School of Science
& Technology

www.city.ac.uk

Systems Architecture

IN1006

Logic Gates and Circuits

—Dr H. Asad

Contents

CITY
UNIVERSITY OF LONDON
EST 1894

Question?

What do you think the following addition is implemented inside a computer using circuits?

X= 347+589

Boolean Algebra

- Boolean algebra is a **mathematical system** for the manipulation of variables that can have one of two values.
- In formal logic, these values are **“true”** and **“false.”**
- In digital systems, these values are **“on”** and “**off**,” **1** and **0**, or **“high”** and **“low.”**
- Boolean expressions are created by performing **operations** on **Boolean variables**.
- Common Boolean operators include **AND**, **OR,** and **NOT.**

AND & OR Operators

A Boolean operator can be completely described using a **truth table.**

The truth tables for the Boolean operators **AND** and **OR** are shown on the right.

- X AND Y is true (1) when both X and Y are true.
- X OR Y is true (1) when at least one of X and Y is true

- The **AND** operator is also known as a **Boolean product.**
- The **OR** operator is the **Boolean sum.**

X AND Y

X	Y	XY
0	0	0
0	1	0
1	0	0
1	1	1

X OR Y

X	Y	X+Y
0	0	0
0	1	1
1	0	1
1	1	1

Boolean Algebra and Digital Computers

- Digital computers contain **circuits** that implement **Boolean functions.**
- The **simpler** that we can make a **Boolean function**, the **smaller the circuit** that will result.
- Simpler circuits are cheaper to build, consume less power, and run faster than complex circuits.

The Hardware Hierarchy

Silicon ➔ Transistors ➔ Logic Gates

Transistors Logic Gates build:

- **Combinational Circuits**
  - **the output** depends on **the combination of inputs** at that point
  - Total **disregard to the past state** of the inputs.
  - Functional units; adder/subtractor/Logical
- **Sequential Circuits**
  - **outputs depend** on a combination of both the **present inputs** as well as the **previous outputs.**
    - Memory units

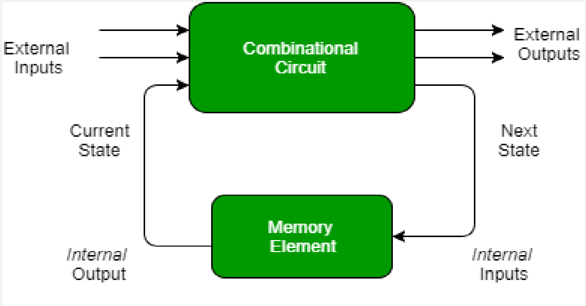


Figure: Sequential Circuit

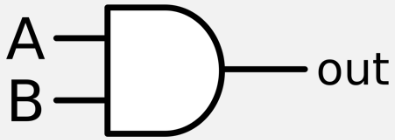
How to build circuits to implement Boolean logic?

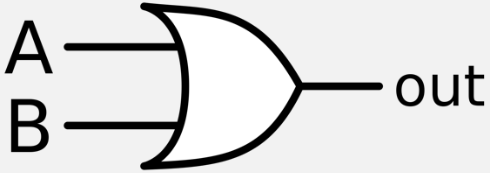
https://www.geeksforgeeks.org/difference-between-combinational-and-sequential-circuit/

Logic Gates

- Boolean functions are implemented in digital computer circuits called **gates.**
- **A gate is an electronic device that produces a result based on two or more input values.**
- Gates consist of one to six **transistors**, but digital designers think of them as a single unit.
- **Integrated circuits** contain collections of gates suited to a particular purpose.

- **Transistor:** the basic physical component of a computer.
- **Gate:** the basic logic element.

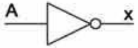




Fundamental Logic Gates

AND, OR, NOT, NAND, NOR, XOR

Logic Gates

Name	NOT	AND	NAND	OR	NOR	XOR	XNOR
Alg. Expr.	$\overline{A}$	AB	$\overline{AB}$	$A+B$	$\overline{A+B}$	$A \oplus B$	$\overline{A \oplus B}$
Symbol							
Truth Table	A X 0 1 1 0	B A X 0 0 0 0 1 0 1 0 0 1 1 1	B A X 0 0 1 0 1 1 1 0 1 1 1 0	B A X 0 0 0 0 1 1 1 0 1 1 1 1	B A X 0 0 1 0 1 0 1 0 0 1 1 0	B A X 0 0 0 0 1 1 1 0 1 1 1 0	B A X 0 0 1 0 1 0 1 0 0 1 1 1

Constructing an AND Gate

- How could an AND gate be built from transistors?
  - Transistors are **switches** that control the flow of electricity
  - Transistors are **controlled by electrical signals** from inputs
- AND gate can be built using two switches in series

A representation of a two-input AND gate as switches in serial controlled by inputs A and B.

The AND Gate

- The output of an **AND** gate is **true**,1, if **all** of its inputs are true.
  - Therefore: The output of an AND gate is false, 0,

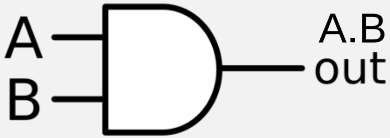
    if any of its inputs are false.

- AND is a **multiplicative operator**
- **A AND B** can also be written as **A·B**, or simply **AB**

  (so in an expression AND is written as a dot)

The AND Gate: Circuit Symbols

- Truth table describes relationship between inputs and outputs
  - How many entries are there in a truth table with n inputs?
  - For an **n-input gate** the truth table will have 2^n entries.



A two-input AND gate

Inputs		Outputs
A	B	AB
0	0	0
0	1	0
1	0	0
1	1	1

Truth table for 2 input AND

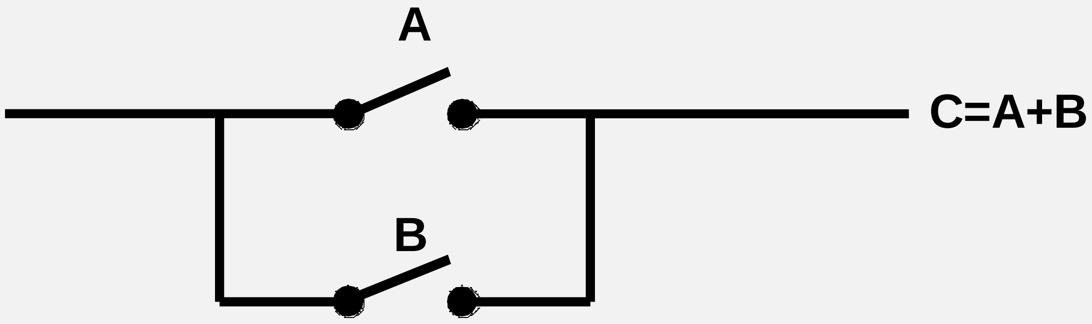
A three-input AND gate

Input			Outputs
A	B	C	ABC
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Truth table for 3 input AND

Constructing an OR Gate

- OR gate can be built using two switches in parallel



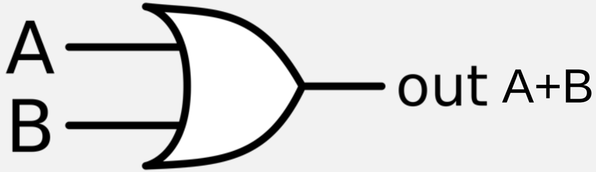
A representation of a two-input OR gate as switches in parallel controlled by inputs A and B.

The OR Gate

- The output of an **OR** gate is **true**, 1, if **any** of its inputs are true.
  - Therefore: The output of an OR gate is false, 0, if all of its inputs are false.
- OR is an **additive operator**
- In a Boolean expression, **A OR B** can also be written as **A+B**

The OR Gate: Circuit Symbols, Truth Table & Word level

OR applied to 8-bit input words A and B to produce C = A+B



A two-input OR gate

Inputs		Outputs
A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1

Truth table for 2 input OR

1 1 0 1 1 1 0 0 ¬ word A

0 1 1 0 0 1 0 1 ¬ word B

1 1 1 1 1 1 0 1 ¬ word C=A+B

A three-input OR gate

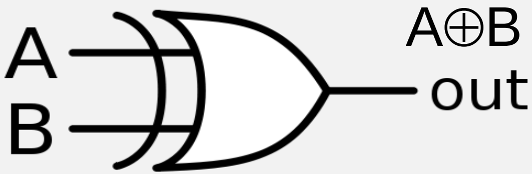
Input			Outputs
A	B	C	A+B+C
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Truth table for 3 input OR

Exclusive OR --- XOR Logic, Symbol & Truth Table

- Logical OR is unlike "or" in English
  - "Would you like tea or coffee?"
    - English: "Would you like one of tea or coffee but not both?"
    - Logical: "Would you like tea, coffee or both?"
- English "or" is like Logical Exclusive OR
  - The output of an Exclusive OR **(XOR)** gate is **true** if **one and only one** of its inputs is true.
  - XOR is written as A ⊕ B

A two-input XOR gate



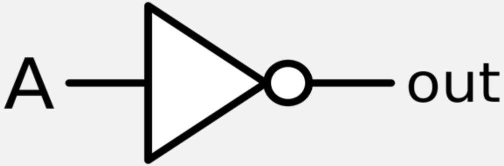
Inputs		Outputs
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Truth table for 2 input XOR

The NOT Gate or Inverter

- NOT gates have only one input
  - NOT gates are also known as **inverters**
- The output of a **NOT** gate is **true** if its **input is false**.
  - Corollary: The output of a NOT gate is false if its input is true.
- In a formula, the logical symbol for negation is a bar over the expression
  - NOT A can be written as **A** or **A**’

NOT gate circuit symbol



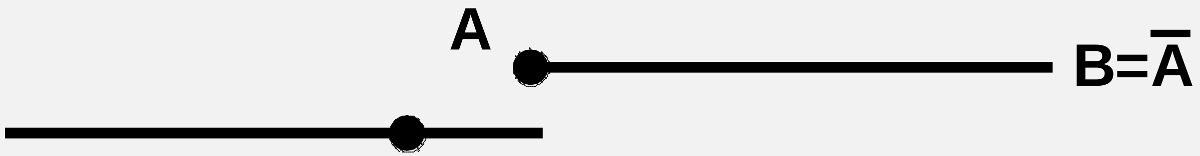
Inputs	Outputs
A	A’
0	1
1	0
Truth table for NOT	

NOT applied to an 8-bit input word A to produce B

0	1	1	0	0	1	0	1	¬ word A
1	0	0	1	1	0	1	0	¬word B

Constructing a NOT Gate

- NOT gate can be built using an inverted switch
  - i.e. a switch that is closed when its input is false and open when its input is true

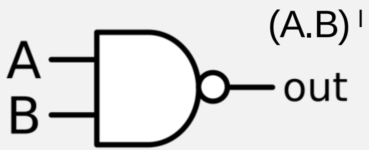


A representation of a NOT gate as an inverted switch.

