Ministry of Education and Science of Ukraine
Lutsk National Technical University

COMPUTER ARCHITECTURE AND MICROPROCESSOR DEVICE PROGRAMMING

Lecture Notes
for applicants of the first (bachelor's) level of higher education
educational program
«Computerized Telecommunication Networks»
field of knowledge 17 – Electronics, Automation, and Electronic Communications
172 – Electronic Communications and Radio Engineering
for full-time and part-time forms of study

Lutsk – 2025

UDC 004.2(07)
C - 73
The electronic copy of the printed edition has been submitted for inclusion in the repository of Lutsk National Technical University
Library Director: _____ Nataliia POLISHCHUK
Recommended for publication by the Academic Council of the Faculty of Computer and Information Technologies of LNTU, protocol No. __ dated «_____» _____ 2025.
Head of the Academic Council of FCIT: _____ Inna KONDIUS
Reviewed and approved at the meeting of the Department of Electronics and Telecommunications of LNTU, protocol No. ___ dated «___» _____ 2025.

Head of the Department of Electronics and Telecommunications:	_____	Valentyn ZABLOTSKYI, Ph.D. in Engineering, Associate Professor, LNTU
Compiled by:	_____	Mykola KHVYSHCHUN, Ph.D. in Physics and Mathematics, Associate Professor of the Department of Electronics and Telecommunications, LNTU
Reviewer:	_____	Nataliia LISHCHYNA, Ph.D. in Engineering, Associate Professor, Head of the Department of Software Engineering, LNTU
Responsible Editor:	_____	Valentyn ZABLOTSKYI, Ph.D. in Engineering, Associate Professor, Head of the Department of Electronics and Telecommunications, LNTU

C-73 Computer Architecture and Microprocessor Device Programming. Lecture notes for students of the first (bachelor's) level of higher education of the educational program «Computerized Telecommunication Networks» of the field of knowledge 17 – Electronics, Automation and Electronic Communications, specialty 172 – Electronic Communications and Radio Engineering, for all forms of study.Compiled by M.V. Khvyshchun, Lutsk: LNTU, 2025. – 56 p.
The publication presents lecture topics of the course «Computer Architecture and Microprocessor Device Programming», intended for students of the first (bachelor's) level of higher education in the educational program «Computerized Telecommunication Networks».

CONTENTS

INTRODUCTION

The course «Computer Architecture and Microprocessor Device Programming» provides students with comprehensive knowledge of how modern computer systems are structured, how microprocessors function, and how to program these devices effectively. This course is designed for undergraduate students specializing in electronics, telecommunications, or computer engineering, and aims to bridge the gap between theoretical foundations of computer hardware and practical programming skills.

Students will begin by exploring the fundamental components of computer architecture, including the CPU, memory, buses, and peripheral interfaces. Special attention will be given to the internal structure and operation of microprocessors and microcontrollers, with practical implementation using both low-level assembly language and high-level programming in C.

By the end of the course, students will have gained essential skills for designing and programming embedded systems, understanding real-time computing constraints, and integrating hardware and software components in modern computing environments.
This course lays the foundation for more advanced studies in embedded systems, digital design, and real-time application development.

Topic 1. Introduction to Computer Architecture

Purpose of the Lecture: to introduce students to the fundamental concepts of computer architecture, including definitions, the difference between architecture and organization, historical evolution, and the classification of computing systems.

1. What is Computer Architecture? Computer architecture is the conceptual design and fundamental operational structure of a computer system. It includes:

1. Instruction Set Architecture (ISA). The part visible to the programmer; defines operations, registers, addressing modes.
2. Microarchitecture. The hardware implementation of the ISA.
3. System Architecture: Interconnections of components (CPU, memory, I/O).

Architecture refers to what the computer does.Organization refers to how it does it.

2. Levels of Abstraction in Computer Systems.

Level	Description
Application Programs	Software running on the system
High-Level Language	Programming languages like C, Python
Assembly Language	Human-readable machine instructions
Machine Language	Binary instructions executed by hardware
Microarchitecture	Internal CPU components
Logic Gates	Built using AND, OR, NOT
Transistors	Physical components on chips

3. Historical Evolution of Computer Systems.

Generation	Technology Used	Key Characteristics	Example Devices
1	2	3	4
1st	Vacuum Tubes	Large, fragile, expensive	ENIAC
2nd	Transistors	Smaller, more reliable	IBM 1401

1	2	3	4
3rd	Integrated Circuits	Compact, energy-efficient	PDP-11
4th	Microprocessors	Single-chip CPU	Intel 4004, Apple I
5th	AI-based Architectures	Parallelism, intelligent systems	Quantum computers (early)

4. Classification of Computers.

a) By Size and Capability:

Type	Description	Example
Supercomputer	High-performance parallel systems	IBM Summit, Fugaku
Mainframe	Large-scale enterprise servers	IBM zSeries
Minicomputer	Mid-range, now largely obsolete	DEC VAX
Microcomputer	Personal and embedded devices	PCs, Raspberry Pi, laptops

b) By Architecture:

Architecture	Memory Structure	Use Case
Von Neumann	Shared instruction/data	General-purpose computers
Harvard	Separate instruction/data	Embedded and signal processors
RISC	Reduced instruction set	ARM, microcontrollers
CISC	Complex instruction set	Intel x86, AMD

5. Key Terms Defined:

1. CPU: Executes program instructions.
2. ALU: Arithmetic Logic Unit.
3. Register: Temporary data storage inside the CPU.
4. Bus: Pathway for data transfer.
5. Instruction Set: Collection of machine instructions a CPU can execute.

Literature

1. Hennessy, J., & Patterson, D. (2022). Computer Architecture: A Quantitative Approach. Morgan Kaufmann. 936 p.
2. Stallings, W. (2023). Computer Organization and Architecture. Pearson Education. 880 p.

Topic 2. Computer Structure – Main Components

Purpose of the Lecture: to provide an in-depth understanding of the physical and logical organization of a computer system, focusing on its major components, their functions, and how they work together to process information. This knowledge serves as the foundation for understanding how software interacts with hardware.

1. Overview of Computer Structure. A computer system is composed of a combination of hardware components that are responsible for executing software instructions and processing data.

These components include:

1. Central Processing Unit (CPU): the brain of the system, responsible for executing instructions.
2. Memory: stores data and instructions for processing.
3. Input / Output (I / O) Devices: Interface through which data enters and exits the system.
4. Storage Devices: Preserve data and programs permanently.
5. System Buses: electrical pathways that transfer data and signals among components.

All these components are integrated onto the motherboard, which acts as the backbone of the computer.

3. Basic Components of a Computer

Component	Description
1	2

1	2
CPU	Executes arithmetic and logic instructions, manages system operations.
Memory (RAM)	Temporarily holds data and instructions that the CPU uses during operation.
Storage (HDD/SSD)	Stores programs, files, and operating systems permanently.
Motherboard	The main PCB that houses and connects all essential system components.
Input Devices	Allow users to input data (e.g. keyboard, mouse, scanner).
Output Devices	Display results and feedback (e.g. monitor, printer, speakers).
Power Supply Unit (PSU)	Converts high-voltage AC to low-voltage DC required by computer components.

3. CPU – The Central Processing Unit. The CPU is a complex chip with multiple subcomponents that carry out the processing of data. It includes:

1. Arithmetic Logic Unit (ALU): performs mathematical and logical operations.
2. Control Unit (CU): directs operations inside the CPU, sending control signals.
3. Registers: small, fast storage locations used during execution.

The CPU constantly communicates with memory and I / O devices via buses. Its speed is measured in GHz and directly impacts system performance.

4. Memory. Memory is a volatile component used for temporary storage. It holds both the instructions that the processor needs and the data being processed. The primary memory types include:

1. RAM (Random Access Memory): stores active programs and data. Data is lost when power is off.
2. Cache: Ultra-fast memory located close to or inside the CPU. It reduces access time by storing frequently used data.

Type	Description
DRAM	Dynamic RAM; slower, used for main system memory
SRAM	Static RAM; faster, more expensive, used in CPU cache

The amount and type of RAM influences how many applications can run simultaneously.

5. Storage. Storage devices retain data when the computer is turned off. Two main types are:

1. HDD (Hard Disk Drive): uses spinning disks and magnetic heads. Good for bulk storage but slower access.
2. SSD (Solid-State Drive): uses flash memory. Faster, more durable, and uses less energy.

Feature	HDD	SSD
Speed	100–200 MB/s	500–5000 MB/s
Durability	Mechanical parts prone to damage	No moving parts, shock resistant
Noise	Audible spinning noise	Silent operation
Cost	Lower cost per GB	Higher cost per GB

6. Motherboard. The motherboard is the central PCB that connects all components:

1. Contains sockets for CPU, RAM, and expansion cards (GPU, sound cards).
2. Includes chipset responsible for data transfer among CPU, memory, and peripherals.
3. Provides interfaces such as SATA, USB, Ethernet, and audio ports.

