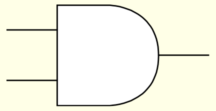
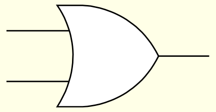
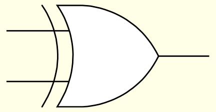
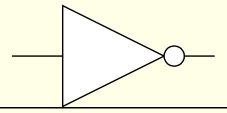
Boolean Logic KO
Define problems using Boolean logic

AND		∧	A ∧ B A AND B
OR		∨	A ∨ B A OR B
XOR		⊻	A ⊻ B A XOR B
NOT		¬	¬A NOT A
The same as		≡	A ≡ B A is the same as B

Truth tables:

AND ∧		
A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1

OR ∨		
A	B	Output
0	0	0
0	1	1
1	0	1
1	1	1

NOT ¬	
A	Output
0	1
1	0

The XOR gate produces a 1 output if either, but not both of the inputs are 1.

XOR ⊻		
A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

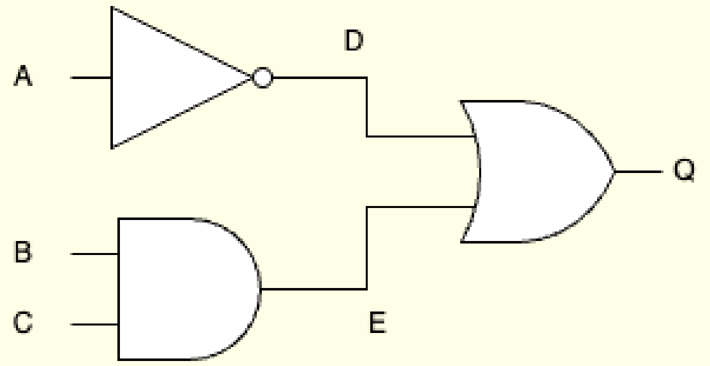
NAND		
A	B	Output
0	0	1
0	1	1
1	0	1
1	1	0

Multiple logic gates can be connected to produce an output based on multiple inputs.

This circuit can be represented by the expression

Q = ¬A ∨ (B ∧ C)

or alternatively as Q = (NOT A) OR (B AND C)

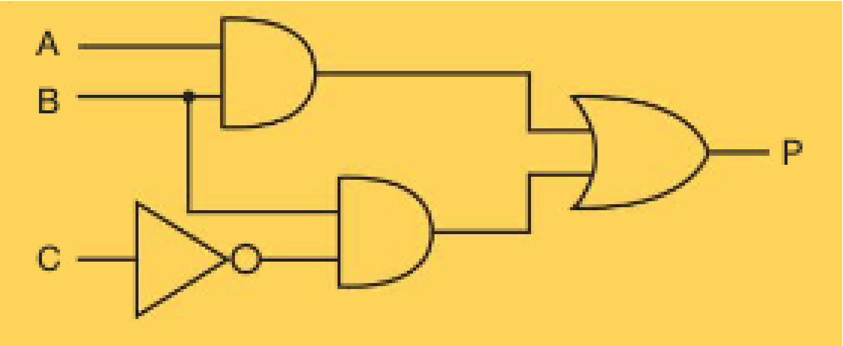


Evaluate the brackets first

Input A	Input B	Input C	D = ¬ A	E = B ∧ C	Output Q = D ∨ E
0	0	0	1	0	1
0	0	1	1	0	1
0	1	0	1	0	1
0	1	1	1	1	1
1	0	0	0	0	0
1	0	1	0	0	0
1	1	0	0	0	0
1	1	1	0	1	1

How to write Boolean expression represented in a logic diagram:

Write the Boolean expression represented by the logic diagram below, using AND, OR and NOT instead of symbols. Then write the same expression using symbols.



First write (A AND B).

Then write (B AND NOT C)

These are the inputs to the OR gate

so the expression is:

P = (A AND B) OR (B AND NOT C)

P = (A ∧ B) ∨ (B ∧ ¬ C)

Defining problems with Boolean logic

A boiler has two sensors, a pressure sensor and a temperature sensor. If either the temperature (T) or the pressure (P) is too high, a valve (V) will close.

This can be expressed as V = T v P or alternatively as V = T OR P

The table representing these conditions could be drawn as follows:

Input	Binary value	Condition
T	1	Temperature too high
	0	Temperature not too high
P	1	Pressure too high
	0	Pressure not too high

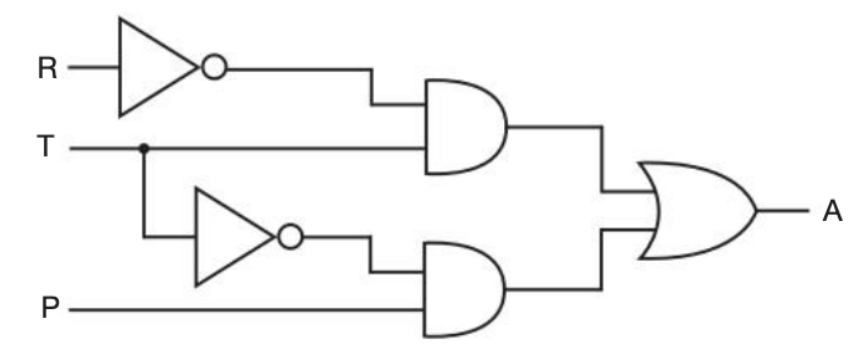
Worked Example

A chemical process has a sensor to detect a dangerous situation, in which case it sounds an alarm (A). The alarm is sounded if:

either temperature >= 100°C AND rotator is OFF
or
PH > 6 AND temperature < 100°C

A table can be drawn to represent these conditions as Boolean values.

