LESSON 67

CPU and Memory Architecture

80 minutes (2 periods) | Computer Systems and Networks | Outcome 2.11 | Outcome 2.13 | Outcome 1.6 | Outcome 1.9 | Outcome 1.13 | Outcome 2.1 | Outcome 2.4

WHAT THIS LESSON DOES

In this lesson, you'll explore the core components of a computer's CPU and memory architecture. Learn about the ALU, registers, and memory hierarchy, then simulate the fetch-execute cycle using Python in VS Code to understand instruction processing.

CURRICULUM OUTCOMES

Outcome	What the specification asks for
2.11	describe the different components within a computer and the function of those components
2.13	describe the rationale for using the binary number system in digital computing and how to convert between binary, hexadecimal and decimal
1.6	explain the operation of a variety of algorithms
1.9	use modelling and simulation in relevant situations
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
2.1	use abstraction to describe systems and to explain the relationship between wholes and parts
2.4	illustrate examples of abstract models

WHAT STUDENTS SHOULD BE ABLE TO DO

- Understand the primary components of the CPU and their specific functions.
- Grasp the structure and significance of the memory hierarchy in computer systems.
- Comprehend the stages of the fetch-execute cycle and their role in instruction processing.
- Recognise how CPU and memory interactions enhance system performance.
- Apply theoretical knowledge to visualise processes through practical tasks like flowcharts.

BEFORE THE LESSON

- This is theory plus one drawing task. Confirm students can reach draw.io through the school filter, or have paper and pencils ready as the fallback the lesson offers.
- Have the four cycle stages clear in your own head before class: Fetch, Decode, Execute, Store, with the loop back to Fetch.

CPU and Memory Architecture

HOW THE LESSON RUNS

Introduction
CPU Components
Memory Hierarchy
The Fetch-Execute Cycle
Tying it All Together
Practical Task: Flowchart
Wrap Up

TEACHING MOVES

- 2. CPU Components. Give each component a job title as you go: ALU is the calculator, control unit is the traffic director, registers are the desk it works on. Students remember roles better than definitions.
- 3. Memory Hierarchy. Draw the hierarchy as a pyramid on the board: fast, small, expensive at the top, slow, large, cheap at the bottom. Make the trade-off the point, not the numbers.
- 4. The Fetch-Execute Cycle. Stress that von Neumann means instructions and data share the same memory. That single idea is why the program counter matters and is examinable at Higher Level.
- 5. Tying it All Together. Walk one imaginary instruction all the way through: program counter gives an address, control unit decodes, ALU adds, result stored. Name each component as it acts.
- 6. Practical Task: Flowchart. Insist on the loop back to Fetch. A flowchart that stops after Store misses the whole point that the cycle repeats. Check for correct symbols: ovals for start and end, rectangles for processes.

WHAT TO WATCH FOR

- Students think the fetch-execute cycle runs once, not continuously for every instruction in a program. During the flowchart task, require the arrow back to Fetch and ask them how many times the cycle runs for a ten-instruction program.
- Registers, cache and RAM get treated as the same thing because they are all 'memory'. Use the pyramid: registers are inside the CPU and tiny, RAM is outside and large. Ask which one holds the address of the next instruction.
- Students confuse the program counter (holds the next address) with the instruction register (holds the current instruction). Point at each on your worked example. Counter points forward, register holds what is being worked on right now.

DIFFERENTIATION

- Emerging: Give them a partly completed flowchart with the four boxes drawn and only the labels and arrows missing. Let them describe the cycle out loud to you before committing anything to the flowchart.
- Developing: Ask them to annotate their flowchart with which CPU component acts at each stage.
- Proficient: Have them extend the flowchart or explain how a cache hit versus a cache miss changes where the fetch reads from, connecting the cycle back to the memory hierarchy. Ask them to explain what von Neumann architecture makes possible that a fixed-program machine could not.