclusive-OR (XOR) logic is used to identify and correct errors.
- **SEC (Single Error Correcting)** and **SEC-DED (Single Error Correcting, Double Error Detecting)** are two common codes. The SEC-DED method detects two errors and corrects one error, providing higher reliability with minimal additional memory overhead.

1. Types of Errors:

- **Hard Failures**:
  - Là lỗi vĩnh viễn do các yếu tố như lỗi sản xuất, môi trường tác động mạnh hoặc sự mài mòn theo thời gian. Những lỗi này khiến các ô nhớ bị "kẹt" ở giá trị 0 hoặc 1 hoặc thay đổi ngẫu nhiên giữa hai giá trị, không thể lưu trữ hoặc truy xuất dữ liệu đúng cách.
- **Soft Errors**:
  - Là lỗi tạm thời không gây phá hủy vật lý ô nhớ. Những lỗi này xảy ra do các sự kiện ngẫu nhiên như vấn đề nguồn điện hoặc sự tác động của các hạt alpha phát ra từ sự phân rã phóng xạ, rất phổ biến trong các vật liệu bán dẫn. Soft errors không làm hỏng phần cứng nhưng có thể thay đổi dữ liệu mà không gây thiệt hại vĩnh viễn.

2. ECC Operation – Error Detection and Correction:

- Khi dữ liệu được ghi vào bộ nhớ, hệ thống sẽ tạo ra một **mã lỗi (K bits)** dựa trên dữ liệu **(M bits)** và lưu trữ cả hai cùng nhau.
- Khi dữ liệu được đọc ra, một mã mới được tạo từ dữ liệu đã đọc và so sánh với mã được lưu trữ để phát hiện lỗi. Kết quả của quá trình này sẽ là một trong ba trường hợp sau:
  - **Không có lỗi**: Dữ liệu được xuất ra nguyên trạng.
  - **Lỗi có thể sửa chữa**: Nếu phát hiện lỗi trong phạm vi hệ thống có thể sửa chữa, dữ liệu sẽ được sửa và sau đó xuất ra.
  - **Lỗi không thể sửa chữa**: Nếu lỗi vượt quá khả năng sửa chữa của hệ thống, lỗi sẽ được báo cáo.

Công thức liên quan đến số lượng **bit kiểm tra (check bits)** trong mã Hamming thường xuất hiện khi chúng ta muốn xác định số lượng **bit kiểm tra (K)** cần thiết để bảo vệ **M bit dữ liệu** khỏi lỗi 1-bit.

Công thức tính số lượng bit kiểm tra:

Để sửa lỗi 1-bit trong một từ dữ liệu có M bit, ta cần thỏa mãn điều kiện sau:

$$2^K \geq M + K + 1$$

Trong đó:

- **M** là số bit dữ liệu (data bits).
- **K** là số bit kiểm tra (check bits).
- **1** là thêm vào để đảm bảo mã có thể phát hiện trạng thái không có lỗi.

3. Hamming Code:

- **Mã Hamming** là một phương pháp mã hóa giúp phát hiện và sửa lỗi 1-bit trong một từ dữ liệu.
- Ví dụ: Với một từ dữ liệu 4-bit, hệ thống thêm 3 **bit chẵn lẻ** vào để tạo thành một mã Hamming 7-bit. Các bit chẵn lẻ này được tính toán sao cho tổng số bit "1" trong các nhóm dữ liệu nhất định là chẵn.
- Khi một lỗi xảy ra, sự bất thường trong các nhóm chẵn lẻ sẽ chỉ ra vị trí của lỗi. Lỗi đó sau đó có thể được sửa chữa bằng cách đảo ngược giá trị của bit bị lỗi.

4. Syndrome Calculation (Tính toán hội chứng):

- **Syndrome word** (từ hội chứng) là kết quả của việc so sánh mã đã lưu trữ và mã được tạo ra khi đọc dữ liệu. Từ hội chứng giúp xác định xem có lỗi hay không và nếu có, bit nào bị lỗi.
- **Các kết quả của tính toán hội chứng**:
  - **Hội chứng = 0**: Không có lỗi.
  - **Hội chứng có một bit duy nhất bằng 1**: Lỗi nằm trong một bit kiểm tra (check bit) chứ không phải trong dữ liệu thực tế, không cần sửa.
  - **Hội chứng có nhiều bit bằng 1**: Điều này cho biết một bit dữ liệu đã bị lỗi, và hội chứng sẽ chỉ ra bit nào cần sửa. Bit đó sẽ được đảo ngược (từ 0 thành 1 hoặc ngược lại).

5. Check Bit Calculation (Tính toán các bit kiểm tra):

- Đối với một từ dữ liệu 8-bit, cần thêm 4 bit kiểm tra (C1, C2, C4, C8). Mỗi bit kiểm tra giám sát một nhóm bit dữ liệu cụ thể. Ví dụ:
  - **C1** kiểm tra các bit dữ liệu 1, 2, 4, 5, và 7.
  - **C2** kiểm tra các bit 1, 3, 4, 6, và 7.
  - **C4** kiểm tra các bit 2, 3, 4, 8.
  - **C8** kiểm tra các bit 5, 6, 7, và 8.
- Các bit kiểm tra này được tính bằng cách sử dụng phép XOR (exclusive-OR) trên các bit dữ liệu mà chúng giám sát. Nếu một lỗi xảy ra trong một bit dữ liệu, kết quả của phép XOR giữa các bit kiểm tra và dữ liệu sẽ cho ra một hội chứng không bằng 0, giúp xác định và sửa lỗi.

6. Memory Overhead (Chi phí bộ nhớ bổ sung):

- ECC yêu cầu thêm bộ nhớ để lưu trữ các bit kiểm tra ngoài dữ liệu thực tế. Điều này tạo ra một chi phí phụ, ví dụ:
  - **IBM 30xx systems**: Sử dụng **mã SEC-DED 8-bit** (Single Error Correction, Double Error Detection – sửa một lỗi và phát hiện hai lỗi) cho mỗi 64-bit dữ liệu, với khoảng 12% bộ nhớ bổ sung.
  - **Hệ thống VAX**: Sử dụng **mã SEC-DED 7-bit** cho mỗi 32-bit dữ liệu, với khoảng 22% bộ nhớ bổ sung.

7. SEC-DED Code (Single Error Correction, Double Error Detection):

- **Mã SEC-DED** cho phép sửa các lỗi 1-bit và phát hiện các lỗi 2-bit, nhưng không thể sửa lỗi 2-bit. Đây là loại mã phổ biến vì nó cung cấp sự cân bằng tốt giữa độ phức tạp và tính tin cậy.

Ví dụ Tính Toán Chi Tiết (Phần bổ sung):

- Giả sử một từ dữ liệu 8-bit là `00111001`, với các bit dữ liệu từ D1 đến D8.
  - **C1 = D1 ⊕ D2 ⊕ D4 ⊕ D5 ⊕ D7**
  - **C2 = D1 ⊕ D3 ⊕ D4 ⊕ D6 ⊕ D7**
  - **C4 = D2 ⊕ D3 ⊕ D4 ⊕ D8**
  - **C8 = D5 ⊕ D6 ⊕ D7 ⊕ D8**
- Kết quả tính toán:
  - **C1 = 1**, **C2 = 1**, **C4 = 1**, **C8 = 0**.

- Nếu một lỗi xảy ra, các bit kiểm tra sẽ phát hiện và chỉ ra vị trí lỗi bằng cách so sánh hội chứng với các giá trị mong đợi.

Kết Luận:

- ECC tăng cường tính tin cậy của bộ nhớ bằng cách phát hiện và sửa lỗi bit đơn lẻ, cũng như phát hiện lỗi nhiều bit. Việc sử dụng ECC sẽ tốn thêm khoảng 7%-22% bộ nhớ, nhưng là cần thiết để bảo vệ dữ liệu trong các hệ thống quan trọng như máy chủ hoặc các máy tính lớn.

-

5.3 DDR DRAM

The interface to main memory, which remains the DRAM chip, is one of the critical system bottlenecks in high-performance processors. To mitigate this, SRAM caches have been placed between the processor and DRAM, but they are costly and offer diminishing returns beyond a certain point. Thus, innovations such as SDRAM (Synchronous DRAM) and DDR-DRAM (Double Data Rate DRAM) have been developed.

Synchronous DRAM (SDRAM):

- SDRAM is synchronized with the system clock, allowing data exchange at the full speed of the processor/memory bus.
- It introduces burst mode to eliminate delays from address setup and precharging. This is especially useful when accessing data in sequence.
- SDRAM uses a multiple-bank internal architecture for on-chip parallelism, which enhances its performance.
- The mode register allows customization of the SDRAM's performance, including the burst length and CAS latency.

DDR SDRAM:

- DDR (Double Data Rate) SDRAM improves on SDRAM by synchronizing data transfer to both the rising and falling edges of the clock, doubling the data rate.
- DDR uses a prefetch buffer, starting with 2 bits in DDR1, then expanding to 4 bits in DDR2, 8 bits in DDR3, and adding bank groups in DDR4.
- The prefetch buffer allows data to be transferred more quickly by pre-loading bits into the buffer, enabling faster I/O.

