Assignment - 5

Q1. Explain the differences between RAM, ROM and Flash memory. Discuss the primary use cases for each type of memory within a computer system and explain why each is suited to its respective purpose?

⇒ RAM (Random Access Memory)

- Volatile: Data is lost when power is off.
- Primary Use: Temporary storage for active programs and data.
- Suitability: Fast read/write speeds make it ideal for applications and OS tasks that need quick access.

2. ROM (Read-Only Memory):

- Non-Volatile: Retains data without power.
- Primary use: Permanent storage for firmware and boot instructions.
- Suitability: Since it's read only (or seldom written to), it's ideal for strong critical, unchanging code.

3. Flash Memory:

- Non-Volatile: Data persists without power, but can be rewritten.
- Primary Use: Storage for data that requires occasional updates, like SSDs and USB drives.
- Suitability: Combines non-volatility with moderate speed, making it practical for data that needs both

Q2. ⇒

be preferred over other I/O other techniques and justify your choice?

⇒ 1. Isolated I/O:
- Separate address space for I/O.
- Pros: Reduces memory space usage for devices.
- Cons: More complex as it requires separate instructions for I/O.

2. Memory-Mapped I/O:
- Shares address space with memory
- Pros: Simplifies addressing as I/O devices used with
- Cons: Reduces available memory address space.

I/O Techniques:
1. Programmed I/O: CPU manages I/O directly; simple but can lead to idle CPU time.
2. Interrupt-Driven I/O: CPU can perform other tasks and is alerted I/O completes; efficient for moderate speed devices.
3. Direct Memory Access (DMA): DMA controller handles data transfer, freeing the CPU for other tasks.

Example:

performance. Using a diagram, illustrate how cache interacts with main memory and the CPU.

⇒ • Cache Memory sits between the CPU and main memory, storing frequency accessed data to speed up retrival.
• Impact on Performance: Cache significantly reduces data access time, as it avoids the slower main memory.
• Memory Hierarchy:
  • CPU (fastest) → L1 Cache → L2 Cache → L3 cache → RAM (main Mem...

Digram

CPU ↔ L1 cache ↔ L2 Cache ↔ Main Memory.

Q4. Explain different cache memory techniques, such as direct mapping, associative mapping, and set-associative mapping. Compare each technique in terms of performance and complexity, and provide examples of how data would be stored in cache under each mapping technique?

⇒ 1. Direct Mapping:
• Mechanism: Each memory block maps to a single cache line.
• Pros: Simple & fast

- Cons: Prone to conflicts if multiple addresses map to the same line.
- Ex:- Address 10 in memory always maps to cache line 10%.

2. Associative Mapping:
- Mechanisms: Any memory cache block can go into any cache line.
- Pros: Reduces conflicts.
- Cons: Slower and more complex due to full search.
- Ex: Addresses 10 can be stored in any available cache line.

3. Set-Associative Mapping:
- Mechanism: Divides cache into sets where each block can go easily into any line within a specific set.
- Pros: Balances b/w flexibility & speed.
- Cons: Moderately complex.
- Example: Address 10 maps to a set, but can fit into any line within that set.

Q5: Compare isolated I/O and memory-mapped I/O in terms of design complexity and their data access. Then, describe the advantages and disadvantages of programmed I/O, interrupt-driven I/O, and direct memory access (DMA) in handling input/output operations. Explain a scenario where DMA would

durability and occasional updates.

Q2. Design a simple memory system using a combination of RAM and ROM chips. Specify how you would organize the memory addresses and manage data storage for efficient access. Describe the factors to consider in terms of read/write process, power, and speed?

⇒ Micro-operations are the low-level steps needed to execute an instruction, forming the basis of the fetch-decode-execute cycle.

