CEA201 (Summary theories)

Computer Architecture (Trường Đại học FPT)

Chapter 1: Introduction

Key Concepts:

- **Turing Model**:
  - **Universal Computation**: Concept that any computable problem can be solved by a Turing machine with the correct program.
  - **Programmability**: Allows different outputs from the same input by changing the program.
  - **Data Processor**: Described as a "black box" that processes input data based on the program to produce output.
- **Von Neumann Model**:
  - **Stored Program Concept**: Both data and instructions are stored in memory.
  - **Sequential Execution**: Instructions are executed one after another in a sequence.
  - **Subsystems**:
    - **Memory**: Stores data and instructions.
    - **Arithmetic Logic Unit (ALU)**: Performs arithmetic and logic operations.
    - **Control Unit**: Directs and manages other subsystems.
    - **Input/Output (I/O)**: Manages data exchange with external devices.
- **Computer Generations**:
  - **First Generation (1945-1956)**: Based on vacuum tubes, large, costly, and unreliable.
  - **Second Generation (1959-1965)**: Utilized transistors, more reliable, and supported high-level programming languages.
  - **Third Generation (1965-1975)**: Integrated circuits enabled miniaturization and software packages.
  - **Fourth Generation (1975-1985)**: Introduction of microprocessors led to personal computers and desktop calculators.
  - **Fifth Generation (Present and Beyond)**: Features AI, voice recognition, and natural language processing.
- **Computer Subsystems**:
  - **Central Processing Unit (CPU)**: Executes instructions and performs calculations, including ALU, control unit, and registers.
  - **Main Memory**:
    - Stores data and instructions actively used.
    - Types include volatile **RAM** and non-volatile **ROM**.
    - **Memory Hierarchy**: Registers > Cache > Main Memory > Secondary Storage.
  - **Input/Output (I/O) Subsystem**:
    - Interfaces with peripherals like keyboard, monitor, and storage.
    - **I/O Addressing**:
      - **Isolated I/O**: Separate address spaces for memory and I/O devices.
      - **Memory-Mapped I/O**: Integrates I/O devices into memory space.

- **Different Architectures**:
    - **CISC (Complex Instruction Set Computer)**: Large instruction set with complex instructions, easier to program.
    - **RISC (Reduced Instruction Set Computer)**: Small instruction set with simple instructions, more efficient.
    - **Pipelining**: Executes overlapping instruction stages to increase throughput.
    - **Parallel Processing**: Uses multiple processing units to perform tasks simultaneously (SISD, SIMD, MIMD).

Chapter 2: Computer Evolution and Performance

1. Designing for Performance:

- Computer systems have become drastically cheaper and more powerful over time.
- Modern applications demand high performance, driving the need for sophisticated design techniques.

2. Microprocessor Speed and Performance Balance:

- **Processor Speed:** Increased by factors such as:
    - Pipelining (overlapping instruction execution stages).
    - Branch prediction (predicting program flow).
    - Superscalar execution (issuing multiple instructions per cycle).
    - Data flow analysis (optimizing instruction scheduling).
    - Speculative execution (executing instructions before they are definitely needed).
- **Performance Balance:** Balancing the performance of different components (CPU, memory, I/O) to avoid bottlenecks. Improved by:
    - Increasing hardware speed of the processor.
    - Increasing size and speed of caches.
    - Changing processor organization and architecture.
- **Challenges:**
    - Power consumption and heat dissipation.
    - RC delay (resistance and capacitance in interconnects).
    - Memory latency and throughput (memory speed lagging behind CPU speed).

3. Multicore and MICs (Many Integrated Cores):

- **Multicore:** Multiple processor cores on a single chip, enhancing performance through parallelism.
- **MIC (Many Integrated Core):** Chips with a large number of cores (50 or more), posing software challenges to exploit parallelism effectively.
- **Benefits:** Increased performance without increasing clock rate, allows for larger caches.

4. GPUs (Graphics Processing Units) and GPGPUs:

- **GPU:** Specialized core for parallel processing of graphics data.
- **GPGPU (General-Purpose Computing on GPUs):** Using GPUs for non-graphics tasks, leveraging their parallel processing capabilities.

5. Amdahl's Law:

- **Concept:** Shows the limit to speedup achievable through parallelization.
- **Key Factors:**
    - The fraction of a program that can be parallelized (f).
    - The number of processors (N).
- **Limitations:** Too pessimistic for some applications (e.g., servers with multiple tasks).

6. Little's Law:

- **Concept:** Relates the average number of items in a queuing system to arrival rate and time spent in the system.
- **Formula:** Average number of items = Arrival rate * Average time in system.
- **Applications:** Applicable to various systems (processors, networks, I/O devices) for performance analysis.

7. Basic Performance Measures:

- **Clock Speed:** Frequency of the system clock, measured in Hertz (Hz).
- **Instruction Execution Rate:** Number of instructions executed per second (MIPS).
- **MFLOPS:** Millions of Floating-Point Operations Per Second.

8. Calculating the Mean:

- **Arithmetic Mean:** Average of all values, useful for execution times.
- **Harmonic Mean:** Useful for comparing rates (e.g., MFLOPS).
- **Geometric Mean:** Appropriate for comparing normalized results (relative performance).

9. Benchmarks and SPEC:

- **Benchmark Suite:** A collection of programs used to evaluate system performance.
- **SPEC (Standard Performance Evaluation Corporation):** An industry consortium that defines and maintains benchmark suites.
- **SPEC CPU2017:** A widely used benchmark suite for processor-intensive applications, measuring both speed and rate.

Key Takeaways:

- Achieving high performance requires careful attention to processor speed, memory speed, I/O performance, and overall system balance.
- Multicore and MIC architectures are key strategies for future performance improvements, requiring software adaptation for effective parallelism.
- Benchmarks provide a standardized way to compare system performance, with SPEC suites being a prominent example.

Quizlet Flashcards - Chapter 2: Computer Evolution and Performance

Term	Definition
Pipelining	Overlapping the execution stages of multiple instructions to increase processor throughput.
Branch Prediction	Predicting the outcome of conditional branches to prefetch instructions and reduce stalls.
Superscalar Execution	Issuing multiple instructions per clock cycle using parallel pipelines.
Data Flow Analysis	Analyzing dependencies between instructions to create an optimized execution schedule.
Speculative Execution	Executing instructions ahead of time based on predictions, even if they might not be needed.
Performance Balance	Balancing the performance of different components (CPU, memory, I/O) to avoid bottlenecks.
Multicore Processor	A processor with multiple processing cores on a single chip.
MIC (Many Integrated Core)	A chip with a large number of processing cores (50 or more).
GPU (Graphics Processing Unit)	A specialized core designed for parallel processing of graphics data.
GPGPU	Using GPUs for non-graphics tasks, leveraging their parallel processing capabilities.
Amdahl's Law	Describes the theoretical speedup limit from parallelization.
Little's Law	Relates average number of items in a queuing system to arrival rate and average time in the system.
Clock Speed	The frequency at which a processor operates, measured in Hertz (Hz).
MIPS	Millions of Instructions Per Second.
MFLOPS	Millions of Floating-Point Operations Per Second.
Arithmetic Mean	The average of a set of values.
Harmonic Mean	Useful for comparing rates, often used for MFLOPS calculations.
Geometric Mean	Appropriate for comparing normalized results (relative performance).

Benchmark Suite	A collection of programs used to evaluate system performance.
SPEC	Standard Performance Evaluation Corporation.
SPEC CPU2017	A benchmark suite for measuring processor performance.

Chapter 3: Computer Function and Interconnection

(Note: *Bộ nhớ đánh địa chỉ từng byte một* and the statements about 8-bit and 16-bit reads/writes are assumed to be general concepts related to memory addressing and data bus width. They are integrated into the relevant sections below.)

1. Computer Components:

- **Von Neumann Architecture:**
  - Data and instructions stored in a single memory space.
  - Memory contents addressable by location (typically byte-addressable).
  - Sequential instruction execution.
- **Components:**
  - CPU (Central Processing Unit): Instruction interpreter, ALU (Arithmetic Logic Unit), and registers.
  - Memory: Stores data and instructions. (Byte-addressable memory allows access to individual bytes. Data bus width determines how many bits are read/written at a time - e.g., 8 bits, 16 bits).
  - I/O (Input/Output) Components: Input and output modules for data transfer with external devices (e.g., USB devices).
  - PC (Program Counter): Register holding the address of the next instruction in memory.
  - IR (Instruction Register): Register holding the current instruction, including opcode and address.

2. Computer Function:

- **Instruction Cycle:**
  - Fetch Cycle: Retrieving an instruction from memory. (The CPU fetches the instruction from the memory location pointed to by the PC, then increments the PC to point to the next instruction.)
  - Execute Cycle: Decoding and performing the operation specified by the instruction.
- **Instruction Cycle States:** (These provide a more detailed breakdown of the fetch and execute cycles)
  - Instruction Address Calculation (IAC): Determining the address of the next instruction.
  - Instruction Fetch (IF): Retrieving the instruction from memory.

- Instruction Operation Decoding (IOD): Decoding the opcode to determine the operation.
- Operand Address Calculation (OAC): Calculating the address of the operands.
- Operand Fetch (OF): Retrieving the operands from memory or registers.
- Data Operation (DO): Executing the operation on the operands.
- Operand Store (OS): Storing the result in memory or registers.
- Interrupt Check: Checking for pending interrupts.

- **Instruction address calculation (Tính toán địa chỉ lệnh)**: Đây là bước đầu tiên, khi bộ xử lý tính toán địa chỉ của lệnh cần thực hiện.
- **Cách nhớ**: "Bắt đầu với địa chỉ của lệnh."
- **Instruction fetch (Nạp lệnh)**: Sau khi tính toán địa chỉ, bộ xử lý sẽ nạp lệnh từ bộ nhớ.
- **Cách nhớ**: "Nạp lệnh từ địa chỉ đã tính."
- **Instruction operation decoding (Giải mã lệnh)**: Lệnh sau đó được giải mã để xác định hành động cần thực hiện.
- **Cách nhớ**: "Giải mã lệnh để biết cần làm gì."
- **Operand address calculation (Tính toán địa chỉ toán hạng)**: Bộ xử lý tính toán địa chỉ của các toán hạng (dữ liệu cần thiết cho lệnh).
- **Cách nhớ**: "Tính toán địa chỉ của các dữ liệu cần thiết."
- **Operand fetch (Nạp toán hạng)**: Bộ xử lý nạp các toán hạng từ bộ nhớ.
- **Cách nhớ**: "Nạp dữ liệu để sử dụng cho lệnh."
- **Data Operation (Thực hiện phép toán dữ liệu)**: Thực hiện các phép tính hoặc thao tác với dữ liệu nạp vào.
- **Cách nhớ**: "Thực hiện các phép toán trên dữ liệu."
- **Operand store (Lưu trữ kết quả)**: Kết quả của phép toán được lưu trữ vào bộ nhớ hoặc thanh ghi.
- **Cách nhớ**: "Lưu kết quả của phép toán vào bộ nhớ."
- **Interrupt check (Kiểm tra ngắt)**: Cuối cùng, bộ xử lý kiểm tra xem có bất kỳ ngắt nào đang chờ xử lý không.
- **Cách nhớ**: "Kiểm tra có ngắt hay không trước khi tiếp tục."

