Learning Objectives (Key Points)

A1.1.1 CPU components – CU, ALU, registers, buses, cores.
A1.1.2 GPU role – graphics rendering, parallel processing, AI, mining.
A1.1.3 CPU vs GPU – design, architecture, memory, applications.
A1.1.4 Primary memory – RAM, ROM, cache, registers.
A1.1.5 Fetch–decode–execute cycle – stages, registers, buses.
A1.1.6 Secondary storage – HDD, SSD, optical, cloud.
A1.1.7 Input devices – keyboards, sensors, scanners, specialized devices.
A1.1.8 Output devices – monitors, printers, speakers, specialized devices.
A1.1.9 Embedded systems – definition, uses, pros/cons.
A1.1.10 Networking hardware – routers, switches, access points.
A1.1.11 System software – OS, utilities, drivers.
A1.1.12 Application software – productivity, creative, specialist.
A1.1.13 Virtualization – purpose, benefits, limitations.
A1.1.14 Cloud computing – IaaS, PaaS, SaaS, advantages/challenges.
A1.1.15 Emerging technologies – AI, IoT, quantum, edge computing.

A1.1.1 Function and interaction of the main central processing unit components

Bullet Points

- CPU = “brain” of the computer, performs most processing.
- Main CPU units:
  - Control Unit (CU) – directs processor operations, fetch–decode–execute cycle.
  - Arithmetic Logic Unit (ALU) – carries out arithmetic and logical operations.
  - Registers – small, fast storage inside CPU (IR, PC, MAR, MDR, AC).
- Buses connect CPU, memory, and peripherals:
  - Control bus – manages signals (read, write, interrupt).
  - Data bus – carries actual data.
  - Address bus – carries memory addresses.
- CPU configurations: single-core, multi-core, co-processor.

The **central processing unit (CPU)** is the part of the computer that interprets and executes instructions. It is often called the “brain” because all key operations depend on it.

1. Control Unit (CU)

- The CU acts like a manager inside the CPU.

- Its primary role is to **direct the flow of data and instructions** between the ALU, registers, memory, and input/output devices.
- It ensures the CPU follows the **fetch–decode–execute cycle**:
    - **Fetch:** Collect the next instruction from memory.
    - **Decode:** Interpret the instruction so the CPU knows what to do.
    - **Execute:** Carry out the instruction by activating the correct components.
- Example: If the instruction is an addition, the CU sends the required data to the ALU and then directs where the result should be stored.

2. Arithmetic Logic Unit (ALU)

- The ALU performs all mathematical and logical operations.
- **Arithmetic operations:** addition, subtraction, multiplication, division.
- **Logical operations:** AND, OR, NOT, XOR.
- Results are often stored in the **accumulator (AC)**, a special register, for quick access.
- Without the ALU, the CPU would not be able to process data, only move it around.

3. Registers

Registers are **tiny storage areas inside the CPU**. They are faster than RAM because they are built directly into the processor. Their purpose is to hold data that the CPU is actively using.

- **Instruction Register (IR):** Holds the current instruction being executed.

- **Program Counter (PC):** Keeps track of the address of the next instruction to fetch. Automatically increments after each fetch.
- **Memory Address Register (MAR):** Stores the address in memory where data is to be fetched from or written to.
- **Memory Data Register (MDR):** Holds the actual data that has been fetched from or is about to be written to memory.
- **Accumulator (AC):** Stores temporary results from the ALU, such as intermediate arithmetic or logic outcomes.

Registers are crucial because they allow the CPU to work at very high speeds without waiting for slower main memory.

4. Buses

Think of buses as **communication highways** in the computer. They connect the CPU with memory and input/output devices.

- **Control Bus:** Carries control signals like “read”, “write”, “interrupt”, and “clock”. It is **bidirectional** (signals can go both ways).
- **Data Bus:** Transfers the actual data (numbers, instructions). It is usually **bidirectional** because data must be both read and written.
- **Address Bus:** Carries the memory address the CPU wants to access. It is usually **unidirectional** (CPU to memory). The width of the address bus determines how many memory locations can be addressed. For example, a 32-bit address bus can address 2^{32} memory locations.