The NAND and NOR Gates

- Not logically fundamental: derived from AND, OR, and NOT
  - **NAND** = NOT AND
  - **NOR** = NOT OR
- But they are **universal**
- I.e., we can **build any logic function** with NAND/NOR
- Thus, they are widely used gates in real circuits

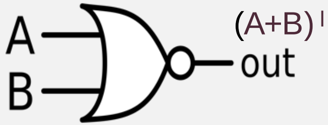
The NAND and NOR Gates: Symbols



A NAND gate

Inputs		Outputs
A	B	(A.B) ˡ
0	0	1
0	1	1
1	0	1
1	1	0

Truth table for 2 input NAND



A NOR gate

Inputs		Outputs
A	B	(A+B) ˡ
0	0	1
0	1	0
1	0	0
1	1	0

Truth table for 2 input NAND

Exercise @Home

- Q1) Work out how the NAND gate in the previous slide can be either written as $(xy)'$ or $x' + y'$.
- Q2) Do the same for the NOR gate.
- Q3)Assembly programmers use Boolean operators to speedup program performance. For example:
  - Can you use the XOR gate to clear the value of a storage location? (Hint: Think of A XOR A)
  - The XOR operator can be used to swap the values of two variables A and B (Hint: A = A XOR B, B = A XOR B, A = A XOR B)

School of Science
& Technology

www.city.ac.uk

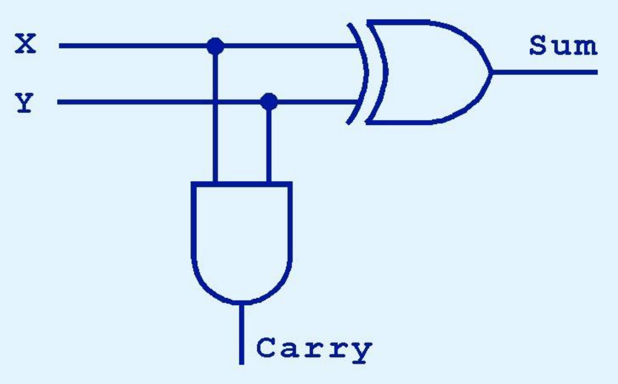
Systems Architecture

IN1006

Logic Gates and Circuits: Application and Design of Combinational Circuits

—Dr H. Asad

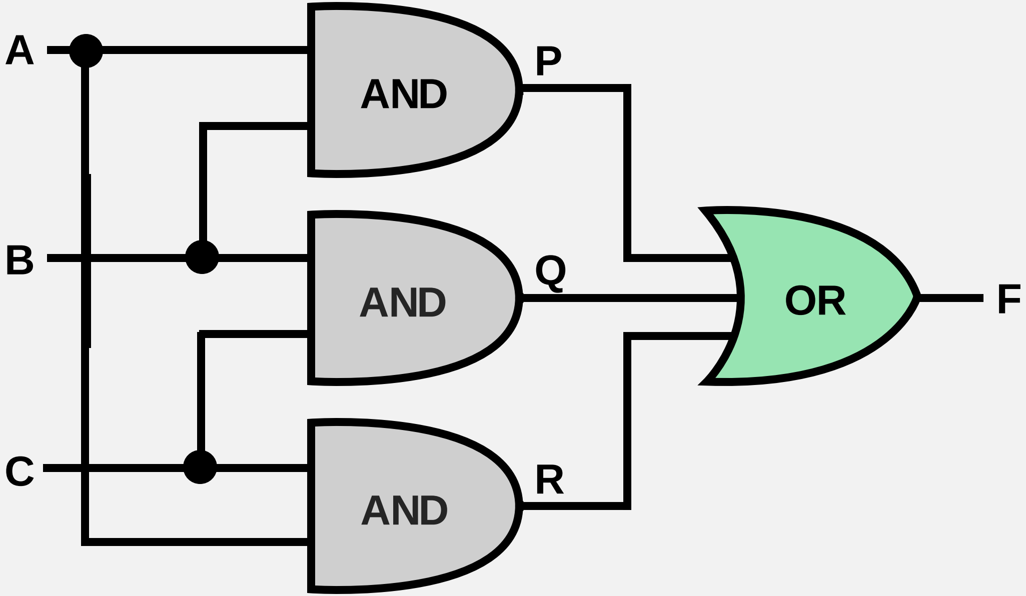
Combinational Circuits, Example 1



Inputs		Outputs	
X	Y	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

As we can see the sum can be found using the XOR operation and the carry using the AND operation.

Example 2: Circuit Diagram: From Circuit to Boolean Expression



What does it do?
What does F mean?

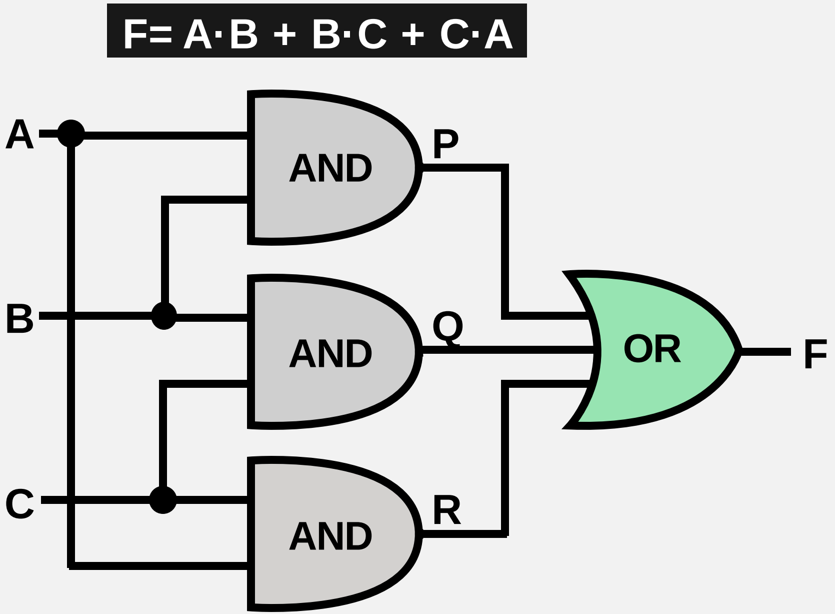
From the circuit diagram:

P= A·B
Q= B·C
R= C·A
F= P+Q+R

Substituting P, Q, and R into equation for F:

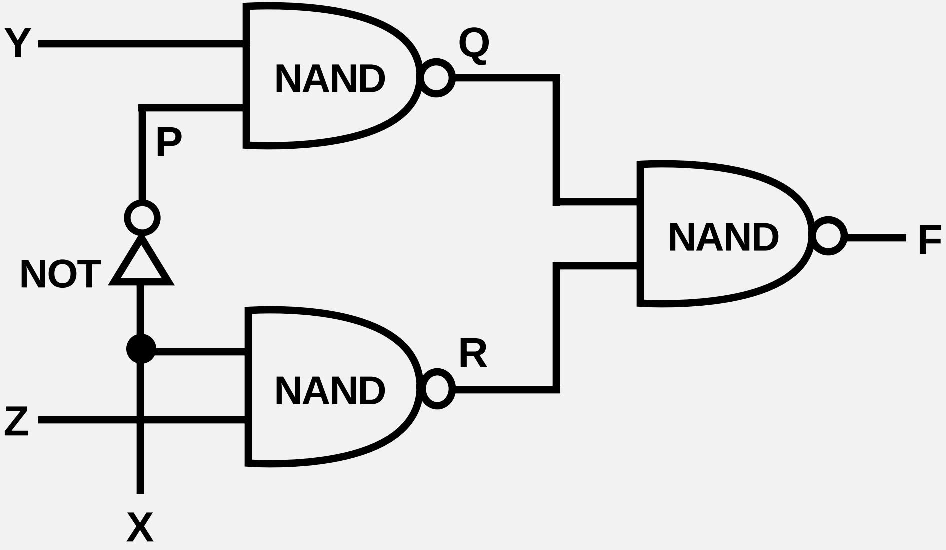
F= A·B + B·C + C·A

Example 2: Circuit Diagram ➔ Truth table



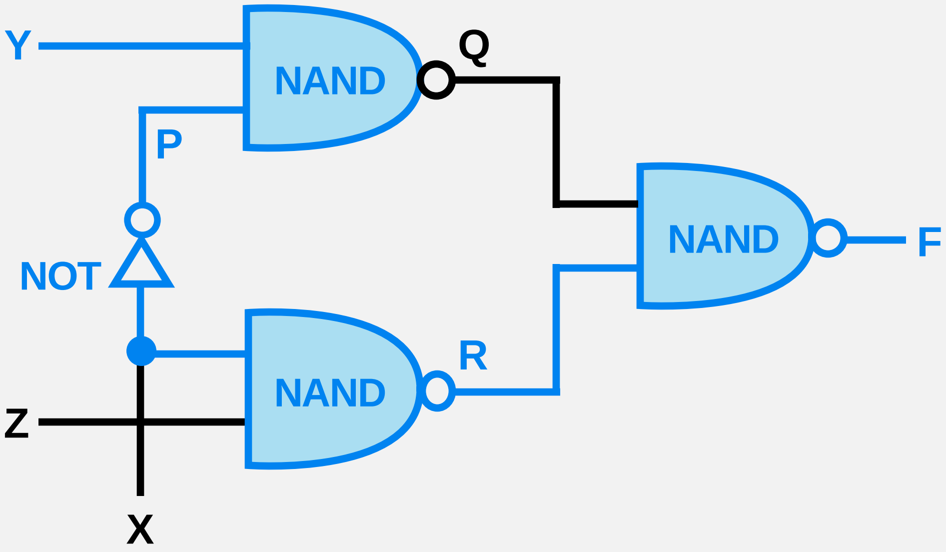
Inputs			Intermediate values			Output
A	B	C	P=A·B	Q=B·C	R=C·A	F=P+Q+R
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	0
0	1	1	0	1	0	1
1	0	0	0	0	0	0
1	0	1	0	0	1	1
1	1	0	1	0	0	1
1	1	1	1	1	1	1

Example 3: Circuit Diagram & Truth Table of Multiplexer



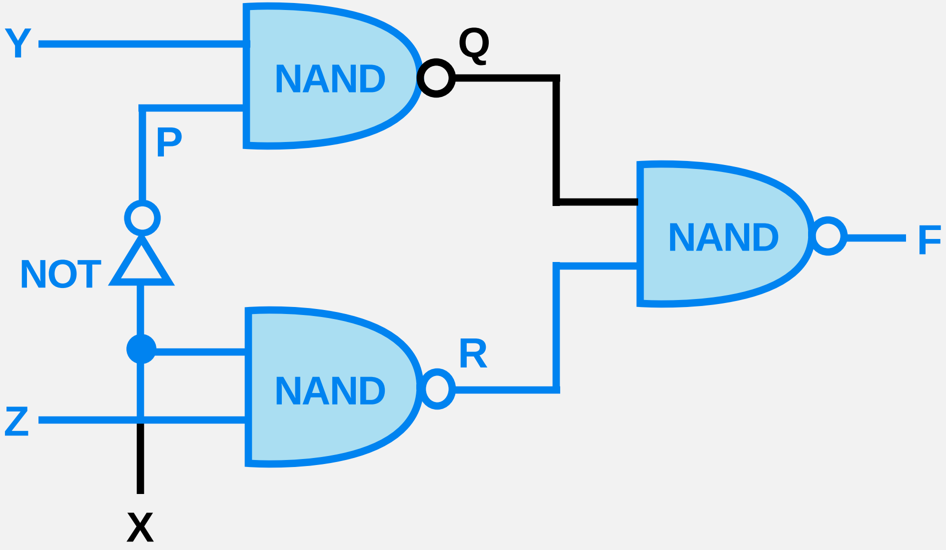
Inputs			Intermediate values			Output
X	Y	Z	P=X’	Q=(P·Y)’	R=(X·Z)’	F=(Q.R)’
0	0	0	1	1	1	0
0	0	1	1	1	1	0
0	1	0	1	0	1	1
0	1	1	1	0	1	1
1	0	0	0	1	1	0
1	0	1	0	1	0	1
1	1	0	0	1	1	0
1	1	1	0	1	0	1

