MATH 8

LOGIC II: MORE LOGIC OPERATIONS — SHORT 2026–27

Last time we met NOT, AND, OR, truth tables, and the logic laws. Today: formulas that are *always* true, four new operations, and how all of logic turns into electronic circuits that add numbers and even remember. In 1937 a 21-year-old student, Claude Shannon, showed that switch circuits are exactly Boolean formulas — switches in series act like AND, in parallel like OR — and the digital age began.

Tautologies and Contradictions

A **tautology** is a formula that is always true, whatever the truth values of its variables; a **contradiction** is always false. $A \vee \neg A$ is a tautology (the *law of excluded middle*); $A \wedge \neg A$ is a contradiction. To show a formula is a tautology, check *every* row of its truth table; to show it is not, one false row is enough. Every logic law is secretly a tautology: two formulas X and Y are equivalent exactly when "X and Y have the same truth value" is always true.

Four New Operations

A	B	$A \text{ XOR } B$	$A \text{ NAND } B$	$A \text{ NOR } B$	$A \text{ XNOR } B$
T	T	F	F	F	T
T	F	T	T	F	F
F	T	T	T	F	F
F	F	F	T	T	T

- **XOR** (exclusive or): exactly one is true — "at least one, but not both": $A \text{ XOR } B \equiv (A \vee B) \wedge \neg(A \wedge B) \equiv (A \wedge \neg B) \vee (\neg A \wedge B)$. XOR is commutative and associative, $A \text{ XOR } F \equiv A$, $A \text{ XOR } T \equiv \neg A$, $A \text{ XOR } A \equiv F$. A chain $A_1 \text{ XOR } A_2 \text{ XOR } \cdots \text{ XOR } A_n$ is true exactly when an *odd* number of the A_i are true.
- **NAND** (not and): $A \text{ NAND } B \equiv \neg(A \wedge B)$ — false only when both are true.
- **NOR** (not or): $A \text{ NOR } B \equiv \neg(A \vee B)$ — true only when both are false.
- **XNOR** (exclusive nor): $A \text{ XNOR } B \equiv \neg(A \text{ XOR } B)$ — true when A and B have the *same* value. (Next time it returns as "A if and only if B.")

NAND Alone Is Enough

Step 1. Any truth table can be written with $\neg, \wedge, \vee$: for each row where the output is T, write an AND that is true only in that row (e.g. $A \wedge \neg B$ for the row $A = T$, $B = F$), then join these ANDs with OR.

Step 2. $\neg, \wedge, \vee$ can be built from NAND alone:

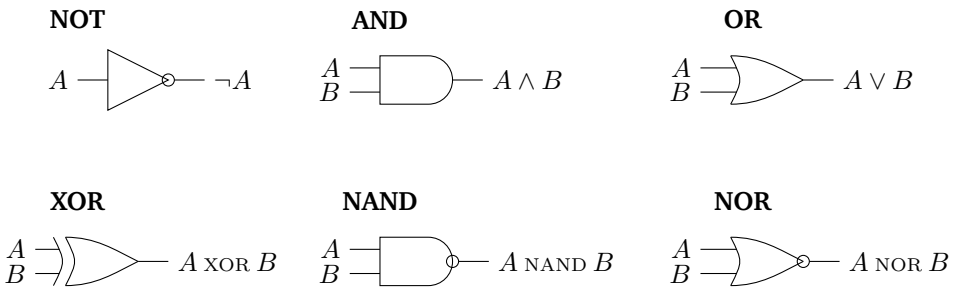
Building NOT, AND, OR from NAND

$$\neg A \equiv A \text{ NAND } A, \qquad A \wedge B \equiv (A \text{ NAND } B) \text{ NAND } (A \text{ NAND } B), \qquad A \vee B \equiv (A \text{ NAND } A) \text{ NAND } (B \text{ NAND } B).$$

The first holds since $\neg(A \wedge A) \equiv \neg A$; the second negates a NAND; the third is De Morgan: $A \vee B \equiv \neg(\neg A \wedge \neg B)$. So *every* logical function can be built from NAND gates only. The same is true for NOR ($\neg A \equiv A \text{ NOR } A$).

Logic Gates

In circuits, 1 means T and 0 means F. Each operation has a standard symbol (a small circle means "negate"; XNOR is an XOR symbol with a circle):



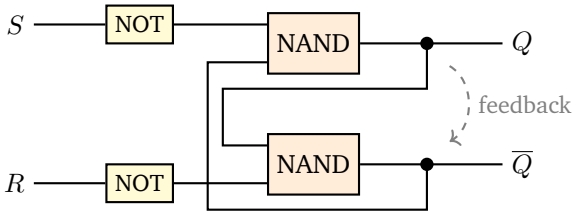
Binary Addition with Logic Gates

Adding two bits gives a **sum bit** and a **carry bit**: $0 + 0 = 0$, $0 + 1 = 1 + 0 = 1$, $1 + 1 = 10_2$. Comparing columns, the sum is $S = A \text{ XOR } B$ and the carry is $C = A \wedge B$: one XOR gate and one AND gate make a **half adder**. Adding longer numbers column by column, each column must also add the carry coming in from the right — three bits at once. That circuit is called a **full adder**.

Memory: The SR Flip-Flop

So far a circuit's output depended only on its current inputs. The **SR flip-flop** can *remember* one bit. It runs on one fact: **if at least one input of a NAND gate is 0, its output is 1.**

SR Flip-Flop (NOT-NAND Implementation)



The top NAND gate gets $\overline{S}$ and the feedback wire $\overline{Q}$, and outputs Q; the bottom one gets $\overline{R}$ and the feedback wire Q, and outputs $\overline{Q}$.