Together, buses ensure that instructions and data move efficiently between the CPU and other parts of the computer.

5. CPU Configurations

- **Single-core processor:** One processing unit. Can handle one instruction stream at a time. Adequate for basic tasks but slow at multitasking.
- **Multi-core processor:** Two or more cores in a single CPU. Each core can handle its own instruction stream. Improves multitasking and performance-heavy tasks. However, performance gain depends on whether software is written to use multiple cores.
- **Co-processors:** Special-purpose processors designed to help the CPU with certain types of work. Examples:
    - GPU (graphics processing unit) for image rendering.
    - Audio processors for sound.
    - Digital signal processors (DSPs) for telecommunications.

A common misconception is that doubling cores automatically doubles performance. In reality, it depends on how well software can use multiple cores and other factors like memory speed.

Practice Questions (with IB Command Terms)

1. **Define** the function of the control unit (CU) in the CPU.
2. **Outline** the role of the arithmetic logic unit (ALU) in processing instructions.
3. **Describe** the purpose of the program counter (PC) during the fetch–decode–execute cycle.
4. **Explain** how the memory address register (MAR) and memory data register (MDR) work together to access memory.
5. **Compare** the roles of the data bus and address bus.

6. **Discuss** why adding more cores to a CPU does not always guarantee faster performance.
7. **Analyze** the importance of registers for CPU speed compared to relying only on RAM.
8. **Evaluate** the significance of co-processors such as GPUs in enhancing overall system performance.

A1.1.2 Role of a graphics processing unit

Bullet Points

- A GPU is a specialized processor designed for rapid image, video, and animation rendering.
- Originally built for video games, GPUs are now used in AI, machine learning, and cryptocurrency mining.
- GPUs have a highly parallel structure with thousands of smaller cores.
- Key uses:
    - Rendering 3D graphics (shaders, textures, lighting).
    - Video processing (encoding, decoding).
    - Machine learning and artificial intelligence (matrix and vector calculations).
    - Cryptocurrency mining (proof-of-work calculations).
- Offloads tasks from CPU, improving system performance.

The **graphics processing unit (GPU)** is a specialized chip created to handle visual tasks like drawing images, videos, and animations. While the CPU can handle these tasks, GPUs are **optimized for repetitive and large-scale calculations**, which makes them much faster for graphics and similar workloads.

1. GPU Structure and Parallel Processing

- Unlike CPUs, which usually have a small number of powerful cores, GPUs contain **thousands of smaller, simpler cores**.
- These cores are arranged to work in parallel, meaning they can carry out the same operation on many pieces of data at once.
- This makes GPUs excellent for processing large data sets, especially when the same calculation needs to be repeated across many pixels, images, or datasets.

2. Traditional Role in Graphics

- **Shaders and Textures:**
    - Shaders are small programs that determine how pixels, lighting, and effects appear in a scene.
    - Textures are image patterns applied to 3D models to give surfaces detail (e.g., wood, metal, skin).
- The GPU applies shaders and textures to 3D models, calculates lighting and shadows, and renders realistic scenes in video games or animations.

3. Video Processing

- GPUs are highly efficient at **video decoding and encoding**.

- Tasks like playing a high-resolution 4K video or editing and rendering video footage require rapid processing of large amounts of data.

- By offloading these tasks from the CPU, the GPU allows smoother video playback and faster rendering.

4. GPUs in Artificial Intelligence and Machine Learning

- Many machine learning and AI models rely on **matrix and vector operations**.

- GPUs are far better at these operations than CPUs because of their parallel structure.

- This allows researchers to train deep learning models much faster.

- Today, GPUs are essential tools in fields like self-driving cars, natural language processing, and medical image analysis.

5. GPUs in Cryptocurrency Mining

- In the early 2010s, miners discovered GPUs could solve **proof-of-work puzzles** (like Bitcoin hashing) much faster than CPUs.

- This made GPUs the standard for mining cryptocurrencies for many years.