Example 3: Circuit Diagram & Truth Table of Multiplexer



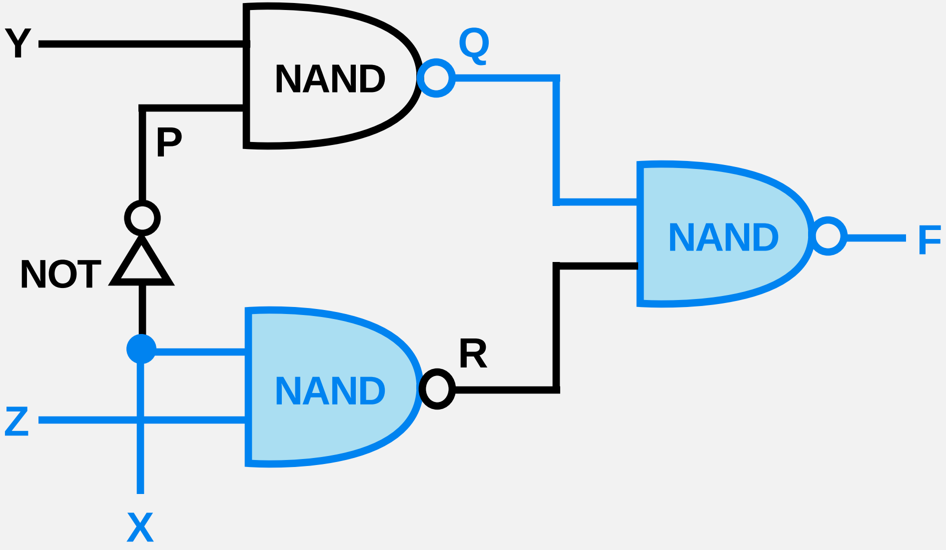
Inputs			Intermediate values			Output
X	Y	Z	P=X’	Q=(P·Y)’	R=(X·Z)’	F=(Q.R)’
0	0	0	1	1	1	0
0	0	1	1	1	1	0
0	1	0	1	0	1	1
0	1	1	1	0	1	1
1	0	0	0	1	1	0
1	0	1	0	1	0	1
1	1	0	0	1	1	0
1	1	1	0	1	0	1

Example 3: Circuit Diagram & Truth Table of Multiplexer



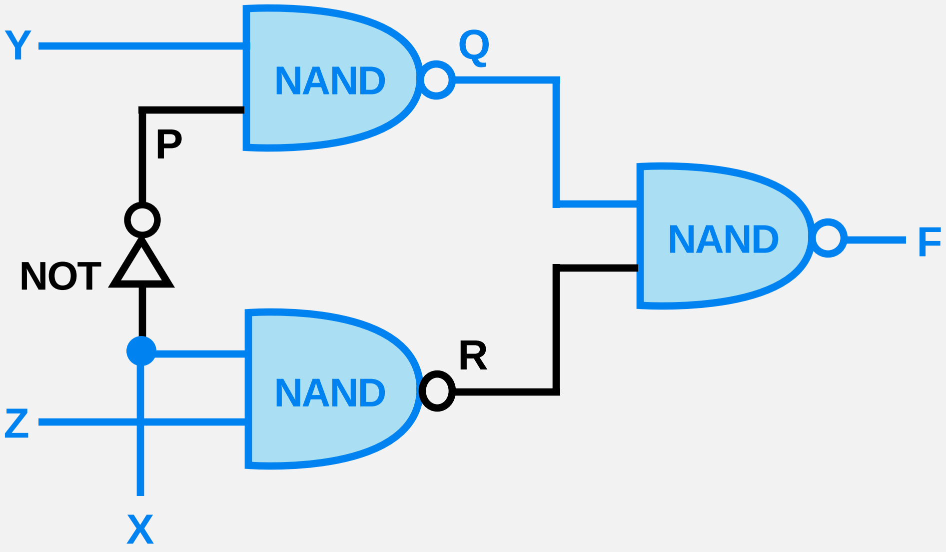
Inputs			Intermediate values			Output
X	Y	Z	P=X'	Q=(P·Y)'	R=(X·Z)'	F=(Q.R)'
0	0	0	1	1	1	0
0	0	1	1	1	1	0
0	1	0	1	0	1	1
0	1	1	1	0	1	1
1	0	0	0	1	1	0
1	0	1	0	1	0	1
1	1	0	0	1	1	0
1	1	1	0	1	0	1

Example 3: Circuit Diagram & Truth Table of Multiplexer



Inputs			Intermediate values			Output
X	Y	Z	P=X'	Q=(P·Y)'	R=(X·Z)'	F=(Q.R)'
0	0	0	1	1	1	0
0	0	1	1	1	1	0
0	1	0	1	0	1	1
0	1	1	1	0	1	1
1	0	0	0	1	1	0
1	0	1	0	1	0	1
1	1	0	0	1	1	0
1	1	1	0	1	0	1

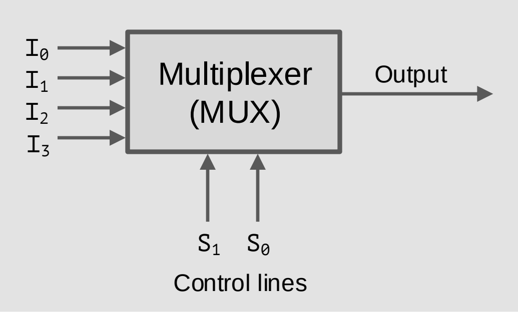
Example 3: Circuit Diagram & Truth Table of Multiplexer



Inputs			Intermediate values			Output
X	Y	Z	P=X’	Q=(P·Y)’	R=(X·Z)’	F=(Q.R)’
0	0	0	1	1	1	0
0	0	1	1	1	1	0
0	1	0	1	0	1	1
0	1	1	1	0	1	1
1	0	0	0	1	1	0
1	0	1	0	1	0	1
1	1	0	0	1	1	0
1	1	1	0	1	0	1

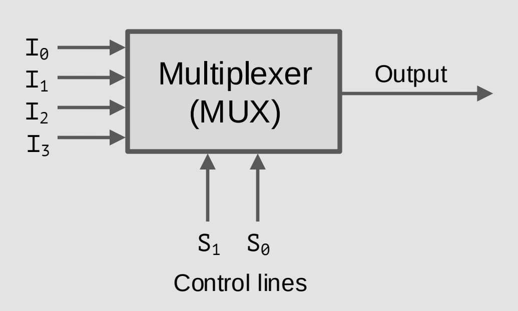
Example 3: Truth Table of Multiplexer

Inputs			Intermediate values			Output
X	Y	Z	P=X’	Q=(P·Y)’	R=(X·Z)’	F=(Q.R)’
0	0	0	1	1	1	0
0	0	1	1	1	1	0
0	1	0	1	0	1	1
0	1	1	1	0	1	1
1	0	0	0	1	1	0
1	0	1	0	1	0	1
1	1	0	0	1	1	0
1	1	1	0	1	0	1



Example 3: Truth Table of Multiplexer

Inputs			Intermediate values			Output
X	Y	Z	P=X’	Q=(P·Y)’	R=(X·Z)’	F=(Q.R)’
0	0	0	1	1	1	0
0	0	1	1	1	1	0
0	1	0	1	0	1	1
0	1	1	1	0	1	1
1	0	0	0	1	1	0
1	0	1	0	1	0	1
1	1	0	0	1	1	0
1	1	1	0	1	0	1



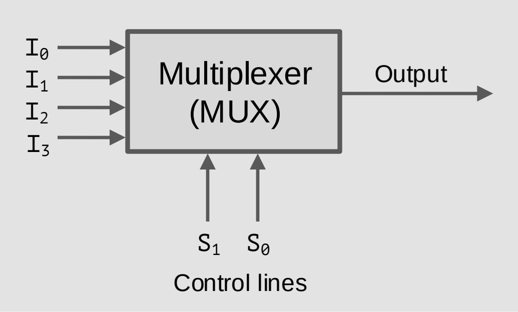
Let’s highlight the rows of the truth table to show:

All the 0s for X together.

All the 1s for X together.

Example 3: Truth Table of Multiplexer

Inputs			Intermediate values			Output
X	Y	Z	P=X’	Q=(P·Y)’	R=(X·Z)’	F=(Q.R)’
0	0	0	1	1	1	0
0	0	1	1	1	1	0
0	1	0	1	0	1	1
0	1	1	1	0	1	1
1	0	0	0	1	1	0
1	0	1	0	1	0	1
1	1	0	0	1	1	0
1	1	1	0	1	0	1

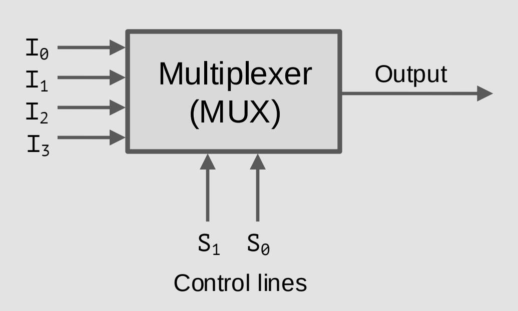


Inspecting the truth table:

F=Y when X is false (0)

Example 3: Truth Table of Multiplexer

Inputs			Intermediate values			Output
X	Y	Z	P=X’	Q=(P·Y)’	R=(X·Z)’	F=(Q.R)’
0	0	0	1	1	1	0
0	0	1	1	1	1	0
0	1	0	1	0	1	1
0	1	1	1	0	1	1
1	0	0	0	1	1	0
1	0	1	0	1	0	1
1	1	0	0	1	1	0
1	1	1	0	1	0	1



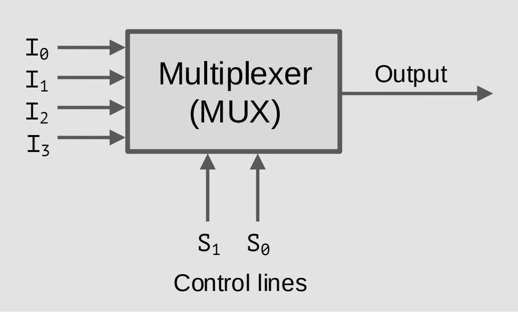
Inspecting the truth table:

F=Y when X is false (0)

F=Z when X is true (1)

Example 3: Truth Table of Multiplexer

Inputs			Intermediate values			Output
X	Y	Z	P=X’	Q=(P·Y)’	R=(X·Z)’	F=(Q.R)’
0	0	0	1	1	1	0
0	0	1	1	1	1	0
0	1	0	1	0	1	1
0	1	1	1	0	1	1
1	0	0	0	1	1	0
1	0	1	0	1	0	1
1	1	0	0	1	1	0
1	1	1	0	1	0	1



Inspecting the truth table:

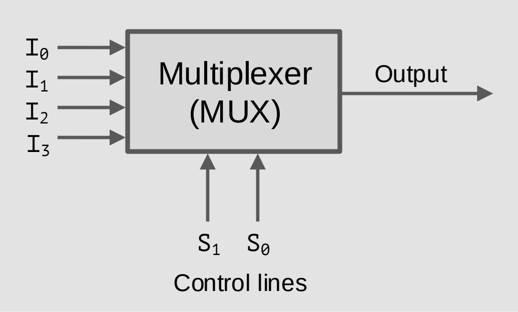
F=Y when X is false (0)

F=Z when X is true (1)

The circuit acts like a switch connecting the output, F, to one of two inputs, Y or Z, depending on the control input X.