Input	Binary value	Condition
T	1	Temperature >= 100°C
	0	Temperature < 100°C
R	1	Rotator ON
	0	Rotator OFF
P	1	PH > 6
	0	PH <= 6



The conditions can be written as

$A = (T \wedge \neg R) \vee (P \wedge \neg T)$ or alternatively as A = (T AND NOT R) OR (P AND NOT T)

Input R	Input T	Input P	$X = T \wedge \neg R$	$Y = P \wedge \neg T$	$A = X \vee Y$
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	1	0	1
0	1	1	1	0	1
1	0	0	0	0	0
1	0	1	0	1	1
1	1	0	0	0	0
1	1	1	0	0	0

Karnaugh maps (K Maps)

The four-variable problem

With four variables, each row or column represents a combination of two variables.
Represent the expression and hence simplify the expression.

AB \ CD	00	01	11	10
00				
01				
11	1	1	1	1
10	1	1	1	1

This simplifies to A.

A Karnaugh map is shown below.

AB \ CD	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	1	1	1
10	0	0	0	0

What Boolean expression does this map show?

A Karnaugh map is shown below.

AB \ CD	00	01	11	10
00	0	1	0	0
01	0	1	0	0
11	1	1	1	1
10	1	1	1	1

What Boolean expression does this map show?

Worked Example: (¬C ∧ ¬D) V (C ∧ ¬D) Step 1: What is repeated in both backets? Step 2: Apply reverse distribution (factoring) (F*actoring* out of an expression by identifying the common factor - Finding what to multiply together to get an *expression*. It is like "splitting" an *expression* into a multiplication of simpler *expressions*.) Step 3: Apply General OR rules NOT X OR X = 1 **Practice Questions:** State the simplified versions of the following Boolean expressions and the rule applied ¬ ¬A (¬ A ^ ¬ B) A v (A^ B) v ¬ A.	Rules for simplifying Boolean expressions X – is a variable (0 or 1) <table><tr><th>General AND rules</th><th>General OR rules</th></tr><tr><td>X AND 0 = 0 X AND 1 = X X AND X = X NOT X AND X = 0</td><td>X OR 0 = X X OR 1 = 1 X OR X = X NOT X OR X = 1</td></tr></table> **De Morgan's Laws -** Breaking a negation and changing the operator between two inputs. $\neg(A \wedge B) \equiv \neg A \vee \neg B$ $\neg(A \vee B) \equiv \neg A \wedge \neg B$ **Distribution** - Expanding brackets **Reverse Distribution –** Taking out the common factor from multiple terms and expressing them in bracketed form. This is also known as **factoring** or **factoring out the common factor** $A \wedge (B \vee C) \equiv (A \wedge B) \vee (A \wedge C)$ $A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$ $A \wedge (B \wedge C) \equiv (A \wedge B) \wedge (A \wedge C)$ $A \vee (B \vee C) \equiv (A \vee B) \vee (A \vee C)$ **Association** – allows for the removal of brackets and regrouping of variables $(A \wedge B) \wedge C \equiv A \wedge (B \wedge C) \equiv A \wedge B \wedge C$ $(A \vee B) \vee C \equiv A \vee (B \vee C) \equiv A \vee B \vee C$ **Commutation – order not important** $A \vee B \equiv B \vee A$ $A \wedge B \equiv B \wedge A$ **Double Negation** $\neg\neg A \equiv A$ **Absorption** This means that if a is true, the entire expression is true regardless of b. $a \wedge (a \vee b) = a \vee (a \wedge b) = a$

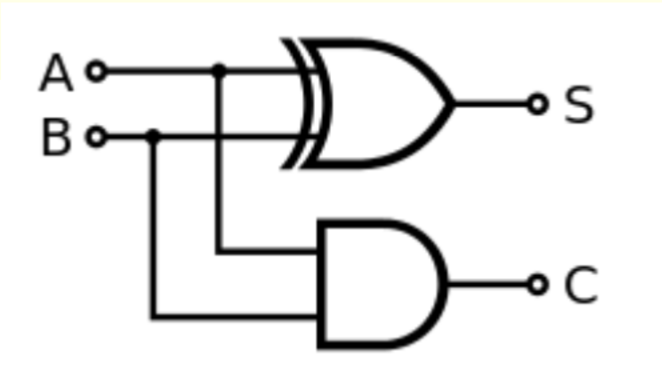
Adders and Flip Flops

Half Adder
A half adder has two inputs, A and B, and two outputs, Sum and Carry. The circuit is formed from just two logic gates: AND and XOR.