3. Interrupts:

- **Purpose:** Improve processing efficiency by allowing the CPU to handle events asynchronously.
- **Types:** Program, timer, I/O, hardware failure.
- **Interrupt Cycle:** Inserted into the instruction cycle to check for and handle interrupts.
- **Interrupt Handling:**
  - Suspend current program, save context.
  - Set PC to the interrupt handler routine address.
  - Execute the interrupt handler.
  - Restore the context of the interrupted program.

- **Multiple Interrupts:**
    - Prioritized interrupt handling: Higher priority interrupts are handled first.
    - Interrupt vector table: A table of interrupt handler addresses.

4. I/O Function:

- **Processor-I/O Data Transfer:** CPU can directly read/write data to I/O modules.
- **Direct Memory Access (DMA):** Allows I/O modules to directly exchange data with memory, improving efficiency by freeing the CPU.

5. Interconnection Structures:

- **Bus Interconnection:**
    - Shared communication pathway.
    - System bus contains data, address, and control lines. (Data bus width significantly impacts system performance).
    - Limited by electrical constraints at high speeds and with many devices (bus contention).
- **Point-to-Point Interconnect:**
    - Direct connections between components.
    - Layered protocol architecture.
    - Packetized data transfer.

6. QuickPath Interconnect (QPI):

- Intel's point-to-point interconnect for high-performance systems.
- **Features:**
    - Multiple direct connections.
    - Layered protocol (physical, link, routing, protocol).
    - Packetized data transfer (phits and flits).

7. Peripheral Component Interconnect Express (PCIe):

- High-speed, point-to-point interconnect for I/O devices.
- **Features:**
    - Layered protocol (physical, data link, transaction).
    - Multi-lane distribution for high data rates.
    - Scrambling and encoding for reliable data transmission.

Quizlet Flashcards - Chapter 3: Computer Function and Interconnection (Enhanced and reorganized)

Term	Definition
Von Neumann Architecture	Computer architecture where data and instructions share the same memory space.
CPU (Central Processing Unit)	Fetches and executes instructions, performs arithmetic and logic operations.

ALU (Arithmetic Logic Unit)	Performs arithmetic and logic operations within the CPU.
Memory	Stores data and instructions.
Byte-Addressable Memory	Memory organized so that each byte has a unique address.
Data Bus	Carries data between CPU, memory, and I/O devices. Its width determines how much data is transferred at once.
I/O Module	Interface for connecting peripheral devices to the computer system.
Program Counter (PC)	Holds the address of the next instruction to be fetched.
Instruction Register (IR)	Holds the currently executing instruction.
Instruction Cycle	The sequence of steps to fetch and execute an instruction.
Fetch Cycle	Retrieving an instruction from memory.
Execute Cycle	Decoding and performing the operation specified by the instruction.
Interrupt	A signal that disrupts normal program execution to handle an event.
Interrupt Handler	A routine that responds to a specific type of interrupt.
Interrupt Vector Table	A table of interrupt handler addresses.
DMA (Direct Memory Access)	Allows I/O devices to transfer data directly to/from memory without CPU intervention.
Bus Interconnection	A shared communication link connecting multiple devices.
System Bus	The main bus connecting CPU, memory, and I/O.
Point-to-Point Interconnect	Dedicated links between specific components.
QPI (QuickPath Interconnect)	Intel's point-to-point interconnect technology.
Phit	20-bit data transfer unit in QPI.
Flit	80-bit data transfer unit in QPI.
PCIe (Peripheral Component Interconnect Express)	High-speed serial interconnect for I/O devices.
PCIe Lane	A pair of wires for differential data transmission in PCIe.

This revised structure provides a more logical flow and incorporates the initial notes more effectively. It also expands on some definitions for greater clarity. Remember to further customize these flashcards with specific details from your textbook and course.

Chapter 4: Cache Memory

1. Principle of Locality:

- **Concept:** Programs tend to access memory locations in clusters, both spatially (nearby addresses) and temporally (recently accessed).
- **Types:**
    - **Temporal Locality:** Accessing the same memory location repeatedly in a short time.
    - **Spatial Locality:** Accessing memory locations near each other.
- **Importance:** Enables efficient use of a memory hierarchy to improve performance.

2. Characteristics of Memory Systems:

- **Location:** Internal (registers, cache, main memory) or external (disks, tapes).
- **Capacity:** Amount of data that can be stored (bytes, words – typically 1 word = 2 bytes or more).
- **Unit of Transfer:** Size of data transferred in a single operation (word or block – block size is typically a power of 2 bytes, e.g., 32 bytes, 64 bytes).
- **Access Method:**
    - Sequential: Linear access (e.g., tape).
    - Direct: Access based on physical location (e.g., disk).
    - Random: Direct access via unique address (e.g., main memory, cache).
    - Associative: Access based on content (e.g., some caches).
- **Performance:**
    - Access Time (Latency): Time to read or write data.
    - Memory Cycle Time: Time between successive accesses.
    - Transfer Rate: Data transfer speed (bits per second).
- **Physical Type:** Semiconductor, magnetic, optical, etc.
- **Physical Characteristics:**
    - Volatile: Data lost when power is off.
    - Nonvolatile: Data retained without power.
    - Erasable: Data can be rewritten.
    - Nonerasable: Data is permanent (e.g., ROM).

3. Memory Hierarchy:

- **Concept:** Using multiple levels of memory with varying speed, cost, and capacity.
- **Levels:** (Fastest to slowest)
    - Registers
    - Cache (L1, L2, L3, L4)
    - Main Memory
    - Secondary Memory (disks, tapes)
- **Design Principles:**
    - Locality: Exploiting locality to place frequently used data in faster memory.
    - Inclusion: Higher levels contain a subset of lower-level data.
    - Coherence: Maintaining consistency across levels.

4. Two-Level Memory Access (Cache):

- **Concept:** Cache acts as a buffer between CPU and main memory.
- **Hit Ratio:** Percentage of memory accesses found in cache.
- **Performance Impact:** High hit ratio greatly reduces average access time.

5. Multilevel Memory Access:

- **Extending the Two-Level Model:** Using more than two levels for further optimization.
- **Example:** Registers > L1 > L2 > Main Memory > Disk.

6. The IBM z13 Memory Hierarchy (Example):

- Illustrates multilevel caching (L1, L2, L3, L4) with different technologies (e.g., SRAM, eDRAM).
- Demonstrates shared caches (L3, L4) for multiple cores.

Key Takeaways: (As provided)

Quizlet Flashcards - Chapter 4: Cache Memory (Revised and expanded)

Term	Definition
Principle of Locality	Programs tend to access data and instructions in predictable patterns.

Temporal Locality	Accessing the same memory location repeatedly.
Spatial Locality	Accessing nearby memory locations.
Memory Hierarchy	A system of memory levels with different speeds, costs, and capacities.
Registers	Fastest, smallest memory within the CPU.
Cache Memory	Small, fast memory holding frequently accessed data.
L1 Cache	Smallest, fastest cache, typically private to a core.
L2 Cache	Larger, slower cache, may be private or shared.
L3 Cache	Largest, slowest cache, often shared by multiple cores.
Main Memory (RAM)	Primary memory for active data and instructions.
Secondary Memory	Non-volatile, long-term storage (e.g., hard drive).
Hit	Data found in cache.
Miss	Data not found in cache.
Hit Ratio	Percentage of memory accesses that are hits.

Miss Penalty	Time to retrieve data from a lower memory level on a cache miss.
Access Time (Latency)	Time to read/write data from a memory location.
Memory Cycle Time	Time between successive memory accesses.
Transfer Rate	Rate of data transfer (bits per second).
Volatile Memory	Loses data when power is off.
Nonvolatile Memory	Retains data without power.
Inclusion Property	Higher memory levels contain a subset of lower level data.
Coherence Property	Maintaining consistency across memory levels.
SRAM (Static RAM)	Faster, more expensive memory technology often used for cache.
DRAM (Dynamic RAM)	Slower, less expensive memory technology often used for main memory.

Chapter 5: Internal Memory Technology

1. Semiconductor Main Memory:

- **Memory Cell:** Basic unit of semiconductor memory, storing a single bit.
- **Random Access Memory (RAM):**
    - Volatile: Data loss upon power off.
    - Read/Write: Supports both read and write operations.
    - **Types:**
        - DRAM (Dynamic RAM): Stores data as charge, requires refresh.
        - SRAM (Static RAM): Uses flip-flops, faster but more expensive than DRAM.

2. Types of ROM (Read-Only Memory):

- **ROM:** Non-volatile, permanent storage, programmed during fabrication.
- **PROM (Programmable ROM):** One-time programmable after fabrication.
- **EPROM (Erasable PROM):** Erasable with UV light, reprogrammable.
- **EEPROM (Electrically Erasable PROM):** Electrically erasable and reprogrammable, byte-level access.
- **Flash Memory:** Non-volatile, block-erasable, faster erase than EPROM, denser than EEPROM.

3. Memory Organization:

- **Memory Chip Organization:** Logical arrangement of cells on a chip (e.g., 16-Mbit DRAM as 4M x 4).
- **Memory Module Organization:** Arrangement of multiple chips using logic like group select.
- **Interleaved Memory:** Organizes memory banks for simultaneous access, boosting performance.

4. Error Correction:

- **Hard Failure:** Permanent physical defect.
- **Soft Error:** Random, non-destructive data alteration.
- **Error-Correcting Codes (ECC):** Detect and correct errors.
    - Hamming Code (SEC): Single Error Correction.
    - SEC-DED: Single Error Correction, Double Error Detection.