Example 3: Truth Table of Multiplexer

Inputs			Intermediate values			Output
X	Y	Z	P=X’	Q=(P·Y)’	R=(X·Z)’	F=(Q.R)’
0	0	0	1	1	1	0
0	0	1	1	1	1	0
0	1	0	1	0	1	1
0	1	1	1	0	1	1
1	0	0	0	1	1	0
1	0	1	0	1	0	1
1	1	0	0	1	1	0
1	1	1	0	1	0	1



Inspecting the truth table:

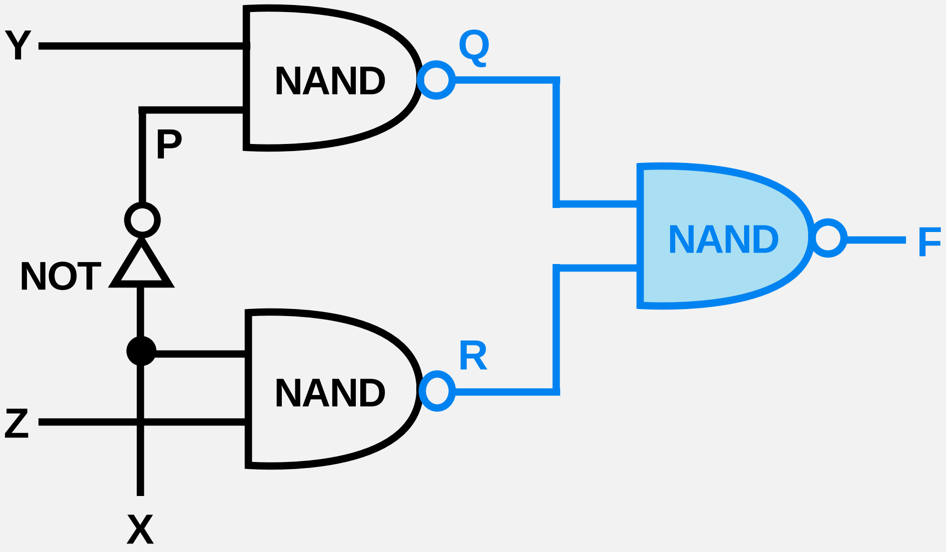
F=Y when X is false (0)

F=Z when X is true (1)

The circuit acts like a switch connecting the output, F, to one of two inputs, Y or Z, depending on the control input X.

The circuit is a two-input multiplexer or MUX for short. It selects a single output from several inputs. The particular input chosen for output is determined by the value of the multiplexer’s control lines.

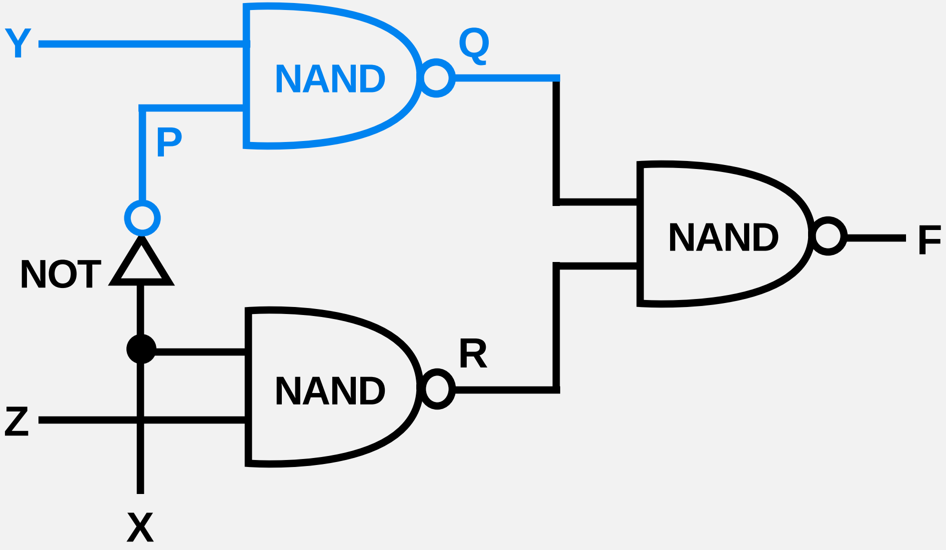
Example 3: Boolean Equation (1)



From the circuit diagram:

- **F=(Q·R)'**
- Q=(Y·P)'
- P=X'
- R=(X·Z)'

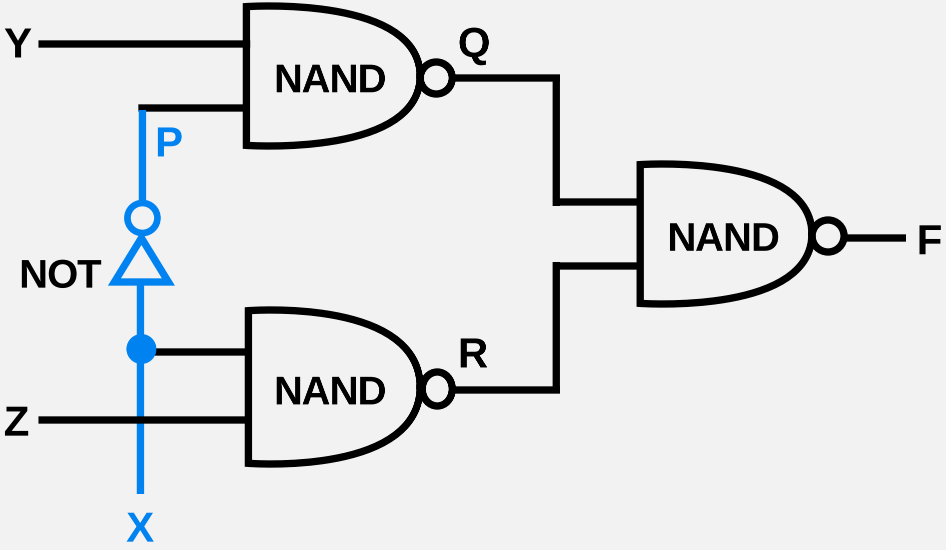
Example 3: Boolean Equation (1)



From the circuit diagram:

- $F=(Q \cdot R)'$
- **$Q=(Y \cdot P)'$**
- $P=X'$
- $R=(X \cdot Z)'$

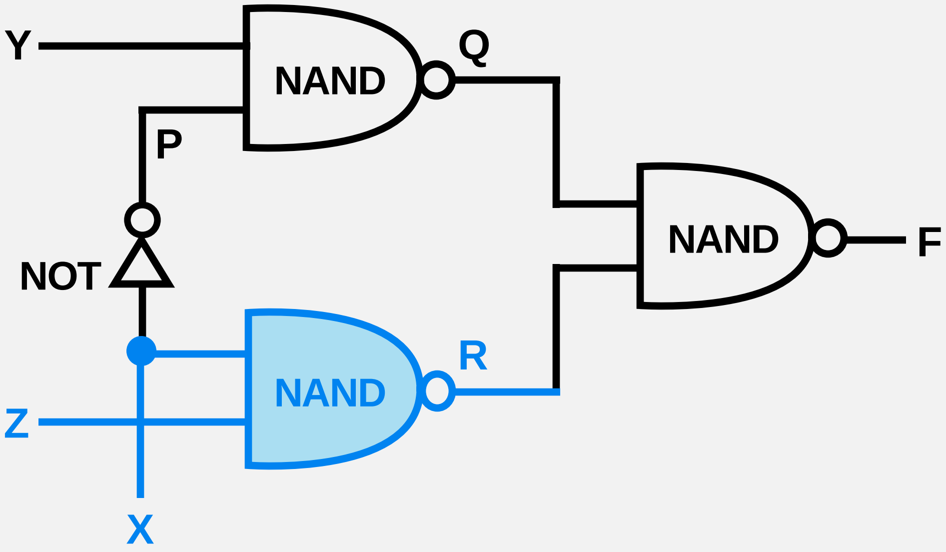
Example 3: Boolean Equation (1)



From the circuit diagram:

- F=(Q·R)'
- Q=(Y·P)'
- **P=X'**
- R=(X·Z)'

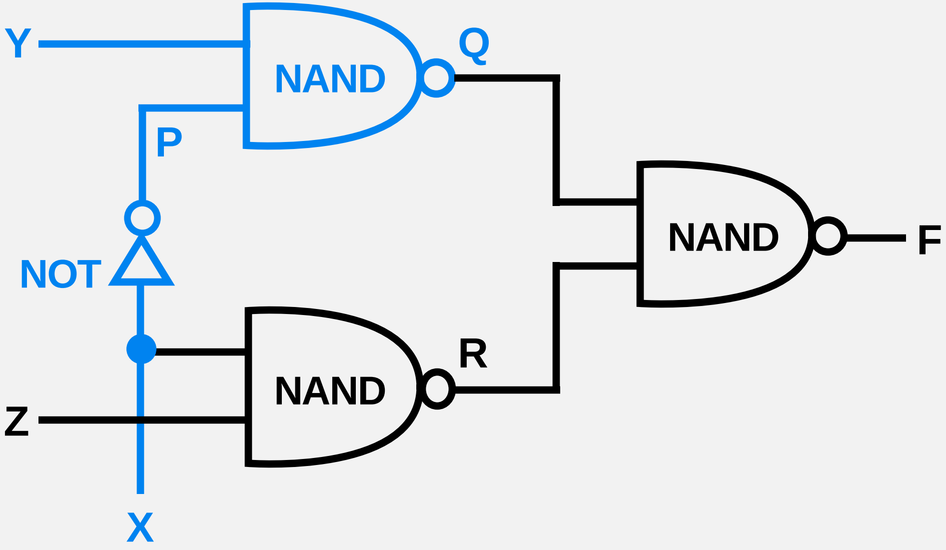
Example 3: Boolean Equation (1)



From the circuit diagram:

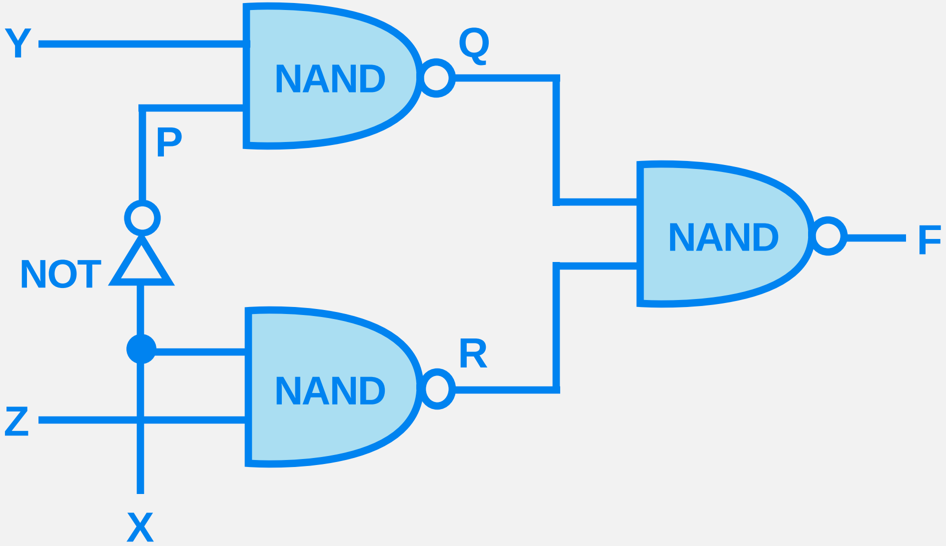
- F=(Q·R)'
- Q=(Y·P)'
- P=X'
- **R=(X·Z)'**

Example 3: Boolean Equation (1)



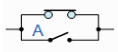
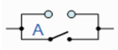
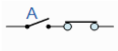
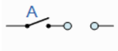
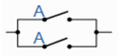
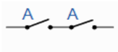
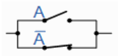
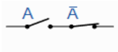
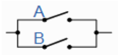
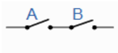
Substituting P into Q:
$Q=(Y \cdot X')'$

Example 3: Boolean Equation (1)



Substituting Q and R into F:

F=((Y·X')' · (X·Z)')'