Each subsequent generation of DDR (DDR2, DDR3, DDR4) increases the prefetch size and introduces new techniques like bank groups to improve data rates without significantly changing the DRAM core speed.

5.4 Flash Memory

Flash memory is a type of semiconductor memory used for both internal and external storage. It was first introduced in the mid-1980s and sits between EPROM and EEPROM in terms of cost and functionality. Flash memory uses electrical erasing technology, allowing it to erase data faster than EPROM. Unlike EPROM, which erases the entire chip, flash memory allows block-level erasure but not byte-level erasure.

Operation: Flash memory cells include a floating gate insulated by a thin oxide layer. In the initial state, the cell represents binary 1. By applying a high voltage, electrons tunnel into the floating gate, representing binary 0. The state of the cell is persistent, meaning the data remains even when power is disconnected, making it suitable for secondary storage.

NOR and NAND Flash Memory:

- **NOR Flash**: Organized with memory cells connected in parallel, allowing high-speed random access, making it possible to read, write, and erase individual bits.
- **NAND Flash**: Cells are connected in series in blocks of 16 or 32 transistors. Data must be accessed in blocks, but NAND offers higher bit density and faster write speeds than NOR.

NOR flash is more commonly used for internal memory in embedded systems, while NAND flash is ideal for external memory like USB drives, memory cards, and SSDs (discussed further in Chapter 6).

CHAPTER 6: EXTERNAL MEMORY.

6.1 Magnetic Disk

- Magnetic disks are a cornerstone of external memory in almost every computer system. Here's a more thorough exploration of the section:

- Đĩa từ là một thành phần quan trọng của bộ nhớ ngoài trong hầu hết các hệ thống máy tính. Dưới đây là phân tích chi tiết hơn về phần này:

Structure and Substrates

- A magnetic disk consists of a circular **platter** made from a non-magnetic material called the **substrate**, which is coated with a magnetizable layer.
- **Traditional substrates**: Aluminum or aluminum alloy.
- **Glass substrates** (newer): Offer several advantages over aluminum, including:
    - Improved uniformity of the magnetic film surface, which enhances disk reliability.
    - Reduced surface defects, lowering read-write errors.
    - The ability to support **lower fly heights**, which allows the read/write head to be closer to the disk surface, improving accuracy.
    - Greater stiffness, minimizing disk vibrations.
    - Better shock resistance.

Cấu trúc và Chất nền

- Đĩa từ bao gồm một **đĩa tròn** được làm từ vật liệu không từ tính gọi là **chất nền**, được phủ một lớp vật liệu từ hóa.
- **Chất nền truyền thống**: Nhôm hoặc hợp kim nhôm.
- **Chất nền thủy tinh** (mới hơn): Có nhiều ưu điểm so với nhôm, bao gồm:
    - Cải thiện độ đồng đều của bề mặt màng từ, tăng độ tin cậy của đĩa.
    - Giảm đáng kể các khuyết điểm bề mặt, giảm lỗi đọc-ghi.
    - Khả năng hỗ trợ **độ cao bay thấp hơn**, cho phép đầu đọc/ghi gần hơn với bề mặt đĩa, cải thiện độ chính xác.
    - Độ cứng tốt hơn, giảm thiểu rung động của đĩa.
    - Chịu va đập tốt hơn.

Magnetic Read and Write Mechanisms

- The **read/write head** interacts with the disk's surface during data operations.
- **Writing process**: Electricity flows through a coil in the write head, creating a magnetic field that magnetizes areas on the disk. Changing the current's polarity results in different magnetization patterns for 1s and 0s.
- **Reading process**: Magnetic fields on the disk induce an electric current in the read head as it moves across the disk surface. In older systems, the same head was used for both reading and writing.
- Modern systems use a **magnetoresistive (MR) head** for reading, which is more efficient than older methods because its resistance varies based on the magnetization direction of the medium, allowing faster and higher-density reads.

Cơ chế Đọc và Ghi Từ

- **Đầu đọc/ghi** tương tác với bề mặt đĩa trong quá trình hoạt động dữ liệu.
- **Quá trình ghi**: Dòng điện đi qua một cuộn dây trong đầu ghi, tạo ra từ trường từ hóa các vùng trên đĩa. Thay đổi cực tính của dòng điện tạo ra các mô hình từ hóa khác nhau cho 1s và 0s.
- **Quá trình đọc**: Từ trường trên đĩa tạo ra dòng điện trong đầu đọc khi nó di chuyển qua bề mặt đĩa. Trong các hệ thống cũ, cùng một đầu được sử dụng cho cả đọc và ghi.
- Các hệ thống hiện đại sử dụng **đầu cảm biến từ trở (MR)** cho việc đọc, hiệu quả hơn vì điện trở của nó thay đổi dựa trên hướng từ hóa của bề mặt, cho phép đọc nhanh hơn và với mật độ cao hơn.

Data Organization and Formatting

- Data is stored in concentric rings called **tracks**, each separated by **inter-track gaps** to reduce interference.
- Tracks are divided into **sectors**, each usually holding **512 bytes**. Modern systems use fixed-length sectors.
- Disks employ **constant angular velocity (CAV)**, where the disk spins at a constant speed, meaning bits near the disk's center travel slower than those near the edge.
    - **Multiple zone recording (MZR)** adjusts this by dividing the disk into zones with varying numbers of sectors per track, maximizing storage capacity.

Tổ chức Dữ liệu và Định dạng

- Dữ liệu được lưu trữ trong các vòng đồng tâm gọi là **rãnh**, mỗi rãnh được ngăn cách bằng **khoảng cách giữa các rãnh** để giảm nhiễu.
- Các rãnh được chia thành các **sector**, mỗi sector thường chứa **512 byte**. Các hệ thống hiện đại sử dụng các sector có độ dài cố định.
- Đĩa sử dụng **tốc độ góc không đổi (CAV)**, trong đó đĩa quay với tốc độ không đổi, nghĩa là các bit gần trung tâm di chuyển chậm hơn các bit ở rìa ngoài.
    - **Ghi nhiều vùng (MZR)** điều chỉnh điều này bằng cách chia đĩa thành các vùng với số lượng sector khác nhau mỗi rãnh, tối đa hóa dung lượng lưu trữ.

Physical Characteristics

- **Head movement**: Disks can have either fixed or movable heads. A **fixed-head disk** has one read/write head per track, while a **movable-head disk** uses one head per surface that moves between tracks.

- **Disk types**:
  - **Non-removable disks**: Permanently mounted (e.g., hard disks in computers).
  - **Removable disks**: Disks that can be removed and replaced (e.g., floppy disks, ZIP drives).
  - **Single vs. double-sided**: Most modern disks are **double-sided**, with data stored on both sides.
  - **Multiple platters**: Some disk drives stack multiple platters, each with its own read/write head, allowing more data to be stored.
- **Disk heads**:
  - Traditional disks have a **fixed air gap** between the head and the disk surface.
  - **Contact-based systems**, like **floppy disks**, have heads that physically touch the disk.
  - **Winchester disks**: A newer design with heads that hover extremely close to the disk surface, thanks to the air pressure from the spinning disk. This allows for greater data density.

Đặc điểm Vật lý

- **Di chuyển của đầu**: Đĩa có thể có đầu cố định hoặc di chuyển. Đĩa **đầu cố định** có một đầu đọc/ghi cho mỗi rãnh, trong khi đĩa **đầu di động** sử dụng một đầu cho mỗi bề mặt, có thể di chuyển giữa các rãnh.
- **Các loại đĩa**:
  - **Đĩa không tháo rời**: Được gắn vĩnh viễn (ví dụ: đĩa cứng trong máy tính).
  - **Đĩa tháo rời**: Có thể tháo ra và thay thế (ví dụ: đĩa mềm, đĩa ZIP).
  - **Một mặt và hai mặt**: Hầu hết các đĩa hiện đại là **hai mặt**, lưu trữ dữ liệu ở cả hai mặt.
  - **Nhiều đĩa**: Một số ổ đĩa xếp chồng nhiều đĩa, mỗi đĩa có đầu đọc/ghi riêng, cho phép lưu trữ nhiều dữ liệu hơn.
- **Cơ chế đầu đọc/ghi**:
  - Đĩa truyền thống có **khoảng cách không khí cố định** giữa đầu và bề mặt đĩa.
  - **Hệ thống tiếp xúc**, như **đĩa mềm**, có các đầu tiếp xúc vật lý với đĩa.
  - **Đĩa Winchester**: Một thiết kế mới hơn với các đầu bay rất gần bề mặt đĩa, nhờ áp suất không khí từ đĩa quay, cho phép mật độ dữ liệu cao hơn.

Performance Parameters

- **Seek time**: The time required to move the disk arm to the desired track.
- **Rotational delay**: The time for the desired sector to rotate under the read/write head.
- **Access time**: The sum of seek time and rotational delay.
- **Transfer time**: Time taken to actually read/write the data after reaching the right position.

Tham số Hiệu suất

- **Thời gian tìm kiếm**: Thời gian cần để di chuyển cánh tay đĩa đến rãnh mong muốn.
- **Trì hoãn quay**: Thời gian để sector mong muốn quay đến vị trí dưới đầu đọc/ghi.
- **Thời gian truy cập**: Tổng của thời gian tìm kiếm và trì hoãn quay.
- **Thời gian truyền**: Thời gian thực sự đọc/ghi dữ liệu sau khi đạt vị trí chính xác.

