Concepts of Computer Science

Gates and Circuits

Chapter Goals

- Lecture 1:
  - Identify basic gates
  - Observe gate behaviour via truth table, logic diagram, and Boolean expression
  - Build circuits from gate combinations
- Lecture 2 and 3:
  - Discuss circuit equivalence and Boolean algebra
  - Discuss several common circuits in computing
  - Build adders, multiplexers, S-R latches

- Lecture 1:
  - Identify basic gates
  - Observe gate behaviour via truth table, logic diagram, and Boolean expression
  - Build circuits from gate combinations

Gates and Circuits

- **Gates**

  A device that performs a basic operation on electrical signals.

- **Circuits**

  Gates combined to perform more complicated tasks.

Describing gates

Boolean expressions: Uses Boolean algebra, mathematical notation for expressing two-valued logic. Same algebra, but different symbols as CS-170.

Logic diagrams: A graphical representation of a circuit; each gate has its own symbol

Truth tables: A table showing all possible input values and the associated output values

Logic Gates

- Six types of gates
  - NOT
  - AND
  - OR
  - XOR
  - NAND
  - NOR

In CS-170 we don’t consider XOR, NAND, and NOR as basic operations.

Likewise, from an electronics perspective, implication and equivalence are not basic gates

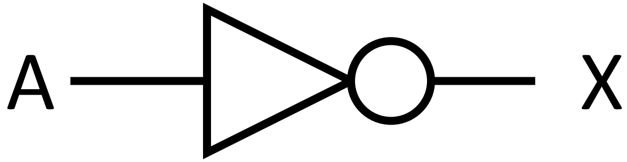
NOT

A NOT gate accepts one input signal (0 or 1) and returns the complementary (opposite) signal as output

Boolean Expression

$X = A'$

Logic Diagram



Truth Table

A	X
0	1
1	0

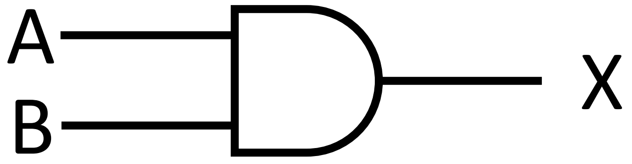
AND

An AND gate accepts two input signals. If both are 1, the output is 1; otherwise the output is 0.

Boolean Expression

$$X = A \cdot B$$

Logic Diagram



Truth Table

A	B	X
0	0	0
1	0	0
0	1	0
1	1	1

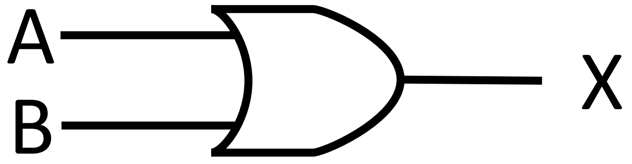
OR

An AND gate accepts two input signals. If both are 0, the output is 0; otherwise the output is 1.

Boolean Expression

$$X = A + B$$

Logic Diagram



Truth Table

A	B	X
0	0	0
1	0	1
0	1	1
1	1	1

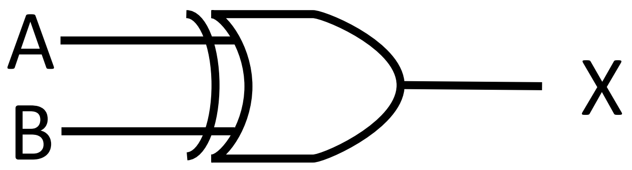
XOR

An XOR gate accepts two input signals. If both are the same, the output is 0; otherwise, the output is 1

Boolean Expression

$X = A \oplus B$

Logic Diagram



Truth Table

A	B	X
0	0	0
1	0	1
0	1	1
1	1	0

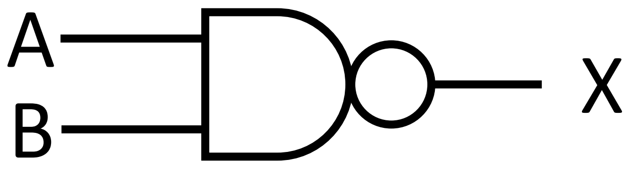
NAND

A NAND ("NOT of AND") gate accepts two input signals. If both are 1, the output is 0; otherwise, the output is 1.