Boolean Expression	Description	Equivalent Switching Circuit	Boolean Algebra Law or Rule
A + 1 = 1	A in parallel with closed = "CLOSED"		Annulment
A + 0 = A	A in parallel with open = "A"		Identity
A . 1 = A	A in series with closed = "A"		Identity
A . 0 = 0	A in series with open = "OPEN"		Annulment
A + A = A	A in parallel with A = "A"		Idempotent
A . A = A	A in series with A = "A"		Idempotent
NOT $\overline{A}$ = A	NOT NOT A (double negative) = "A"		Double Negation
A + $\overline{A}$ = 1	A in parallel with NOT A = "CLOSED"		Complement
A . $\overline{A}$ = 0	A in series with NOT A = "OPEN"		Complement
A+B = B+A	A in parallel with B = B in parallel with A		Commutative
A.B = B.A	A in series with B = B in series with A		Commutative
$\overline{A+B} = \overline{A}.\overline{B}$	invert and replace OR with AND		de Morgan's Theorem
$\overline{A.B} = \overline{A}+\overline{B}$	invert and replace AND with OR		de Morgan's Theorem

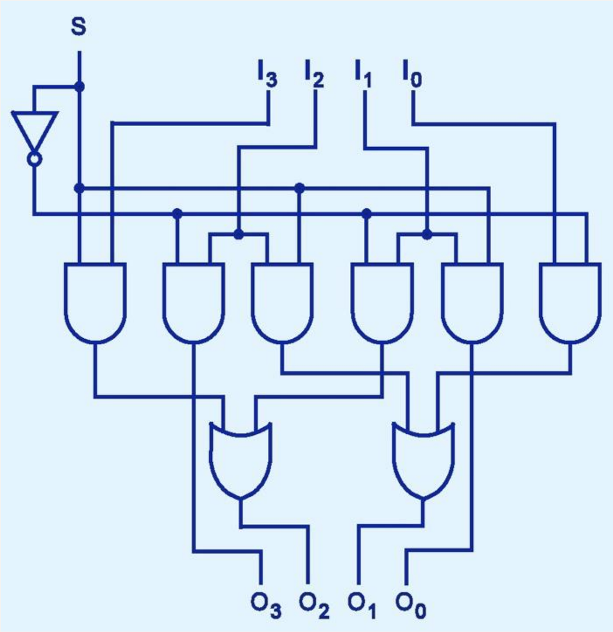
Example 3: Boolean Equation (2)

Q =	(A + B).(A + C)	
	A.A + A.C + A.B + B.C	– Distributive law
	A + A.C + A.B + B.C	– Idempotent AND law (A.A = A)
	A(1 + C) + A.B + B.C	– Distributive law
	A.1 + A.B + B.C	– Identity OR law (1 + C = 1)
	A(1 + B) + B.C	– Distributive law
	A.1 + B.C	– Identity OR law (1 + B = 1)
Q =	A + (B.C)	– Identity AND law (A.1 = A)

Shifter Circuit (@home)

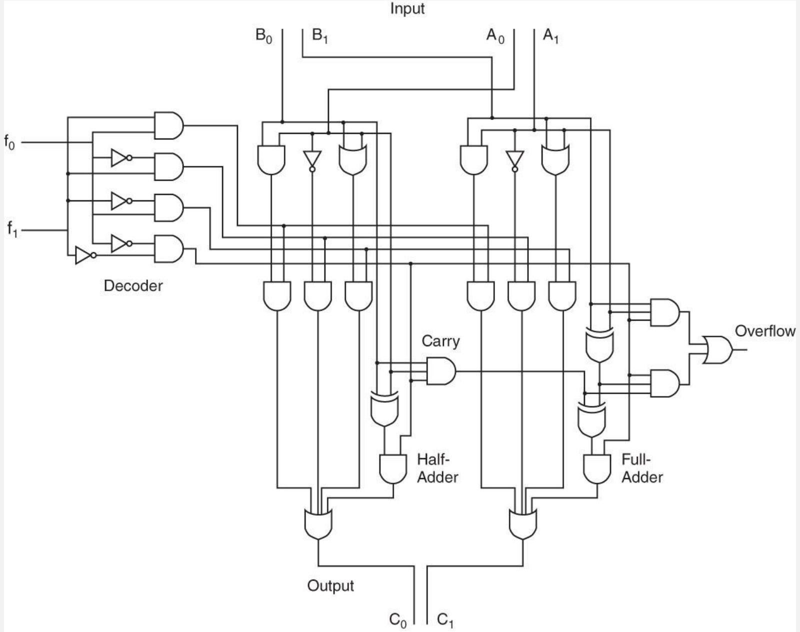
- This shifter moves the bits of a nibble (4-bits) one position to the left or right.

- Why is this a useful operator to have in programming?



Combinational Circuits: A simple ALU

- Combinational logic circuits produce a specified output (almost) at the instant when input values are applied.
- Example: Arithmetic Logic Unit (ALU)



School of Science
& Technology

www.city.ac.uk

Systems Architecture

IN1006

Logic Gates and Circuits: Sequential Circuits

—Dr H. Asad

Combinational circuit vs Sequential (Logic) Circuits

The logic circuits can be characterised as:

- Combinational circuits are **memoryless elements.**
- **Sequential logic elements:** these are circuits which remember their previous inputs
- the output of a sequential circuit depends not only on its current inputs but also its previous inputs/outputs
- Sequential circuits are ***memory* elements**

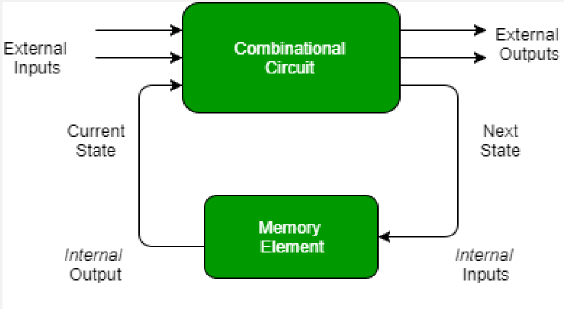


Figure: Sequential Circuit

https://www.geeksforgeeks.org/difference-between-combinational-and-sequential-circuit/

Sequential Logic Circuit

A sequential circuit consists of:

- a **combinational circuit** that transforms a set of inputs into an output
- a **sequential logic element** that acts **as a memory**, storing the data to feedback into the combinational circuit.

The data held in **memory** is called the **internal state** of the circuit.

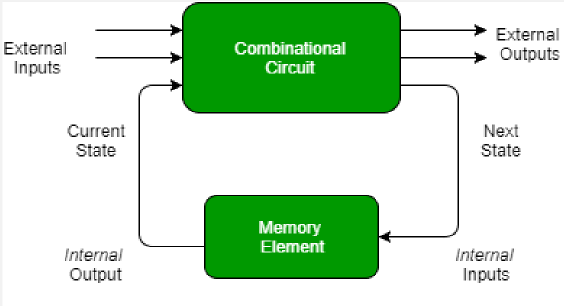
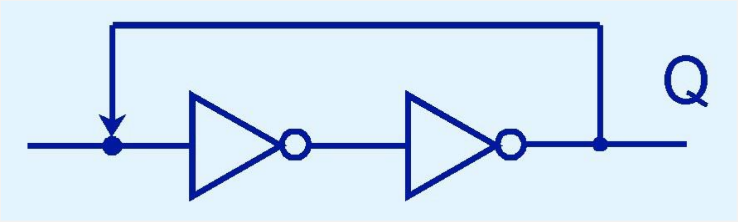


Figure: Sequential Circuit

https://www.geeksforgeeks.org/difference-between-combinational-and-sequential-circuit/

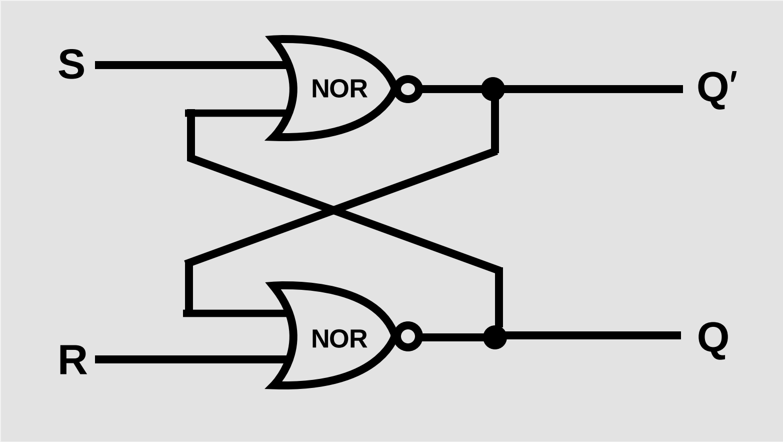
Sequential Circuits

- The **previous output** of a sequential circuit together **with its current input** is used to generate the **next output** and the **next state** in the circuit.
- To **retain their state** values, sequential circuits **rely on** ***feedback***.
- **Feedback** in digital circuits occurs when an output is **looped** back to the input.
- A simple example of this concept is shown below.
  - If Q is 0 it will always be 0, if it is 1, it will always be 1. Why?



Two Cross-Coupled NOR Gates -- SR flip-flop (latch)

- You can see how feedback works by examining the most basic sequential logic components:
  the **SR flip-flop or SR latch**
- The "SR" stands for Set/Reset.
- The internals of an SR latch are shown, along with its block diagram.

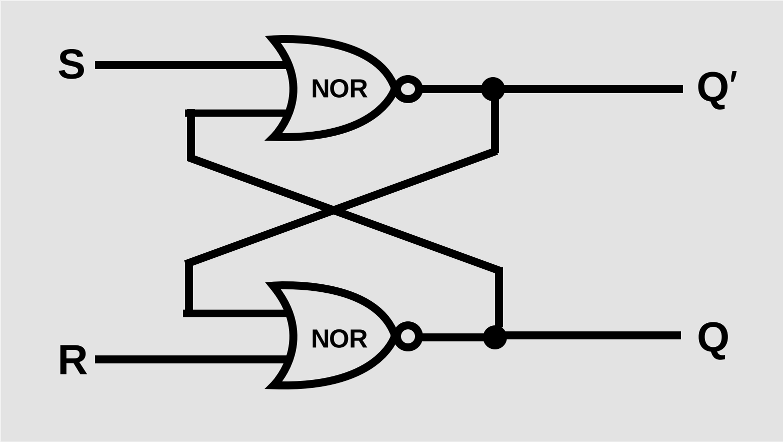


Analysing a Sequential Circuit

- To analyse a sequential circuit, like the cross-coupled NOR gates, **we first assume some initial conditions**:

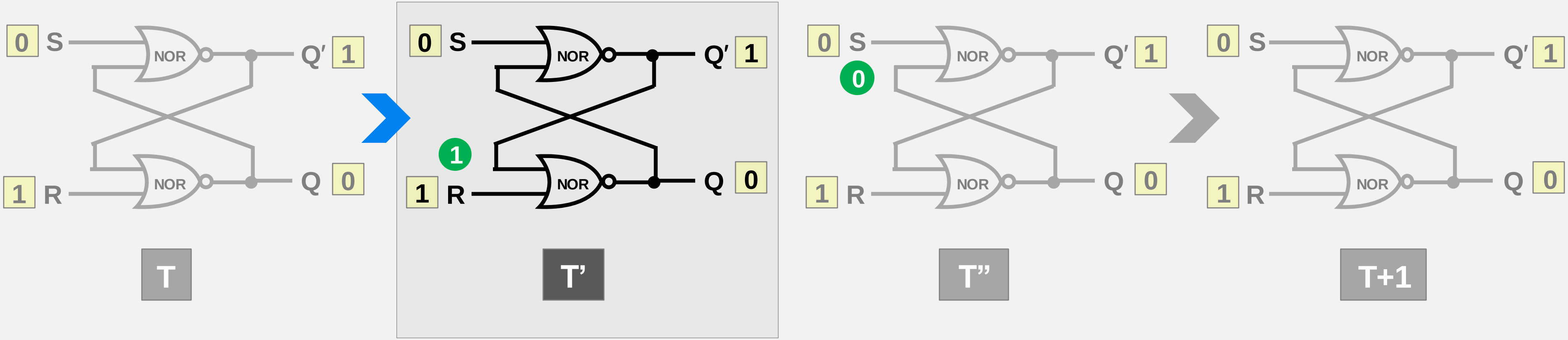
  e.g. Q=1 and S=R=0.