5. Advanced DRAM Organization:

- **SDRAM (Synchronous DRAM):** Synchronized with external clock.
- **DDR SDRAM (Double Data Rate SDRAM):** Uses both clock edges and prefetch.
- **DDR Generations:** Evolving data rates and features (DDR1, DDR2, DDR3, DDR4, etc.).

6. Embedded DRAM (eDRAM):

- **Concept:** DRAM on the same chip as processor/ASIC.
- **Advantages:** Faster access than off-chip DRAM, lower power than SRAM.
- **Uses:** L3/L4 cache (IBM z13), memory-side cache (Intel Core).

7. Flash Memory in Detail:

- **Operation:** Uses floating-gate transistors.
- **Types:**
    - NOR Flash: Bit-level access, code storage, small data.
    - NAND Flash: Block-level access, higher density/write speed, SSDs, memory cards.

8. Newer Nonvolatile Memory Technologies:

- **STT-RAM (Spin-Transfer Torque RAM):** Magnetic RAM, fast, durable.
- **PCRAM (Phase-Change RAM):** Uses phase-change materials.
- **ReRAM (Resistive RAM):** Alters resistance for storage.
- **Potential:** Could replace/supplement DRAM and secondary storage.

Quizlet Flashcards - Chapter 5: Internal Memory Technology (Further refined)

Term	Definition
Memory Cell	Basic unit of memory, stores one bit.
RAM (Random Access Memory)	Volatile, read/write memory.
DRAM (Dynamic RAM)	Stores data as charge, needs refreshing.
SRAM (Static RAM)	Uses flip-flops, faster than DRAM, more expensive.
ROM (Read-Only Memory)	Non-volatile, permanent memory.
PROM (Programmable ROM)	One-time programmable ROM.
EPROM (Erasable PROM)	UV-erasable, reprogrammable ROM.
EEPROM (Electrically Erasable PROM)	Electrically erasable/reprogrammable ROM.
Flash Memory	Non-volatile, block-erasable memory.
NOR Flash	Bit-level access flash memory.
NAND Flash	Block-level access flash memory, higher density.
Memory Chip Organization	Arrangement of cells on a chip (e.g., rows x columns).
Memory Module Organization	Combining multiple chips to form larger memory units.
Interleaved Memory	Distributing addresses across banks for simultaneous access.

Hard Failure	Permanent physical defect in memory.
Soft Error	Random, non-destructive change in memory content.
ECC (Error-Correcting Code)	Detects and corrects memory errors.
Hamming Code (SEC)	Single Error Correction code.
SEC-DED	Single Error Correction, Double Error Detection code.
SDRAM (Synchronous DRAM)	DRAM synchronized with a clock signal.
DDR SDRAM (Double Data Rate SDRAM)	Transfers data on both rising/falling clock edges.
Prefetch Buffer (DDR)	Buffer in DDR SDRAM for faster data transfer.
eDRAM (Embedded DRAM)	DRAM integrated on the same chip as a processor.
STT-RAM (Spin-Transfer Torque RAM)	Magnetic RAM, fast and durable.
PCRAM (Phase-Change RAM)	Uses phase-change materials for storage.
ReRAM (Resistive RAM)	Stores data by changing resistance.

This further refined set of flashcards focuses on concise definitions and clear distinctions between related concepts, making them ideal for quick review and efficient learning. They also highlight key differences between memory types and technologies. Remember to adapt them based on your course material.

Chapter 6: External Memory

1. Magnetic Disk (Hard Disk Drive - HDD):

- **Structure:** Circular platter(s) with magnetizable coating.
- **Read/Write Mechanisms:**
    - Write: Current in head magnetizes disk area.
    - Read (Traditional): Moving magnetic field induces current.
    - Read (Modern): Magnetoresistive (MR) sensor detects magnetization changes.
- **Data Organization:**
    - Tracks: Concentric rings.
    - Sectors: Divisions within a track.
    - Cylinder: All tracks at the same radial position on multiple platters.

- **Formatting:** Organizes tracks/sectors with control data.
- **Disk Layout Methods:**
    - Constant Angular Velocity (CAV): Constant bits per sector, varying density.
    - Multiple Zone Recording (MZR): Zones with different bit densities for near-constant linear density.
- **Physical Characteristics:** Head motion (fixed/movable), portability, sides, platters.
- **Performance Parameters:**
    - Seek Time: Head movement to track.
    - Rotational Delay (Latency): Sector rotation under head.
    - Transfer Time: Data transfer to/from disk.
    - Access Time: Seek time + rotational delay + transfer time.

2. RAID (Redundant Array of Independent Disks):

- **Concept:** Multiple disks for performance/reliability.
- **RAID Levels:** (Focus on key differences and functionality)
    - RAID 0 (Striping): High performance, no redundancy.
    - RAID 1 (Mirroring): High reliability, data duplication.
    - RAID 5 (Distributed Parity): Good performance and redundancy, parity spread across disks.
    - RAID 6 (Dual Parity): Higher reliability than RAID 5, two parity calculations.

3. Solid-State Drives (SSDs):

- **Definition:** Semiconductor-based storage (typically NAND flash).
- **Advantages over HDDs:** Higher IOPS, durability, lower power, quieter/cooler, faster access.
- **Architecture:** Controller, addressing, buffer/cache, error correction, flash chips.
- **Practical Issues:** Performance degradation (wear leveling and TRIM command help mitigate this), write endurance.

4. Optical Memory:

- **CD:** Original audio format. CD-ROM (read-only), CD-R (write-once), CD-RW (rewritable).
- **DVD:** Higher capacity than CD. DVD-ROM, DVD-R, DVD-RW.
- **Blu-ray Disc (BD):** High-definition video, large capacity.

5. Magnetic Tape:

- **Technology:** Sequential access magnetic storage.
- **Data Organization:** Tracks, physical records, inter-record gaps.
- **Recording Techniques:** Parallel, serial, serpentine.
- **Advantages:** Low cost, high capacity (archival).
- **Disadvantage:** Slow sequential access.

Key Takeaways: (As provided)

Quizlet Flashcards - Chapter 6: External Memory (Further refined and focused)

Term	Definition
HDD (Hard Disk Drive)	Uses rotating magnetic platters for storage.
Track	Concentric ring on a disk platter.
Sector	A division within a disk track.
Cylinder	All tracks at the same radial position on multiple platters.
Seek Time	Time for the read/write head to move to the correct track.
Rotational Delay/Latency	Time for the desired sector to rotate under the head.
Transfer Time	Time to transfer data to/from the disk.
Access Time (HDD)	Seek time + rotational delay + transfer time.
RAID (Redundant Array of Independent Disks)	Combining multiple disks for performance and/or reliability.
RAID 0 (Striping)	Data striped across disks, high performance, no redundancy.
RAID 1 (Mirroring)	Data mirrored on two disks, high reliability.
RAID 5 (Distributed Parity)	Data and parity striped across disks, good performance and redundancy.
RAID 6 (Dual Parity)	Two parity calculations, higher reliability.
SSD (Solid State Drive)	Uses semiconductor memory (e.g., NAND flash) for storage.
NAND Flash Memory	Type of flash memory used in SSDs.
Wear Leveling	Distributes writes evenly to extend SSD lifespan.
TRIM Command	Informs SSD about unused blocks for efficient erasure.
Optical Disc	Uses laser technology for data storage.
CD-ROM	Read-only optical disc.
CD-R	Write-once optical disc.
CD-RW	Rewritable optical disc.
DVD	Higher capacity optical disc than CD.

Blu-ray Disc (BD)	High-definition optical disc, very large capacity.
Magnetic Tape	Sequential access storage using magnetic tape.
Serpentine Recording	Data recorded back and forth on tape.

This refined set of flashcards concentrates on the most essential concepts and terminology, making them more manageable for studying. They also emphasize the key differences between various storage technologies and RAID levels. Continue to adapt and expand upon them as needed for your specific course requirements.

Chapter 7: Input/Output (I/O)

1. Introduction:

- External devices interact with the computer through I/O modules.
- I/O modules are intermediaries between CPU, memory, and external devices.

2. External Devices:

- **Categories:**
    - Human Readable: User interaction (monitors, keyboards, printers).
    - Machine Readable: Equipment interaction (disk drives, sensors, actuators).
    - Communication: Remote device interaction (modems, network cards).
-
- **Components:**
    - Control Logic: Handles I/O module commands.
    - Transducer: Converts data between electrical and other energy forms.
    - Buffer: Temporary data storage.

3. I/O Module Functions:

- Control and Timing: Coordinates data flow.
- Processor Communication: Handles commands, data, status, addresses.
- Device Communication: Manages communication with the device.
- Data Buffering: Handles speed differences.
- Error Detection: Detects transmission errors.

4. I/O Techniques:

- **Programmed I/O:** CPU-controlled, simple but inefficient for large transfers.
- **Interrupt-Driven I/O:** CPU initiates I/O, gets interrupted when done. More efficient but still CPU-involved per transfer.
- **DMA (Direct Memory Access):** I/O module accesses memory directly, most efficient for bulk transfers.

- **Direct Cache Access (DCA):** I/O accesses cache directly, bypassing main memory, for highest speed (e.g., high-speed networking).

5. Programmed I/O and I/O Instructions:

- I/O Commands: Control, Test, Read, Write.
- **Addressing Modes:**
    - Memory-Mapped I/O: I/O shares memory address space.
    - Isolated I/O: Separate address space for I/O.

6. Interrupt Handling:

- Interrupt Signal: Device to CPU.
- Interrupt Acknowledge: CPU to device.
- Context Saving: Saving the current program state.
- Interrupt Handler: Routine to handle the interrupt.
- Context Restoring: Restoring the program state.

7. DMA (Direct Memory Access):

- DMA Controller: Manages I/O-memory data transfer.
- Cycle Stealing: DMA briefly takes bus control.
- Fly-by DMA: Data transfers directly between I/O and memory.

8. Direct Cache Access (DCA):

- Motivation: High-speed network traffic.
- Cache Injection: NIC writes to cache directly.
- Benefits: Lower latency, better network performance.

9. Evolution of I/O: From simple CPU-controlled I/O to specialized I/O modules and processors.

10. External Interconnection Standards: (Concise descriptions)

- USB: Widely used, hot-pluggable, various speeds.
- FireWire: High-speed, hot-pluggable.
- SCSI: Parallel interface, enterprise storage.
- Thunderbolt: Very high-speed, combines data, video, audio, power.
- InfiniBand: High-speed server/storage interconnect.
- PCIe: High-speed serial bus, internal peripherals.
- SATA: Serial interface for disk storage.
- Ethernet: Dominant wired networking.
- Wi-Fi: Dominant wireless networking.