A well-designed motherboard influences system stability and upgradability.

7. Input and Output Devices.

Input Devices: devices like the keyboard, mouse, microphone, and scanner allow users to provide commands or input data.

Output Devices: devices such as monitors, printers, and speakers present processed data to the user.

These devices connect via standardized interfaces like USB, HDMI, audio jacks, and wireless technologies (Bluetooth, Wi-Fi).

8. System Buses. Buses are electrical pathways that connect different components and allow communication:

1. Data Bus: transfers binary data between CPU and memory or I / O.
2. Address Bus: carries the memory addresses of data.
3. Control Bus: Sends control signals (e.g., read/write instructions).

Modern buses include PCI Express (PCIe), SATA, USB, and front-side bus (FSB).

9. Power Supply and Cooling.

Power Supply Unit (PSU):

1. Converts 230V or 110V AC to DC voltages needed by components (12V, 5V, 3.3V).
2. Must be rated for sufficient wattage to support the entire system.

Cooling Mechanisms:

1. Passive Cooling: Heat sinks that dissipate heat.
2. Active Cooling: Fans, liquid cooling systems.
3. Critical for maintaining performance and prolonging hardware life.

10. Summary:

1. A computer is a combination of subsystems working together to execute instructions and process data.
2. The CPU, memory, storage, motherboard, I/O devices, and power supply form the system's hardware foundation.
3. Effective communication via buses and proper cooling are essential for efficient operation.
4. Understanding component roles helps in troubleshooting, upgrading, and programming systems.

Literature

1. Andrews J., A+ Guide to Hardware: Managing and Maintaining PCs, Cengage Learning, 2022. 752 p.
2. Necaise R., Computer Hardware: Installation, Interfacing, Troubleshooting and Maintenance, Wiley, 2023 512 p.

Topic 3. Microprocessors – Architecture and Types

Purpose of the Lecture: to explain the structure and classification of microprocessors, their evolution over time, the difference between RISC and CISC architectures, and the role of embedded and multicore processors in modern computing systems. This lecture establishes a fundamental understanding required for later topics on programming and integration of microcontrollers.

1. What is a Microprocessor? A microprocessor is an integrated circuit (IC) designed to execute a set of instructions. It serves as the central computing unit in modern computers and embedded systems.

Key functions of a microprocessor:

1. Executes arithmetic and logical operations.
2. Controls data flow between memory and I/O.
3. Manages program execution and control signals.

A microprocessor is the core of a computer's Central Processing Unit (CPU) on a single chip.

2. Internal Architecture of a Microprocessor Microprocessors typically consist of the following components:

1. ALU (Arithmetic Logic Unit): Performs mathematical and logic operations.
2. CU (Control Unit): Directs all operations by interpreting instructions.
3. Registers: Temporary high-speed storage used during processing.
4. Bus Interface: Connects the processor to system memory and peripherals.
5. Clock Unit: Regulates instruction cycles.

3. Evolution of Microprocessors. Microprocessors have evolved through several generations:

Generation	Bit Width	Notable Example	Characteristics
1	2	3	4
1st	4-bit	Intel 4004	Basic computation, slow, minimal memory

1	2	3	4
2nd	8-bit	Intel 8080, Zilog Z80	Improved speed, wider data paths
3rd	16-bit	Intel 8086	Introduction of segmented memory
4th	32-bit	Intel 80386	Multitasking and protected mode
5th	64-bit	AMD64, Intel Core i7	Multicore, high-speed buses, virtualization

4. Instruction Set Architectures: RISC vs CISC The Instruction Set Architecture (ISA) defines the operations a processor can perform.

1. RISC (Reduced Instruction Set Computer):
   - Uses a small set of simple instructions;
   - Emphasizes performance and efficiency;
   - Used in ARM, MIPS processors.
2. CISC (Complex Instruction Set Computer):
   - Large, complex instruction set;
   - Designed to reduce the number of instructions per program;
   - Used in Intel x86, AMD.

Feature	RISC	CISC
Instruction Size	Fixed	Variable
Execution Time	One clock cycle per instruction	Multiple cycles per instruction
Complexity	Low	High
Energy Efficiency	High	Lower

5. Multicore Processors. A multicore processor contains two or more processing units (cores) on a single chip. Each core can run separate threads or processes simultaneously.

Benefits:

1. Improved performance through parallelism.

2. Lower power consumption per task.
3. Better support for multitasking and multithreaded applications.

Type	Example	Use Case
Dual-core	Intel Core i3	Office, multimedia
Quad-core	AMD Ryzen 5	Gaming, design software
Octa-core	Apple M1 Pro	Video editing, virtualization

6. Embedded Microprocessors. Embedded microprocessors are specialized CPUs designed to perform dedicated tasks in electronic systems.

Common in:

1. Automotive systems.
2. Industrial automation.
3. Consumer electronics.
4. Internet of Things (IoT).

Characteristics:

1. Low power consumption.
2. Real-time processing support.
3. Integrated peripherals (UART, ADC, GPIO).
4. Often used with real-time operating systems (RTOS)

Feature	Embedded Processor	General-purpose Processor
Application Focus	Dedicated task	General computing
OS Requirement	RTOS or bare-metal	Windows, Linux, macOS
Cost and Size	Low	High
Examples	ARM Cortex-M, ESP32	Intel Core i7, AMD Ryzen

7. Case Study: ARM Cortex-M4 The ARM Cortex-M4 is a widely used embedded microprocessor in real-time systems.

Key features:

1. 32-bit RISC architecture.

2. Integrated NVIC (Nested Vectored Interrupt Controller).
3. Thumb-2 instruction set for compact code.
4. Low power modes.
5. Peripheral support: UART, SPI, I2C, ADC.

8. Summary:

1. Microprocessors are the heart of computing systems, executing instructions and managing data flow.
2. They vary by instruction complexity (RISC vs CISC), number of cores, and application type.
3. Multicore processors enhance parallel processing, while embedded processors are optimized for low power and dedicated control.
4. Understanding microprocessor architecture is key to software development and hardware design.

Literature

1. Furber S., ARM System-on-Chip Architecture, Pearson Education. 2023. 512 p.
2. Stokes J., Inside the Machine: An Illustrated Introduction to Microprocessors and Computer Architecture. No Starch Press. 2022. 392 p.

Topic 4. Random Access Memory (RAM): Types and Characteristics

Purpose of the Lecture: To study the structure, types, and functioning of Random Access Memory (RAM), its role in the memory hierarchy, and how its characteristics affect overall computer performance. This topic is essential for understanding how programs are loaded and executed in real-time.

1. Introduction to RAM Random Access Memory (RAM) is a form of volatile memory, meaning it loses its content when the power is turned off. It serves as the computer's short-term memory, temporarily storing data that the CPU actively uses.

Main characteristics of RAM:

1. High-speed access.
2. Read and write capabilities.
3. Volatile nature (non-permanent).

RAM acts as a buffer between the processor and the slower permanent storage (HDD or SSD), significantly increasing the speed at which data can be accessed.

2. Role of RAM in the Memory Hierarchy RAM sits between the CPU Cache (faster) and secondary storage (slower).

Level	Type	Speed	Volatility	Use Case
L1/L2/L3 Cache	Inside CPU	Very High	Yes	Immediate CPU data access
RAM	Main memory	High	Yes	Running programs and data
SSD/HDD	Storage	Low	No	Permanent file storage

3. Types of RAM There are two main types of RAM:

a). DRAM – Dynamic RAM:

- Stores data in capacitors.
- Needs constant refreshing to retain data;
- Cheaper and denser (more capacity);
- Used as main memory in desktops and laptops.

b). SRAM – Static RAM:

- Uses flip-flops to store bits;
- Does not require refreshing;
- Faster but more expensive;
- Used in CPU cache (L1, L2, L3).

Feature	DRAM	SRAM
Speed	Slower	Faster
Cost	Lower	Higher
Power Consumption	Higher (refreshing)	Lower
Use	Main system memory	Processor cache memory

4. RAM Form Factors and Modules RAM is installed as modules (sticks) on the motherboard. The form factors differ depending on the device:

1. DIMM (Dual In-line Memory Module): Used in desktops.
2. SO-DIMM (Small Outline DIMM): Used in laptops.

Modern RAM modules use DDR (Double Data Rate) technology:

Type	Transfer Rate	Typical Use
DDR3	Up to 2133 MT/s	Legacy systems
DDR4	Up to 3200 MT/s	Current desktops/laptops
DDR5	>4800 MT/s	High-performance PCs

5. RAM Timing and Latency RAM performance depends not only on clock speed but also on latency, which is the delay before data transfer begins.