6.2 RAID (Redundant Array of Independent Disks)

- RAID systems address the performance limitations of individual disks by using multiple disks in parallel, achieving higher performance and reliability. Here's a more granular breakdown of RAID:

- Hệ thống RAID giải quyết các hạn chế về hiệu suất của đĩa đơn lẻ bằng cách sử dụng nhiều đĩa song song, giúp đạt được hiệu suất và độ tin cậy cao hơn. Dưới đây là phân tích chi tiết về RAID:

General Characteristics of RAID

RAID arrays distribute data across multiple disks and often include redundancy for reliability. The key features across RAID levels are:

1. The operating system views multiple physical disks as a single logical drive.
2. Data is **striped** across the disks to allow parallel operations.
3. **Redundancy** (in most RAID levels) ensures data recovery in case of a disk failure.

Đặc điểm chung của RAID

RAID phân phối dữ liệu trên nhiều đĩa và thường bao gồm tính năng dự phòng để đảm bảo độ tin cậy. Các tính năng chính của RAID bao gồm:

1. Hệ điều hành xem nhiều đĩa vật lý như một ổ đĩa logic duy nhất.
2. Dữ liệu được **phân dải** trên các đĩa để cho phép hoạt động song song.
3. **Dự phòng** (trong hầu hết các cấp RAID) đảm bảo khôi phục dữ liệu trong trường hợp đĩa bị hỏng.

RAID Levels

- **RAID 0 (Striping Only)**:
  - No redundancy; data is striped across disks for performance.
  - Improves data throughput by allowing simultaneous reads and writes from/to different disks.
  - **Risk**: If one disk fails, all data is lost. Tăng tốc độ đọc và ghi.
- **RAID 1 (Mirroring)**: 2 ổ cứng. Cấu hình mang tính dự phòng, copy y chang để tránh ở khác bị hư. Nếu gắn ổ mới thì dữ liệu được đồng hóa.
  - Data is duplicated (mirrored) across two disks, offering high reliability.
  - **Reads** can be performed from either disk, improving read performance.
  - **Writes** must be done on both disks simultaneously, which may limit write performance.
  - Ideal for critical data as recovery from a failure is instant (no need to rebuild).
- **RAID 2 (Hamming Code Error Correction)**: 2 3 4 làm tiền đề ra đời cho 5
  - Employs a Hamming code for error detection and correction.
  - Used for environments with frequent errors, but rarely implemented due to the high number of parity disks required.
- **RAID 3 (Bit-level Parity)**:
  - Data is split into small strips, and a **single parity disk** stores error-detection information.
  - If one disk fails, data can be reconstructed from the parity disk.
- **RAID 4 (Block-level Parity)**:
  - Data is stored in blocks, and parity is stored on a single disk.
  - Similar to RAID 3 but allows independent access to blocks (useful for high transaction environments).
  - The parity disk can become a bottleneck during write operations.
- **RAID 5 (Distributed Parity)**: 3 ổ đĩa, lưu chéo để đảm bảo hư ổ nào đó để ko bị mất dữ liệu.
  - Parity data is distributed across all disks, eliminating the bottleneck of a dedicated parity disk.
  - This level is popular for general-purpose use, balancing performance, capacity, and fault tolerance.
- **RAID 6 (Dual Distributed Parity)**: bổ sung thêm 2 parity nhưng hiệu suất không bằng

    - Adds a second parity block, allowing recovery even if two disks fail simultaneously.
    - More fault-tolerant than RAID 5 but with a higher overhead in write performance.

RAID 0 + 1:

RAID 1+-0: ngược lại

RAID 5+ 0: VỀ TÌM HIỂU(ÁP DỤNG NHƯ NHÀ HÀNG)

Các cấp RAID

- **RAID 0 (Chỉ phân dải)**:
    - Không có tính năng dự phòng; dữ liệu được phân dải trên các đĩa để cải thiện hiệu suất.
    - Tăng cường thông lượng dữ liệu bằng cách cho phép đọc và ghi đồng thời từ/to nhiều đĩa.
    - **Rủi ro**: Nếu một đĩa bị hỏng, toàn bộ dữ liệu bị mất.
- **RAID 1 (Nhân bản)**:
    - Dữ liệu được sao chép (nhân bản) trên hai đĩa, đảm bảo độ tin cậy cao.
    - **Đọc** có thể được thực hiện từ bất kỳ đĩa nào, cải thiện hiệu suất đọc.
    - **Ghi** phải được thực hiện trên cả hai đĩa cùng lúc, có thể hạn chế hiệu suất ghi.
    - Lý tưởng cho dữ liệu quan trọng vì khôi phục từ sự cố là ngay lập tức (không cần khôi phục lại).
- **RAID 2 (Sửa lỗi Hamming)**:
    - Sử dụng mã Hamming để phát hiện và sửa lỗi.
    - Sử dụng trong môi trường có nhiều lỗi, nhưng ít được triển khai do yêu cầu số lượng lớn đĩa dự phòng.
- **RAID 3 (Dải bit với Parity)**:
    - Dữ liệu được chia thành các dải nhỏ, và một **đĩa parity** lưu trữ thông tin phát hiện lỗi.
    - Nếu một đĩa bị hỏng, dữ liệu có thể được tái tạo từ đĩa parity.
- **RAID 4 (Parity mức khối)**:
    - Dữ liệu được lưu trữ trong các khối, và parity được lưu trên một đĩa.
    - Tương tự như RAID 3 nhưng cho phép truy cập độc lập vào các khối (hữu ích cho các môi trường giao dịch cao).
    - Đĩa parity có thể trở thành nút cổ chai trong các hoạt động ghi.
- **RAID 5 (Parity phân phối)**:
    - Parity được phân phối trên tất cả các đĩa, loại bỏ nút cổ chai của đĩa parity chuyên dụng.
    - Cấp độ này phổ biến cho việc sử dụng chung, cân bằng giữa hiệu suất, dung lượng và khả năng chịu lỗi.
- **RAID 6 (Parity phân phối kép)**:
    - Thêm một khối parity thứ hai, cho phép khôi phục ngay cả khi hai đĩa bị hỏng cùng lúc.
    - Chịu lỗi tốt hơn RAID 5 nhưng với chi phí hiệu suất ghi cao hơn.

6.3 Solid State Drives (SSD)

- SSDs have gained popularity due to their performance advantages over traditional magnetic hard drives (HDDs). They rely on **NAND flash memory** and are made entirely from electronic components with no moving parts.

- SSD ngày càng được sử dụng rộng rãi do những ưu điểm về hiệu suất so với ổ cứng từ (HDD). SSD dựa trên **bộ nhớ flash NAND** và hoàn toàn được làm từ các thành phần điện tử, không có bộ phận chuyển động.

Compared SSD with HDD

- As the cost of SSDs using NAND flash memory decreases and performance and bit density increase, SSDs are increasingly competing with HDDs. Table 6.5 shows popular comparison metrics between SSD and HDD at the present time.
- Khi giá thành của SSD sử dụng bộ nhớ flash NAND giảm và hiệu suất cũng như mật độ bit tăng lên, SSD ngày càng cạnh tranh với HDD. Bảng 6.5 cho thấy các chỉ số so sánh phổ biến giữa SSD và HDD tại thời điểm hiện tại.

Advantages of SSDs compared with HDD

- **High IOPS**: SSDs provide much higher input/output operations per second (IOPS), making them suitable for tasks that require rapid data access.
- **Durability**: SSDs are resistant to physical shocks and vibrations due to the absence of moving parts.
- **Power Efficiency**: They consume less power, resulting in lower operating costs and improved battery life in portable devices.
- **Faster Access Times**: SSDs provide much lower latency and faster data access compared to HDDs.
- **No Mechanical Wear**: SSDs do not suffer from the mechanical wear typical of spinning hard disks.

Ưu điểm của SSD so với HDD

- **Hiệu suất cao trong thao tác I/O trên giây (IOPS)**: SSD cung cấp số lượng thao tác nhập/xuất mỗi giây cao hơn nhiều so với HDD, giúp cải thiện đáng kể hiệu suất tổng thể của hệ thống.
- **Độ bền**: SSD không có các bộ phận chuyển động, nên ít bị tổn thương trước các tác động vật lý như sốc hoặc rung động.
- **Tuổi thọ dài hơn**: SSD không chịu sự hao mòn cơ học như HDD, điều này làm tăng tuổi thọ của SSD.
- **Tiêu thụ năng lượng thấp**: SSD tiêu thụ ít năng lượng hơn nhiều so với HDD có cùng kích thước, điều này đặc biệt hữu ích cho các thiết bị di động.
- **Hoạt động êm ái và ít tỏa nhiệt**: SSD không có các bộ phận quay nên vận hành êm hơn và tỏa nhiệt ít hơn, tiết kiệm không gian và năng lượng.
- **Thời gian truy cập và độ trễ thấp**: SSD có thời gian truy cập nhanh hơn HDD hơn 10 lần, giúp xử lý dữ liệu nhanh hơn rất nhiều.