- **Set** ($S = 1$, $R = 0$): $\overline{S} = 0$, so the top gate outputs $Q = 1$; then the bottom gate gets 1 and 1, so $\overline{Q} = 0$.
- **Reset** ($S = 0$, $R = 1$): $\overline{R} = 0$, so $\overline{Q} = 1$; then the top gate gets 1 and 1, so $Q = 0$.
- **Hold** ($S = 0$, $R = 0$): each NAND gate just negates the feedback wire, and the circuit *keeps* whichever state it was in — that is memory!
- $S = R = 1$ makes both outputs 1, so it is **forbidden**.

Common Mistakes

- **Confusing OR and XOR**: OR is true when at least one input is true; XOR when exactly one is.
- **NAND/NOR truth values**: NAND is false only when both inputs are true; NOR is true only when both are false.
- **Calling a formula a tautology after checking a few rows**: one false row is enough to show it is *not* a tautology.

Homework

1. Determine whether each is a tautology, a contradiction, or neither:
   (a) $(A \text{ NAND } B) \vee A$
   (b) $(A \text{ NOR } B) \wedge A$
   (c) $(A \text{ XOR } B) \text{ XOR } (A \text{ XNOR } B)$
   (d) $(A \text{ NAND } B) \vee (A \text{ NOR } B)$

2. Simplify each expression to T, F, A, or $\neg A$:
   (a) $A \text{ NAND } T$
   (b) $A \text{ NOR } F$
   (c) $A \text{ XOR } \neg A$
   (d) $A \text{ NAND } \neg A$
   (e) $A \text{ NOR } \neg A$
   (f) $(A \text{ XOR } B) \text{ XOR } B$

3. (a) Show with a truth table that $A \text{ XOR } B \equiv (A \text{ NAND } N) \text{ NAND } (B \text{ NAND } N)$, where $N = A \text{ NAND } B$. Draw the circuit using the gate symbols from this handout. How many NAND gates does it use?
   (b) Suppose instead you translated $(A \wedge \neg B) \vee (\neg A \wedge B)$ into NAND gates piece by piece (NOT = 1 gate, AND = 2 gates, OR = 3 gates). How many gates would that take?

4. M Using only NOR gates, build a circuit for $A \text{ XOR } B$. (It can be done with 5 gates.)

5. **Mystery circuit.** An XOR gate and an AND gate both receive the inputs A and B. Their two outputs go into a second XOR gate, whose output is the output of the circuit. Which basic operation does this circuit compute? Explain why.

6. **Three switches.** A long hallway has three doors, each with a switch for the same ceiling light. Flipping any one switch changes the light, and when all three switches are down, the light is off. Let A, B, C mean "switch 1 / 2 / 3 is up."
   (a) Explain why the light is on exactly when an odd number of switches are up.
   (b) Write a formula for "the light is on" using XOR.
   (c) What would the formula be with four switches?

7. **Secret codes.** As in a "one-time pad," a message M is coded by XORing it, bit by bit, with a secret key K: the coded message is $C = M \text{ XOR } K$.
   (a) Code the message `110010` with the key `101011`, then decode the result with the same key.

(b) Prove, using the properties of XOR, that $(M \text{ XOR } K) \text{ XOR } K \equiv M$ for every M and K, so decoding always works.

(c) A spy learns both a message M and its coded version C. Show that she can now find the key.

8. **Finding an error.** Nine data bits are written in a 3×3 square. Then a parity bit is added at the end of each row, a parity bit at the bottom of each column, and one more in the corner, so that *every row and every column* of the 4×4 grid has an even number of 1s. During transmission exactly one bit was flipped, and the grid below arrived. Which bit was flipped? Explain how the parity bits find it.

1	0	1	0
1	1	1	0
1	1	1	1
0	0	1	1

9. M **Full adder from half adders.** A *half adder* takes two bits and outputs their sum bit (XOR) and carry (AND). A *full adder* adds three bits A, B, C_{in}, and outputs a sum bit S and a carry-out C_{out} so that $A + B + C_{\text{in}} = 2 \cdot C_{\text{out}} + S$ (for example, $1 + 1 + 1 = 11$ in binary).

(a) Show how to build a full adder from two half adders and one OR gate. Describe the wiring and check all 8 rows with a table.

(b) Explain why the OR gate could be replaced by an XOR gate without changing anything.

10. H An SR flip-flop (built as in this handout) starts with $Q = 0$. The following sequence of inputs is applied: $(S, R) = (1, 0)$, then $(0, 0)$, then $(0, 1)$, then $(0, 0)$, then $(1, 0)$. What is the value of Q after each step?

11. M **Mixed operators.**

(a) Rewrite $(A \text{ NAND } B) \text{ NOR } C$ using only $\neg$, $\wedge$, $\vee$, and simplify as much as you can.

(b) Do the same for $A \text{ NAND } (B \text{ NOR } C)$.

(c) Describe in words exactly when $(A \text{ XOR } B) \text{ XNOR } C$ is true.

(d) Show that $(A \text{ XNOR } B) \text{ XOR } C \equiv (A \text{ XOR } B) \text{ XNOR } C$.

12. H **A Knight-or-Knave detector.** On the island of knights (who always tell the truth) and knaves (who always lie), you may ask an islander yes/no questions. Write "yes" as T and "no" as F, and let N be the statement "this islander is a knave."

(a) Explain why, if you ask "Is P true?", the answer you hear is $P \text{ XOR } N$.

(b) You are at a fork in the road and want to know whether the left road leads to the village (call this statement V), but you don't know whether the islander is a knight or a knave. Show that if you ask "Is $V \text{ XOR } N$ true?" — in words, "Is exactly one of these true: the left road leads to the village, you are a knave?" — the answer you hear is always V.