1. CAS Latency (CL): Time between command and data availability.
2. Bandwidth: The volume of data transferred per second.

Lower latency + higher bandwidth = better overall performance.

6. Virtual Memory and RAM When RAM is full, the OS uses a section of the hard drive as virtual memory:

1. Slower than physical RAM.
2. Enables multitasking beyond installed RAM.
3. Implemented using page files or swap partitions.

7. RAM in Embedded Systems In embedded microcontrollers (e.g., ARM Cortex-M), RAM is:

1. Smaller in size (typically KBs).
2. Used for stack, heap, and data.
3. Paired with Flash memory (for program code).

Efficient RAM use is crucial for real-time performance in embedded applications.

8. Summary:

1. RAM is a vital component for temporary data storage during program execution.
2. It balances speed and cost within the memory hierarchy.
3. DRAM is used for main system memory, while SRAM is used for CPU cache.
4. RAM types and specifications (e.g., DDR4, latency) influence system speed.
5. Embedded systems use RAM very efficiently due to hardware limitations.

Literature

1. Jacob B., Ng S., Wang D., Memory Systems: Cache, DRAM, Disk, Morgan Kaufmann. 2023. 880 p.

2. Anderson T., Dahlin M., Operating Systems: Principles and Practice. Recursive Books. 2022. 512 p.

Topic 5. Data Storage Systems – Hard Disk Drives (HDDs) and Solid-State Drives (SSDs)

Purpose of the Lecture: To provide a comprehensive understanding of the structure, principles of operation, advantages, limitations, and comparative performance of Hard Disk Drives (HDDs) and Solid-State Drives (SSDs). The lecture also covers types of interfaces, form factors, practical applications, and best practices for integrating storage systems in modern computing environments.

1. Introduction to Data Storage Data storage devices are essential components of all modern computer systems. They are responsible for storing the operating system, software applications, user data, and system configuration files.

These devices are non-volatile, meaning they retain information even after the power is turned off, in contrast to RAM.

The two most common types are:

1. HDD (Hard Disk Drive): An electromechanical storage device that uses rotating platters and magnetic heads.
2. SSD (Solid-State Drive): A flash-based memory device with no moving parts, providing faster access speeds and higher shock resistance.

Both types use standard communication interfaces to connect to the motherboard such as SATA and NVMe.

2. Hard Disk Drive (HDD)

Structure and Components:

1. One or more circular platters coated with magnetic material.
2. A spindle motor spins the platters at speeds such as 5400 or 7200 RPM.
3. An actuator arm moves the read/write head to specific locations on the platters.

4. Data is written to or read from the magnetic surface by changing magnetic polarity.

Operation: The read/write heads float extremely close to the disk surface and move across it to access data arranged in concentric circles called tracks. Each track is divided into sectors.

Advantages of HDDs:

1. Cost-effective for large storage capacities (1–20 TB or more).
2. Ideal for bulk data storage and archival purposes.

Disadvantages:

1. Slower read/write access due to mechanical movement.
2. Vulnerable to physical damage from vibration or shock.
3. Generates noise and heat.

3. Solid-State Drive (SSD)

Structure and Components:

1. Composed of flash memory chips (NAND).
2. Managed by a built-in controller that handles data placement, wear leveling, and error correction.

Operation. Data is electronically stored and accessed in memory cells. There are no moving parts, which drastically improves access time and reliability.

Types of NAND flash:

1. SLC (Single-Level Cell) – High performance, high cost.
2. MLC (Multi-Level Cell) – Balanced cost/performance.
3. TLC (Triple-Level Cell) – Common in consumer SSDs.
4. QLC (Quad-Level Cell) – Large capacity, lower endurance.

Advantages of SSDs:

1. Fast boot and load times (0.1 ms latency vs ~10 ms for HDD).
2. Shock resistance and low heat generation.
3. Silent operation and lower energy consumption.

Disadvantages:

1. Higher price per GB.

2. Finite number of write/erase cycles (although modern wear-leveling extends lifespan).

4. Performance Comparison.

Feature	HDD	SSD
Read/Write Speed	100–200 MB/s	500–7000 MB/s (NVMe)
Latency	~10 ms	<0.1 ms
Durability	Prone to mechanical failure	High (no moving parts)
Noise	Audible spinning and clicks	Silent
Cost per GB	Low	Higher
Power Efficiency	Higher consumption	Lower power usage

5. Interface Types

1. SATA (Serial ATA): Widely used in both HDDs and older SSDs. Max speed ~600 MB/s.

2. NVMe (Non-Volatile Memory Express): Uses PCI Express lanes, much faster. Designed for modern SSDs.

3. USB: External storage devices use USB 3.0, 3.1, or USB-C connections.

4. SAS (Serial Attached SCSI): Used in enterprise-level servers for high-performance HDDs and SSDs.

6. Form Factors:

1. 3.5-inch HDD: Common in desktops. High capacity.

2. 2.5-inch HDD/SSD: Used in laptops, game consoles.

3. M.2 SSD: Stick-like design, fits directly on motherboard. Can support SATA or NVMe.

4. U.2 SSD: Larger enterprise SSD used in servers.

5. PCIe AIC (Add-In Card): SSD installed into PCIe slot for maximum speed.

7. Use Cases and Applications.

Storage Type	Common Applications
1	2

1	2
HDD	Archiving, backups, CCTV, file servers
SSD	Operating system drive, databases, video editing, gaming

Hybrid Storage Configuration:

- Boot/OS + frequently used programs → SSD;
- Media storage, backups → HDD.

This hybrid model ensures speed and capacity at an affordable cost.

8. Summary:

1. Storage devices are essential for data retention and system operation.
2. HDDs offer large storage capacity at low cost but suffer from mechanical drawbacks.
3. SSDs provide superior performance and durability with higher costs.
4. NVMe SSDs significantly outperform traditional SATA devices in speed.
5. Understanding performance metrics and application needs helps choose the right combination of storage technologies.

Literature

1. Reinsel D., Gantz J., Rydning J., The Digitization of the World From Edge to Core, IDC / Seagate,.2022. 64 p.
2. Chen W., Solid State Drives: Modeling, Analysis, and Applications, Springer. 2023. 372 p.

Topic 6. Graphics Cards and Video Adapters

Purpose of the Lecture: To explain the structure, functionality, types, and applications of graphics cards and video adapters. This lecture focuses on how graphical data is processed and rendered in modern computer systems, the role of GPUs (Graphics Processing Units), memory requirements, and how to choose the right graphics solution based on computational and visual needs.

1. Introduction to Graphics Hardware Graphics cards and video adapters are specialized hardware components responsible for rendering images, animations, and video to display devices. While integrated graphics are sufficient for everyday tasks, dedicated graphics cards are essential for 3D rendering, gaming, video editing, and scientific visualization.

A graphics processing unit (GPU) is the core component of a video adapter. It is designed to process and manipulate graphical data independently of the CPU, freeing the system processor for other tasks.

2. Types of Graphics Solutions

Type	Description
Integrated Graphics	Built into the CPU or motherboard, suitable for basic tasks.
Dedicated GPU	Standalone graphics card with its own processor and video memory (VRAM).
Discrete GPU (External)	Connects via Thunderbolt or USB-C; used in laptops for GPU acceleration.

Integrated GPUs share system memory (RAM), which limits their performance in intensive applications.

Dedicated GPUs feature their own VRAM (e.g., GDDR6), which significantly improves rendering performance, particularly in high-resolution displays or multitasking environments.

3. Structure of a Graphics Card.

A typical dedicated GPU includes the following components:

1. GPU Core: Executes thousands of parallel threads for pixel and vertex calculations.
2. VRAM (Video RAM): High-speed memory for storing frame buffers, textures, and shaders.
3. Cooling System: Fans or heatsinks to dissipate heat.
4. Power Connectors: Additional power supply (6-pin or 8-pin) for high-end GPUs.
5. Output Ports: HDMI, DisplayPort, VGA, or DVI connectors for monitors.

The GPU operates on a highly parallel architecture, which is ideal for large-scale matrix and pixel operations, making it popular not only for graphics but also for general-purpose computation (GPGPU).

4. GPU Memory (VRAM).

Video RAM is specialized memory that allows the GPU to quickly access textures, buffers, and images. Types of VRAM include:

VRAM Type	Description
GDDR5	Older standard, still in use in budget cards.
GDDR6	Higher bandwidth, common in modern GPUs.
HBM2	High Bandwidth Memory, used in data centers.

Higher amounts of VRAM are crucial when:

1. Running games at 1440p or 4K resolution.
2. Using multiple monitors.
3. Performing complex 3D rendering or simulation tasks.

