LESSON 2

Introduction to Computer Systems

40 minutes | Foundations of Computer Science | Outcome 1.13 | Outcome 1.14 | Outcome 1.18 | Outcome 1.22 | Outcome 2.11 | Outcome 3.11 | Outcome 3.13

WHAT THIS LESSON DOES

Explore the basics of computer science and understand how computer systems work. Follow step-by-step guidance to learn about hardware, software, and key components like the CPU, memory, and storage, with practical tasks to reinforce your knowledge.

CURRICULUM OUTCOMES

Outcome	What the specification asks for
1.13	identify important computing developments that have taken place in the last 100 years and consider emerging trends that could shape future computing technologies
1.14	explain when and what machine learning and AI algorithms might be used in certain contexts
1.18	recognise the diverse roles and careers that use computing technologies
1.22	read, write, test, and modify computer programs
2.11	describe the different components within a computer and the function of those components
3.11	use and control digital inputs and outputs within an embedded system
3.13	develop a program that utilises digital and analogue inputs

WHAT STUDENTS SHOULD BE ABLE TO DO

- Understand the fundamental concepts of computer science and its real-world applications.
- Identify the key components of computer systems, including hardware and software.
- Explain the roles of the CPU, memory, and storage in processing and managing data.
- Apply basic coding skills to demonstrate computer system functions using practical tools.
- Recognise the importance of computer systems in everyday technology and potential career paths.

BEFORE THE LESSON

- Open makecode.microbit.org yourself first and confirm it loads through the school filter. All four practical tasks depend on it and the simulator runs in the browser, so no physical micro:bit is strictly required.
- If you have class micro:bits, cable them up and check they are recognised, but decide in advance whether the class runs on simulator only to avoid mid-lesson USB troubleshooting.
- Have the three code snippets ready to paste or project, since students copy them rather than write them from scratch.

Introduction to Computer Systems

HOW THE LESSON RUNS

- What is Computer Science?
- Overview of Computer Systems
- The Microbit and Makecode.com Code Editor
- The CPU
- Practical task: CPUs
- Memory
- Practical task: Memory
- Storage
- Practical task: Storage
- Wrap Up

TEACHING MOVES

- 1. What is Computer Science? Ask the class for their own examples before reading the list. Social media feeds, navigation apps and game recommendations will come up and anchor the abstract definition in things they used this morning.
- 2. Overview of Computer Systems. Draw the hardware/software split on the board and have students sort a few items you name: keyboard, browser, RAM, an app. The confusion at the boundary is the useful part.
- 3. The Microbit and Makecode.com Code Editor. Walk everyone to makecode.microbit.org together and confirm the simulator appears on screen before moving on. Stress that no physical board is needed today.
- 4. The CPU. Slow down on fetch-decode-execute and use the word cycle deliberately. Students will meet this exact term on the written paper, so name each of the three stages explicitly.
- 5. Practical task: CPUs. After the LED blinks, ask what the CPU is doing 500 milliseconds at a time. Link the forever loop to repeated fetch-decode-execute rather than treating it as just a blink.
- 6. Memory. Hammer the word volatile: RAM loses everything when power goes. Contrast it now with storage in the next step so the distinction sticks.
- 7. Practical task: Memory. Have students press button A several times and watch the count update. Point out the variable is the memory: it holds a value that changes between events.
- 8. Storage. Emphasise non-volatile: storage survives power off. Ask why a device needs both RAM and storage rather than just one, which surfaces the speed-versus-permanence trade-off.
- 9. Practical task: Storage. Be honest that the micro:bit variable is still volatile here and only illustrates accumulation. Note that unplugging loses the sequence, which is exactly what real storage would not do.
- 10. Wrap Up. Ask three students to define CPU, memory and storage in one sentence each without notes. Correct any that swap memory and storage before ending.

WHAT TO WATCH FOR

- Students treat memory and storage as the same thing, since both hold data. Anchor the difference on one word each: memory is volatile and temporary, storage is non-volatile and permanent. Ask what survives switching the device off.
- Students think the storage task actually saves the sequence permanently on the micro:bit. State plainly that the variable is volatile and is lost on power off. The task illustrates accumulation only, not true persistence.
- Students see the forever loop as unrelated to the CPU, just a way to make an LED blink. Reframe each pass of the loop as the CPU repeating fetch-decode-execute. The blink is the visible result of a repeated cycle.

DIFFERENTIATION

- Emerging: If the simulator will not load, pair the student with a neighbour whose screen works and let them read and predict the code together before typing. Give the code pre-pasted and ask them only to change one number, for example the pause value, and describe what changed.
- Developing: Ask them to predict the LED behaviour before running, then check. Prediction forces reasoning about the loop rather than just watching output.
- Proficient: Ask them to modify the counter so it resets on button B, then explain what happens to the stored value. Push toward the Higher Level distinction between volatile and non-volatile memory using their own program as evidence.