Boolean Expression

$X = (A \cdot B)'$

Logic Diagram



Truth Table

A	B	X
0	0	1
1	0	1
0	1	1
1	1	0

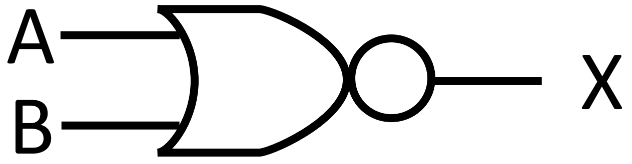
NOR

The NOR ("NOT of OR") gate accepts two inputs. If both are 0, the output is 1; otherwise, the output is 0.

Boolean Expression

$X = (A + B)'$

Logic Diagram



Truth Table

A	B	X
0	0	1
1	0	0
0	1	0
1	1	0

A note on notation

Here we have seen the use of **+**, **•**, and **′**

You may use **∨**, **∧** , and **¬** from propositional logic.

You may prefer the words **OR**, **AND**, and **NOT**, or even **disjunction**, **conjunction**, and **negation**.

You may even be familiar with **~** or **!** for negation.

<u>Just don’t mix them. Stick to a convention.</u>

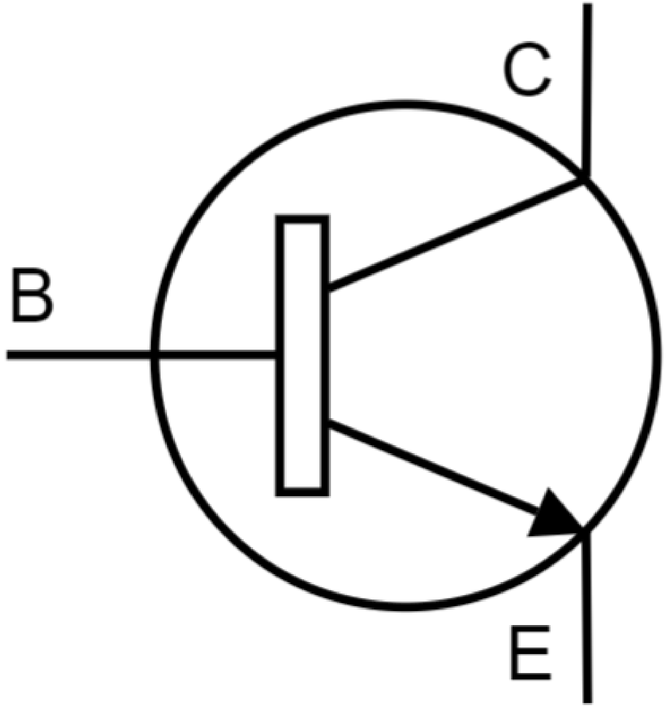
Constructing gates

- Device that acts either as a **wire that conducts electricity** or as **a resistor that blocks the flow of electricity**, depending on the **voltage level of an input** signal.
- A **transistor** has no moving parts, yet it acts like a switch.
- Transistors are made of a **semiconductor** material, which is neither a particularly good conductor of electricity nor a particularly good insulator.
- Transistors are the basic building blocks for gates.