- However, newer **application-specific integrated circuits (ASICs)** have replaced GPUs in large-scale mining, since they are more efficient for this specific task.

6. Contribution to System Performance

- By taking over resource-heavy tasks such as rendering and AI calculations, GPUs free up the CPU to focus on other operations.

- This collaboration between CPU and GPU results in overall smoother and faster system performance.

Practice Questions

1. Define the role of a graphics processing unit in a computer system.
2. Outline how shaders and textures improve the realism of 3D graphics.
3. Explain why GPUs are better than CPUs for artificial intelligence and machine learning tasks.
4. Describe how GPUs contribute to smooth playback and editing of high-resolution videos.
5. Discuss how cryptocurrency mining led to a surge in GPU demand, and why this trend has slowed in recent years.
6. Analyze how the GPU and CPU working together can enhance gaming performance.

A1.1.3 Differences between the CPU and the GPU

Bullet Points

- CPU = general-purpose processor; GPU = specialized processor.
- CPU has fewer but more powerful cores; GPU has thousands of smaller cores.
- CPU excels at sequential tasks requiring quick decisions; GPU excels at parallel tasks with repetitive operations.

- CPU uses small, high-speed caches; GPU uses high-bandwidth VRAM.
- CPU is versatile, running the OS and general software; GPU is optimized for graphics, rendering, video, and AI.
- Together, CPU and GPU complement each other for high-performance computing (e.g., gaming, simulations).

The **central processing unit (CPU)** and the **graphics processing unit (GPU)** are both essential in modern computers, but they are designed with different goals in mind. Understanding these differences explains why some tasks run better on the CPU, while others are faster on the GPU.

1. Design Philosophy

- **CPU (general-purpose processor):**
  - Designed to handle a wide range of tasks.
  - Executes operating system instructions, user commands, and most application logic.
  - Prioritizes flexibility and decision-making.
- **GPU (specialized processor):**
  - Built to perform specific types of calculations extremely efficiently.
  - Focused on parallel processing of large amounts of data, such as graphics rendering.
  - Ideal for tasks where the same operation must be performed many times simultaneously.

2. Core Architecture

- **CPU cores:**
    - Few in number (commonly 4–16 in consumer devices, higher in servers).
    - Each core is powerful, capable of handling complex, varied instructions.
    - Supports features like branch prediction and out-of-order execution to improve performance.
- **GPU cores:**
    - Thousands of smaller, simpler cores.
    - Each core is not as powerful as a CPU core, but together they perform massive parallel computations.
    - This is why GPUs are better for tasks such as image rendering, where thousands of pixels need to be processed simultaneously.

3. Memory Access

- **CPU:**
    - Relies on a small but extremely fast **cache memory** (L1, L2, L3).
    - Optimized for frequently accessing small amounts of data quickly.
    - Works best with irregular tasks where quick switching is required.
- **GPU:**
    - Uses its own memory called **VRAM (Video RAM)**.

    - VRAM has high bandwidth, allowing large amounts of data (such as frames, textures, or training data for AI) to move rapidly.
    - Essential for tasks that involve massive datasets.

4. Power Efficiency

- **CPU:**
    - Consumes less power for general tasks.
    - Better suited for long, mixed workloads like running operating systems or business applications.
- **GPU:**
    - Consumes more power, especially when rendering graphics or running AI models.
    - The higher power demand comes from handling thousands of simultaneous operations.

5. Practical Applications

- **CPU best for:**
    - Operating system tasks.
    - Running everyday software (word processors, browsers).
    - Decision-heavy operations that require versatility.
- **GPU best for:**
    - Rendering 3D graphics and video processing.

    - Scientific simulations, machine learning, and deep learning.
    - Cryptocurrency mining (though now often replaced by ASICs).

6. How CPU and GPU Work Together

- In gaming, the **CPU** processes game logic (rules, AI, physics, player inputs).
- The **GPU** simultaneously renders the visuals (3D models, textures, lighting, shadows).
- Together, they create a smooth and responsive gaming experience.