- Following through the circuit, we see that this is a **stable state** with Q continuing to be 1 after an iteration.



Analysing a Sequential Circuit

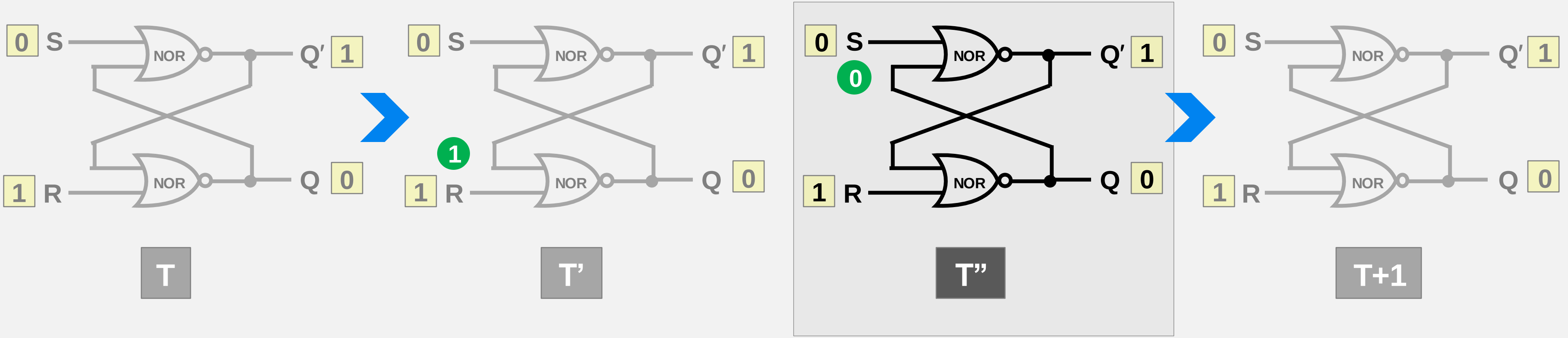
- If we assumed an initial condition with **Q=0 and S=R=0,** we find that this is also a **stable state** with Q continuing to be 0 after each iteration.
- What happens if we change the inputs S and R?
- If **Q=0 and S=0 while R=1**, what happens?



Q' = NOR(0,0) = 1 and Q = NOR(1,1) = 0, so changing R to 1 **resets** Q from 0 to 0.

Analysing a Sequential Circuit

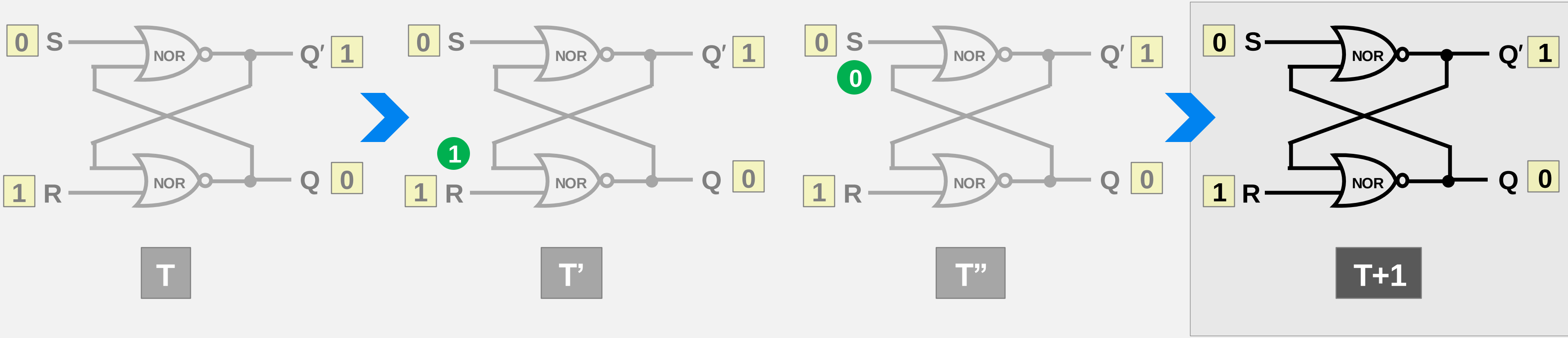
- If we assumed an initial condition with **Q=0 and S=R=0,** we find that this is also a **stable state** with Q continuing to be 0 after each iteration.
- What happens if we change the inputs S and R?
- If **Q=0 and S=0 while R=1**, what happens?



Q’ = NOR(0,0) = 1 and Q = NOR(1,1) = 0, so changing R to 1 **resets** Q from 0 to 0.

Analysing a Sequential Circuit

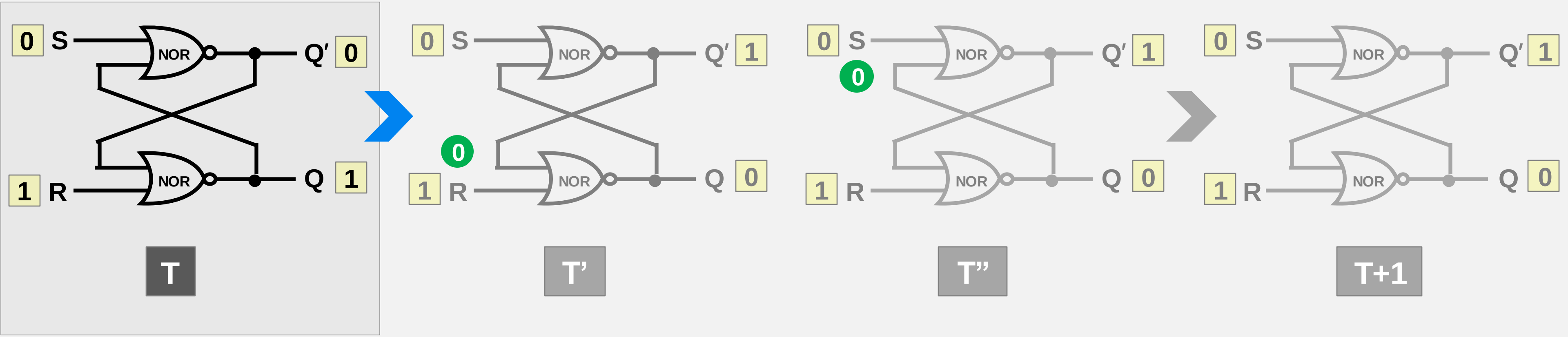
- If we assumed an initial condition with **Q=0 and S=R=0,** we find that this is also a **stable state** with Q continuing to be 0 after each iteration.
- What happens if we change the inputs S and R?
- If **Q=0 and S=0 while R=1**, what happens?



Q’ = NOR(0,0) = 1 and Q = NOR(1,1) = 0, so changing R to 1 **resets** Q from 0 to 0.

Analysing a Sequential Circuit (cont'd)

If **Q=1 and S=0 while R=1**, what happens?

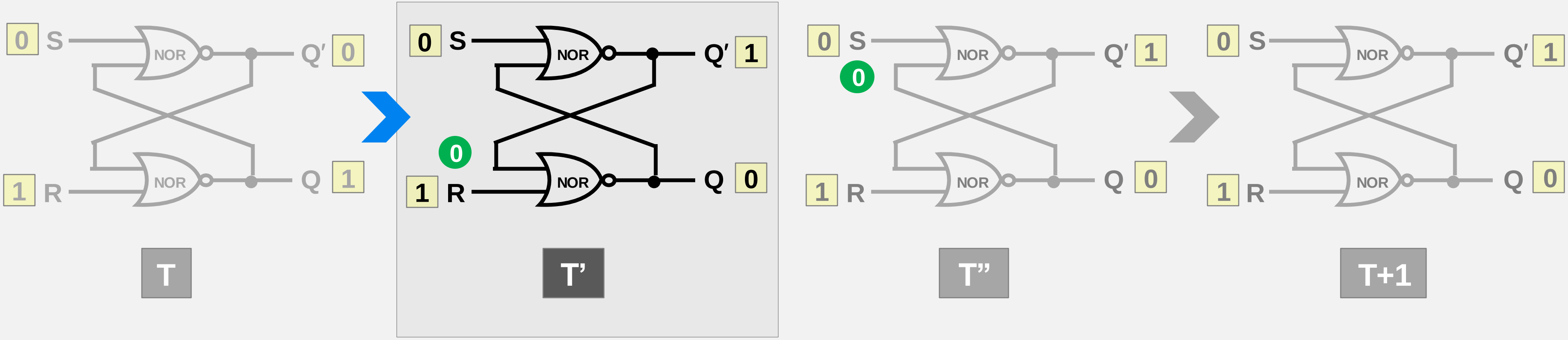


Q' = NOR(0,1) = 0 and Q = NOR(1,0) = 0, so changing **R to 1 resets Q from 1 to 0.**

So, when R is set to 1, then **Q is reset to 0**.

Analysing a Sequential Circuit (cont’d)

If **Q=1 and S=0 while R=1**, what happens?

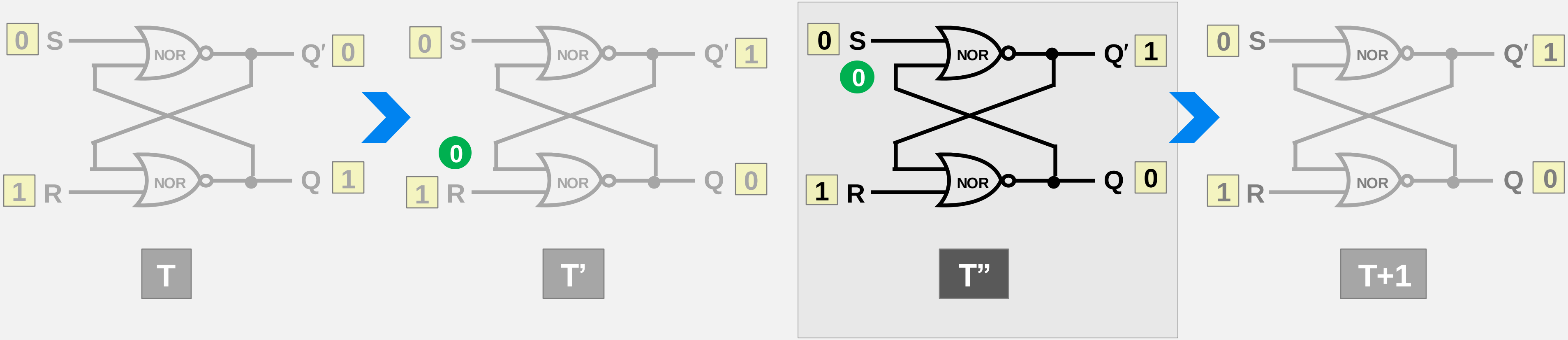


Q’ = NOR(0,1) = 0 and Q = NOR(1,0) = 0, so changing **R to 1 resets Q from 1 to 0.**

So, when R is set to 1, then **Q is reset to 0**.