Mặc dù HDD vẫn có ưu thế về **chi phí mỗi bit** và **dung lượng**, nhưng sự chênh lệch này đang dần được thu hẹp.

Structure of SSD

- SSD includes the following main components:
  - Controller: Controls interactions between components within the SSD and provides firmware management capabilities.
  - Cache: High-speed cache helps speed up data transfer and speed matching between flash memory and the host system.
  - Error correction: A system that detects and corrects errors to ensure data is not corrupted during read/write operations.
  - Flash memory chips: These are the main data storage chips of the SSD, using NAND flash technology.

Cấu trúc của SSD

SSD bao gồm các thành phần chính sau:

- **Bộ điều khiển**: Điều khiển các tương tác giữa các bộ phận trong SSD và cung cấp khả năng quản lý firmware.
- **Bộ nhớ đệm (cache)**: Bộ nhớ đệm có tốc độ cao giúp tăng tốc độ truyền dữ liệu và khớp tốc độ giữa bộ nhớ flash và hệ thống chủ.
- **Chỉnh sửa lỗi (error correction)**: Một hệ thống phát hiện và sửa lỗi để đảm bảo dữ liệu không bị hỏng trong quá trình đọc/ghi.
- **Các chip bộ nhớ flash**: Đây là các chip lưu trữ dữ liệu chính của SSD, sử dụng công nghệ NAND flash.

Challenges with SSDs

- **Performance Degradation**: Over time, SSD performance may degrade due to **fragmentation**. SSDs write data in **pages** (typically 4 KB), but pages are grouped into larger **blocks** (often 512 KB). Modifying a page requires erasing and rewriting the entire block, which can slow down write operations as the SSD fills up.
    - **TRIM** commands and **over-provisioning** (allocating extra space) help mitigate this issue by maintaining SSD efficiency.
- **Write Endurance**: Flash memory cells wear out after a certain number of writes (typically around 100,000 cycles).

    - Techniques like **wear leveling** and **bad block management** are used to distribute writes evenly across the SSD and extend its lifespan.

Vấn đề thực tế của SSD

SSD gặp hai vấn đề chính:

1. **Giảm hiệu suất theo thời gian**: SSD có xu hướng giảm hiệu suất khi dung lượng trống giảm dần. Lý do là dữ liệu trong SSD được lưu trữ dưới dạng **các trang** (page) và các trang này được lưu trong **các khối** (block). Để ghi đè một trang, toàn bộ khối phải được xóa và viết lại, điều này làm chậm quá trình ghi dữ liệu khi ổ đĩa bị phân mảnh.
    - Các kỹ thuật như **TRIM** và **over-provisioning** giúp duy trì hiệu suất bằng cách xóa trước các trang không cần thiết trong thời gian nhàn rỗi của SSD và dự trữ không gian trống cho các hoạt động ghi.
2. **Giới hạn về số lần ghi**: Bộ nhớ flash của SSD có giới hạn về số lần ghi trước khi các ô nhớ bị hao mòn. Sau một số lần ghi nhất định (khoảng 100.000 lần), các ô nhớ sẽ mất khả năng lưu trữ dữ liệu.
    - Các kỹ thuật **phân phối ghi đều** (wear leveling) giúp phân phối đồng đều các thao tác ghi lên
    - bộ ổ đĩa để kéo dài tuổi thọ SSD.

CHAPTER 11: DIGITAL LOGIC

Section 11.1: Boolean Algebra

Boolean algebra is the mathematical foundation of digital logic. It was first introduced by George Boole in 1854, and later applied to digital circuit design by Claude Shannon in 1938. Boolean algebra is essential for both the **analysis** and **design** of digital circuits:

- **Analysis**: Describes circuit functions in a concise form.
- **Design**: Simplifies the implementation of circuit functions.

Mục 11.1: Đại Số Boole

Đại số Boole là nền tảng toán học cho thiết kế logic số. Nó được giới thiệu bởi George Boole vào năm 1854 và được Claude Shannon áp dụng vào thiết kế mạch điện tử vào năm 1938. Đại số Boole cần thiết cho cả hai việc **phân tích** và **thiết kế** mạch số:

- **Phân tích**: Miêu tả chức năng của mạch một cách ngắn gọn.
- **Thiết kế**: Giúp đơn giản hóa việc thực hiện các chức năng mạch

Key Concepts of Boolean Algebra:

- **Variables**: Represent binary values (0 for FALSE and 1 for TRUE).
- **Basic Operations**:
  - **AND** (· or simply concatenation): Produces a TRUE output if both inputs are TRUE.
  - **OR** (+): Produces a TRUE output if at least one input is TRUE.
  - **NOT** (¯A or A'): Inverts the value.

Các khái niệm chính của Đại số Boole:

- **Biến**: Biểu thị các giá trị nhị phân (0 cho FALSE và 1 cho TRUE).
- **Các phép toán cơ bản**:
  - **AND** (· hoặc nối tiếp): Cho kết quả TRUE nếu cả hai đầu vào đều TRUE.
  - **OR** (+): Cho kết quả TRUE nếu ít nhất một đầu vào là TRUE.
  - **NOT** (`¯A` hoặc `A'`): Đảo ngược giá trị.

Additional Logical Operations:

- **XOR** (exclusive OR): Produces a TRUE output if exactly one of the inputs is TRUE.
- **NAND**: Produces the opposite of the AND operation.
- **NOR**: Produces the opposite of the OR operation.

Các phép toán logic bổ sung:

- **XOR** (hoặc loại trừ): Cho kết quả TRUE nếu chỉ có một đầu vào TRUE.
- **NAND**: Cho kết quả ngược với phép AND.
- **NOR**: Cho kết quả ngược với phép OR.

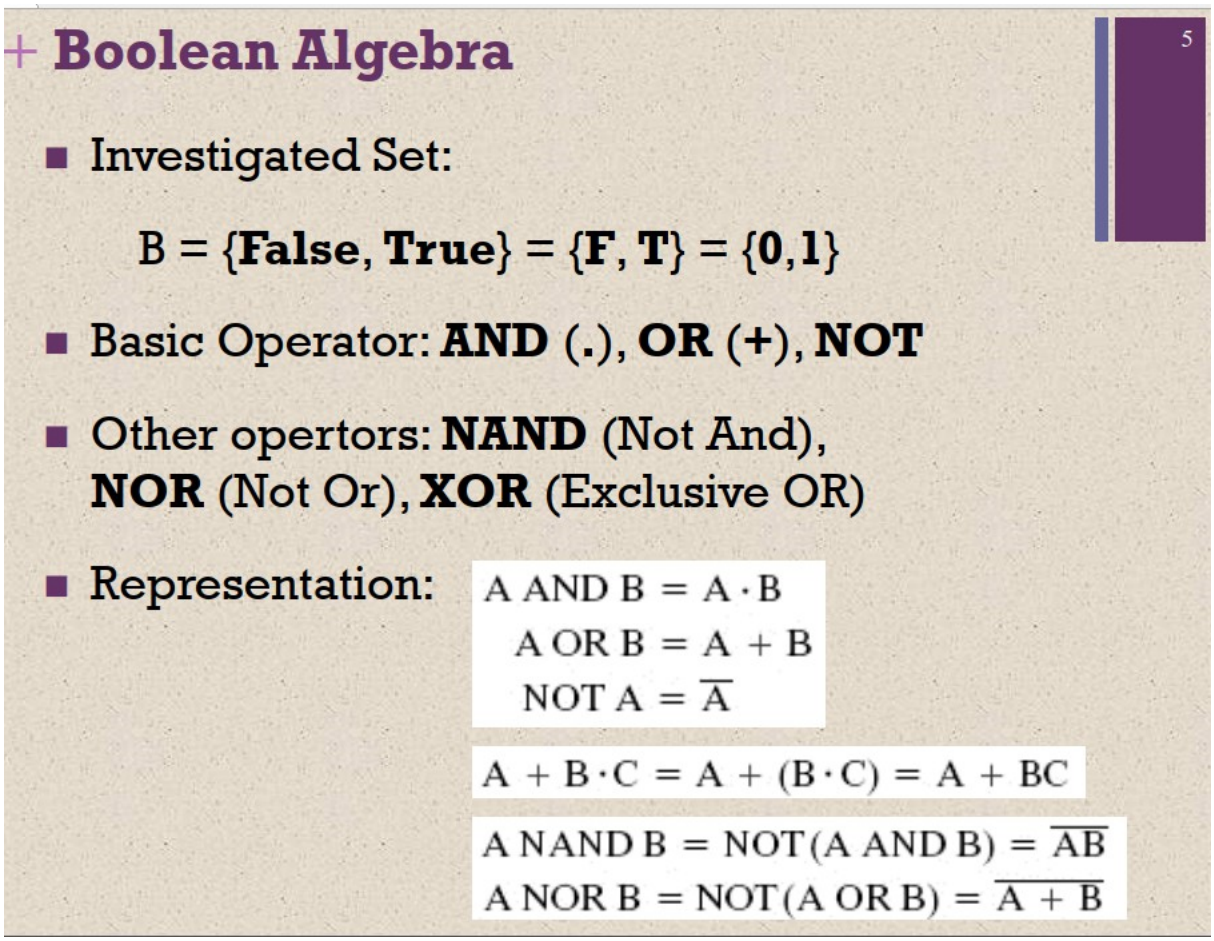
Key Identities in Boolean Algebra:

- **Commutative Laws**: The order of variables does not affect the result. For AND, A·B=B·AA \cdot B = B \cdot AA·B=B·A, and for OR, A+B=B+AA + B = B + AA+B=B+A.
- **Distributive Laws**: AND distributes over OR, and OR distributes over AND. A+(B·C)=(A+B)·(A+C)A + (B \cdot C) = (A + B) \cdot (A + C)A+(B·C)=(A+B)·(A+C).