11. IBM z13 I/O Structure (Example): Dedicated I/O subsystem with specialized processors, channels, hierarchical structure.

Key Takeaways: (As provided)

Quizlet Flashcards - Chapter 7: Input/Output (I/O) (Concise and focused)

Term	Definition
I/O Module	Interface between CPU/memory and external devices.
Peripheral Device	External device connected to the computer.
I/O Techniques	Methods for transferring data between CPU/memory and peripherals.
Programmed I/O	CPU-controlled I/O.
Interrupt-Driven I/O	I/O using interrupts.
DMA (Direct Memory Access)	Direct memory access by I/O devices.
DCA (Direct Cache Access)	Direct cache access by I/O devices.
Memory-Mapped I/O	I/O uses memory addresses.
Isolated I/O	I/O uses separate address space.
Interrupt	Signal indicating an event needing attention.
Interrupt Handler	Routine that handles an interrupt.
DMA Controller	Hardware for managing DMA transfers.
Cycle Stealing	DMA temporarily borrows the bus.
USB (Universal Serial Bus)	Common peripheral interface.
FireWire (IEEE 1394)	High-speed serial bus.
SCSI (Small Computer System Interface)	Parallel interface, often used for storage.
Thunderbolt	Very high-speed interface combining data/video/audio/power.
InfiniBand	High-speed interconnect for servers/storage.
PCIe (Peripheral Component Interconnect Express)	High-speed serial bus for internal peripherals.
SATA (Serial ATA)	Serial storage interface.
Ethernet	Dominant wired networking technology.
Wi-Fi (IEEE 802.11)	Dominant wireless networking technology.

This set of flashcards is even more streamlined, prioritizing core concepts and commonly used terminology. It's a good starting point for basic understanding, and you can add more detailed cards as needed. Remember to tailor the content to your specific textbook and course requirements.

Chapter 8: Operating System Support

1. Operating System (OS) Overview:

- **Purpose:** Manages hardware/software resources, provides user interface.
- **Objectives:** Convenience, efficiency.

2. OS Services:

- Program creation (tools).
- Program execution (loading, resource management).
- I/O device access (simplified interface).
- Controlled file access (format, storage, access control).
- System access (control).
- Error detection and response.
- Accounting (usage statistics).

3. Key Interfaces:

- **Instruction Set Architecture (ISA):**
    - User ISA: Application instructions.
    - System ISA: OS resource management instructions.
-
- **Application Binary Interface (ABI):** Binary compatibility standards.
- **Application Programming Interface (API):** High-level system access for applications.

4. OS as a Resource Manager:

- **Resources:** CPU time, memory, I/O, files.
- **Responsibilities:** Resource allocation, process management, access control.

5. Types of Operating Systems:

- Interactive: Direct user interaction.
- Batch: Grouped job processing.
- Multiprogramming: Multiple programs in memory, sharing CPU.
- Uniprogramming: One program at a time.

6. Scheduling:

- Long-Term: Job admission.

- Medium-Term: Swapping, degree of multiprogramming.
- Short-Term (Dispatcher): Next process to run.
- I/O: I/O device access.

7. Process Management:

- Process: Program in execution.
- **Process States:** New, Ready, Running, Waiting, Halted.
- Process Control Block (PCB): Process information.

8. Memory Management:

- Swapping: Moving processes between memory and disk.
- **Partitioning:**
    - Fixed: Fixed-size partitions.
    - Variable: Dynamic allocation.
-
- Paging: Fixed-size frames and pages.
- Virtual Memory: Disk space as RAM extension.
- Demand Paging: Loading pages on demand.
- Page Fault: Accessing a non-resident page.
- Thrashing: Excessive swapping.
- Translation Lookaside Buffer (TLB): Page table entry cache.

9. Segmentation:

- Dividing memory logically.
- **Advantages:** Handles growing data, independent modification, sharing/protection.

10. Intel x86 Memory Management:

- Segmentation and paging (optional).
- Segmentation: Segment selector and offset, protection.
- Paging: Two-level tables, page sizes, TLB.

11. ARM Memory Management:

- Virtual memory with sections, supersections, pages.
- Two-level translation tables.
- Access control via domains.

Key Takeaways: (As provided)

Quizlet Flashcards - Chapter 8: Operating System Support (Streamlined and reorganized)

Term	Definition

Operating System (OS)	Manages hardware/software, provides user interface.
Kernel	Core of the OS, handles essential functions.
System Call	Request from application to OS.
ISA (Instruction Set Architecture)	Processor instruction set.
ABI (Application Binary Interface)	Binary compatibility standards.
API (Application Programming Interface)	High-level interface for applications.
Process	Program in execution.
Process State	Current condition of a process (e.g., Running, Ready).
PCB (Process Control Block)	Data structure with process information.
Scheduling	Deciding which process gets CPU time.
Multiprogramming	Multiple programs in memory concurrently.
Memory Management	Managing memory allocation and access.
Swapping	Moving processes between memory and disk.
Paging	Dividing memory into frames and programs into pages.
Virtual Memory	Using disk space as an extension of RAM.
Demand Paging	Loading pages only when needed.
Page Fault	Trying to access a non-resident page.
TLB (Translation Lookaside Buffer)	Cache for page table entries.
Segmentation	Dividing memory into logical segments.

This refined set of flashcards focuses on the most crucial terms and concepts, providing a strong foundation for understanding operating system support. You can add more detailed cards for specific topics or examples as necessary. Remember to customize the content to match your course materials.

Chapter 9: Computer Arithmetic

Key Concepts:

- **Integer Representation**:
  - **Two's Complement**: A system for representing signed integers, allowing straightforward binary arithmetic with positive and negative values.
- **Floating-Point Representation (IEEE 754)**:
  - **Single Precision**: 32-bit format (1 sign bit, 8 exponent bits, 23 fraction bits).
  - **Double Precision**: 64-bit format (1 sign bit, 11 exponent bits, 52 fraction bits).
  - Represents real numbers, with techniques for handling rounding and precision errors.
- **Binary Arithmetic**:
  - **Addition/Subtraction**: Binary operations with overflow detection and two's complement handling.
  - **Multiplication/Division**: Performed in binary, similar to decimal operations, with methods for managing signed and floating-point numbers.

Chapter 10: Instruction Sets: Characteristics and Functions

Key Concepts:

- **Instruction Set Architecture (ISA)**:
  - Defines the processor's supported operations and encoding format.
- **Instruction Types**:
  - **Data Transfer**: Instructions for moving data (e.g., MOV, LOAD, STORE).
  - **Arithmetic Operations**: Basic math operations (e.g., ADD, SUB, MUL).
  - **Logical Operations**: Boolean operations (e.g., AND, OR, NOT).
  - **Control**: Instructions to manage program flow (e.g., JUMP, CALL).

IV. Instruction Set Design:

Key design issues include:

- Operation Repertoire: Number and complexity of supported instructions.
- Data Types: Supported data types (numbers, characters, logical).
- Instruction Format: Length, number of addresses, field sizes.
- Registers: Number and usage of processor registers.
- Addressing: Methods for specifying operand addresses.

V. Types of Operands:

- Numbers: Integers (signed, unsigned), floating-point, packed decimal.
- Addresses: Memory locations or I/O device identifiers.
- Characters: ASCII, EBCDIC.
- Logical Data: Bits treated individually (Boolean values).

VI. Data Types (x86 and ARM):

- x86 Data Types: Byte, word, doubleword, quadword, various numeric formats (Figure 13.4), SIMD data types.
- ARM Data Types: Byte, halfword, word, support for signed/unsigned integers, endianness.

VII. Types of Operations:

- Data Transfer: Moving data between locations (e.g., MOV, LOAD, STORE).

- Arithmetic: ADD, SUB, MUL, DIV, INC, DEC, NEG.
- Logical: AND, OR, XOR, NOT, shifts, rotates.
- Conversion: Changing data formats (e.g., decimal to binary).
- Input/Output: Transferring data between I/O and memory/registers (IN, OUT).
- System Control: Privileged operations managed by the operating system.
- Transfer of Control: Branching, skipping, procedure calls (CALL, RET, JMP, Jcc).

VIII. x86 and ARM Operation Types:

- x86: Complex array of operations including specialized instructions for procedures (CALL, ENTER, LEAVE, RETURN), SIMD instructions (MMX, SSE). Status flags (Table 13.8) and condition codes (Table 13.9) influence control flow.
- ARM: Load/store, branch, data processing, multiply, parallel addition/subtraction, extend, status register access. Conditional execution based on status flags (Table 13.11).

Quizlet Flashcards - Chapter 13: Instruction Sets

Term	Definition
Instruction Set	Collection of all instructions a processor can execute.
Opcode	Specifies the operation to be performed.
Operand	Data or location on which the operation is performed.
Instruction Format	Structure of an instruction (opcode, operands).
Mnemonic	Abbreviation representing an opcode.
Addressing Mode	Method for specifying operand location.
Data Transfer	Moving data between registers or memory.
Arithmetic Operation	Operations like ADD, SUB, MUL, DIV.
Logical Operation	Operations like AND, OR, XOR, NOT.
Shift Operation	Moving bits left or right within a word.
Rotate Operation	Shifting bits circularly within a word.
Conversion	Changing data format.
Input/Output	Transferring data to/from I/O devices.
System Control	Privileged instructions for OS.
Transfer of Control	Changing instruction execution sequence.
Branch Instruction	Conditional or unconditional jump to another instruction.
Skip Instruction	Skipping the next instruction.

Procedure Call	Invoking a subroutine.
Stack Frame	Data structure on the stack for procedure context.
SIMD	Single Instruction, Multiple Data.
MMX	Multimedia extensions for x86.
SSE	Streaming SIMD Extensions for x86.
Endianness	Byte order within a word (big-endian, little-endian).
Condition Code	Flags representing the outcome of an operation.

- **Addressing Modes**:
    - Various methods to specify data addresses, including **immediate**, **direct**, **indirect**, and **indexed** addressing.

Chapter 11: Addressing Modes and Formats

This chapter delves into addressing modes, which specify how operands are accessed, and instruction formats, which define the structure of instructions.