5. Rendering Technologies Modern GPUs support several technologies for improving visual quality:

1. Ray Tracing: Simulates light for realistic reflections and shadows.
2. DLSS (Deep Learning Super Sampling): Uses AI to upscale graphics for better performance.
3. OpenGL / DirectX / Vulkan: APIs that facilitate rendering through software-to-hardware communication.

GPU drivers and software optimizations are as important as hardware. Frequent driver updates ensure compatibility and performance improvements.

6. Performance Metrics.

Metric	Description
Core Clock Speed	Measured in MHz/GHz; higher speed = faster execution.
CUDA Cores / Shaders	Number of parallel processors; more cores = better multitasking.
Memory Bandwidth	Determines how fast data moves between VRAM and GPU.
TDP (Thermal Design Power)	Indicates power consumption and cooling needs.

7. Applications of GPUs.

Graphics cards are used not only for gaming but in a wide range of professional and scientific domains:

1. Gaming: High-FPS rendering, VR support.
2. Video Editing: Real-time preview, GPU-accelerated encoding.
3. Machine Learning: Parallel data processing on large datasets.
4. CAD / 3D Modeling: Precise rendering of models and simulations.
5. Cryptocurrency Mining: Solving hash algorithms using parallel computation.

8. Choosing the Right GPU Consider the following factors when selecting a graphics card:

1. Workload Type: Gaming, editing, AI training, etc.
2. Monitor Resolution: Higher resolution requires more VRAM.
3. System Compatibility: Power supply wattage, PCIe slots, case size.
4. Budget: Entry-level (e.g., GTX 1650), Mid-range (RTX 3060), High-end (RTX 4080).

9. Summary:

1. Graphics cards process and render visual data for display.
2. GPUs are optimized for parallelism, making them powerful in visual and non-visual computation.
3. Dedicated GPUs offer superior performance over integrated solutions.

4. The choice of graphics hardware should match the user's workload, system specs, and budget.

Literature

1. Owens J.D., Houston M., Luebke D., GPU Computing Gems, Morgan Kaufmann. 2022. 632 p.

2. Sandler M., Howard A., Zhu M., Efficient Processing of Deep Neural Networks, Springer. 2023. 430 p.

Topic 7. Power Supply Units and Cooling Systems

Purpose of the Lecture: To explore the architecture and functionality of computer power supply units (PSUs) and cooling systems, understand their critical roles in system stability, safety, and performance, and learn how to choose appropriate power and thermal solutions for different computing tasks.

1. Introduction Every electronic component in a computer requires power to function. The Power Supply Unit (PSU) delivers that power, converting electrical energy from AC (Alternating Current) mains into stable DC (Direct Current) voltages.

Equally important is the cooling system, which protects components from overheating. As performance and power draw increase (especially with CPUs and GPUs), effective cooling ensures operational reliability and component longevity.

2. Power Supply Units (PSUs).

Main Functions:

1. Converts 110V/220V AC to +12V, +5V, and +3.3V DC.
2. Provides power to the motherboard, CPU, drives, GPU, and peripherals.
3. Protects against electrical anomalies (short circuit, over-voltage, over-current).

Key PSU Specifications:

1. Wattage Rating (W): Total power the PSU can deliver. Common values range from 300W to 1000W+.
2. Efficiency Rating: Percentage of AC power converted to DC. Higher efficiency = less heat: Rated via the 80 PLUS standard (Bronze, Silver, Gold, Platinum, Titanium).

3. Modularity:
   - Non-modular: All cables are fixed;
   - Semi-modular: Some cables detachable;
   - Fully modular: All cables detachable for cleaner cable management.

Connectors Provided:
   - 24-pin ATX connector (motherboard);
   - 8/4-pin CPU power;
   - PCIe 6/8-pin (GPU);
   - SATA and Molex for storage and peripherals.

3. Cooling Systems in Computers.

Purpose: Heat is a byproduct of electrical resistance. As CPUs and GPUs execute billions of instructions per second, they generate substantial thermal output that must be dissipated to maintain safe operating temperatures.

Consequences of Inadequate Cooling:
   - System instability or crashes;
   - Thermal throttling (reduced performance);
   - Long-term component degradation or failure.

4. Types of Cooling Systems

Cooling Type	Description	Use Case
Air Cooling	Uses heatsinks and fans to blow hot air away from components.	Standard desktops, moderate workloads
Liquid Cooling	Circulates coolant through water blocks, radiators, and pumps.	High-performance, gaming, overclocking
Passive Cooling	Uses heat sinks without fans; relies on natural convection.	Low-power systems (Raspberry Pi, IoT)

Components of a Typical Air Cooling System:

1. Heatsink: Made from aluminum or copper; attached directly to the CPU or GPU.
2. Fan: Mounted to push or pull air through the heatsink fins.

3. Case Fans: Ensure airflow within the chassis, removing hot air and bringing in cooler air.
4. Liquid Cooling Components:
5. Water block: Mounted to CPU/GPU.
6. Pump: Moves coolant through the loop.
7. Radiator: Dissipates heat via fans.
8. Reservoir: Stores excess coolant (in custom loops).

5. Thermal Interface Material (TIM) A thermal paste or pad is applied between the CPU/GPU and heatsink to improve thermal conductivity. Poor application can result in overheating even with powerful coolers.

6. Monitoring and Controlling Temperatures.

Modern systems provide thermal sensors and management features:

1. BIOS/UEFI: Displays system temperatures and fan speeds.
2. Software Tools: e.g., HWMonitor, SpeedFan, MSI Afterburner
3. Fan Control: Automatic or user-defined via PWM (Pulse Width Modulation).

Safe Operating Temperatures:

– CPU: 30–85°C (95°C max in high-performance chips;)
– GPU: 40–90°C (depends on load and manufacturer).

7. Choosing the Right PSU and Cooling Solution

For the PSU:

1. Calculate total system power requirement with an online PSU calculator.
2. Add 20–30% headroom for safety and future upgrades.
3. Choose higher efficiency (Gold or above) for better thermals and lower electricity bills.

For Cooling:

1. Use air cooling for budget or general-purpose systems.
2. Consider liquid cooling for high-performance, gaming, or overclocked systems.
3. Ensure good airflow with properly positioned case fans (intake/exhaust).

8. Summary:

1. The power supply is vital for converting and regulating electricity in a PC.
2. Cooling systems prevent overheating, enabling stable and prolonged operation.

3. Different use cases demand different power capacities and thermal management solutions.

4. Efficient and reliable PSU and cooling configurations enhance both performance and longevity.

Literature

1. Mueller S., Upgrading and Repairing PCs, Pearson Education. 2023. 944 p.

2. White R., The Hardware Hacker: Adventures in Making and Breaking Hardware, No Starch Press. 2022. 448 p.

Topic 8. Monitor Types – Technologies and Interfaces

Purpose of the Lecture: To provide a comprehensive understanding of monitor technologies, screen types, display specifications, and video interface standards. This lecture will equip students with the knowledge to select appropriate display hardware for various applications, considering image quality, compatibility, and ergonomic factors.

1. Introduction to Monitors. Monitors are the primary output devices used to display visual data generated by the computer's graphics subsystem. They are essential for user interaction, media consumption, design work, gaming, and general system monitoring.

Modern monitors come in various sizes, resolutions, panel technologies, and connection standards, making it essential to understand how these parameters affect usability and performance.

2. Types of Display Technologies

Technology	Description	Pros	Cons
1	2	3	4
LCD	Liquid Crystal Display; uses backlighting (CCFL or LED)	Affordable, widely used	Limited contrast and color depth
LED	LCD with LED backlight; more efficient than CCFL	Energy-efficient, better brightness	Still limited viewing angles

1	2	3	4
OLED	Organic Light Emitting Diodes; pixels emit light individually	Deep blacks, high contrast, thin panels	Expensive, risk of burn-in
IPS	In-Plane Switching; a type of LCD with improved color and angle performance	Great color accuracy and wide viewing	Typically slower response times
TN	Twisted Nematic; older LCD type, fast response	Low cost, fast refresh for gaming	Poor color reproduction
VA	Vertical Alignment; compromise between TN and IPS	Good contrast and black levels	Slower response and color shift

3. Monitor Resolutions and Aspect Ratios.

Resolution defines the number of pixels a screen can display. Higher resolutions offer sharper images but require more GPU processing.

Resolution	Pixel Count	Common Use
1280x720	HD (720p)	Older monitors, entry-level laptops
1920x1080	Full HD (1080p)	Standard for desktops and office work
2560x1440	Quad HD (2K)	Gaming, multimedia editing
3840x2160	Ultra HD / 4K	Professional design, high-end systems

Aspect Ratios:

- 16:9 – Standard widescreen;
- 21:9 – Ultrawide (productivity, immersive gaming);
- 4:3 – Legacy screens.