- **DeMorgan's Theorems**: Useful for transforming expressions. For instance, A·B=A¯+B¯A \cdot B = \overline{\overline{A} + \overline{B}}A·B=A+B.

Các định lý quan trọng trong Đại số Boole:

- **Luật giao hoán:** Thứ tự của các biến không ảnh hưởng đến kết quả. Đối với phép AND, $A \cdot B = B \cdot A$, và đối với phép OR, $A + B = B + A$.
- **Luật phân phối:** AND phân phối qua OR và ngược lại. $A + (B \cdot C) = (A + B) \cdot (A + C)$.
- **Định lý DeMorgan:** Hữu ích để biến đổi biểu thức. Ví dụ, $\overline{A \cdot B} = \overline{A} + \overline{B}$.



Section 11.2: Gates

Gates are fundamental components used in digital circuits to perform Boolean operations. There are six basic types of gates, each with its own unique truth table and behavior:

1. **AND Gate**: Produces a TRUE output when all inputs are TRUE.
2. **OR Gate**: Produces a TRUE output when at least one input is TRUE.
3. **NOT Gate**: Inverts the input (produces the opposite value).
4. **NAND Gate**: Produces the opposite of an AND gate.
5. **NOR Gate**: Produces the opposite of an OR gate.

6. **XOR Gate**: Produces TRUE if only one input is TRUE.

Mục 11.2: Cổng Logic

Cổng logic là các thành phần cơ bản được sử dụng trong các mạch số để thực hiện các phép toán Boole. Có sáu loại cổng cơ bản, mỗi loại có bảng chân lý và hành vi riêng:

1. **Cổng AND**: Cho kết quả TRUE khi tất cả các đầu vào đều TRUE.
2. **Cổng OR**: Cho kết quả TRUE khi ít nhất một đầu vào là TRUE.
3. **Cổng NOT**: Đảo ngược đầu vào (cho giá trị ngược lại).
4. **Cổng NAND**: Cho kết quả ngược với cổng AND.
5. **Cổng NOR**: Cho kết quả ngược với cổng OR.
6. **Cổng XOR**: Cho kết quả TRUE nếu chỉ có một đầu vào TRUE.

Table 11.1 Boolean Operators

(a) Boolean Operators of Two Input Variables

P	Q	NOT P ($\overline{P}$)	P AND Q ($P \cdot Q$)	P OR Q ($P + Q$)	P NAND Q ($\overline{P \cdot Q}$)	P NOR Q ($\overline{P + Q}$)	P XOR Q ($P \oplus Q$)
0	0	1	0	0	1	1	0
0	1	1	0	1	1	0	1
1	0	0	0	1	1	0	1
1	1	0	1	1	0	0	0

(b) Boolean Operators Extended to More than Two Inputs (A, B, . . .)

Operation	Expression	Output = 1 if
AND	$A \cdot B \cdot \ldots$	All of the set {A, B, ...} are 1.
OR	$A + B + \ldots$	Any of the set {A, B, ...} are 1.
NAND	$\overline{A \cdot B \cdot \ldots}$	Any of the set {A, B, ...} are 0.
NOR	$\overline{A + B + \ldots}$	All of the set {A, B, ...} are 0.
XOR	$A \oplus B \oplus \ldots$	The set {A, B, ...} contains an odd number of ones.

Exercise (p.433)

Draw a logic diagram for following expressions:

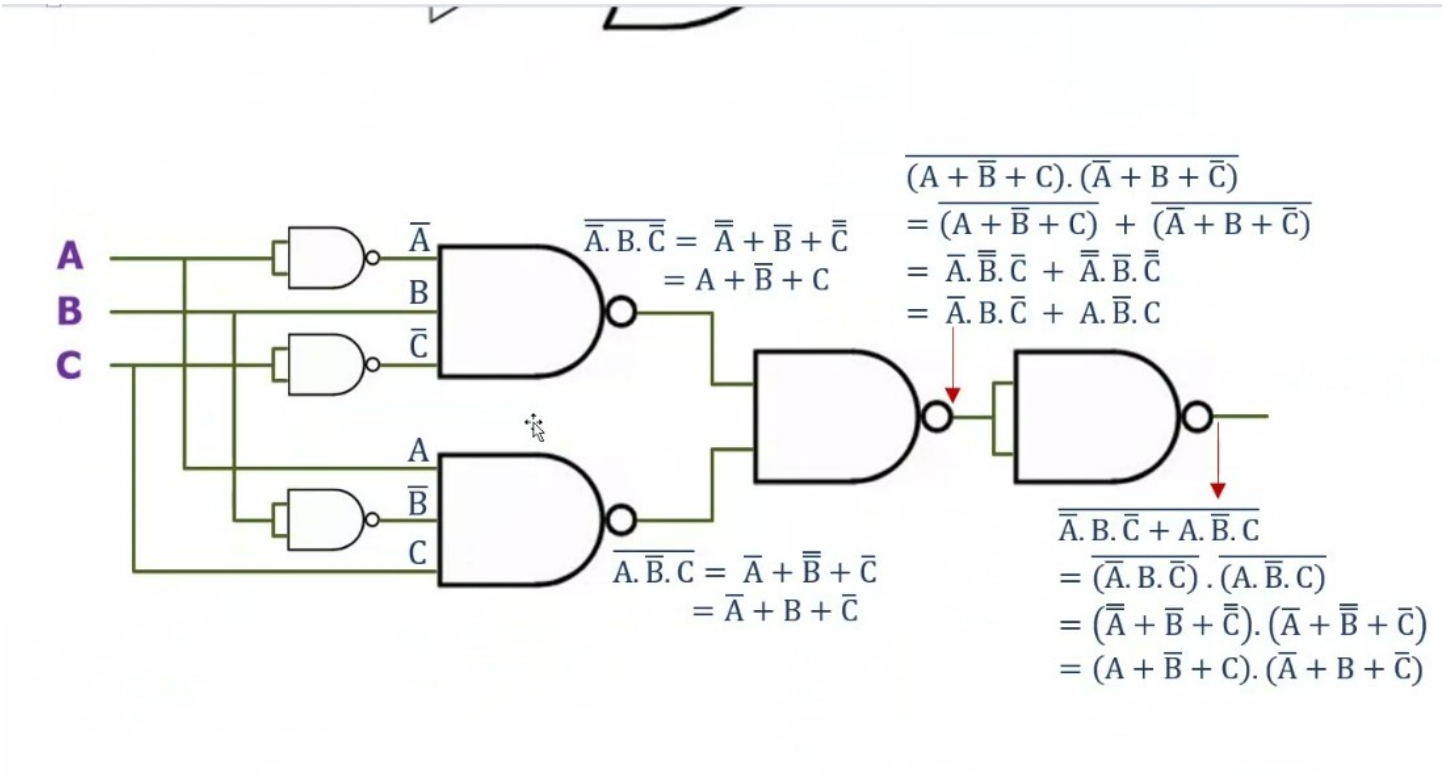
a) $(A + \overline{B} + C).(\overline{A} + B + \overline{C})$

b) $\overline{A} + ((B + C).(\overline{B} + \overline{C}))$

c) $(A.B) + (\overline{A}.\overline{B}) + (A.\overline{C})$

Draw a logic diagram for following expressions uses NAND gate only.

Draw a logic diagram for following expressions uses NOR gate only.



Functional Completeness of Gates:

- **AND, OR, and NOT**: These three gates can implement any Boolean function.

- **NAND and NOR**: Either of these gates alone can be used to build all other gates, making them functionally complete. This simplifies circuit design, as it allows the entire circuit to be built using only one type of gate.

Tính chất hoàn chỉnh của các cổng:

- **AND, OR, NOT**: Ba cổng này có thể thực hiện mọi hàm Boole.
- **NAND và NOR**: Một trong hai cổng này có thể được sử dụng để xây dựng tất cả các cổng khác, làm đơn giản hóa thiết kế mạch.

Section 11.3: Combinational Circuits

A **combinational circuit** is a set of interconnected gates where the output at any given time depends only on the current inputs. Combinational circuits do not have memory, meaning they do not store previous states, and the output changes immediately when the inputs change, subject only to gate delays.

Mục 11.3: Mạch Tổ Hợp

Mạch tổ hợp là một tập hợp các cổng được kết nối với nhau, trong đó đầu ra tại bất kỳ thời điểm nào chỉ phụ thuộc vào đầu vào tại thời điểm đó. Mạch tổ hợp không có bộ nhớ, nghĩa là chúng không lưu trữ các trạng thái trước đó và kết quả đầu ra thay đổi ngay lập tức khi có sự thay đổi ở đầu vào, chỉ bị trì hoãn do thời gian trễ của các cổng logic.