RTL for LOAD R1, [Addresses]:

1. fetch:
   - MAR ← PC
   - MDR ← M[MAR]
   - IR ← MDR; PC ← PC+1
2. Decode: Identify LOAD
3. Execute:
   - MAR ← Address
   - MDR ← M[MAR]
   - R1 ← MDR

Q3. Describe the concept of cache memory and explain its role within the memory hierarchy. Discuss the impact of cache memory on system

Assignment 3

1. Define RISC & CISC architectures. What are the key differentiating characteristics b/w the 2?

Sol. RISC is a CPU design philosophy that emphasizes a small, highly optimized instruction set. Features:

a. Simplicity – Fewer, highly optimized instructions
b. Load/Store – Operations are primarily register-based.
c. Pipelining – Designed for efficient instruction execution.
d. Fixed instruction length – Simplifies decoding.

CISC features a more extensive instruction set that includes complex instructions set that includes complex instructions capable of performing multiple operations. Features:

a. Complex Instructions – More extensive & versatile instruction set.
b. Variable Instruction length – Allows for complex operations in fewer instructions.
c. Memory-to-memory Operations – Directly operates on memory.
d. Less Pipelining efficiency – Complexity can hinder pipelining.

Differences –

a. Instruction Set – RISC has a smaller set; CISC has a larger, more complex set.
b. Execution – RISC targets one instruction per cycle, CISC may require multiple cycles.
c. Design Focus – RISC emphasizes simplicity; CISC handles complexity within instructions.
d. Performance – RISC optimizes through pipelining; CISC relies on compiler optimization.

2. Compare & Contrast the instruction sets of a typical RISC & CISC processor. Provide examples of instructions from each architecture.

Sol. RISC characteristics :

a. Simplicity - Fewer, uniform instructions
b. Fixed Length - Typically 32 bits.
c. Load / Store Model - Memory accessed only thru specific load/ store instructions.

eg. a. ADD R1, R2, R3: Adds R2 & R3, stores in R1.
b. LW R1, 0(R2): Loads from memory into R1.
c. SW R1, 0(R2): Stores R1 into memory
d. BEQ R1, R2, Label: Branches if R1 equals R2

CISC characteristics:
a. Larger, more complex set of instructions.
b. Direct memory access in many instructions.
c. Variable instructions length.

eg. a. Mov AX, [BX] (move from memory to register)
b. ADD AX, [BX] (add memory value to register)

Key Differences:
a. Instruction count - RISC has fewer, simpler instructions; CISC has many complex ones.

b. Execution - RISC aims for fast execution; CISC can perform more per instruction but may be slower.

c. Design focus - RISC emphasizes efficiency; CISC emphasizes reducing code size.

3. Explain the philosophy behind the RISC approach, emphasizing simplicity & reduced instruction complexity. How does this philosophy impact instruction execution & performance?

Sol. The RISC philosophy focuses on simplicity & efficiency thru a small, optimized set of instructions. Key principles:
a. Simplicity - Fewer, easier-to-implement instructions streamline design
b. Fixed instruction length - Uniform instruction size simplifies decoding & enhances pipelining.
c. Load / Store architecture - Only specific instructions access

memory, reducing complexity.
d. Emphasis on registers - More general-purpose registers minimize memory access, speeding up computations.
e. Pipelining - The straight forward design allow for more efficient instruction processing in parallel.

Impact on performance:
a Faster Execution
b. Improved Pipelining
c. Higher throughput
d. Easier Optimization
e. Reduced Complexity.

4. Discuss the advantages of CISC architectures, including the ability to perform complex operations with a single instruction. Give examples of situations where CISC architectures might be advantageous.

Sol. Advantages:
a. Complex Instructions - CISC can perform complex operations with a single instruction, reducing the no. of instructions needed.
b. Memory efficiency - More compact code can lead to reduced memory usage, beneficial in limited-memory environments.
c. Reduced Code Size - Fewer instructions can improve cache utilization & overall performance.
d. Ease of high-level language mapping - CISC makes it easier to translate high-level constructs into machine code.
e. Backward Compatibility - Supports older instructions, facilitating the use of legacy software.