S = SUM
C – Carry

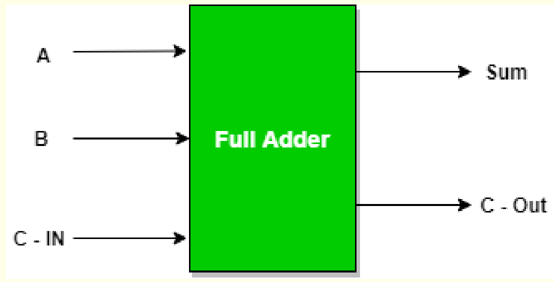
0+0 = 0 SUM 0 No Carry 0
1+0 = 1 SUM 1 No Carry 0
1+1 =1 0 SUM 0 Carry 1
1+1+1=11 SUM 1 Carry 1

A	B	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



Full Adder A full adder is similar to a half adder, but has an additional input, allowing for carry in to be represented.

A	B	C_{in}	C_{out}	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



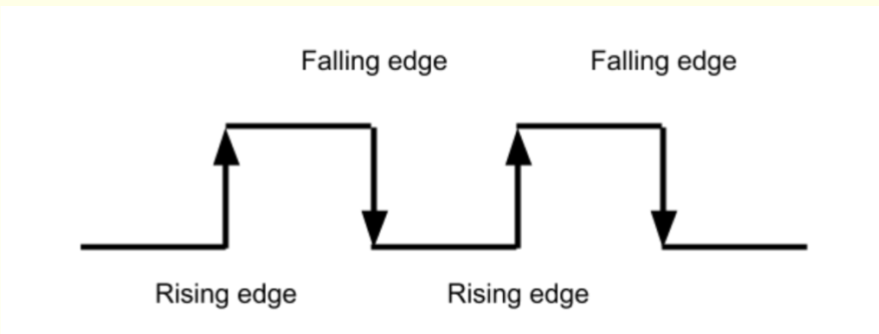
Cover the Cout and SUM columns in the Truth Table and fill in the table yourself

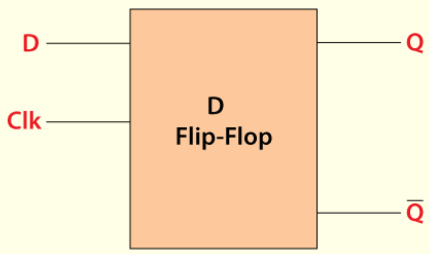
D Type Flip Flops

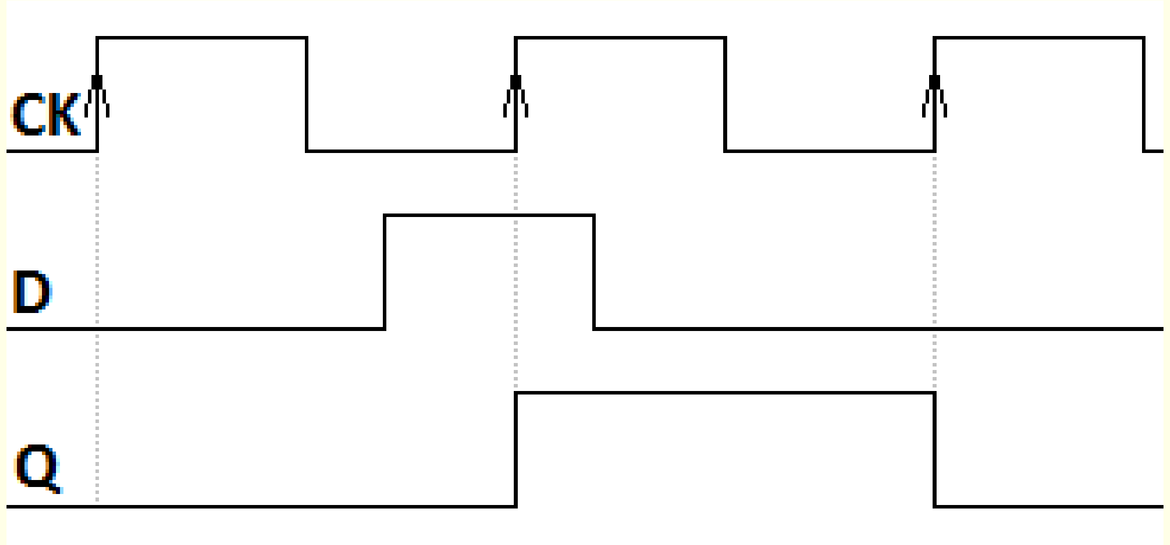
A flop flop is a type of logic circuit which can store the value of one bit.

A flip flop has two inputs, a control signal and a clock input.

Clock Pulse







A D-type flip flop can only change at a rising edge, the start of a clock tick.

D – INPUT
Q – OUTPUT