Key Components of Combinational Circuits:

- **Mối Quan Hệ Đầu Vào/Đầu Ra:** Đầu ra là hàm của đầu vào tại thời điểm cụ thể.
- Được Định Nghĩa Theo Ba Cách:

- **Truth Table**: Lists all possible combinations of input values and specifies the corresponding output for each combination. For example, if a circuit has `n` inputs, the truth table will have 2n2^n2n rows, each corresponding to a different combination of input values.
- **Graphical Symbols**: Represents the structure and interconnection of gates in a circuit diagram.
- **Boolean Equations**: Describes the relationship between inputs and outputs using Boolean expressions.

Các thành phần chính của mạch tổ hợp:

- **Bảng chân lý**: Liệt kê tất cả các tổ hợp giá trị đầu vào có thể và chỉ ra giá trị đầu ra tương ứng cho mỗi tổ hợp. Ví dụ, nếu một mạch có n đầu vào, bảng chân lý sẽ có 2n2^n2n hàng, mỗi hàng tương ứng với một tổ hợp của các giá trị đầu vào.
- **Biểu tượng đồ họa**: Biểu diễn cấu trúc kết nối của các cổng logic trong sơ đồ mạch.
- **Phương trình Boole**: Diễn tả mối quan hệ giữa các đầu vào và đầu ra của mạch thông qua các hàm Boole.

Ví Dụ Về Hàm Boolean

Khi làm việc với các hàm Boolean có ba biến, hai dạng phổ biến là:

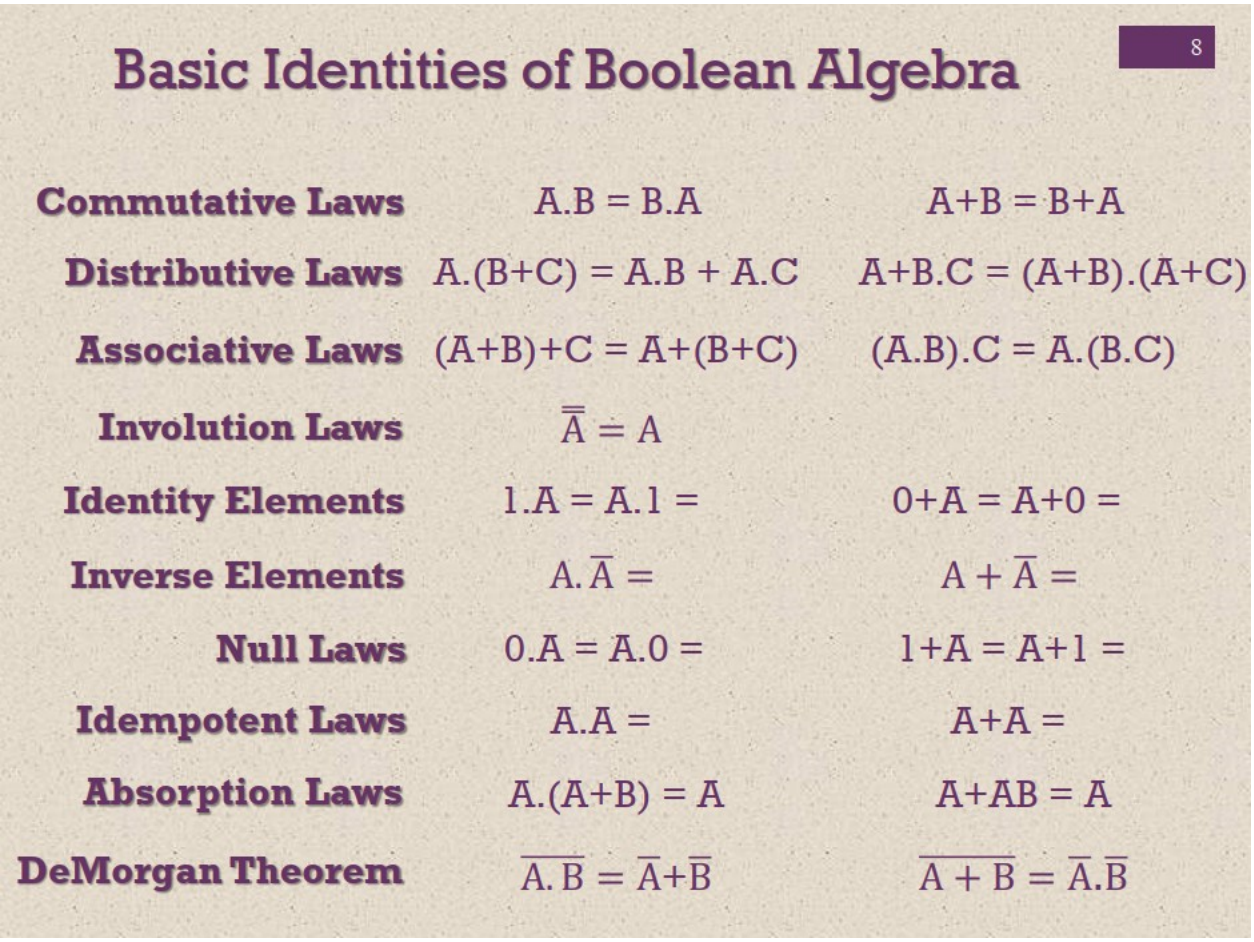
1. **Tổng của Các Tích (SOP):** Biểu thức này đại diện cho tổng (hoặc logic) của nhiều tích (và logic).
2. **Tích của Các Tổng (POS):** Biểu thức này là tích (và logic) của nhiều tổng (hoặc logic).

Design of Boolean Functions: Every Boolean function can be implemented as a network of gates. There are multiple ways to realize a given Boolean function, but the two most common forms are:

- **Sum of Products (SOP)**: Expresses the Boolean function as a sum of ANDed terms. For example: $F = A \cdot B + \overline{A} \cdot C$.
- **Product of Sums (POS)**: Expresses the Boolean function as a product of ORed terms. For example: $F = (A + B) \cdot (\overline{A} + C)$.

Thiết kế các hàm Boole: Mọi hàm Boole đều có thể được thực hiện dưới dạng một mạng các cổng logic. Có nhiều cách hiện thực hóa khác nhau cho một hàm Boole cụ thể, nhưng thường có hai dạng chính:

- **Tổng các tích (Sum of Products - SOP)**: Biểu diễn hàm Boole dưới dạng tổng các tích của các biến đầu vào. Ví dụ: $F = A \cdot B + \overline{A} \cdot C$.
- **Tích của tổng (Product of Sums - POS)**: Biểu diễn hàm Boole dưới dạng tích của các tổng. Ví dụ: $F = (A + B) \cdot (\overline{A} + C)$.



Simplification of Boolean Functions: To reduce the number of gates required in a combinational circuit, Boolean expressions must be simplified. Three main methods for simplification are:

1. **Algebraic Simplification**: Uses Boolean identities and rules to transform and reduce expressions.
2. **Karnaugh Maps**: A visual tool used to simplify Boolean expressions when the number of variables is small (up to 4 variables). Each square in the Karnaugh map corresponds to a combination of input values, and adjacent squares containing 1s can be grouped together to simplify the Boolean function.

- Example: A Karnaugh map with two adjacent squares containing 1s simplifies $F = A \cdot B + A \cdot \overline{B}$ to $F = A$.

Đơn giản hóa các hàm Boole: Để giảm số lượng cổng cần thiết trong một mạch tổ hợp, ta cần đơn giản hóa các biểu thức Boole. Có ba phương pháp đơn giản hóa chính:

1. **Đơn giản hóa đại số**: Sử dụng các định lý và quy tắc trong đại số Boole để biến đổi và rút gọn các biểu thức.

Các Phương Pháp Đơn Giản Hóa

1. **Bản Đồ Karnaugh (K-map):** Một phương pháp trực quan để đơn giản hóa các biểu thức Boolean.
2. **Phương Pháp Quine-McCluskey:** Một phương pháp bảng để đơn giản hóa có thể xử lý nhiều biến hơn so với K-map.

2. **Bản đồ Karnaugh**: Là một phương pháp trực quan dùng để biểu diễn và đơn giản hóa hàm Boole khi số biến ít (tối đa 4 biến). Các ô trong bản đồ Karnaugh tương ứng với các tổ hợp giá trị đầu vào, và các nhóm ô có chứa giá trị 1 có thể được kết hợp để rút gọn biểu thức Boole.

Bản Đồ Karnaugh

Bản đồ Karnaugh cung cấp một cách thuận tiện để trực quan hóa và tối thiểu hóa các hàm Boolean cho tối đa bốn biến. Nó là một mảng các ô vuông, với mỗi ô đại diện cho một tổ hợp trạng thái biến (0 hoặc 1).