Analysing a Sequential Circuit (cont'd)

If **Q=1 and S=0 while R=1**, what happens?

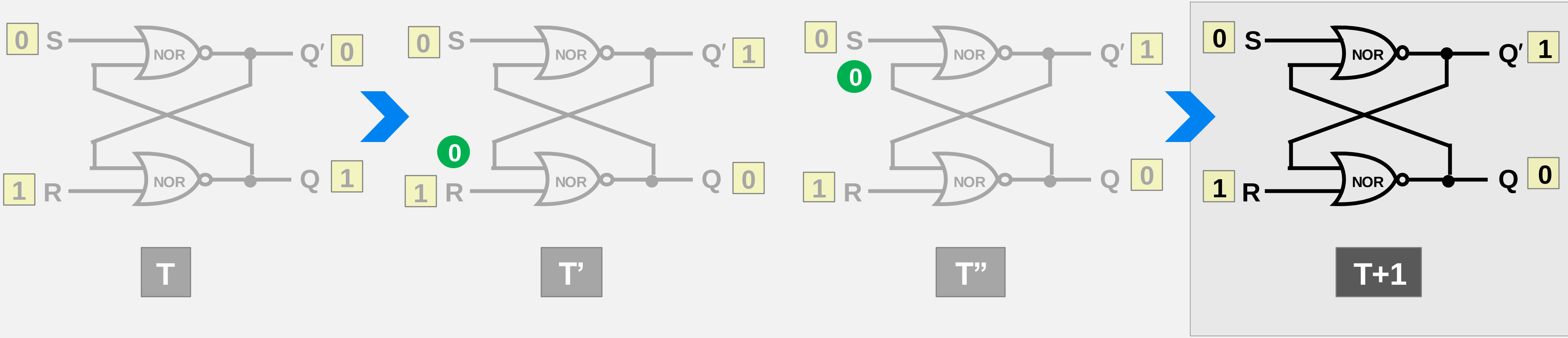


Q' = NOR(0,1) = 0 and Q = NOR(1,0) = 0, so changing **R to 1 resets Q from 1 to 0.**

So, when R is set to 1, then **Q is reset to 0**.

Analysing a Sequential Circuit (cont'd)

If **Q=1 and S=0 while R=1**, what happens?

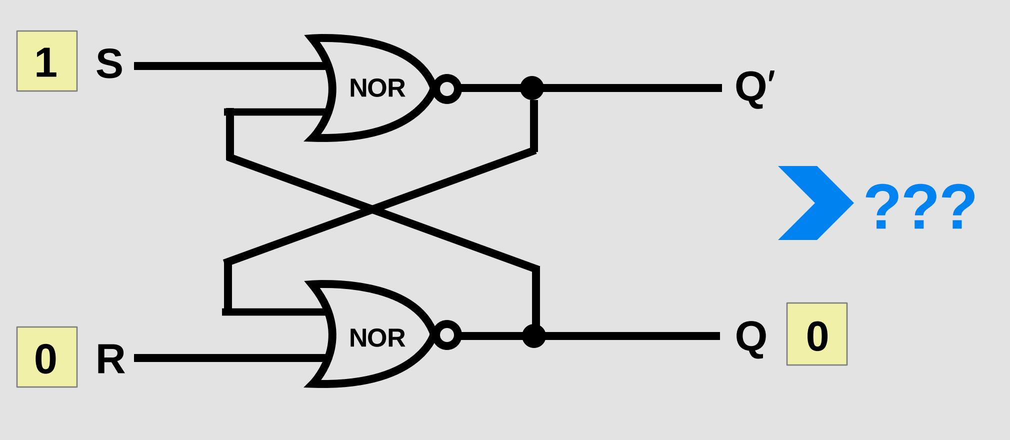


Q' = NOR(0,1) = 0 and Q = NOR(1,0) = 0, so changing **R to 1 resets Q from 1 to 0.**

So, when R is set to 1, then **Q is reset to 0**.

Analysing a Sequential Circuit (cont'd)

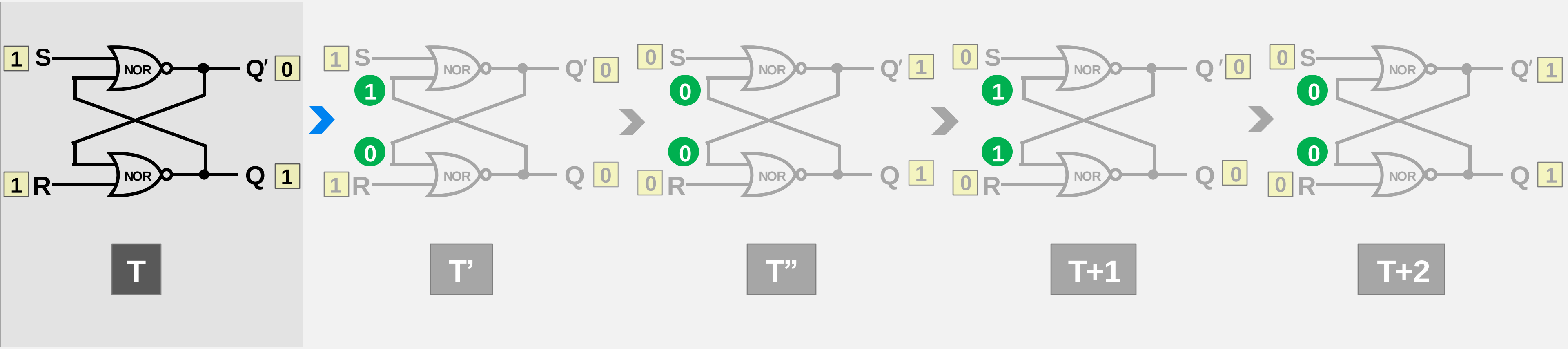
Question: What happens when S=1, R=0 and Q= 0?



@Home: Can you work out what happens when S=1, R=0 and Q= 1?
(These is the **set** part of the SR latch).

Analysing a Sequential Circuit (cont'd)

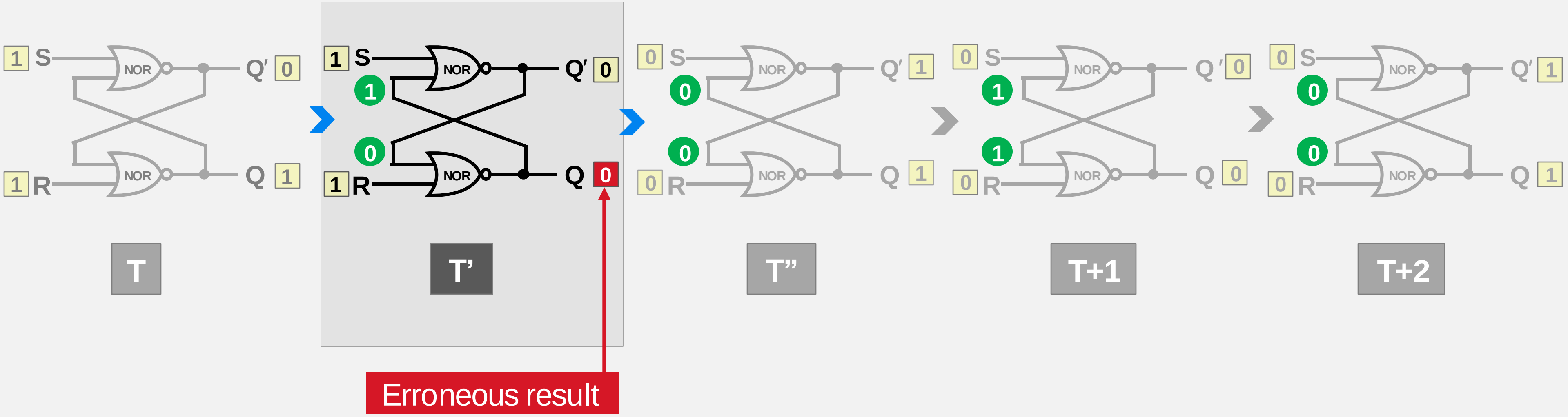
Question: What happens when S=1, R=1 and Q=1/Q=0?



OR RACE CONDITIONS (as gates may not produce their outputs in exactly the same time).

Analysing a Sequential Circuit (cont'd)

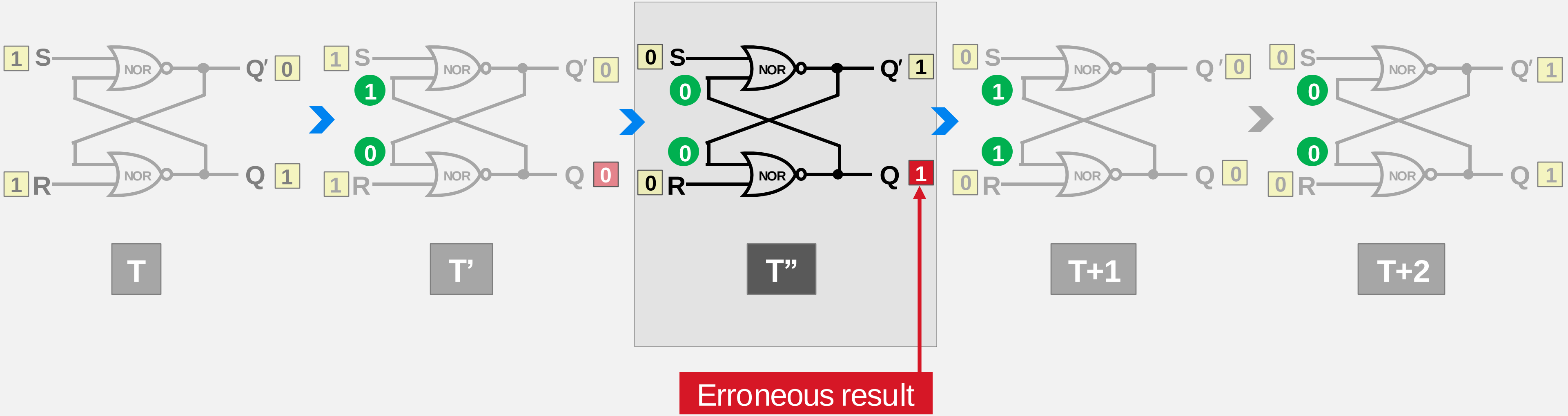
Question: What happens when S=1, R=1 and Q=1/Q=0?



OR RACE CONDITIONS (as gates may not produce their outputs in exactly the same time).

Analysing a Sequential Circuit (cont’d)

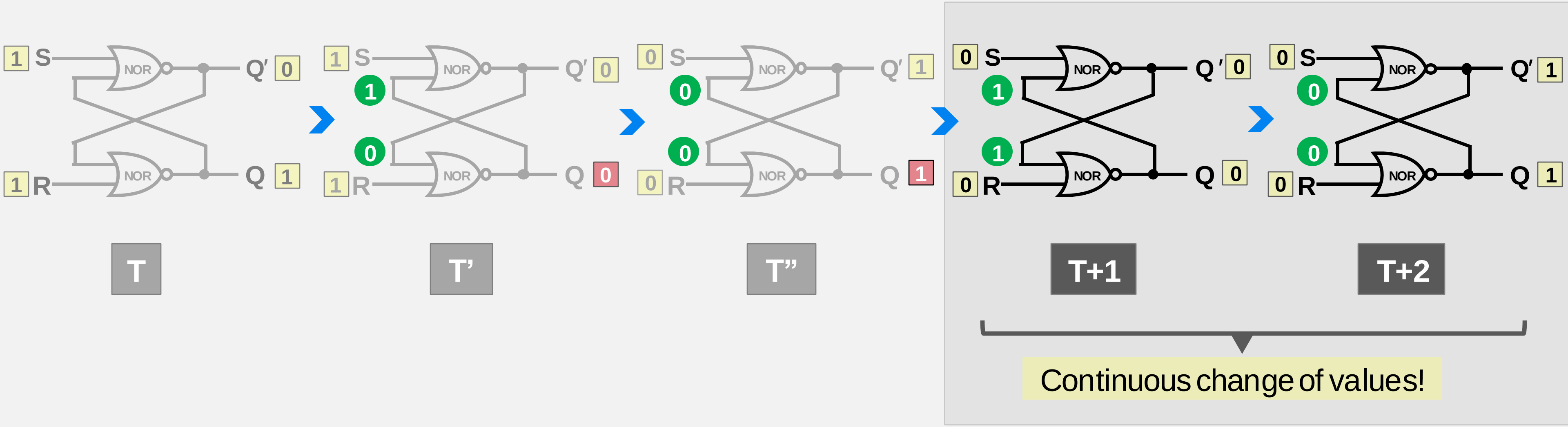
Question: What happens when S=1, R=1 and Q=1/Q=0?



