ANSWERS

A8.2 Circuits, expressions and truth tables

Boolean algebra and logic circuits · A level · OCR H446 1.4.3, AQA 7517 4.6.4.1, Eduqas A500QS 1.2 · about 20 min

What this lesson is about

Reading and drawing circuits, defining a problem in Boolean logic, and sum of products from a truth table.

- From circuit to expression
- From expression to truth table
- From expression to circuit
- Defining a problem with Boolean logic
- From truth table to expression: sum of products

Questions

6 marks in all

1. Q = ¬(A ∨ B) ∨ (B ∧ C). A = 0, B = 1 and C = 0. What is Q? [1 mark]

> **Answer:** 0. A ∨ B is 1, so ¬(A ∨ B) is 0; B ∧ C is 0; 0 ∨ 0 is 0.

2. For how many of the 8 input rows is Q = ¬(A ∨ B) ∨ (B ∧ C) equal to 1? [1 mark]

> **Answer:** 4. Rows 000, 001, 011 and 111: the NOR gives the first two and B ∧ C gives the last two.

3. "The alarm sounds if the robot has bumped (K), or if it is close (C) while the battery is low (L)." Which expression matches? [1 mark]

- ☐ **A** K ∨ (C ∧ L)
- ☐ **B** (K ∨ C) ∧ L
- ☐ **C** K ∧ C ∧ L
- ☐ **D** K ∨ C ∨ L

> **Answer:** A. "While" joins C and L with AND; "or" adds K on its own.

4. A truth table for A, B and C has Q = 1 only on the row A=1 B=0 C=1. Write its minterm in OCR notation, with the letters in the order A, B, C. [1 mark]

> **Answer:** A ∧ ¬B ∧ C. Each input that is 0 in the row gets a NOT; the letters are joined by AND.

5. How many gates does the circuit for (A ⊻ B) ∧ ¬C need? [1 mark]

- ☐ **A** 3
- ☐ **B** 2
- ☐ **C** 4
- ☐ **D** 5

> **Answer:** A. One XOR, one NOT and one AND: one gate per operator.

6. On a circuit diagram, what does a dot where two wires meet show? [1 mark]

- ☐ **A** The wires are connected (a junction)
- ☐ **B** The wires cross without connecting
- ☐ **C** A NOT gate
- ☐ **D** The output of the circuit

> **Answer:** A. A junction dot means the signal splits to both wires; crossing wires without a dot are not connected.

The task: truth table to expression

Write `sum_of_products(outputs)`. Its input `outputs` is a list of eight integers, each 0 or 1: the Q column of a truth table for inputs A, B and C, with the rows in binary order, so `outputs[0]` is the row A=0 B=0 C=0 and `outputs[7]` is A=1 B=1 C=1. It returns a string: `Q =` followed by the minterms joined with `OR`. Each minterm is in brackets, lists A, B and C in that order joined with `AND`, and writes an input that is 0 in that row as `NOT` and a space before the letter. If no row is 1, it returns `Q = 0`. For example, `[0, 0, 1, 0, 0, 0, 0, 1]` gives `Q = (NOT A AND B AND NOT C) OR (A AND B AND C)`. Print the result for these three tables, one per line, in this order: `[0, 1, 1, 1, 1, 1, 1, 1]`, then `[0, 0, 1, 0, 0, 0, 1, 1]`, then `[0, 0, 0, 0, 0, 0, 0, 0]`.

```
# the two lines every program starts with: the commands, then the robot
from bugbot import *
connect()

def sum_of_products(outputs):
    return "Q = "

print(sum_of_products([0, 1, 1, 1, 1, 1, 1, 1]))
print(sum_of_products([0, 0, 1, 0, 0, 0, 1, 1]))
print(sum_of_products([0, 0, 0, 0, 0, 0, 0, 0]))
```

The hint students can ask for: Row number i, written as three binary digits, is the values of A, B and C on that row. For each row whose output is 1, turn each digit into a letter or NOT and a letter, join them with AND inside brackets, then join the terms with OR. Think about what to print when no row is 1.

A solution

```
# the two lines every program starts with: the commands, then the robot
from bugbot import *
connect()

def sum_of_products(outputs):
    names = ["A", "B", "C"]
    terms = []
    for i in range(8):
        if outputs[i] == 1:
            bits = format(i, "03b")
            literals = []
            for k in range(3):
                if bits[k] == "1":
                    literals.append(names[k])
                else:
                    literals.append("NOT " + names[k])
            terms.append("(" + " AND ".join(literals) + ")")
    if len(terms) == 0:
        return "Q = 0"
    return "Q = " + " OR ".join(terms)

stop_rule = [0, 1, 1, 1, 1, 1, 1, 1]
led_rule = [0, 0, 1, 0, 0, 0, 1, 1]
never = [0, 0, 0, 0, 0, 0, 0, 0]
print(sum_of_products(stop_rule))
print(sum_of_products(led_rule))
print(sum_of_products(never))
```

Any program that meets the task's checks is marked correct in the simulator; this is one way, not the only way.