I. Addressing Modes:

Addressing modes determine how the effective address (EA) of an operand is calculated. Common modes include:

- **Immediate:** Operand is part of the instruction itself (Operand = A).
- **Direct:** EA is the address specified in the instruction (EA = A).
- **Indirect:** EA is fetched from the memory location pointed to by the address in the instruction (EA = (A)).
- **Register:** Operand is in a processor register (EA = R).
- **Register Indirect:** EA is in the register specified by the instruction (EA = (R)).
- **Displacement:** EA is the sum of a constant and the contents of a register (EA = A + (R)). Subtypes:
    - **Relative:** Uses the program counter (PC) as the register (EA = (PC) + A).
    - **Base-Register:** Uses a base register for addressing data structures.
    - **Indexing:** Uses an index register for accessing arrays. Includes autoindexing (pre/post).
- **Stack:** EA is the top of the stack; implicit addressing.

II. x86 Addressing Modes:

x86 offers a variety of addressing modes (Table 14.2) including combinations of displacement, base, index, and scaling. Segment registers play a role in address calculation (Figure 14.2).

III. ARM Addressing Modes:

ARM addressing also includes variations of register, immediate, and displacement modes. Indexing methods include offset, preindex, and postindex (Figure 14.3). Load/Store Multiple instructions support accessing multiple registers (Figure 14.4).

IV. Instruction Formats:

- **Instruction Formats:** Define the layout of bits within an instruction, including opcode and operand specifications.
- **Instruction Length:** A crucial design decision influenced by memory size, organization, bus structure, and processor complexity.
- **Allocation of Bits:** Balancing the number of addressing modes, operands, register sets, address range, and address granularity.

V. Instruction Formats (PDP-8, PDP-10, PDP-11, VAX, x86, ARM):

- **PDP-8:** Simple instruction formats for memory reference, I/O, and register reference instructions (Figure 14.5).
- **PDP-10:** Single instruction format with opcode, register, index register, and memory address fields (Figure 14.6).
- **PDP-11:** Variable-length instructions with different formats for different operations (Figure 14.7).
- **VAX:** Complex instruction formats supporting various addressing modes (Figure 14.8).
- **x86:** Variable-length instructions with complex encoding using prefixes, opcode, ModR/m byte, SIB byte, displacement, and immediate fields (Figure 14.9).
- **ARM:** Fixed-length (32-bit) instructions with various formats for data processing, load/store, and branch operations (Figure 14.10). Immediate constants are formed using rotation (Figure 14.11). Thumb instructions are 16-bit; Thumb-2 combines 16-bit and 32-bit instructions (Figures 14.12, 14.13).

Quizlet Flashcards - Chapter 14: Addressing Modes and Formats

Term	Definition
Addressing Mode	Method for specifying the location of an operand.
Effective Address (EA)	The actual memory address or register number of an operand.
Immediate Addressing	Operand is included in the instruction.
Direct Addressing	EA is the address specified in the instruction.

Indirect Addressing	EA is at the memory location whose address is in the instruction.
Register Addressing	Operand is in a processor register.
Register Indirect Addressing	EA is in a register specified by the instruction.
Displacement Addressing	EA = A + (R), where A is a constant and R is a register.
Relative Addressing	Displacement addressing using the PC as the register.
Base-Register Addressing	Displacement addressing using a base register.
Indexing	Displacement addressing using an index register.
Autoindexing	Automatically incrementing or decrementing the index register.
Preindexing	Indexing before indirection.
Postindexing	Indexing after indirection.
Stack Addressing	Operand is on the top of the stack.
Instruction Format	Layout of bits in an instruction.
Instruction Length	Number of bits in an instruction.
Variable-Length Instruction	Instructions with varying lengths.
ModR/m Byte (x86)	Specifies operands and addressing modes in x86.
SIB Byte (x86)	Provides scaled indexing in x86.
Thumb Instruction Set (ARM)	16-bit instruction set for ARM.
Thumb-2 Instruction Set (ARM)	Combines 16-bit and 32-bit instructions in ARM.

Chapter 12: Processor Structure and Function

This chapter explores the internal organization of the processor, including registers, instruction cycle, data flow, pipelining, and interrupt processing. It also examines the x86 and ARM architectures as examples.

I. Processor Organization:

A processor must be able to:

- **Fetch Instruction:** Read instructions from memory.

- **Interpret Instruction:** Decode the instruction to determine the action.
- **Fetch Data:** Retrieve data from memory or I/O if needed.
- **Process Data:** Perform arithmetic or logical operations on data.
- **Write Data:** Store results in memory or I/O.
- **Internal Memory (Registers):** Small, fast storage within the processor for temporary data and control information.

II. Register Organization:

- **Registers:** Function as a higher-speed memory layer within the processor.
- **User-Visible Registers:** Accessible by programmers to minimize memory access.
    - **General Purpose Registers:** Used for various operations.
    - **Data Registers:** Hold data.
    - **Address Registers:** Hold memory addresses or address offsets.
    - **Condition Codes (Flags):** Set by the processor to indicate the outcome of operations.
- **Control and Status Registers:** Used by the control unit and operating system.
    - **Program Counter (PC):** Holds the address of the next instruction.
    - **Instruction Register (IR):** Holds the current instruction.
    - **Memory Address Register (MAR):** Holds the memory address for access.
    - **Memory Buffer Register (MBR):** Holds data being read from or written to memory.
    - **Program Status Word (PSW):** Contains status flags and other control bits.

III. Instruction Cycle:

- **Fetch:** Retrieve the next instruction from memory (Figure 16.5).
- **Execute:** Perform the operation specified by the instruction.
- **Indirect:** Fetch the effective address of the operand if indirect addressing is used (Figure 16.6).
- **Interrupt:** Handle interrupts if they occur (Figure 16.7).
- **Instruction Cycle State Diagram (Figure 16.4):** Shows the detailed steps of the instruction cycle, including operand fetch, address calculation, and data operation.

IV. Instruction Pipelining:

- **Pipelining Strategy:** Overlapping the execution of multiple instructions, analogous to an assembly line (Figure 16.8, 16.9).
- **Additional Stages:** Fetch Instruction (FI), Decode Instruction (DI), Calculate Operands (CO), Fetch Operands (FO), Execute Instruction (EI), Write Operand (WO).
- **Pipeline Performance:** Increased instruction throughput.
- **Pipeline Hazards:** Conditions that stall the pipeline:
    - **Resource Hazards:** Conflicts over hardware resources (e.g., memory).
    - **Data Hazards:** Dependencies between instructions (RAW, WAR, WAW).
    - **Control Hazards (Branch Hazards):** Incorrect branch predictions.
- **Dealing with Branches:**
    - **Multiple Streams:** Duplicating pipeline stages (expensive).

    - **Prefetch Branch Target:** Fetching both branch target and next sequential instruction.
    - **Loop Buffer:** Small, fast memory for storing recently fetched instructions.
    - **Branch Prediction:** Predicting the outcome of a branch using static or dynamic techniques (Figure 16.18, 16.19, 16.20).

V. x86 Processor Family:

- **Register Organization:** General-purpose registers, segment registers, EFLAGS register (status and control), control registers (Figure 16.2, 16.25, 16.26).
- **Intel 80486 Pipelining:** Five-stage pipeline (Figure 16.21).
- **MMX Registers:** Mapping to floating-point registers (Figure 16.27).
- **Interrupt Processing:** Interrupts and exceptions, interrupt vector table (Table 16.3).

VI. ARM Processor:

- **RISC Architecture:** Moderate register set, load/store architecture, fixed-length instructions, separate ALU and shifter, few addressing modes, conditional instruction execution.
- **Organization:** Register file, barrel shifter, ALU, multiply/accumulate unit, control unit, CPSR (Figure 16.28).
- **Processor Modes:** User mode and six privileged modes (Supervisor, Abort, Undefined, Interrupt, Fast Interrupt, System).
- **Register Organization with Modes:** Banked registers for different modes (Figure 16.29).
- **CPSR and SPSR:** Program Status Registers (Figure 16.30).
- **Interrupt Processing:** Interrupt vector table (Table 16.4).

Quizlet Flashcards - Chapter 16: Processor Structure and Function

Term	Definition
Register	Small, fast storage location within the CPU.
Program Counter (PC)	Holds the address of the next instruction to be fetched.
Instruction Register (IR)	Holds the currently executing instruction.
Memory Address Register (MAR)	Holds the address of the memory location being accessed.
Memory Buffer Register (MBR)	Holds the data being read from or written to memory.
Program Status Word (PSW)	Contains status flags and control bits.
Instruction Cycle	Sequence of steps to fetch and execute an instruction.
Pipelining	Overlapping the execution of multiple instructions.
Pipeline Hazard	Condition that stalls the pipeline.

Resource Hazard	Pipeline stall due to resource conflict.
Data Hazard	Pipeline stall due to data dependency.
Control Hazard (Branch Hazard)	Pipeline stall due to branch instruction.
Branch Prediction	Predicting the outcome of a branch.
Loop Buffer	Small cache for instructions to optimize loop execution.
Interrupt	Signal from hardware requiring processor attention.
Exception	Software-generated interrupt.
Interrupt Vector Table	Table of addresses for interrupt handlers.
RISC	Reduced Instruction Set Computer.
CISC	Complex Instruction Set Computer.
User Mode	Processor mode for application programs.
Privileged Mode	Processor mode for operating system.
CPSR (ARM)	Current Program Status Register.
SPSR (ARM)	Saved Program Status Register.

Chapter 13: Reduced Instruction Set Computers (RISC)

This chapter examines the characteristics, advantages, and implementations of Reduced Instruction Set Computers (RISC).

I. Instruction Execution Characteristics:

These characteristics influence instruction set design:

- **Operations:** Types and frequency of operations in high-level languages (HLLs).
- **Operands:** Types and frequency of operands used in HLLs (Tables 17.2, 17.3).
- **Procedure Calls:** Frequency and number of arguments/local variables (Table 17.4).
- **Implications:** These characteristics suggest optimizing for simple, frequent operations and efficient register usage.

II. The Use of a Large Register File:

- **Software Solution:** Compiler allocates registers based on program analysis.
- **Hardware Solution:** Larger register file directly supports more variables in registers.
- **Register Windows:** Overlapping register sets for efficient procedure calls (Figures 17.1, 17.2).

- **Global Variables:** Using a set of global registers for frequently accessed global variables.
- **Large Register File vs. Cache:** Comparison of characteristics (Table 17.5, Figure 17.3).
- **Compiler Optimization for Register Allocation:** Graph coloring approach (Figure 17.4).