4. Refresh Rate and Response Time:

1. Refresh Rate (Hz): Number of times the screen updates per second:
    - common values: 60Hz, 75Hz, 120Hz, 144Hz, 240Hz;
    - higher rates benefit gaming and reduce eye strain in fast motion.

2. Response Time (ms): Time taken for a pixel to change color:
   - Lower is better (e.g., 1–5 ms is ideal for gaming);
   - High response time can lead to ghosting and motion blur.

5. Color Accuracy and Calibration.

Color accuracy is critical in tasks like photo editing, graphic design, and video production:

1. Color Gamut: Range of colors a monitor can reproduce (sRGB, AdobeRGB, DCI-P3).
2. Bit Depth: Defines number of shades per color (8-bit, 10-bit panels).
3. Hardware Calibration: Uses colorimeters to adjust monitor output to real-world standards.

6. Monitor Interfaces and Cables.

Interface	Description	Max Resolution/Features
HDMI	Common digital interface for video and audio	4K @ 60Hz (HDMI 2.0), 8K @ 60Hz (HDMI 2.1)
DisplayPort	Designed for computers; supports higher bandwidth	8K+ @ 60Hz; supports Daisy-Chaining
VGA	Analog legacy interface	Max ~1080p, no audio support
DVI	Digital interface; now largely obsolete	Up to 2560x1600 (DVI-D Dual-Link)
USB-C	Modern universal port; carries video, data, and power	Used in laptops, tablets, portable monitors

Adapters are available to convert between standards, but quality may vary.

7. Specialized Monitor Features:

- G-Sync / FreeSync: Syncs GPU output with monitor refresh rate to avoid screen tearing;
- HDR (High Dynamic Range): Expands contrast and brightness range for better realism;

- Curved Displays: Reduce distortion on ultrawide screens, provide immersive experience;
- Touchscreens: Used in point-of-sale, kiosks, and tablets;
- Blue Light Filters & Flicker Reduction: For eye health in prolonged use.

8. Ergonomics and Mounting:

- Adjustable stands (tilt, height, swivel);
- VESA mount compatibility for arm brackets;
- Anti-glare coatings and matte finishes for better visibility;
- Screen size vs. viewing distance: 24–32" optimal for desks at 60–90 cm distance.

9. Summary:

1. Monitor quality significantly impacts the user experience, productivity, and visual accuracy.
2. Choosing the right monitor involves evaluating resolution, panel type, refresh rate, and connectivity.
3. Understanding display technologies helps tailor system builds for specific needs like gaming, content creation, or office use.

Literature

1. Poynton C., Digital Video and HD: Algorithms and Interfaces. Morgan Kaufmann, 2022. 720 p.
2. Sellers J., Display Interfaces: Fundamentals and Standards, Springer. 2023. 416 p.

Topic 9. Keyboard Types – Connectivity and Functionality

Purpose of the Lecture: To explore the structural design, communication interfaces, key-switch mechanisms, and ergonomic considerations of keyboards. Students will learn the differences between mechanical and membrane keyboards, wired and wireless options, and how keyboard choice influences user performance, comfort, and application domain.

1. Introduction to Keyboards. Keyboards are one of the most fundamental human-computer interaction (HCI) devices. They convert keystrokes into digital signals that are

interpreted by the computer. Modern keyboards vary widely in layout, key technology, connectivity, and features.

Depending on use case (office, gaming, industrial control, or accessibility), the keyboard's build and interface may differ significantly.

2. Keyboard Layouts.

Layout Type	Description	Use Case
QWERTY	Standard layout based on English typing	Global default
AZERTY	French-speaking countries	France, Belgium
DVORAK	Optimized for faster typing and less finger movement	Niche, ergonomic enthusiasts
Compact / 60%	Fewer keys, no function row or numpad	Portability, minimal setups
Tenkeyless (TKL)	No numeric keypad	Space-saving, gaming
Full-size (104+)	Includes all function, navigation, and number keys	Office and productivity work

3. Key Technologies: Mechanical vs. Membrane.

Type	Description	Pros	Cons
Membrane	Uses pressure pads and conductive traces	Quiet, inexpensive	Poor tactile feedback, less durable
Mechanical	Individual switches under each key	Excellent feedback, long-lasting	More expensive, louder
Scissor Switch	Hybrid of both; used in laptops	Low profile, moderate tactile feedback	Limited lifespan

Mechanical Switch Varieties:

1. Linear (e.g., Cherry MX Red): Smooth keystroke, no tactile bump.
2. Tactile (e.g., Cherry MX Brown): Gentle bump at actuation point.
3. Clicky (e.g., Cherry MX Blue): Audible click and tactile bump.

4. Connectivity Options.

Interface	Description	Typical Use
USB Wired	Standard plug-and-play; low latency	Most desktops, gaming
PS/2	Legacy interface; supports N-key rollover	Older systems, some specialized applications
Bluetooth	Wireless via short-range radio	Laptops, tablets, mobile users
RF Wireless	Uses USB dongle and 2.4GHz signal	Good range, stable connection

Latency & Power:

- Wired connections offer the lowest input delay;
- Wireless keyboards may introduce latency but offer portability;
- Bluetooth and RF devices require batteries or recharging.

5. Features and Customization;

- Backlighting (RGB/Monochrome): Enhances aesthetics and visibility in low light;
- Programmable Keys / Macros: Useful in gaming and professional software (e.g., CAD, video editing);
- Media Controls: Volume, play/pause, brightness directly accessible;
- Ergonomic Design: Split keyboards, tenting, wrist rests to reduce strain;
- Hot-swappable Switches: Allows changing switch types without soldering.

6. Ergonomics and Typing Comfort Prolonged keyboard use can result in strain or repetitive stress injuries (RSI). Ergonomic keyboards aim to reduce this by:

- Promoting natural wrist angles;
- Reducing finger travel distance;
- Offering support for palm and wrist.

Tips for Ergonomic Use:

– Keep wrists straight and level;

– Use light keystrokes;

– Maintain elbows at a 90° angle;

– Take regular breaks.

7. Use-Specific Keyboards:

– Gaming Keyboards: Mechanical switches, RGB lighting, anti-ghosting, fast polling rates;

– Office Keyboards: Quiet operation, compact layout, ergonomic shape;

– Industrial/Medical: Sealed, spill-resistant, washable designs;

– Accessibility Keyboards: Large print keys, one-handed layouts, switch-compatible.

8. Summary:

– Keyboards differ in size, layout, switch mechanism, and connectivity;

– Mechanical keyboards provide better performance and feedback but are more expensive;

– Ergonomic and specialized keyboards enhance comfort and accessibility for specific needs;

- Selecting a keyboard depends on workload, typing style, and environment.

Literature

1. Haverbeke D., Computer Input Devices: Design and Application, Springer. 2022. 358 p.

2. Bender R., Keyboard Engineering and Ergonomics, CRC Press. 2023. 404 p.

Topic 10. Lecture 10: Modern Trends in Computer Hardware Development

Purpose of the Lecture: To examine the current trends and technological advancements shaping the evolution of computer hardware. The lecture explores key developments in processors, memory, storage, connectivity, miniaturization, energy efficiency, and the integration of AI and machine learning at the hardware level.

1. Introduction Computer hardware is evolving rapidly, driven by demands for higher performance, lower energy consumption, portability, and support for emerging technologies like AI and virtual reality. Innovations in architecture, materials, and design are enabling more powerful, compact, and intelligent devices.

Staying informed about these trends is essential for future engineers, technicians, and IT professionals who will be designing, configuring, or optimizing hardware systems.

2. Advances in Microprocessors Modern CPUs are significantly more powerful and efficient than their predecessors.

Key trends:

1. Smaller transistor nodes: 7nm, 5nm, and approaching 3nm processes improve speed and reduce power use.
2. Multi-core and multi-threading architectures: Increase parallel processing (e.g., 8-core, 16-thread CPUs).
3. Heterogeneous computing: Integration of CPU, GPU, and AI cores on a single chip (e.g., Apple M1/M2).
4. Chiplet design: Combines multiple dies in a single package for improved modularity and yield.

Example: AMD's Ryzen and Intel's Alder Lake architectures combine high-performance and high-efficiency cores for balanced performance.

3. Graphics and Parallel Processing Modern GPUs not only power gaming and content creation but also accelerate scientific computations, deep learning, and data analysis.

Key developments:

1. Real-time ray tracing for photorealistic graphics.
2. AI-enhanced rendering (e.g., NVIDIA DLSS, AMD FSR).
3. Use of GPGPU (General-Purpose computing on GPUs) in AI, simulations, and big data.

