LESSON 8

Logic Gates and Boolean Algebra

80 minutes (2 periods) | Foundations of Computer Science | Outcome 2.12 | Outcome 2.13 | Outcome 1.4 | Outcome 1.9 | Outcome 2.6 | Outcome 1.22 | Outcome 3.13

WHAT THIS LESSON DOES

In this lesson, you'll learn the essentials of logic gates and Boolean algebra, key concepts in computer science. Explore how computers make decisions using AND, OR, and NOT gates, and apply these ideas through practical tasks with MakeCode.

CURRICULUM OUTCOMES

Outcome	What the specification asks for
2.12	describe the different types of logic gates and explain how they can be arranged into larger units to perform more complex tasks
2.13	describe the rationale for using the binary number system in digital computing and how to convert between binary, hexadecimal and decimal
1.4	solve problems using skills of logic
1.9	use modelling and simulation in relevant situations
2.6	construct algorithms using appropriate sequences, selections/conditionals, loops and operators to solve a range of problems, to fulfil a specific requirement
1.22	read, write, test, and modify computer programs
3.13	develop a program that utilises digital and analogue inputs

WHAT STUDENTS SHOULD BE ABLE TO DO

- Understand the fundamental concepts of logic gates and their role in computer decision-making.
- Identify and interpret circuit diagrams and truth tables for basic logic gates.
- Grasp the principles of Boolean algebra, including variables, operators, and key laws.
- Apply logic gate concepts through practical simulations and programming tasks.
- Develop skills in simplifying Boolean expressions to optimise logical designs.

BEFORE THE LESSON

- Open makecode.microbit.org yourself first and confirm it loads through the school filter. The task uses the on-screen simulator, so physical micro:bits are optional, but plug and test a couple if you want students to flash to hardware.
- Have the three gate symbols and their truth tables ready to display, and write out the Boolean operator notation students will use: AND as a dot, OR as plus, NOT as a bar over the variable.
- Print or have the four evaluation expressions and the simplification task visible for the pen-and-paper activity.

Logic Gates and Boolean Algebra

HOW THE LESSON RUNS

- Introduction
- What are Logic Gates?
- Circuit Diagrams & Truth Tables
- Practical Task: Logic Gates
- Introduction to Boolean Algebra
- Practical Activity: Boolean Algebra
- Wrap-Up

TEACHING MOVES

- 2. What are Logic Gates? Anchor the whole idea in one image: a gate is a decision-maker that only knows 0 and 1. Use the light switch, then have students predict the output of an AND gate before you show the table.
- 3. Circuit Diagrams & Truth Tables. Build one truth table together on the board, filling every input row first so students see the pattern of combinations, then the output column. Stress that a table with n inputs has 2 to the power n rows.
- 4. Practical Task: Logic Gates. Make the link explicit: the AND operator in the if-condition is the AND gate in code. Insist students use 'else if' for the OR case, not a second separate if, so only one icon shows at a time.
- 5. Introduction to Boolean Algebra. Frame Boolean algebra as writing down what the gates do in shorthand. Connect the notation straight back to the gates they just simulated so it feels like translation, not new content.
- 6. Practical Activity: Boolean Algebra. Circulate and check working, not just answers. For the simplification task, ask students to name which law they used at each step rather than jumping to the final result.

WHAT TO WATCH FOR

- Students read the AND operator in code as loose everyday 'and', so they expect a heart when only one button is pressed. Return to the AND truth table: output is 1 only when both inputs are 1. Have them trace their code against every row before running it.
- Confusing the OR gate with 'exclusive or', assuming OR is false when both inputs are true. Show the OR row where A=1 and B=1 gives 1. Point out that plain OR means 'at least one', and both counts.
- Treating simplification as guessing rather than applying named laws step by step. Require one law per line with the law named. A shorter expression that cannot be justified does not count as simplified.

DIFFERENTIATION

- Emerging: If MakeCode is fighting them, pair them with a working screen and let them focus on tracing the truth table by hand first. Give the two-input AND table only, fully worked, and ask them to build the OR table by changing just the output column.
- Developing: Ask them to predict the on-screen output for each button combination before pressing, then check against the simulator. Have them explain in words why 'else if' matters here before you accept the code.
- Proficient: Stretch to a three-input gate: build the eight-row truth table and add a NOT condition to the micro:bit program. Give a longer expression to simplify using De Morgan's laws, and ask them to verify the result with a full truth table, evidence they could reuse in an Applied Learning Task.