OR RACE CONDITIONS (as gates may not produce their outputs in exactly the same time).

Analysing a Sequential Circuit (cont’d)

Question: What happens when S=1, R=1 and Q=1/Q=0?



OR RACE CONDITIONS (as gates may not produce their outputs in exactly the same time).

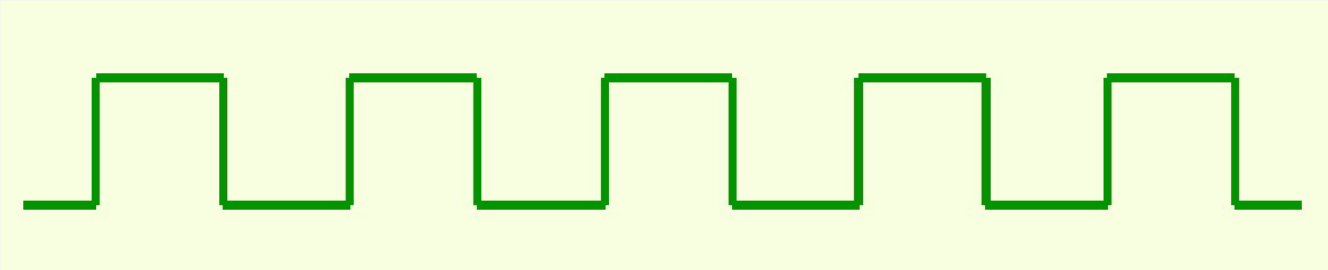
Analysing a Sequential Circuit (cont'd)

- The SR flip-flop actually has three inputs: S, R, and its current output, Q.
- Q(t) means the value of the output at time t. Q(t+1) is the value of Q after the next clock pulse.
- Thus, we can construct a truth table for this circuit, as shown at the right
- **Notice the two undefined values. When both S and R are 1, the SR flip-flop is unstable.**

S	R	Present State Q(t)	Next State Q(t+1)
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	undefined
1	1	1	undefined

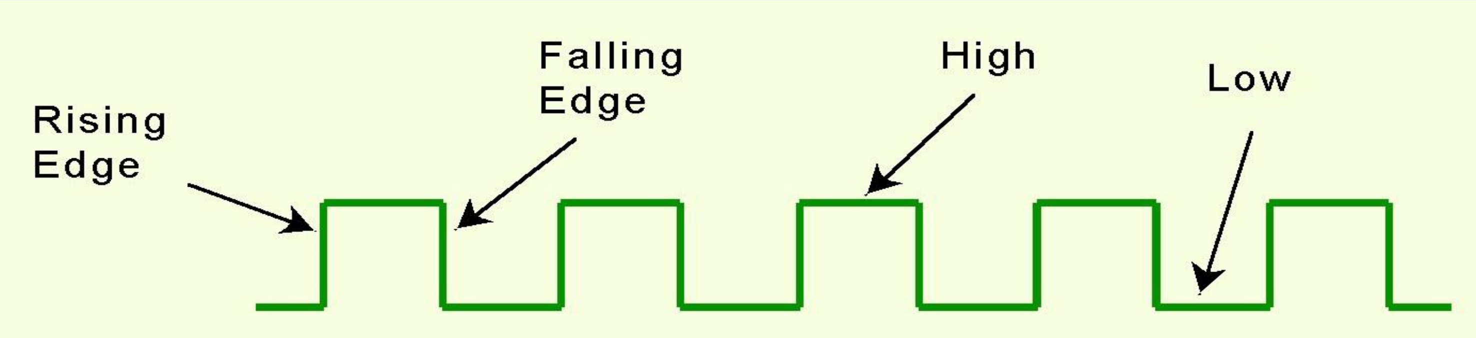
Clocks

- As the name implies, **sequential logic circuits** require a means by which events can be sequenced.
- State changes are controlled by clocks.
  - A “clock” is a special circuit that sends electrical pulses through a circuit.
- Clocks produce electrical waveforms such as the one shown below.



Clocks and Sequential Circuits

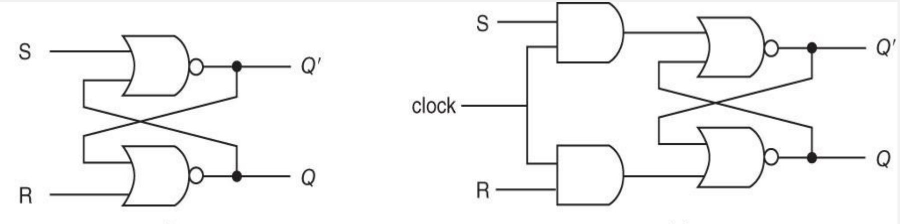
- State changes occur in sequential circuits only when the clock ticks.
- Circuits can change state on the rising edge, falling edge, or when the clock pulse reaches its highest voltage.



The Clocked SR Flip-Flop

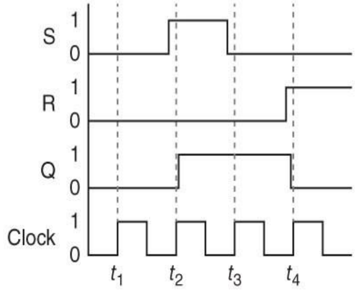
- **Clocked SR flip-flops** only change state when clocked

- An SR flip-flop extended so its inputs R' and S' are derived from the external inputs R and S by AND - ing them with a clock input C



S	R	Q(t+1)
0	0	Q(t) (no change)
0	1	0 (reset to 0)
1	0	1 (set to 1)
1	1	undefined

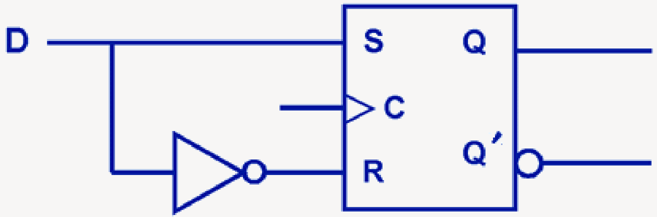
c)

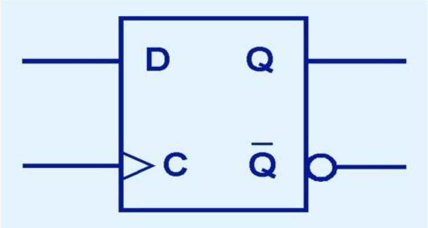


d)

The D-type Flip-Flop

- The D flip-flop has two inputs, D and C.
  - D = data input, C = clock input.
- When a D flip-flop is clocked, the value of the D input is set to its Q output and the output remains constant until the next time is clocked
- The D flip-flop is actually a modified SR flip-flop which **removes the non determinism of SR flip flop**





The D-type Flip-Flop (cont’d)

When C=0
The circuit is not active so at the next state it continues to store what it had
(➔ Q(t+1) = Q(t))

When C=1

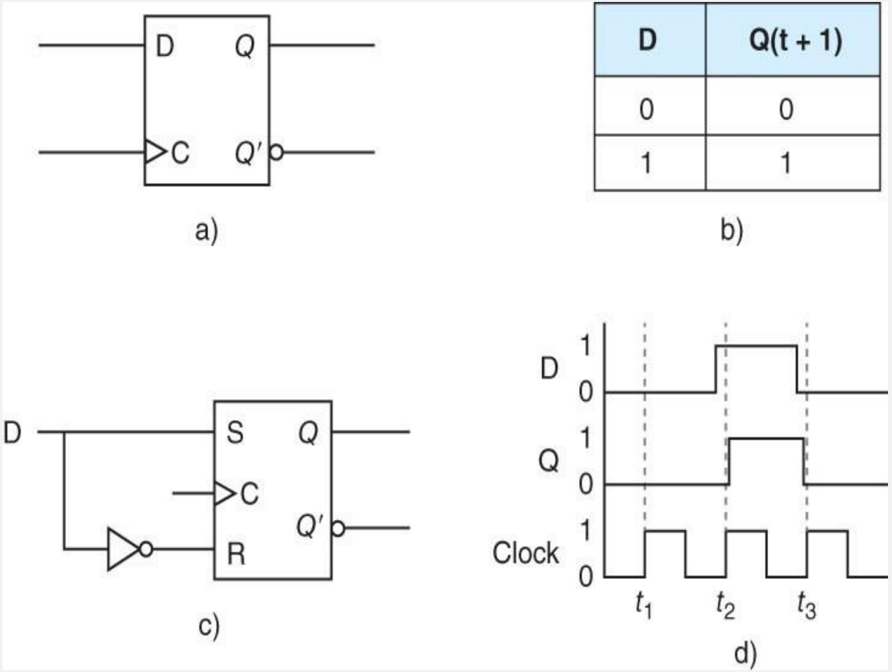
- When D=0 then

  R = 1 (S=0) and Q(t+1) = 0

- When D =1 then

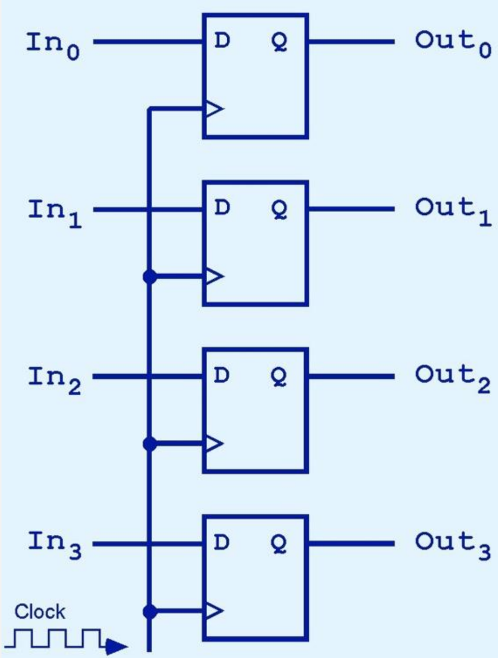
  S = 1 (R=0) and Q(t+1) = 1

- The D flip-flop stores one bit of information ➔ memory



D Flip-Flops Create Registers

- Groups of flip-flops called **registers** are used as on-chip storage
- This illustration shows a 4-bit register consisting of D flip-flops. You will usually see its block diagram (below) instead.



Summary

- Computer systems are built from a few simple types of circuit elements,
  - Combinational circuits (Logic)
  - Sequential elements (Memory)
  - The circuits in real-world computers are much more complex than the simple circuits we've looked at but they fall into these categories

School of Science & Technology
City, University of London
Northampton Square
London
EC1V 0HB
United Kingdom

T: +44 (0)20 7040 5060
E: SST-ug@city.ac.uk
www.city.ac.uk/department