Tạo K-map: Đối với một hàm có nnn biến, K-map có 2n2^n2n ô vuông, mỗi ô tương ứng với một tổ hợp duy nhất của các biến.

a) $A(x,y) = x.y + x.\bar{y}$

00 $(\bar{x}.\bar{y})$	01 $(\bar{x}.y)$	11 $(x.y)$	10 $(x.\bar{y})$
		1	1

$\rightarrow A = x$

b) $B(x,y) = x.y + \bar{x}.y + \bar{x}.\bar{y}$

00 $(\bar{x}.\bar{y})$	01 $(\bar{x}.y)$	11 $(x.y)$	10 $(x.\bar{y})$
1	1	1	

$\rightarrow B = \bar{x} + y$

Các Bài Tập Ví Dụ

1. **Xây Dựng Bảng Karnaugh:**
   - Đối với các hàm như A(x,y)=x·y+x·y¯A(x, y) = x \cdot y + x \cdot \overline{y}A(x,y)=x·y+x·y, và các biểu thức tương tự.
2. **Nhiệm Vụ Tối Thiểu Hóa:**
   - Đơn giản hóa các hàm Boolean bằng cách sử dụng kỹ thuật K-map

- Ví dụ: Một bản đồ Karnaugh với hai ô liền kề chứa giá trị 1 sẽ đơn giản hóa $F = A \cdot B + A \cdot \overline{B}$ thành $F = A$.

3. **Quine–McCluskey Tables**: Used when the number of variables is larger than 4, as Karnaugh maps become more complex.

Bảng Quine–McCluskey: Được sử dụng khi số biến lớn hơn 4, do bản đồ Karnaugh trở nên phức tạp.

Phương Pháp Quine-McCluskey

Phương pháp này liên quan đến một cách tiếp cận có hệ thống để tối thiểu hóa các hàm Boolean, đặc biệt hữu ích cho hơn bốn biến.

- **Giai Đoạn Đầu Tiên:** Tạo một bảng để liệt kê các minterms.
- **Giai Đoạn Cuối:** Kết hợp các thuật ngữ để phát triển một biểu thức đã được tối thiểu hóa.

Examples of Combinational Circuits:

- Combinational circuits play a crucial role in various digital systems, where the output depends solely on the current inputs. Here are a few common examples of combinational circuits:
- Mạch tổ hợp đóng vai trò quan trọng trong các hệ thống kỹ thuật số, nơi đầu ra chỉ phụ thuộc vào các đầu vào hiện tại. Dưới đây là một số ví dụ phổ biến về mạch tổ hợp:

1. **Half Adder**

- A **half adder** is a combinational circuit that performs the addition of two single-bit binary numbers. It has two inputs (A and B) and two outputs:
  - **Sum (S):** The XOR operation between A and B.
  - **Carry (C):** The AND operation between A and B.
  - **Functionality:**
    - Inputs: A, B
    - Outputs: $S = A \oplus B$, $C = A \wedge B$
  - Use Case: It is the basic building block for more complex adders.

1. Bộ Cộng Một Nửa (Half Adder)

- **Half adder** là một mạch tổ hợp thực hiện phép cộng của hai số nhị phân một bit. Nó có hai đầu vào (A và B) và hai đầu ra:
  - **Sum (S):** Phép XOR giữa A và B.
  - **Carry (C):** Phép AND giữa A và B.
  - **Chức Năng:**
    - Đầu vào: A, B
    - Đầu ra: $S = A \oplus B$, $C = A \wedge B$
  - Ứng Dụng: Đây là khối cơ bản để xây dựng các bộ cộng phức tạp hơn.

2. **Full Adder**

- A **full adder** is a step up from the half adder. It adds three binary bits (A, B, and a carry-in, CinC_{in}Cin) and produces a sum and a carry-out:
  - **Sum (S):** XOR operation between A, B, and CinC_{in}Cin.
  - **Carry-out (C):** The OR of multiple AND operations.
  - **Functionality:**
    - Inputs: A, B, CinC_{in}Cin
    - Outputs: $S = A \oplus B \oplus$ CinC_{in}Cin, Cout=(A∧B)∨(A∧Cin)∨(B∧Cin)C_{out} = (A \ land B) \lor (A \land C_{in}) \lor (B \land C_{in})Cout=(A∧B)∨(A∧Cin)∨(B∧Cin)
  - Use Case: Used in multi-bit binary addition.

2. Bộ Cộng Đầy Đủ (Full Adder)

- **Full adder** là một bước phát triển từ half adder. Nó cộng ba bit nhị phân (A, B và carry-in, CinC_{in}Cin) và tạo ra tổng và carry-out:
  - **Sum (S):** Phép XOR giữa A, B và CinC_{in}Cin.
  - **Carry-out (C):** Phép OR của nhiều phép AND.
  - **Chức Năng:**
    - Inputs: A, B, C_{in}
    - Outputs: S = A ⊕ B ⊕ C_{in}, $C_{out} = (A \wedge B) \vee (A \wedge C_{in}) \vee (B \wedge C_{in})$
  - Ứng Dụng: Dùng trong phép cộng nhị phân nhiều bit.

3. **Multiplexer (MUX)**

- A **multiplexer** selects one of the multiple input data lines and forwards it to a single output line. The selection is controlled by select lines.
  - **Functionality:**
    - Inputs: Multiple data inputs, select lines
    - Output: One data output based on the select line's value
  - Use Case: Multiplexers are widely used in communication systems for selecting data signals.

4. **Demultiplexer (DEMUX)**

- A **demultiplexer** takes a single input and routes it to one of many outputs, based on select lines.
  - **Functionality:**
    - Input: Single input line, select lines
    - Outputs: One of many output lines is active based on the select lines.
  - Use Case: Used in applications where one data signal needs to be distributed to multiple outputs.

4. Bộ Phân Chia Tín Hiệu (Demultiplexer - DEMUX)

- **Demultiplexer** lấy một tín hiệu đầu vào và chuyển nó tới một trong nhiều đầu ra, dựa trên các đường chọn.
  - **Chức Năng:**
    - Đầu vào: Một đường vào, các đường chọn
    - Đầu ra: Một trong các đường ra được kích hoạt dựa trên các đường chọn.
  - Ứng Dụng: Dùng trong các ứng dụng cần phân phối một tín hiệu tới nhiều đầu ra.

5. **Decoder**

- A **decoder** converts binary information from 'n' input lines to a maximum of 2n2^n2n unique output lines.
  - **Functionality:**
    - Inputs: n binary inputs
    - Outputs: Up to 2n2^n2n outputs
  - Use Case: Used in memory address decoding, and for enabling specific devices in a system.

5. Bộ Giải Mã (Decoder)

- **Decoder** chuyển đổi thông tin nhị phân từ 'n' đường vào thành tối đa 2n2^n2n đường ra duy nhất.
    - **Chức Năng:**
        - Đầu vào: n đường nhị phân
        - Đầu ra: Tối đa 2n2^n2n đầu ra
    - Ứng Dụng: Dùng trong giải mã địa chỉ bộ nhớ, và để kích hoạt các thiết bị cụ thể trong hệ thống.

6. **Encoder**

- An **encoder** performs the opposite function of a decoder, converting multiple inputs into a smaller number of outputs.
    - **Functionality:**
        - Inputs: Multiple inputs
        - Outputs: Encoded output
    - Use Case: Often used in communication systems to reduce the number of lines needed for data transmission

6. Bộ Mã Hóa (Encoder)

- **Encoder** thực hiện chức năng ngược lại với decoder, chuyển đổi nhiều đầu vào thành một số lượng đầu ra nhỏ hơn.
    - **Chức Năng:**
        - Đầu vào: Nhiều đầu vào
        - Đầu ra: Đầu ra được mã hóa
    - Ứng Dụng: Thường dùng trong hệ thống truyền thông để giảm số lượng đường truyền dữ liệu cần thiết.

CHAPTER 12:

12.1: Đặc điểm của lệnh máy

Việt Nam:

Trong chương này, chúng ta tập trung vào tập lệnh máy và các thành phần cơ bản của một lệnh máy. Đây là những thông tin mà bộ xử lý cần để thực hiện một lệnh. Một lệnh máy bao gồm mã lệnh, tham chiếu đến các toán hạng nguồn, toán hạng kết quả, và tham chiếu đến lệnh tiếp theo.

Các thành phần chính của lệnh máy:

1. **Mã lệnh (Opcode):** Xác định thao tác cần thực hiện (ví dụ: cộng, trừ).
2. **Tham chiếu toán hạng nguồn:** Xác định toán hạng đầu vào cho thao tác.
3. **Tham chiếu toán hạng kết quả:** Xác định vị trí lưu trữ kết quả.
4. **Tham chiếu lệnh tiếp theo:** Xác định địa chỉ của lệnh tiếp theo cần thực hiện.

Lệnh máy cũng có thể được biểu diễn dưới dạng nhị phân hoặc ký hiệu với các tên gọi dễ hiểu như ADD (Cộng), SUB (Trừ), và MUL (Nhân).

English:

This section focuses on machine instructions and their basic elements, which are necessary for the processor to execute a command. A machine instruction includes the operation code, references to source operands, result operands, and the reference to the next instruction.

Main components of a machine instruction:

1. **Operation code (Opcode):** Specifies the operation (e.g., addition, subtraction).
2. **Source operand reference:** Identifies the input operands.
3. **Result operand reference:** Specifies where to store the result.
4. **Next instruction reference:** Determines the address of the next instruction.