Practice Questions

1. Define the difference between the CPU's role and the GPU's role in a computer system.
2. Outline why GPUs have more cores than CPUs, and explain the advantage of this design.
3. Describe how the memory architecture of a CPU differs from that of a GPU.
4. Explain why CPUs are better for sequential tasks, while GPUs excel in parallel processing.
5. Compare how CPUs and GPUs handle tasks in video gaming.
6. Analyze how both CPU and GPU contribute to improving performance in scientific computing or AI applications.

A1.1.4 Purposes of different primary memory types

Bullet Points

- Primary memory = memory directly accessible by the CPU.
- Four main types:
  - RAM – active data and instructions, volatile.
  - ROM – permanent instructions, non-volatile.
  - Cache (L1, L2, L3) – speeds up access between RAM and CPU.
  - Registers – smallest, fastest storage inside CPU.
- Each type has a distinct purpose, contributing to efficient CPU operation.

Primary memory is the set of memory units that the CPU can access directly without delay. Unlike secondary storage (such as SSDs or HDDs), primary memory is optimized for **speed and immediate accessibility**. Each type plays a specific role in ensuring the CPU operates smoothly.

---

1. Random Access Memory (RAM)

- **Function:**
  - RAM stores programs and data currently in use.
  - When you open software, it is loaded into RAM so the CPU can access it instantly.

- **Volatility:**
  - RAM is volatile, meaning contents are erased when power is lost.
  - This is why saving work to secondary storage is essential.
- **Impact on performance:**
  - More RAM allows more programs to run simultaneously without slowing down.
- **Example:**
  - In a smartphone, switching between apps is quick because apps remain in RAM until closed or replaced.

---

2. Read-Only Memory (ROM)

- **Function:**
  - ROM holds critical instructions that rarely change.
  - It stores the **BIOS (Basic Input/Output System)** or firmware, which runs self-tests and loads the operating system when the device powers on.
- **Volatility:**
  - ROM is non-volatile, so it retains its data even when power is off.
- **Modern variation:**
  - Many computers now use **flash memory**, a form of ROM that can be updated (e.g., BIOS updates).

- **Example:**
    - On a phone, ROM stores the operating system so it is always available during startup.

---

3. Cache Memory (L1, L2, L3)

- **Purpose:**
    - Cache acts as a **middle layer** between slow RAM and the fast CPU.
    - It stores recently or frequently used instructions/data so the CPU can avoid fetching them repeatedly from RAM.
- **Levels:**
    - **L1 cache:**
        - Smallest (32KB–128KB), but fastest.
        - Located directly on the CPU core.
        - Often split into instruction (L1i) and data (L1d) cache.
    - **L2 cache:**
        - Larger (256KB–2MB), slightly slower.
        - Usually dedicated per core or very close to the CPU.
    - **L3 cache:**
        - Largest (2MB–64MB), slowest of the three.

        - Shared among multiple CPU cores.

- **Cache hits and misses:**
  - A **cache hit** occurs when the CPU finds needed data in cache → very fast access.
  - A **cache miss** occurs when the data is not there → CPU must fetch from RAM, which is slower.
- **Analogy:**
  - Cache = notes on your desk (quick access).
  - RAM = bookshelf in your room (slower but nearby).
  - Hard drive = library across town (much slower).

---

4. Registers

- **Function:**
  - Registers are the **fastest and smallest form of memory**, built directly into the CPU.
  - They store data and instructions that are actively being processed at that moment.
- **Types of registers:**
  - Program Counter (PC) – tracks the next instruction’s memory address.
  - Instruction Register (IR) – stores the instruction being executed.

    - Memory Address Register (MAR) – stores the memory address to access.
    - Memory Data Register (MDR) – stores the actual data being transferred to/from memory.
    - Accumulator (AC) – stores intermediate results from the ALU.
- **Importance:**
    - Registers allow the CPU to execute instructions at maximum speed by avoiding delays from RAM.
- **Size:**
    - Very small in capacity (measured in bytes or words), but critical for efficiency.

