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

- Confirm makecode.microbit.org loads on the student machines and is not blocked by the school filter.
- Optionally charge micro:bits and set out USB cables. The simulator alone is enough if hardware is short.
- Open the three practical snippets yourself once (forever LED toggle, button A counter, A/B sequence) so you can unstick students quickly.

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 for everyday examples they already use. Map each one back to solving a problem with data or instructions, not to 'knowing computers'.
- 2. Overview of Computer Systems. Hold up the split: hardware is what you can touch, software is the instructions. Keep returning to both working together on data.
- 3. The Microbit and Makecode.com Code Editor. Get everyone into the MakeCode simulator before any theory. Show blocks and the code view once so nobody is hunting menus later.
- 4. The CPU. Walk fetch, decode, execute as three beats on the board. Link 'brain' talk to running instructions, not to thinking.
- 5. Practical task: CPUs. Have them run the forever toggle-and-pause loop on the simulator. Ask what repeats and who is doing the repeating: the CPU on stored instructions.
- 6. Memory. Stress volatility: RAM holds the working set only while power is on. Contrast speed of access with permanence.
- 7. Practical task: Memory. Build the button A counter with a variable. After each press, ask where the count lives between presses and what happens if they refresh.
- 8. Storage. Name non-volatile retention as the point of storage. Contrast it with the RAM story they just saw, using files that survive power-off.
- 9. Practical task: Storage. Run the A-adds-digit, B-shows-sequence task. Be explicit: the variable is still volatile. It only models accumulation, not true storage.
- 10. Wrap Up. Cold-call three roles: CPU executes, memory holds the working data, storage keeps data when power goes. Keep it spoken, no written reflection.

WHAT TO WATCH FOR

- Students treat memory (RAM) and storage as the same thing because both 'hold data'. What to do: Use the power-off test: ask what survives a restart. RAM clears, storage does not. Revisit the counter task after a simulator refresh.
- Students think the sequence task proves the micro:bit has permanent storage for variable values. What to do: State the lesson's own limit: the value is lost on power-off. Call it a model of accumulation, then name real storage examples.
- Students hear 'brain of the computer' and think the CPU understands or decides, rather than fetching and running instructions. What to do: Retell the blink loop as fetch-decode-execute only. The 'intelligence' is in the program, not in the chip.

DIFFERENTIATION

- Emerging: Give the working forever-blink code first. Have them change only the pause time or LED position so they see CPU-driven repetition before writing structure. Keep them on the simulator with a partner narrating each block aloud before they drag it.
- Developing: After each practical, ask them to point to which lines are instructions, which name is a variable, and when the value changes. Have them predict the LED or count result before they run, then reconcile with what the simulator shows.
- Proficient: Ask them to explain the blink or counter in fetch-decode-execute language, tying each line to a stage of the cycle. Extend the counter or sequence with a clear reset rule and justify whether that behaviour is memory-like or storage-like.
- Level note: Ordinary Level: name CPU, memory and storage and match each to a simple role. Higher Level: explain fetch-decode-execute and volatility versus permanence with tighter vocabulary.