Machine instructions can be represented in binary or symbolically with recognizable terms like ADD, SUB, and MUL.

12.2: Các loại toán hạng

Việt Nam:

Máy tính xử lý dữ liệu với nhiều loại toán hạng khác nhau như địa chỉ, số, ký tự và dữ liệu logic. Những loại này thường xuất hiện trong các lệnh máy.

1. **Địa chỉ:** Thường được sử dụng trong các phép toán về bộ nhớ và tham chiếu đến một địa chỉ cụ thể.
2. **Số:** Bao gồm các loại số nguyên nhị phân, số thập phân và số dấu phẩy động.
3. **Ký tự:** Được mã hóa dưới dạng nhị phân, phổ biến nhất là mã ASCII hoặc EBCDIC.
4. **Dữ liệu logic:** Các phép toán xử lý từng bit, với hai giá trị là đúng hoặc sai.

English:

Computers handle different types of operands, including addresses, numbers, characters, and logical data, which are commonly used in machine instructions.

1. **Addresses:** Used for memory operations and reference a specific location.
2. **Numbers:** Include binary integers, decimal, and floating-point numbers.
3. **Characters:** Encoded in binary, with ASCII and EBCDIC being common formats.
4. **Logical data:** Involve operations on individual bits, representing true or false values.

12.4: Các loại thao tác

Việt Nam:

Có nhiều loại thao tác khác nhau được hỗ trợ bởi máy tính, bao gồm:

- **Chuyển dữ liệu:** Di chuyển dữ liệu giữa các thanh ghi hoặc bộ nhớ.
- **Toán học:** Thực hiện các phép toán cộng, trừ, nhân, chia.
- **Logic:** Thực hiện các phép toán AND, OR, NOT, XOR trên các bit.
- **Chuyển đổi:** Chuyển đổi dữ liệu từ định dạng này sang định dạng khác.
- **I/O (Nhập/Xuất):** Trao đổi dữ liệu với các thiết bị ngoại vi.
- **Điều khiển hệ thống:** Các lệnh đặc biệt dành cho quản lý hệ thống.
- **Chuyển quyền điều khiển:** Chuyển luồng thực thi giữa các đoạn mã.

English:

There are various types of operations supported by computers, including:

- **Data transfer:** Moving data between registers or memory.
- **Arithmetic:** Performing addition, subtraction, multiplication, and division.
- **Logical:** Conducting AND, OR, NOT, XOR operations on bits.
- **Conversion:** Converting data from one format to another.
- **Input/Output (I/O):** Exchanging data with peripheral devices.
- **System control:** Special instructions for system management.
- **Transfer of control:** Switching execution between different code segments.

CHAPTER 16:

1. Flynn's Taxonomy

This taxonomy classifies parallel processing systems based on instruction and data streams:

- **SISD (Single Instruction, Single Data)**: A single processor executes a single instruction stream to process a single data stream in memory. Traditional single-processor systems belong to this type.
- **SIMD (Single Instruction, Multiple Data)**: A single instruction controls multiple processing elements that operate on different data sets in parallel. Each element has its own memory, allowing for parallel data processing. Examples include vector processors and array processors.
- **MISD (Multiple Instruction, Single Data)**: Multiple processors execute different instructions on the same data stream. MISD systems are rare and do not have common commercial applications.
- **MIMD (Multiple Instruction, Multiple Data)**: Multiple processors execute different instructions on different data sets concurrently. This is the most versatile processing type and is common in modern multi-core systems like SMPs, clusters, and NUMA.

---

1. Phân loại của Flynn

Đây là cách phân loại các hệ thống xử lý song song dựa trên luồng lệnh và luồng dữ liệu:

- **SISD (Single Instruction, Single Data)**: Một bộ xử lý thực hiện một luồng lệnh để xử lý một luồng dữ liệu duy nhất trong không gian bộ nhớ. Hệ thống đơn xử lý truyền thống thuộc loại này.
- **SIMD (Single Instruction, Multiple Data)**: Một lệnh điều khiển nhiều phần tử xử lý hoạt động trên các tập dữ liệu khác nhau song song. Mỗi phần tử có bộ nhớ riêng, giúp xử lý dữ liệu song song. Ví dụ: bộ xử lý vector và bộ xử lý mảng.
- **MISD (Multiple Instruction, Single Data)**: Nhiều bộ xử lý thực hiện các lệnh khác nhau trên cùng một luồng dữ liệu. MISD hiếm và không có ứng dụng thương mại phổ biến.
- **MIMD (Multiple Instruction, Multiple Data)**: Nhiều bộ xử lý thực hiện các lệnh khác nhau trên các tập dữ liệu khác nhau đồng thời. Đây là loại hệ thống xử lý đa năng, phổ biến trong các hệ thống xử lý đa nhân hiện đại như SMPs, clusters, và NUMA.

2. Symmetric Multiprocessors (SMPs)

SMP is a multiprocessor system where processors have similar capabilities and share the same main memory and I/O devices via a common connection system, allowing for uniform memory access.

- Characteristics of SMP:
    1. Two or more processors with equivalent capabilities.
    2. Processors share memory and I/O devices through a common connection.
    3. An integrated operating system manages processor interactions.
- **Advantages of SMP**:
    - **Performance**: Tasks can be parallelized, improving performance.
    - **Reliability**: Failure of one device doesn't affect the whole system.
- **Disadvantages of SMP**:
    - **Performance Limitation**: Multiple processors accessing memory through a shared bus can create bottlenecks.
    - **Cache Coherence**: Protocols are needed to maintain cache consistency among processors.

---

2. Bộ xử lý đa đối xứng (SMP)

SMP là hệ thống đa xử lý trong đó các bộ xử lý có khả năng tương tự, chia sẻ cùng một bộ nhớ chính và các thiết bị I/O qua một hệ thống kết nối chung, cho phép truy cập bộ nhớ đồng nhất.

- Đặc điểm của SMP:
    1. Hai hoặc nhiều bộ xử lý có khả năng tương đương.
    2. Các bộ xử lý chia sẻ bộ nhớ và thiết bị I/O qua kết nối chung.
    3. Hệ điều hành tích hợp quản lý tương tác giữa các bộ xử lý.
- **Ưu điểm của SMP**:
    - **Hiệu suất**: Hệ thống có thể thực hiện các phần của nhiệm vụ song song, cải thiện hiệu suất.
    - **Độ tin cậy**: Các lỗi của một thiết bị không ảnh hưởng đến toàn bộ hệ thống.
- **Nhược điểm của SMP**:
    - **Giới hạn về hiệu suất**: Khi các bộ xử lý cùng truy cập vào bộ nhớ qua bus chia sẻ, có thể gây tắc nghẽn.
    - **Đồng bộ cache**: Cần có các giao thức để duy trì tính nhất quán của cache giữa các bộ xử lý.

3. Multiprocessing System Topologies

- **Shared Bus SMP**: Processors and main memory are connected through a shared bus, allowing access to memory and I/O devices. A shared bus simplifies system design and is scalable, but it can create bottlenecks when multiple devices need simultaneous access.
- **NUMA (Non-Uniform Memory Access)**: Processors have unequal access times to different memory regions, providing better scalability.

---

3. Cấu trúc liên kết của hệ thống đa xử lý

- **SMP với bus chia sẻ**: Các bộ xử lý và bộ nhớ chính được kết nối qua một bus chung, cho phép truy cập vào bộ nhớ và các thiết bị I/O. Bus chia sẻ giúp hệ thống đơn giản và dễ mở rộng, nhưng có thể gây tắc nghẽn khi nhiều thiết bị cần truy cập cùng lúc.
- **NUMA (Non-Uniform Memory Access)**: Các bộ xử lý có thời gian truy cập bộ nhớ không đồng đều tùy thuộc vào vùng bộ nhớ được truy cập. Điều này giúp hệ thống có khả năng mở rộng tốt hơn.

4. Operating System Design Requirements for SMP

- **Concurrent Process Management**: The OS must support code that can be executed by multiple processors simultaneously without causing conflicts.
- **Process Scheduling**: The OS must efficiently assign tasks to processors.
- **Synchronization**: Ensures processors share resources without conflicts.
- **Memory Management**: Maintains memory consistency across processors.
- **Reliability and Fault Tolerance**: Handles failures in one processor without affecting the entire system.

---

4. Các yêu cầu thiết kế hệ điều hành cho SMP

- **Quản lý các tiến trình đồng thời**: Hệ điều hành cần hỗ trợ các mã chương trình có thể được nhiều bộ xử lý thực thi cùng lúc mà không gây ra xung đột.
- **Lập lịch tiến trình**: Hệ điều hành cần phân công các tiến trình sao cho hiệu quả.
- **Đồng bộ hóa**: Đảm bảo các bộ xử lý chia sẻ tài nguyên mà không gây ra xung đột.
- **Quản lý bộ nhớ**: Đảm bảo tính nhất quán bộ nhớ giữa các bộ xử lý.
- **Độ tin cậy và khả năng chịu lỗi**: Xử lý sự cố khi một bộ xử lý gặp lỗi mà không ảnh hưởng đến toàn bộ hệ thống.