ANSWERS

A8.1 Logic gates and notation

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

NOT, AND, OR, XOR, NAND and NOR, the notation each board uses, and why NAND alone can build anything.

- The six gates
- Three ways to write the same thing
- Gates in Python
- NAND can build anything
- A robot example

Questions

6 marks in all

1. When does a NAND gate output 0? [1 mark]

- ☐ **A** Only when both inputs are 1
- ☐ **B** Only when both inputs are 0
- ☐ **C** When the inputs are different
- ☐ **D** Never

> **Answer:** A. NAND is NOT AND: AND is 1 only when both inputs are 1, so NAND is 0 only then.

2. When does a NOR gate output 1? [1 mark]

- ☐ **A** Only when both inputs are 0
- ☐ **B** When at least one input is 1
- ☐ **C** Only when both inputs are 1
- ☐ **D** When the inputs are different

> **Answer:** A. NOR is NOT OR: OR is 0 only when both inputs are 0, so NOR is 1 only then.

3. In OCR notation, which expression means A XOR B? [1 mark]

- ☐ **A** A ⊻ B
- ☐ **B** A ∨ B
- ☐ **C** A ∧ B
- ☐ **D** ¬(A ∧ B)

> **Answer:** A. OCR writes XOR as ⊻, OR as ∨, AND as ∧ and NOT as ¬.

4. What does this program print? [1 mark]

```
a = 1
b = 0
print(1 - (a & b), 1 - (a | b), a ^ b)
```

> **Answer:**
>
> ```
> 1 0 1
> ```
>
> NAND(1, 0) is 1, NOR(1, 0) is 0 and XOR(1, 0) is 1.

5. What does this program print? [1 mark]

```
a = 1
print(~a, 1 - a, a ^ 1)
```

> **Answer:**
>
> ```
> -2 0 0
> ```
>
> ~ flips every bit of a signed integer, so ~1 is -2; 1 - a and a ^ 1 both give the single-bit NOT, 0.

6. Why are NAND and NOR called universal gates? [1 mark]

- ☐ **A** Any logic circuit can be built from NAND gates alone, or from NOR gates alone
- ☐ **B** They are used in every processor made
- ☐ **C** They have the most rows in their truth tables
- ☐ **D** They work with any number of inputs

> **Answer:** A. NOT, AND and OR can all be made from NANDs (or from NORs), and every circuit can be made from those.

The task: everything from NAND

You are given `NAND(a, b)`, which takes two bits (each 0 or 1) and returns 0 or 1. Write four functions, each taking bits and returning 0 or 1, built **only** by calling `NAND` or the gates you have already built from it: `NOT(a)`, `AND(a, b)`, `OR(a, b)` and `XOR(a, b)`. Your program may not use `and`, `or`, `not`, comparisons such as `==`, a minus sign, or the operators `&`, `|`, `^` and `~`. Then use loops to print one line for each of the four input combinations, in binary order, in exactly this form: `A=0 B=1 NOT_A=1 AND=0 OR=1 XOR=1`

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

def NAND(a, b):
    return [1, 1, 1, 0][2 * a + b]

def NOT(a):
    return 0

def AND(a, b):
    return 0

def OR(a, b):
    return 0

def XOR(a, b):
    return 0
```

The hint students can ask for: NOT and AND are worked out in the lesson. For OR, think about what a NAND gives when both of its inputs have already been inverted. For XOR, sketch it on paper first: the NAND of the two inputs is a signal worth computing once and using more than once.

A solution

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

def NAND(a, b):
    return [1, 1, 1, 0][2 * a + b]

def NOT(a):
    return NAND(a, a)

def AND(a, b):
    return NOT(NAND(a, b))

def OR(a, b):
    return NAND(NOT(a), NOT(b))

def XOR(a, b):
    n = NAND(a, b)
    return NAND(NAND(a, n), NAND(b, n))

for a in [0, 1]:
    for b in [0, 1]:
        print(f"A={a} B={b} NOT_A={NOT(a)} AND={AND(a, b)} OR={OR(a, b)} XOR={XOR(a, b)}")
```

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