Situations where CISC is advantageous:
a. Embedded Systems - Ideal for resource-constrained applications due to reduced code size.
b. Legacy Software - Maintains compatibility with older applications in critical industries.

c. Complex Calculations - Efficient for mathematical operations & data manipulation.

d. Text Processing - Handles string operations with fewer instructions.

e. Development Simplicity - Simplifies programming with high-level languages.

5. Analyze the impact of RISC & CISC architectures on the design of compilers & assembly language programming. How do these architectures affect code generation & optimization?

Sol. RISC Architecture:

Impact on compilers -

a. Simplicity - RISC's limited instructions set simplifies code generation, mapping high-level constructs easily.

b. Register Emphasis - Compilers focus on efficient register allocation to minimize memory access.

c. Instruction Scheduling - Advanced scheduling techniques optimize instruction order to pipelining.

d. Uniform length - Simplifies parsing & optimization processes

Impact on assembly language:

a. Straight: RISC assembly is simpler, with one instruction per operation, making it easier to learn.

b. Optimization awareness - programmers need to consider register usage & pipeline efficiency.

CISC architecture:

Impact on compilers:

a. Complex Instructions - CISC allows for fewer but more complex instructions, simplifying code for high-level operations.

b. High-Level Mapping - Compilers can map complex constructs directly to single instructions.

c. Less focus on registers - More reliance on memory operations can reduce the emphasis on register management.

d. Complex code generations - Requires sophisticated logic for optimal instruction selection.

Impact on assembly language:
a. Rich Instruction Set - CISC assembly is more complex with many operations possible in single instructions
b. Efficiency Considerations - Careful use of complex instructions is necessary to avoid inefficiencies.
Code Generation Optimization:
RISC - Generates more, simpler instructions; Optimization focuses on scheduling & register use
CISC - Generates fewer, complex instructions; optimization involves careful instruction selection.

6. Given a real-world application (eg. mobile devices, servers, embedded systems), justify whether a RISC or CISC, architecture would be more suitable, considering factors like power efficiency, performance & code size.
Sol Application : Mobile Devices
RISC Architecture Justification:
a. Power efficiency - RISC is designed for low power consumption, ideal for battery-operated devices.
b. Performance - Focus on pipelining & efficient execution of simple instructions leads to high responsiveness for mobile tasks like browsing & gaming.
c. Code Size - While RISC may generate more instructions, modern compilers optimize this, mitigating impact on code size.
d. Ecosystem Support: ARM dominates the mobile market, providing extensive developer support & optimized libraries.

Comparision with CISC:
a. Power Consumption - CISC generally consumes more power, making it less suitable for mobile devices.
b. Performance : CISC's complex instructions may offer some efficiency but at higher energy costs & heat generation.
c. Code Size - CISC can reduce code size, but RISC's efficiency has diminished this advantage in mobile applications.

7. Describe the evolution of microprocessors from early CISC designs to more modern RISC hybrid architectures. What were the driving forces behind these changes?

Sol Evolution of microprocessors:

Early CISC designs (1970s - 1980s) -

Characteristics - CISC architectures like x86 used complex instruction sets of multiple operations.

Driving forces - a. Hardware limitations - complex instructions maximized efficiency with limited resources.

b. Software Compatibility - Extensive instruction sets eased transitions from older systems.

Rise of RISC (1980s - 1990s):

Characteristics - RISC architectures focused on a small set of simple, fast instructions.

Driving forces - a. Performance - Simplicity allowed for efficient pipelining.

b. Power efficiency - Low power consumption suited mobile & embedded systems.

c. Technological advances - More transistors enabled complex features without CISC.

Hybrid architectures (2000s - Present):

Characteristics - Modern processors combine RISC & CISC principles, like x86 using RISC-like operations.

Driving forces - a. Performance & efficiency - Demands for high-performance & low power led to hybrid designs.

b. Diverse applications - Varied computing needs required flexible architectures.

c. Market Competitions - Innovation driven by competition among manufacturers.