4. Memory and Storage. Innovations Memory and storage technologies are evolving to match CPU/GPU performance:

- DDR5 RAM: Higher bandwidth and lower latency compared to DDR4;
- LPDDR5/6: Used in mobile devices and ultrabooks for power efficiency;

– PCIe 5.0 and 6.0: Doubles bandwidth every generation for faster data exchange;

– NVMe SSDs: Provide >7000 MB/s speeds; replacing SATA SSDs;

– Persistent memory: Technologies like Intel Optane aim to bridge RAM and SSD.

5. Energy Efficiency and Sustainability Hardware manufacturers are prioritizing green computing to reduce energy usage and environmental impact:

– ARM architecture is increasingly adopted in data centers for its efficiency;

– Low TDP processors and dynamic power scaling;

– Use of recyclable materials and fanless passive cooling systems;

– EPEAT, Energy Star, and RoHS compliance in manufacturing.

6. Miniaturization and Portability. Hardware is becoming more compact while maintaining performance:

– System-on-Chip (SoC): Combines CPU, GPU, memory controller, and I/O on a single chip;

– Ultrabooks, tablets, and convertibles replace bulky laptops;

– Single-board computers (SBCs): Raspberry Pi, Jetson Nano enable powerful embedded computing.

7. Artificial Intelligence and Edge Computing Integration of AI accelerators into hardware is reshaping computing architectures:

– TPUs (Tensor Processing Units): Designed for matrix math in neural networks;

– Edge AI: Processing data locally instead of relying on cloud — faster and more secure;

– AI cores now integrated into CPUs and SoCs (e.g., Apple Neural Engine, Intel NPU).

8. High-Speed Connectivity. Connectivity standards continue to evolve to support faster data transfer and lower latency.

Standard	Description	Max Speed
1	2	3
USB 4	Universal connector with PCIe & DisplayPort	Up to 40 Gbps

1	2	3
Thunderbolt 4	Combines USB-C, PCIe, and video	40 Gbps, daisy chaining
Wi-Fi 6/6E/7	Higher speeds, lower latency, better spectrum	Up to 30 Gbps (Wi-Fi 7)
Bluetooth 5.3	Enhanced range and power efficiency	Reliable IoT communication
5G/6G	High-speed mobile connectivity	Latency <1 ms, high density

9. Summary:

– Modern computer hardware trends prioritize performance, efficiency, compactness, and intelligent functionality;

– Microprocessors and GPUs are becoming more modular, parallel, and integrated;

– Memory, storage, and interfaces are advancing to support faster, broader, and more efficient data access;

– AI is increasingly present not only in software but in hardware-level computation;

– Understanding these trends is crucial for professionals designing the next generation of computing systems.

Literature

1. Hennessy J., Patterson D., Computer Architecture: A Quantitative Approach, Morgan Kaufmann. 2022. 936 p.

2. Kaeli D., Heterogeneous Computing with OpenCL and CUDA. Morgan Kaufmann. 2023. 480 p.

Topic 11. Assembly Language Programming for Microprocessors

Purpose of the Lecture: To introduce students to assembly language programming and its close interaction with microprocessor architecture. This lecture covers the fundamentals of low-level programming, instruction formats, addressing modes, control structures, and practical examples using a typical instruction set (such as x86 or ARM).

1. Introduction to Assembly Language Assembly language is a low-level programming language that is closely tied to a computer's machine architecture. Unlike high-level languages (C, Python), assembly language provides direct control over hardware, allowing programmers to write highly optimized routines.

Each assembly instruction corresponds to a single machine code instruction that is executed by the CPU. Therefore, understanding assembly is essential for systems programming, embedded development, and performance-critical applications.

2. Role of Assemblers and Machine Code:

– Machine code: Binary code understood directly by the CPU (e.g., 10111000 01100001);

– Assembly language: Mnemonic representation of machine instructions (e.g., MOV AL, 61h);

– Assembler: A tool that converts assembly source code into machine code;

– Disassembler: Converts machine code back into assembly language.

Assembly is processor-specific, meaning each CPU family (e.g., x86, ARM) has its own unique instruction set architecture (ISA).

3. Basic Structure of an Assembly Program A typical assembly program includes:

– Directives: Instructions to the assembler (e.g., .data, .code, .text, .global);

– Labels: Mark positions in the code (e.g., LOOP_START:);

– Instructions: CPU operations (e.g., MOV, ADD, JMP, CALL).

Example (x86 syntax):

```
section .data
  msg db 'Hello, World!', 0
```

```
section .text
    global _start
_start:
    mov edx, 13        ; message length
    mov ecx, msg       ; message to write
    mov ebx, 1         ; file descriptor (stdout)
    mov eax, 4         ; syscall (sys_write)
    int 0x80           ; call kernel
    mov eax, 1         ; syscall (sys_exit)
    xor ebx, ebx       ; status = 0
    int 0x80
```

4. Registers and Memory Access Microprocessors use registers as high-speed storage.

Register	Description
AX, BX	General-purpose registers (accumulators)
SI, DI	Source and Destination Index registers
SP, BP	Stack Pointer and Base Pointer
IP	Instruction Pointer (next instruction)
FLAGS	Status flags (Zero, Carry, Sign, Overflow)

Memory access in assembly is done using pointers and addressing modes:

- Immediate: MOV AX, 5;
- Direct: MOV AX, [1234h];
- Register indirect: MOV AX, [BX];
- Indexed: MOV AX, [SI+5].

5. Instruction Categories Assembly instructions fall into the following categories:

- Data transfer: MOV, PUSH, POP, XCHG;
- Arithmetic: ADD, SUB, MUL, INC, DEC;
- Logic: AND, OR, XOR, NOT;

- Control flow: JMP, CALL, RET, JZ, JNZ, LOOP;
- String operations: MOVSB, LODSB, STOSB.

Example:

```
MOV AX, 2
ADD AX, 3      ; AX now contains 5.
```

6. Procedures and Stack Management Assembly allows modular programming through procedures (subroutines):

- Call a procedure using CALL label;
- Return with RET;
- Use PUSH/POP to manage parameters via the stack.

The stack grows downward in memory and is used for:

- Temporary storage;
- Parameter passing;
- Preserving return addresses.

Stack example:

```
PUSH AX
CALL MyFunction
POP AX
```

7. Addressing Modes and Control Structures Assembly allows fine-grained control over program flow:

- Conditional jumps (based on FLAGS): JZ, JNZ, JC, JS;
- Loop structures using LOOP, CMP, and JMP.

Example – Counting loop:

```
MOV CX, 10
LOOP_START:
  ; do something
  LOOP LOOP_START  ; Decrement CX, jump if not zero.
```

8. Debugging and Tools:

- Use debuggers like GDB, OllyDbg, or NASM + Linux syscalls;
- Set breakpoints, view registers, step through instructions;
- Disassemblers (IDA Pro, objdump) aid in reverse engineering.

Key development tools:

- NASM, MASM: Popular assemblers for x86;
- Keil uVision / STM32CubeIDE: For embedded microcontrollers;
- GNU Assembler (GAS): For Linux systems (AT&T syntax).

9. Summary:

- Assembly language allows low-level control of CPU and memory;
- It provides insights into how high-level code is executed;
- Understanding instruction formats, register usage, and memory operations is essential for systems programming, embedded development, and optimization;
- Though less used for application development, it remains critical in drivers, bootloaders, and real-time systems.

Literature

1. Hyde R.. The Art of Assembly Language, No Starch Press. 2022. 880 p.
2. Kip Irvine. Assembly Language for x86 Processors, Pearson. 2023. 736 p.

Topic 12. Programming ARM Microcontrollers in C for AVR

Purpose of the Lecture: To introduce students to the basic principles of programming ARM-based microcontrollers using the C programming language, with practical examples and comparison to AVR platforms. Emphasis is placed on register-level programming, hardware abstraction, GPIO control, and embedded development tools.

1. Introduction to ARM and AVR Microcontrollers ARM and AVR are two distinct families of microcontrollers widely used in embedded systems. AVR, developed by Atmel (now Microchip), is 8-bit and used for simpler applications. ARM (Advanced RISC Machine) is 32-bit and supports higher performance systems.

ARM Cortex-M microcontrollers (e.g., STM32, NXP, Nordic) are designed for real-time applications and offer extensive peripheral support, low power consumption, and scalable performance.

2. C Language in Embedded Programming C is the most widely used language for microcontroller programming due to:

- Direct hardware control;
- Efficient memory usage;
- Portability;
- Extensive support from toolchains and libraries.

Typical C structure for microcontroller programs:

```
#include <stdint.h>
int main(void) {
  // Initialization
  while(1) {
    // Main loop
  }
}
```

3. ARM Architecture Overview:

- 32-bit RISC architecture: simpler instruction set;
- Registers: R0–R15 (general + special purpose);

- Status Registers: Program Status Register (PSR);
- Nested Vector Interrupt Controller (NVIC): Efficient interrupt handling;
- SysTick Timer: System tick for time base.