III. Reduced Instruction Set Architecture:

- **Characteristics of RISC:**
    - One instruction per machine cycle.
    - Register-to-register operations.
    - Simple addressing modes.
    - Simple instruction formats.
-
- **CISC vs. RISC Characteristics:** RISC simplifies the processor design for improved pipelining and higher clock speeds. CISC aims for code density and complex instructions.
- **Advantages of RISC:** (See "Circumstantial Evidence")
    - More effective compiler optimization.
    - Efficient pipelining.
    - Better interrupt response.
-
- **Code Size Relative to RISC I:** Comparison of code size for various architectures (Table 17.6).
- **Register-to-Register vs. Memory-to-Memory:** RISC's register-to-register approach reduces memory traffic (Figure 17.5).

IV. RISC Pipelining:

- **Pipelining with Regular Instructions:** Simplified instruction formats and execution patterns enable efficient pipelining.
- **Optimization of Pipelining:**
    - **Delayed Branch:** Executing an instruction in the delay slot after a branch (Table 17.8, Figure 17.7).
    - **Delayed Load:** Continuing execution while waiting for a load to complete.
    - **Loop Unrolling:** Replicating loop body to reduce overhead and increase parallelism (Figure 17.8).
-

V. MIPS R4000:

- **Instruction Set:** Simple, fixed-length instructions (Figure 17.9).
- **Instruction Pipeline:** Five-stage pipeline, superpipelining (Figure 17.10, 17.11).
- **Pipeline Stages:** Detailed description of each stage (Page 33).

VI. SPARC:

- **SPARC Register Set:** Register windows for efficient procedure calls (Figure 17.12, 17.13).
- **Instruction Set:** Simple instructions with few addressing modes (Table 17.10, Figure 17.14).
- **Instruction Format:** Fixed-length instructions.

VII. Processor Organization for Pipelining:

- **Buffers and Pre-Decoding:** I-buffer, pre-decoding unit (Figure 17.15).
- **Forwarding, Reorder Buffer, Multiple Reservation Stations:** Techniques to improve pipeline performance (Figure 17.16).

Quizlet Flashcards - Chapter 17: Reduced Instruction Set Computers (RISC)

Term	Definition
RISC	Reduced Instruction Set Computer.
CISC	Complex Instruction Set Computer.
Register File	Set of registers within the CPU.
Register Window	Overlapping register sets for efficient procedure calls.
Delayed Branch	Instruction executed after a branch, filling the delay slot.
Delayed Load	Continuing execution while waiting for a load to complete.
Loop Unrolling	Replicating loop body to reduce overhead.
Superscalar	Multiple pipelines to execute multiple instructions per cycle.
Superpipelining	Deep pipeline with many short stages.
MIPS R4000	Early commercially successful RISC processor.
SPARC	Scalable Processor Architecture (RISC).
Register-to-Register	RISC design principle: operations performed on registers.
Load/Store Architecture	Only LOAD and STORE instructions access memory.
Instruction Pipeline	Overlapping instruction execution stages.
Pipeline Stages (MIPS)	IF, RD, ALU, MEM, WB.
Pipeline Stages (R4000)	IF, IS, RF, EX, DF, DS, TC, WB.

TLB (Translation Lookaside Buffer)	Cache for virtual-to-physical address translations.

Chapter 14: Instruction-Level Parallelism and Superscalar Processors

This chapter discusses techniques for enhancing processor performance by exploiting instruction-level parallelism (ILP), focusing on superscalar processors.

I. Superscalar Overview:

- **Superscalar:** A processor designed to improve the performance of scalar instruction execution by issuing multiple instructions concurrently.
- **Scalar Instructions:** Operate on single data values (as opposed to vector instructions).
- **Parallelism:** Achieved through multiple pipelines and out-of-order execution.
- **Evolution:** Represents the next step in high-performance processor design.

II. Superscalar vs. Superpipelined:

- **Superscalar:** Multiple parallel pipelines (Figure 18.1).
- **Superpipelined:** Deeper pipeline with more stages.
- **Comparison:** Superscalar focuses on parallel execution, superpipelined on faster clock cycles (Figure 18.3).
- **Generic Superscalar Organization:** Includes instruction fetch, decode, issue, dispatch, execute, reorder buffer, write-back (Figure 18.2).
- **Speedup:** Superscalar processors can achieve significant speedup (Table 18.1).

III. Constraints:

- **Instruction-Level Parallelism (ILP):** Degree to which instructions can be executed in parallel.
- **Limitations on ILP:**
    - **True Data Dependency:** One instruction depends on the result of a previous instruction.
    - **Procedural Dependency:** Instructions related to branch prediction and control flow.
    - **Resource Conflicts:** Multiple instructions require the same resource simultaneously.
    - **Output Dependency:** Two instructions write to the same location.
    - **Antidependency (WAR):** One instruction writes to a location that a subsequent instruction reads.
- **Effect of Dependencies:** Dependencies can prevent parallel execution and stall the pipeline (Figure 18.4).

IV. Design Issues:

- **Instruction-Level Parallelism vs. Machine Parallelism:** ILP is the potential for parallelism; machine parallelism is the ability to exploit it through hardware.
- **Instruction Issue Policy:** Protocol for initiating instruction execution.
    - **In-Order Issue:** Instructions issued in program order.
    - **Out-of-Order Issue:** Instructions issued based on operand availability.
- **Completion:** Can be in-order or out-of-order (Figure 18.5).
- **Register Renaming:** Eliminating WAR and WAW hazards by using a larger set of physical registers (Figure 18.7).

V. Superscalar Execution:

- **Out-of-Order Execution Logic:** Includes renaming, dispatching, execution, and committing (Figure 18.6).
- **Reorder Buffer (ROB):** Holds instructions in execution and manages out-of-order completion (Page 25).
- **Register Renaming (Page 26):** Remapping architectural registers to physical registers.
- **Micro-op Queuing:** Holding instructions waiting for execution units.
- **Execution Units:** Integer and floating-point units retrieve data from registers and cache.

VI. Branch Prediction:

- **Importance:** Pipelined machines need to handle branches efficiently.
- **RISC vs. Superscalar:** Delayed branch less effective in superscalar processors.
- **Pre-RISC Techniques:** Superscalar processors use techniques like branch prediction tables.
- **Conceptual Depiction (Figure 18.8):** Shows the flow of instructions in superscalar processing.

VII. Superscalar Implementation:

- **Key Elements:** Instruction fetch strategies, dependency checking, instruction issuing, parallel execution resources, commit mechanisms (Page 17).

VIII. Intel Core Microarchitecture:

- **Front End:** BPU, instruction fetch/predecode, instruction queue/decode (Page 20).
- **Out-of-Order Execution Logic:** Allocate, rename, micro-op queuing, execution, retire.
- **Branch Prediction Unit (BPU):** Predicts branch types and uses a Branch Target Buffer (BTB).
- **Instruction Fetch and Predecode:** Fetches instructions, determines length, decodes prefixes.
- **Instruction Queue and Decode:** Holds fetched instructions and translates to micro-ops.
- **Out-of-Order Execution Logic (Page 24):** Allocates resources and manages ROB.
- **Cache/Memory Parameters and Performance:** L1, L2, L3 cache details (Table 18.2).

IX. ARM Cortex-A8:

- **Integer Pipeline:** 13-stage pipeline (Figure 18.11).
- **Instruction Fetch Unit:** Predicts instruction stream, fetches from L1 cache.
- **Instruction Decode Unit:** Dual pipeline structure, in-order issue.
- **Instruction Processing Stages (Page 31):** D0 (Thumb decode), D1 (decode), D2 (queue), D3 (scheduling), D4 (final decode).
- **Memory System Effects (Table 18.3):** Cache misses, TLB misses, store buffer full, unaligned access.
- **Dual-Issue Restrictions (Table 18.4):** Load/store, multiply, branch, data hazards, multi-cycle instructions.
- **Integer Execute Unit:** Two ALU pipelines, address generator, multiply pipeline.
- **Load/Store Pipeline:** Runs in parallel with integer pipeline.
- **NEON and Floating-Point Pipeline:** Separate pipeline for SIMD and floating-point operations (Figure 18.12).

X. ARM Cortex-M3:

- **Block Diagram:** Simplified architecture (Figure 18.13).
- **Pipeline:** Three-stage pipeline (Fetch, Decode, Execute) (Figure 18.14).

Quizlet Flashcards - Chapter 18: Instruction-Level Parallelism and Superscalar Processors

Term	Definition
Superscalar Processor	Processor that can issue and execute multiple instructions concurrently.
Superpipelined Processor	Processor with a deep pipeline and many stages.
Instruction-Level Parallelism (ILP)	Degree to which instructions in a program can be executed in parallel.
Data Dependency	One instruction depends on the result of a previous instruction.
Control Dependency	Dependency related to branch instructions and control flow.
Resource Conflict	Multiple instructions require the same resource.
Output Dependency (WAW)	Two instructions write to the same location.
Antidependency (WAR)	An instruction writes to a location that a later instruction reads.
Instruction Issue Policy	Protocol for initiating instruction execution (in-order/out-of-order).

Reorder Buffer (ROB)	Hardware buffer for managing out-of-order execution.
Register Renaming	Technique to eliminate WAR and WAW hazards.
Branch Prediction	Predicting the outcome of a branch instruction.
Branch Target Buffer (BTB)	Cache for branch target addresses.
Micro-op	Small, simple instruction used internally by the processor.
Instruction Window	Another name for reservation station.
Out-of-Order Execution (OoOE)	Executing instructions in a different order than the program order.
In-Order Execution	Executing instructions in the program order.

Chapter 15: Control Unit Operation + Chapter 16: Microprogrammed Control

This chapter examines the control unit, the part of the processor responsible for sequencing and executing instructions. It focuses on micro-operations and microprogrammed control.

I. Micro-operations:

- **Micro-operations (μops):** Functional, or atomic, operations of a processor. Simple steps involving processor registers.
- **Program Execution:** Consists of sequential instruction execution.
- **Instruction Cycle:** Made up of subcycles (fetch, indirect, execute, interrupt).
- **Subcycles:** Composed of one or more micro-operations (Figure 19.1).

II. The Fetch Cycle:

- **Purpose:** Retrieves the next instruction from memory.
- **Registers Involved:**
    - **Memory Address Register (MAR):** Holds memory address.
    - **Memory Buffer Register (MBR):** Holds instruction fetched from memory.
    - **Program Counter (PC):** Holds address of next instruction.
    - **Instruction Register (IR):** Holds the fetched instruction.
- **Sequence of Events (Figure 19.2):** PC -> MAR, Memory -> MBR, PC increment, MBR -> IR.