Practice Questions

1. Define the purpose of RAM and explain why it is considered volatile.
2. Describe how ROM contributes to the boot-up process of a computer system.
3. Explain the advantages of cache memory and distinguish between L1, L2, and L3 cache.
4. Outline the role of registers in CPU operations, giving examples of different types.
5. Compare RAM and cache in terms of size, speed, and purpose.
6. Discuss how cache hits and cache misses affect CPU performance.

7. Analyze why registers are considered essential even though RAM already exists.

A1.1.5 The fetch–decode–execute cycle

Bullet Points

- The fetch–decode–execute cycle is the fundamental process that allows a CPU to run programs.
- Steps:
  - Fetch – CPU gets the instruction from memory.
  - Decode – CPU interprets the instruction.
  - Execute – CPU performs the required action.
- Registers involved: PC, MAR, MDR, IR, AC.
- Buses involved: address bus, data bus, control bus.
- This cycle repeats continuously as long as the computer is running.

The **fetch–decode–execute cycle** (also called the **instruction cycle**) is the repetitive sequence of steps the CPU follows to run every instruction in a program. Without this cycle, computers would not function because no instructions would ever be carried out.

---

1. Fetch Stage

- The **Program Counter (PC)** holds the memory address of the next instruction.
- This address is copied into the **Memory Address Register (MAR)**.
- The address bus carries this location to the memory.
- The control bus sends a **read signal** to request the instruction.
- The instruction is sent back from memory via the data bus and temporarily stored in the **Memory Data Register (MDR)**.
- The instruction is then copied into the **Instruction Register (IR)** for processing.
- Meanwhile, the PC increments automatically so it points to the next instruction.

---

2. Decode Stage

- The instruction in the IR is sent to the **Control Unit (CU)**.
- The CU interprets the instruction:
  - If it is an arithmetic operation, it signals the ALU.
  - If it requires data from memory, it sets up the MAR and MDR accordingly.
  - If it is a control operation (like branching), it adjusts the PC.
- The decoding process ensures that the CPU understands what the instruction means before execution.

---

3. Execute Stage

- The CPU carries out the instruction as directed by the CU.
- Possible actions include:
    - Performing arithmetic or logic operations in the **ALU**.
    - Storing results in the **Accumulator (AC)** or memory.
    - Moving data between registers.
    - Sending data to an output device.
    - Loading data from input or memory.
- Once execution is complete, the CPU returns to the fetch stage to process the next instruction.

Example with Little Man Computer (LMC)

The LMC is a simplified model to help visualize the cycle.

- **Instruction example:** `LDA 4` (Load the value at memory address 4 into the accumulator).
    1. Fetch: PC gives the address of the instruction (0). MAR = 0, memory sends instruction 504 to MDR → IR. PC increments to 1.
    2. Decode: CU interprets 5 as "LDA" (load) and 04 as the address.
    3. Execute: The CPU accesses memory location 4, retrieves its value, and stores it in the AC.

The cycle repeats until a **HLT (halt)** instruction is reached, which stops processing.

Role of Buses

- **Address Bus:** Used during fetch to specify the location in memory.
- **Data Bus:** Transfers instructions and data to and from the CPU.
- **Control Bus:** Sends read/write commands, clock pulses, and halt signals.

Importance of the Cycle

- The fetch–decode–execute cycle ensures **orderly and efficient processing** of instructions.
- By automatically updating the PC during the fetch stage, the CPU always knows where the next instruction is, ensuring continuous operation.
- Without this cycle, the CPU could not function as a general-purpose processor.

Practice Questions

1. Define the fetch–decode–execute cycle and outline its three stages.
2. Describe the role of the program counter during the fetch stage.
3. Explain how the memory address register (MAR) and memory data register (MDR) work together during the fetch cycle.

4. Illustrate how the instruction register (IR) is used in the decode stage.
5. Compare the actions carried out in the execute stage when the instruction is arithmetic versus when it is a control instruction.
6. Analyze why updating the program counter during fetch (not after execute) is critical for smooth CPU operation.
7. Discuss how the address bus, data bus, and control bus are all used during one cycle.
8. Evaluate the usefulness of models like the Little Man Computer in understanding how real CPUs work.