Example register-level control (STM32):

```
#define GPIOA_MODER   (*(volatile uint32_t*)0x48000000)
GPIOA_MODER |= (1 << 10); // Set pin to output
```

4. GPIO Programming in C General-Purpose Input/Output (GPIO) is fundamental.

Steps:

1. Enable GPIO clock.
2. Configure pin mode.
3. Write/read pin values.

Example – Blinking an LED (pseudocode):

```
RCC->AHB2ENR |= RCC_AHB2ENR_GPIOAEN;   // Enable GPIOA
GPIOA->MODER &= ~(3 << (5 * 2));       // Set PA5 as output
GPIOA->ODR ^= (1 << 5);                // Toggle PA5
```

5. Timer Configuration in ARM (STM32 example) Timers are used for delays, PWM, and periodic tasks.

Steps:

1. Enable timer peripheral clock.
2. Configure prescaler and auto-reload.
3. Enable counter.

```
RCC->APB1ENR1 |= RCC_APB1ENR1_TIM2EN;
TIM2->PSC = 7999;      // Prescaler
TIM2->ARR = 999;       // Auto-reload
TIM2->CR1 |= TIM_CR1_CEN;
```

6. Interrupts in ARM Cortex-M Interrupts improve responsiveness without constant polling.

Features:

- Vector table holds ISR addresses;
- NVIC handles priorities;

– __enable_irq() and __disable_irq() manage global interrupt state.

Example – Button press ISR:

```
void EXTI0_IRQHandler(void) {
  if (EXTI->PR1 & EXTI_PR1_PIF0) {
    EXTI->PR1 |= EXTI_PR1_PIF0; // Clear flag
    // Handle button event
  }
}
```

7. Comparison: ARM vs AVR in C Programming

Feature	AVR (ATmega328)	ARM Cortex-M (STM32)
Architecture	8-bit Harvard	32-bit RISC
Clock Speed	Up to 20 MHz	Up to 480 MHz
Memory	Limited (e.g., 2 KB RAM)	Extensive (e.g., 512 KB RAM)
Peripheral support	Basic (UART, ADC, PWM)	Advanced (USB, CAN, DMA)
C Programming Style	Direct I/O access via macros	CMSIS and HAL libraries

8. Tools and IDEs:

– ARM: STM32CubeIDE, Keil uVision, PlatformIO;

– AVR: Atmel Studio, AVRDude, Arduino IDE (for simple projects);

– Debugging: ST-Link, J-Link, OpenOCD.

ARM Toolchain Example:

– Compiler: arm-none-eabi-gcc;

– Linker scripts define memory regions;

– CMSIS headers simplify register access.

9. Summary:

– C programming enables full control of ARM microcontrollers;

– ARM offers better performance, scalability, and richer peripherals than AVR;

– Understanding the architecture and hardware abstraction layers (HAL, CMSIS) is crucial for effective development;

– Programming at register level builds insight, while libraries help accelerate development.

Literature

1. Yiu J., *T*he Definitive Guide to ARM® Cortex®-M3 and Cortex®-M4 Processors, Newnes. 2022. 512 p.

2. Perron J., Embedded Systems with ARM Cortex-M Microcontrollers in Assembly Language and C, Elektor. 2023. 648 p.

Topic 13. Programming AVR Microprocessors in C

Purpose of the Lecture: To provide students with an in-depth understanding of how to program AVR microcontrollers using the C language. The lecture includes explanations of AVR architecture, register-level programming, peripheral interfacing, and usage of common development tools.

1. Introduction to AVR Microcontrollers AVR is a family of 8-bit microcontrollers developed by Atmel (now part of Microchip Technology). Popular AVR chips include the ATmega328 (used in Arduino Uno) and ATmega32. These microcontrollers are known for their ease of use, good community support, and suitability for small embedded systems.

2. C Language in AVR Programming The C language allows high-level program structure while offering direct hardware access. Standard AVR development uses avr-gcc compiler and AVR Libc for hardware abstraction.

Basic structure of an AVR C program:

```
#include <avr/io.h>
int main(void) {
   // Initialization
   while (1) {
      // Main loop
   }
}
```

3. AVR Architecture Overview:

- 8-bit RISC architecture: Efficient, reduced instruction set;
- General-purpose registers: 32 (R0–R31);
- Harvard architecture: Separate program and data memory;
- Flash memory: Non-volatile memory for program code;
- SRAM: Volatile memory for variables.

Status Register (SREG): Holds flags like Zero, Carry, Negative, etc.

4. GPIO Programming GPIO is controlled using three registers:

- DDRx: Data Direction Register;
- PORTx: Output register;
- PINx: Input register.

Example – Blinking LED on pin PB0:

```
DDRB |= (1 << PB0);       // Set PB0 as output
while (1) {
   PORTB ^= (1 << PB0);  // Toggle PB0
   _delay_ms(500);
}
```

Include delay header:

```
#include <util/delay.h>
```

5. Timers and Counters AVR microcontrollers have multiple 8- and 16-bit timers.

- Timer/Counter0 and Timer/Counter1 are commonly used;
- Modes: Normal, CTC, PWM.

Example – CTC mode using Timer1:

```
TCCR1B |= (1 << WGM12);    // CTC mode
OCR1A = 15624;             // Compare value
TCCR1B |= (1 << CS12);     // Prescaler 256
TIMSK1 |= (1 << OCIE1A);   // Enable interrupt
sei();                     // Enable global interrupts
```

6. Interrupt Handling Interrupts are essential for efficient real-time response:

- ISR(vector) defines interrupt service routine;
- Enable global interrupts using sei();

– Disable using cli().

Example – Timer1 Compare Match ISR:

```
ISR(TIMER1_COMPA_vect) {
   PORTB ^= (1 << PB0);  // Toggle LED
}
```

7. ADC – Analog to Digital Converter AVR microcontrollers include 10-bit ADCs.

Steps to use ADC:

1. Select reference voltage (e.g., AVCC).
2. Select ADC input channel (MUX).
3. Enable ADC and start conversion.
4. Read result from ADC register.

Example:

```
ADMUX = (1 << REFS0);          // AVCC reference
ADCSRA = (1 << ADEN);          // Enable ADC
ADCSRA |= (1 << ADSC);         // Start conversion
while (ADCSRA & (1 << ADSC));    // Wait for result
uint16_t value = ADC;
```

8. USART – Serial Communication Universal Synchronous/Asynchronous Receiver/Transmitter (USART) is used for serial data communication.

Steps:

1. Set baud rate.
2. Enable TX/RX.
3. Write to data register.

Example – Transmitting a character:

```
UBRR0H = 0;
UBRR0L = 103;                  // Baud 9600 @ 16MHz
UCSR0B = (1 << TXEN0);         // Enable transmitter
UCSR0C = (1 << UCSZ01) | (1 << UCSZ00);  // 8-bit
while (!(UCSR0A & (1 << UDRE0)));
UDR0 = 'A';
```

9. Development Tools:

- Atmel Studio – Official IDE for AVR;
- AVR-GCC – Compiler toolchain;
- AVRDude – Upload tool for flashing firmware;
- Arduino IDE – Simplified environment for rapid development.

Programmer Hardware:

- USBasp;
- AVRISP mkII;
- Arduino as ISP.

10. Summary:

- AVR microcontrollers are ideal for small embedded systems;
- C language provides a balance of high-level structure and low-level control;
- Knowledge of registers, GPIO, timers, and interrupts is critical;
- Development tools like AVR-GCC and Atmel Studio enable efficient embedded development.

Literature

1. Mazidi M.A., AVR Microcontroller and Embedded Systems: Using Assembly and C. Pearson. 2022.800 p.

2. Barnett R.H. Embedded C Programming and the Atmel AVR. Cengage Learning. 2023. 576 p.

Topic 14. Lecture 14: Real-Time Systems

Purpose of the Lecture: To understand the principles, design constraints, and implementation methods of real-time systems (RTS), especially in embedded contexts. This includes understanding timing constraints, task scheduling, hardware timers, interrupts, and real-time operating systems (RTOS).

1. Introduction to Real-Time Systems (RTS) Real-time systems are computing systems that must respond to inputs or events within a strict time deadline. These systems are common in automation, avionics, medical devices, telecommunications, and automotive control systems.

Types of real-time systems:

- Hard real-time: Missing a deadline may lead to catastrophic failure (e.g., airbag systems);
- Soft real-time: Occasional deadline misses are tolerable (e.g., video streaming).