III. Indirect Cycle:

- **Purpose:** Fetches the operand address when using indirect addressing.
- **Sequence:** IR(Address) -> MAR, Memory -> MBR, MBR(Address) -> IR(Address).

IV. Interrupt Cycle:

- **Purpose:** Handles interrupts.
- **Sequence:** PC -> MBR, Save_Address -> MAR, Routine_Address -> PC, MBR -> Memory.

V. Execute Cycle:

- **Instruction Decoding:** Control unit examines opcode and generates the sequence of micro-operations.
- **Example (ADD):** IR(Address) -> MAR, Memory -> MBR, R1 + MBR -> R1.

VI. Instruction Cycle (Figure 19.3):

- **Flowchart:** Shows the sequence of fetch, indirect, execute, and interrupt cycles based on Instruction Control Code (ICC).

VII. Control of the Processor:

- **Control Unit:** Responsible for sequencing and executing micro-operations.
- **Functional Requirements:** Determines the control signals needed for each micro-operation.
- **Control Signals:** Activate specific data paths and functional units (Figure 19.4, 19.5).
- **Control Unit Tasks:**
    - **Sequencing:** Determining the order of micro-operations.
    - **Execution:** Generating control signals to execute micro-operations.
- **Micro-operations and Control Signals (Table 19.1):** Lists control signals for fetch, indirect, and interrupt cycles.

VIII. Internal Processor Organization (Figure 19.6):

- **Internal Bus:** Connects registers and functional units.
- **Data Paths:** Routes data between registers and ALU.
- **Intel 8085:** Example processor with internal organization (Figure 19.7).
- **Intel 8085 External Signals:** Description of control, address, and data signals (Table 19.2, Figures 19.8, 19.9).

IX. Control Unit Implementation:

- **Hardwired Implementation:** Control unit built using logic circuits. Complex and difficult to modify.
- **Microprogrammed Implementation:** Control unit uses microinstructions stored in control memory. More flexible and easier to design.

X. Microprogrammed Control:

- **Concept:** Uses microinstructions to control the processor.
- **Wilkes Control:** Early microprogrammed control unit design (Figure 19.16).
- **Microinstructions:** Control words specifying micro-operations. Can be horizontal (wide, directly controlling hardware) or vertical (encoded, requiring decoding). (Figure 19.12)

- **Control Memory:** Stores microinstructions.
- **Control Unit Microarchitecture:** Sequencing logic, control address register (CAR), control memory, control buffer register (CBR). (Figure 19.14)
- **Microprogrammed Control Unit Functioning:** Instruction decoding triggers microinstruction execution, which generates control signals (Figure 19.15).
- **Wilkes Example:** Illustrates microprogrammed control (Tables 19.4, 19.5).
- **Control Memory Organization:** Microinstructions for different parts of the instruction cycle (Figure 19.13).
- **Control Unit with Decoded Inputs:** Uses a decoder to generate control signals (Figure 19.10).

Quizlet Flashcards - Chapter 19: Control Unit Operation and Microprogrammed Control

Term	Definition
Micro-operation (μop)	Basic operation performed by the processor.
Instruction Cycle	Sequence of steps to fetch and execute an instruction.
Fetch Cycle	Retrieving an instruction from memory.
Indirect Cycle	Fetching the address of an operand.
Interrupt Cycle	Handling an interrupt.
Execute Cycle	Performing the operation specified by the instruction.
Control Unit	Part of the processor responsible for sequencing and executing instructions.
Control Signal	Signal that activates a data path or functional unit.
Hardwired Control	Control unit implemented using logic circuits.
Microprogrammed Control	Control unit implemented using microinstructions.
Microinstruction	Control word specifying micro-operations.
Control Memory	Memory that stores microinstructions.
Control Address Register (CAR)	Holds the address of the next microinstruction.
Control Buffer Register (CBR)	Holds the current microinstruction.
Horizontal Microinstruction	Wide microinstruction directly controlling hardware.
Vertical Microinstruction	Encoded microinstruction requiring decoding.

Chapter 17: Parallel Processing

This chapter explores parallel processing, focusing on different parallel architectures, multiprocessor systems, cache coherence, and multithreading.

I. Multiple Processor Organizations:

- **Flynn's Taxonomy:** Classifies parallel architectures based on instruction and data streams:
    - **SISD (Single Instruction, Single Data):** Uniprocessors (Figure 20.2a).
    - **SIMD (Single Instruction, Multiple Data):** Vector and array processors (Figure 20.2b).
    - **MISD (Multiple Instruction, Single Data):** Not commercially implemented.
    - **MIMD (Multiple Instruction, Multiple Data):** Most common type of parallel processor (Figures 20.2c, 20.2d). Includes SMP, clusters, and NUMA.
- **Taxonomy of Parallel Architectures:** Illustrates the relationship between different parallel processor architectures (Figure 20.1).

II. Symmetric Multiprocessors (SMPs):

- **Definition:** Standalone computer with multiple similar processors sharing memory, I/O, and the operating system (Figure 20.5).
- **Characteristics:**
    - Shared memory and I/O.
    - Shared access to I/O devices.
    - All processors can perform the same functions (symmetric).
    - Controlled by an integrated operating system.
- **Multiprogramming vs. Multiprocessing (Figure 20.3):** Multiprogramming interleaves processes on a single processor; multiprocessing overlaps execution on multiple processors.
- **Tightly Coupled Multiprocessor:** Generic block diagram (Figure 20.4).
- **Bus Organization:**
    - **Advantages:** Simplicity, flexibility, reliability.
    - **Disadvantages:** Performance bottleneck due to bus contention, cache coherence issues.

III. Multiprocessor Operating System Design Considerations:

- **Simultaneous Concurrent Processes:** OS routines need to be reentrant.
- **Scheduling:** Avoid conflicts between processors scheduling processes.
- **Synchronization:** Mechanisms to control access to shared resources.
- **Memory Management:** Consistent paging and page replacement across processors.
- **Reliability and Fault Tolerance:** Graceful degradation in case of processor failure.

IV. Cache Coherence:

- **Problem:** Data inconsistency between multiple caches in a multiprocessor system.

- **Software Solutions:** Compiler and OS manage cache coherence. Less efficient.
- **Hardware Solutions (Cache Coherence Protocols):** Dynamically recognize and resolve inconsistencies.
    - **Directory Protocols:** Centralized directory keeps track of cache line states. Effective in large-scale systems but can create a bottleneck.
    - **Snoopy Protocols:** Each cache controller monitors (snoops) the bus for cache updates. Well-suited for bus-based systems.
        - **Write Invalidate (MESI):** Most common approach. Invalidates other copies when a processor writes to a cache line.
        - **Write Update (Write Broadcast):** Updates all copies when a processor writes.

V. MESI Protocol:

- **MESI:** Modified, Exclusive, Shared, Invalid (Table 20.1, Figure 20.7).
- **State Transition Diagram (Figure 20.6):** Illustrates state transitions based on read/write operations and snooping.
- **Cache Operations:**
    - **Read Miss:** Processor initiates memory read and snoops the bus.
    - **Read Hit:** No state change.
    - **Write Miss:** Processor initiates read-with-intent-to-modify (RWITM) and loads the line.
    - **Write Hit:** Depends on the current state (Modified, Exclusive, Shared).
- **Writeback Cache Transactions:** Examples of read and write transactions (Figures 20.8, 20.9).

VI. Multithreading and Chip Multiprocessors:

- **MIPS Rate:** MIPS rate = f * IPC (frequency * instructions per cycle).
- **Multithreading:** Dividing an instruction stream into multiple threads for parallel execution. Increases ILP without significantly increasing complexity.
- **Threads vs. Processes (Page 29):** Thread is a dispatchable unit of work within a process. Process has resource ownership.
- **Implicit vs. Explicit Multithreading (Page 30):** Explicit: Threads are defined by software or hardware. Implicit: Hardware extracts threads from a sequential program.
- **Approaches to Explicit Multithreading:**
    - **Interleaved (Fine-Grained):** Switching threads at each clock cycle.
    - **Blocked (Coarse-Grained):** Switching threads on events like cache misses.
    - **Simultaneous Multithreading (SMT):** Issuing instructions from multiple threads simultaneously.
- **Chip Multiprocessing:** Replicating processors on a single chip (multicore).
- **Comparison of Approaches (Figure 20.10):** Illustrates how different multithreading approaches schedule threads.

VII. Clusters:

- **Definition:** Group of interconnected computers working together (Page 33).
- **Configurations:** Standby server, shared disk (Figure 20.11).

- **Clustering Methods (Table 20.2):** Passive standby, active secondary, servers connected to disks, servers sharing disks.

VIII. Nonuniform Memory Access (NUMA):

- **NUMA:** All processors can access all memory, but access time varies depending on memory location.
- **CC-NUMA (Cache-Coherent NUMA):** Maintains cache coherence across nodes.
- **Motivation:** Overcome SMP limitations, maintain transparent system-wide memory (Page 37).
- **Organization (Figure 20.12):** Multiple multiprocessor nodes connected via an interconnect network.
- **Pros and Cons:** Performance benefits at larger scales than SMP, but software changes may be required and there are availability concerns.

Quizlet Flashcards - Chapter 20: Parallel Processing

Term	Definition
Parallel Processing	Using multiple processors to perform computations concurrently.
Flynn's Taxonomy	Classification of computer architectures based on instruction and data streams.
SISD	Single Instruction, Single Data.
SIMD	Single Instruction, Multiple Data.
MISD	Multiple Instruction, Single Data.
MIMD	Multiple Instruction, Multiple Data.
SMP (Symmetric Multiprocessor)	Multiple processors share memory and OS.
Cache Coherence	Ensuring data consistency between multiple caches.
Cache Coherence Protocol	Hardware or software mechanism to maintain cache coherence.
Directory Protocol	Centralized directory tracks cache line states.
Snoopy Protocol	Cache controllers monitor bus for cache updates.
MESI Protocol	Modified, Exclusive, Shared, Invalid (cache coherence protocol).
Multithreading	Dividing an instruction stream into multiple threads.
Thread	Dispatchable unit of work within a process.

Process	Instance of a program running on a computer.
Implicit Multithreading	Hardware extracts threads from a sequential program.
Explicit Multithreading	Software or hardware defines separate threads.
SMT (Simultaneous Multithreading)	Issuing instructions from multiple threads concurrently.
Chip Multiprocessing (Multicore)	Multiple processors on a single chip.
Cluster	Group of interconnected computers working together.
NUMA (Nonuniform Memory Access)	All processors can access all memory, but access times vary.
CC-NUMA	Cache-Coherent NUMA.