2. Key Characteristics of RTS:

- Predictability: Tasks must complete within defined time bounds;
- Determinism: System behavior should be repeatable;
- Concurrency: Multiple tasks execute seemingly in parallel;
- Responsiveness: System reacts promptly to external events.

3. Real-Time Constraints:

- Release time: When the task becomes ready;
- Deadline: When the task must complete;
- Execution time: How long the task takes to run.

Example constraint table:

Task	Release Time	Deadline	Execution Time
T1	0 ms	20 ms	5 ms
T2	5 ms	30 ms	10 ms

4. Task Scheduling .Scheduling algorithms decide which task to run and when.

Types of scheduling:

- Static scheduling: Decisions made at compile time;

- Dynamic scheduling: Decisions made during execution;
- Common scheduling algorithms:
- Rate Monotonic Scheduling (RMS): Fixed-priority based on task period;
- Earliest Deadline First (EDF): Prioritize task with nearest deadline.

Example RMS Priority Table:

Task	Period	Priority
T1	10 ms	High
T2	50 ms	Medium
T3	100 ms	Low

5. Timer Usage in Real-Time Systems Hardware timers are critical for:

- Creating time delays;
- Generating interrupts;
- Implementing periodic tasks.

Timer configuration in C (AVR example):

```
TCCR0A = (1 << WGM01);
OCR0A = 249;                // Compare match value
TIMSK0 |= (1 << OCIE0A);    // Enable interrupt
TCCR0B |= (1 << CS01) | (1 << CS00);  // Prescaler 64
sei();
```

ISR for periodic task:

```
ISR(TIMER0_COMPA_vect) {
  // Execute periodic code every 1 ms
}
```

6. Interrupts and Latency. Interrupts enable real-time response without polling.

Important concepts:

- Interrupt latency: Time between event and start of ISR;
- Interrupt jitter: Variability in ISR response time;
- Nested interrupts: High-priority ISR can preempt a lower-priority ISR.

To minimize latency:

- Keep ISRs short;
- Avoid blocking calls in ISRs;
- Use flags/semaphores to delegate tasks to main loop.

7. Real-Time Operating Systems (RTOS). RTOSs provide features for multitasking and precise timing.

Common RTOS features:

- Preemptive multitasking;
- Task priorities;
- Semaphores and mutexes;
- Timers and delays.

Popular RTOSs:

- FreeRTOS;
- Micrium µC/OS-II/III;
- Zephyr.

FreeRTOS example – LED blink task:

```
void vTaskLED(void *pvParameters) {
  for (;;) {
    toggleLED();
    vTaskDelay(pdMS_TO_TICKS(1000));
  }
}
```

8. Design Considerations:

- Worst-Case Execution Time (WCET) analysis;
- Task prioritization based on importance and frequency;
- Resource sharing: Avoid priority inversion using mutexes;
- Debugging tools: Logic analyzers, real-time trace analyzers, simulators.

9. Application Examples:

- Automotive systems: Engine control units (ECUs), adaptive cruise control;
- Medical systems: Infusion pumps, pacemakers;
- Robotics: Motor control, sensor feedback loops.

10. Summary:

- Real-time systems must operate within timing constraints;
- Deterministic task scheduling is key to meeting deadlines;
- Use hardware timers and interrupts effectively;
- RTOSes simplify task management and synchronization;
- Proper design ensures reliability, responsiveness, and safety.

Literature

1. Buttazzo G. Hard Real-Time Computing Systems. Springer. 2023. 524 p.
2. Labrosse J.J. MicroC/OS-II: The Real-Time Kernel. CMP Books. 2022. 592 p.

Topic 15. Integration of Microprocessors into Complex Systems

Purpose of the Lecture: To explore how microprocessors and microcontrollers are integrated into larger, complex electronic and computing systems. This includes understanding system architecture, interfaces, communication protocols, embedded software, and system-level design practices.

1. Overview of System Integration Microprocessors are central components in modern embedded and computing systems. Integration refers to the process of connecting the microprocessor with peripherals, memory, sensors, actuators, and software to perform a coordinated task.

Applications include:

- Smart home automation;
- Industrial control systems;
- Robotics and automation;
- Medical equipment;
- Automotive ECUs.

2. System-Level Architecture A complex system typically includes:

- Microprocessor/Microcontroller Unit (MCU/MPU);
- Memory blocks: Flash, RAM, EEPROM;
- Peripherals: Timers, ADCs, UARTs, GPIOs;

- Power management: Voltage regulators, battery interface;
- Communication interfaces: I2C, SPI, UART, CAN, USB.

Block Diagram Example:

```
[Sensor]--[ADC]--|
                 |--> [MCU] -- [PWM] --> [Motor Driver] --> [Actuator]
[Memory]---------|
[User Interface]--[GPIO/UART]--> [Display/Button]
```

3. Communication Protocols for Integration:

- UART: Asynchronous, serial communication between MCU and devices;
- SPI: Full-duplex, high-speed protocol used with displays, sensors;
- I2C: Two-wire protocol used for connecting multiple peripherals;
- CAN: Reliable network protocol for automotive/industrial;
- Ethernet/Wi-Fi/Bluetooth: For remote connectivity and IoT.

Example – I2C Connection:

```
Wire.begin();          // Master mode
Wire.beginTransmission(0x3C); // OLED address
Wire.write(0x00);      // Command mode
Wire.endTransmission();
```

4. Embedded Software Integration. Combining firmware modules to operate synchronously:

- Hardware abstraction layer (HAL);
- Peripheral drivers (GPIO, UART, ADC, etc.);
- Middleware (file systems, communication stacks);
- Application layer (main logic, user interface).

Layered Software Model:

```
[Application Code]
[Middleware: e.g., TCP/IP stack]
[Drivers: UART, SPI, etc.]
[HAL or direct register access]
```

5. Real-Time Operating Systems (RTOS) in Integration RTOS enables multiple software modules to run concurrently:

- Tasks: each handles a subsystem (e.g., sensor reading, communication);
- Inter-task communication: queues, semaphores;
- Timers and schedulers ensure timing guarantees.

Example:

- Task 1: Poll temperature sensor;
- Task 2: Update OLED display;
- Task 3: Transmit data via UART.

6. Power and Thermal Considerations

- Voltage regulation: Buck/boost converters to step down/up supply;
- Sleep/idle modes: For battery optimization;
- Thermal management: Heatsinks, fans, temperature sensors.

Design Tip: Always check MCU power domains and allowable voltage ranges.

7. Debugging and Testing in System Integration:

- Serial debug interfaces (e.g., SWD, JTAG);
- Logic analyzers & oscilloscopes for signal inspection;
- Software tools: GDB, simulators, in-circuit emulators (ICE).

Testing levels:

- Unit tests for drivers;
- Integration tests for module interaction;
- System validation against specifications.

8. Case Study: IoT Weather Station Components:

- MCU (e.g., ESP32 or STM32);
- DHT22 sensor (temperature, humidity);
- OLED display (SPI/I2C);
- Wi-Fi module;
- Power supply with Li-Ion battery and charging circuit.

System Workflow:

1. Read sensor every 1 s.
2. Update display with new data.
3. Send data via HTTP to cloud server.

4. Sleep for power saving.

9. Challenges in System Integration:

– Signal integrity and PCB design constraints;
– Software timing issues and race conditions;
– Incompatibility between modules/interfaces;
– EMI/EMC considerations in mixed-signal environments.

Best Practices:

– Modular design and layered software;
– Use standard protocols and libraries;
– Maintain clear documentation and signal mappings.

10. Summary:

– Integration combines hardware and software subsystems;
– Protocols and abstraction layers facilitate modular design;
– Testing and power optimization are crucial for stable operation;
– RTOS and middleware support scalability and responsiveness;
– Thorough planning and documentation improve integration success.

Literature

1. Valvano J.W., Embedded Systems: Introduction to the MSP432 Microcontroller. CreateSpace. 2022. 604 p.

2. Barr M., Massa A., Programming Embedded Systems. O'Reilly Media. 2023. 334 p.

FOR NOTES

C -73 Computer Architecture and Microprocessor Device Programming. Lecture notes for students of the first (bachelor's) level of higher education in the educational program «Computerized Telecommunication Networks», Field of Knowledge 17 – Electronics, Automation, and Electronic Communications, Speciality 172 – Electronic Communications and Radio Engineering,for all forms of study. Compiled by M.V. Khvyshchun, Lutsk: LNTU, 2025. – 56p.

Typesetting Mykola Khvyshchun
Editor Mykola Khvyshchun

Signed for printing: «__» _____ 2025
Format: 60×84/16. Offset paper.
Font: Times New Roman. Printed sheets: 1.8.
Print run: 50 copies

Image and Promotion Department
Lutsk National Technical University
75 Lvivska Street, Lutsk, 43018,
UkrainePrinting – VIP LNTU