Chapter 18: Multicore Computers

This chapter explores multicore computer architecture, discussing hardware and software performance issues, different multicore organizations, and cache coherence.

Key Concepts:

- **Multicore Design:**
  - A single processor chip contains multiple cores, allowing for parallel execution and increasing computational power without higher clock speeds.
  - Each core can independently execute tasks, improving overall system efficiency.
- **Cache Coherence:**
  - Shared Cache: Some multicore systems use a shared L3 cache accessible by all cores, while each core may have its own L1 and L2 caches.
  - Coherence Protocols: Protocols like MESI ensure data consistency across cores to prevent issues when multiple cores access shared memory.
- **Memory Bandwidth:** As the number of cores increases, sufficient memory bandwidth is necessary to avoid bottlenecks, as multiple cores access memory concurrently.
- **Software Performance Challenges:**
  - Parallel Programming: Writing software to efficiently use all available cores.
  - Load Balancing: Distributing tasks evenly across cores to prevent idle cores and maximize processing power.
  - Thread Management: Ensuring threads do not conflict and managing task dependencies.

I. Multicore Organization:

- **Alternative Chip Organizations:** Superscalar, simultaneous multithreading (SMT), and multicore (Figure 21.1). Multicore is the dominant approach.
- **Levels of Cache:** L1 (private), L2 (dedicated or shared), L3 (shared).
- **Simultaneous Multithreading (SMT):** Increases ILP by issuing instructions from multiple threads concurrently.

II. Hardware Performance Issues:

- **Increase in Parallelism and Complexity:** Difficult to design and manage complex parallel systems.
- **Power Consumption:** Power density increases with more cores (Figure 21.2).
- **Performance Effect of Multiple Cores:** Amdahl's law limits speedup due to sequential portions of code (Figure 21.3).
- **Scaling of Database Workloads:** Real-world examples of performance scaling (Figure 21.4).

III. Software Performance Issues:

- **Software on Multicore:** Need for parallel programming models and tools.
- **Valve Game Software Example:** Case study illustrating the challenges of multicore programming.

IV. Effective Applications for Multicore Processors:

- **Multi-threaded Native Applications:** Thread-level parallelism.
- **Multi-process Applications:** Process-level parallelism.
- **Java Applications:** Java Virtual Machine (JVM) supports multithreading.
- **Multi-instance Applications:** Virtualization for isolation.

V. Threading Granularity:

- **Granularity:** Size of the unit of work that can be parallelized.
- **Fine-Grained vs. Coarse-Grained:** Fine-grained offers more flexibility but has higher overhead.
- **Tradeoff:** Balancing flexibility and overhead.
- **Hybrid Threading (Figure 21.5):** Example of combining different threading approaches.

VI. Multicore Organization Alternatives (Figure 21.6):

- **Dedicated L1 Cache:** Each core has its own L1 cache.
- **Dedicated L2 Cache:** Each core has its own L2 cache.
- **Shared L2 Cache:** Cores share an L2 cache.
- **Shared L3 Cache:** Cores share an L3 cache.

VII. Heterogeneous Multicore Organization:

- **Heterogeneous Multicore:** Processor chip with different types of cores (e.g., CPU and GPU).
- **CPU-GPU Combination:** Prominent trend, but requires coordination.
- **GPU Characteristics:** Supports thousands of parallel execution threads.
- **GPU Applications:** Well-suited for vector and matrix operations.
- **Chip Elements (Figure 21.7):** CPUs, GPUs, caches, interconnection network, DRAM controllers.
- **AMD 5100K Example:** Operating parameters of a heterogeneous multicore processor (Table 21.1).

VIII. Heterogeneous System Architecture (HSA):

- **Key Features:** Unified virtual memory space, coherent memory policy, unified programming interface.
- **Objective:** Seamlessly integrate CPU and GPU programming.
- **Texas Instruments 66AK2H12 Example:** Heterogeneous multicore chip with ARM CPUs and C66x DSPs (Figure 21.8).

IX. ARM big.LITTLE Technology:

- **big.LITTLE:** Combines high-performance (big) and low-power (LITTLE) cores.
- **Chip Components (Figure 21.9):** Cortex-A15 (big), Cortex-A7 (LITTLE), CCI-400 interconnect, GIC-400 interrupt controller.
- **Pipelines (Figure 21.10):** Cortex-A7 (in-order), Cortex-A15 (out-of-order).
- **Performance Comparison (Figure 21.11):** A15 offers higher performance, A7 lower power.

X. Cache Coherence in Multicore:

- **Software vs. Hardware:** Hardware solutions are preferred for SoC.
- **Directory vs. Snoop Protocols:** Two main hardware approaches.
- **ACE (Advanced Extensible Interface Coherence Extensions):** ARM's hardware coherence solution. Supports various cache configurations and big.LITTLE.

XI. ARM ACE Cache Line States (Figure 21.12):

- **States:** Modified, Owned, Exclusive, Shared, Invalid.
- **Comparison of Snoop Protocols (Table 21.2):** MESIM (Modified, Exclusive, Shared, Invalid, Modified) and MOESI (Modified, Owned, Invalid, Shared, Invalid).

XII. Intel Core i7-5960X Example:

- **Block Diagram:** Eight cores, L1, L2, L3 caches, DDR4 memory controllers, PCIe (Figure 21.13).
- **Physical Layout:** Cores and memory controller arranged for efficient communication.

XIII. ARM Cortex-A15 MPCore:

- **Block Diagram:** Four cores, L1, L2 caches, TLBs, Snoop Control Unit (SCU), GIC (Figure 21.14).
- **Interrupt Handling:** GIC handles masking, prioritization, and distribution of interrupts.
- **GIC Details:** Memory-mapped, placed alongside cores, accessed via SCU (Page 24).
- **Interrupt States:** Inactive, Pending, Active (Page 25).
- **GIC Block Diagram:** Shows the internal structure of the GIC (Figure 21.15).
- **Cache Coherency:** SCU manages cache coherence using MESI and Direct Data Intervention (DDI).

Quizlet Flashcards - Chapter 21: Multicore Computers

Term	Definition
Multicore Processor	Chip with multiple processor cores.
SMT (Simultaneous Multithreading)	Executing instructions from multiple threads concurrently on a single core.
Amdahl's Law	Limits speedup due to sequential portions of code.
Threading Granularity	Size of the parallelizable unit of work.
Fine-Grained Threading	Small units of work, more flexibility, higher overhead.
Coarse-Grained Threading	Large units of work, less flexibility, lower overhead.
Heterogeneous Multicore	Chip with different types of cores (e.g., CPU, GPU).
GPU (Graphics Processing Unit)	Specialized processor for graphics and parallel computing.
HSA (Heterogeneous System Architecture)	Architecture for integrating CPU and GPU programming.
big.LITTLE Technology	Combining high-performance and low-power cores.
Cache Coherence	Ensuring data consistency across multiple caches.
Snoop Control Unit (SCU)	Hardware unit for managing cache coherence.
Direct Data Intervention (DDI)	Copying data directly between L1 caches.

MESI Protocol	Cache coherence protocol (Modified, Exclusive, Shared, Invalid).
MOESI Protocol	Cache coherence protocol (Modified, Owned, Exclusive, Shared, Invalid).
ACE (Advanced Extensible Interface Coherence Extensions)	ARM's cache coherence solution.

Chapter 19: Number Systems

Key Concepts:

- **Binary, Octal, and Hexadecimal Representations**:
  - **Binary (Base-2)**: Uses digits 0 and 1, fundamental to computer systems.
  - **Octal (Base-8)** and **Hexadecimal (Base-16)**: Used as shorthand for binary, simplifying binary representation.
- **Binary Arithmetic**:
  - **Addition**: Binary addition, including carry-over logic.
  - **Subtraction**: Using two's complement representation to perform subtraction as addition.
  - **Multiplication and Division**: Similar to decimal but with binary digits.
- **Two's Complement Representation**:
  - A method for representing signed integers, allowing positive and negative values.
  - Simplifies binary arithmetic for operations involving both positive and negative numbers.
- **Conversions**:
  - **Binary to Decimal**: Summing each binary digit multiplied by its respective power of 2.
  - **Binary to Hexadecimal**: Grouping binary digits into sets of four to convert directly to hex.
  - **Hexadecimal to Binary**: Each hex digit translates into a four-bit binary sequence, allowing quick conversion.

Chapter 20: Digital Logic

Key Concepts:

- **Boolean Algebra**:
  - The mathematical framework for logic circuits, defining operations using binary values.
  - **Basic Laws**:

        - **Commutative**: $A+B=B+A$A + B = B + A; $A \cdot B=B \cdot A$A \cdot B = B \cdot A
        - **Associative**: $A+(B+C)=(A+B)+C$A + (B + C) = (A + B) + C; $A \cdot (B \cdot C)=(A \cdot B) \cdot C$A \cdot (B \cdot C) = (A \cdot B) \cdot C
        - **Distributive**: $A \cdot (B+C)=A \cdot B+A \cdot C$A \cdot (B + C) = A \cdot B + A \cdot C; $A \cdot (B+C)=A \cdot B+A \cdot C$
        - **DeMorgan's Theorem**: $\overline{A \cdot B} = \overline{A} + \overline{B}$; $\overline{A + B} = \overline{A} \cdot \overline{B}$
- **Logic Gates**:
    - **Basic Gates**: AND, OR, NOT.
    - **Derived Gates**: NAND, NOR, XOR, XNOR.
    - Used to build digital circuits, these gates implement Boolean operations.
- **Combinational Circuits**:
    - Circuits where outputs depend only on the current inputs, with no memory of previous states.
    - **Examples**:
        - **Adders**: Used for binary addition, including half-adders and full-adders.
        - **Multiplexers**: Select one of many input signals based on control inputs.
        - **Decoders**: Convert binary input to a specific output line, often used in memory address decoding.
- **Sequential Circuits**:
    - Circuits where outputs depend on both current inputs and previous states, incorporating memory elements.
    - **Key Components**:
        - **Flip-Flops**: Store individual bits, acting as memory elements. Types include SR, JK, D, and T flip-flops.
        - **Counters**: Sequential circuits that count pulses, used in timing and control applications.
        - **Registers**: Group of flip-flops used to store multiple bits, commonly used for holding data or instructions.