Constructing gates

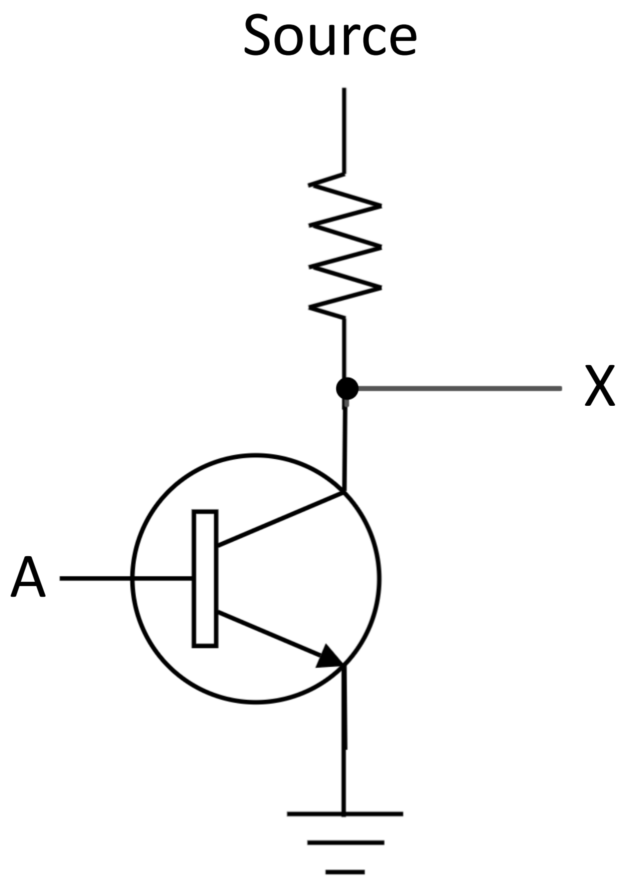
- A transistor has three terminals:
  - A collector
  - A base
  - An emitter

- If current flows into the **Emitter** then this results in the **Source** being connected to the **Ground**. This causes the **output voltage to drop**.



Constructing NOT gates

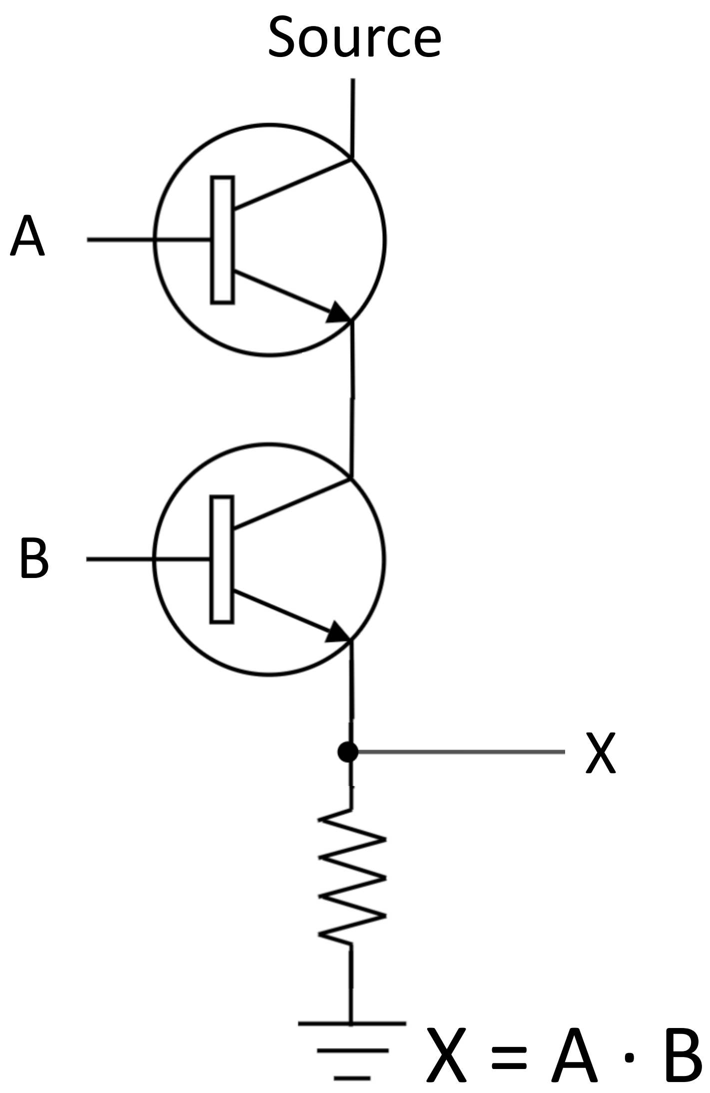
- This diagram shows how an NPN transistor might be connected to give a NOT gate.
- If there is a **high signal** coming into the base of the transistor, then the transistor lets **current flow through**. Thus pulling the out **signal low**.
- If there is a **low signal** coming into the base of the transistor, then the transistor does **not let any current through**. Thus, allowing the out **signal high**.



X = A’

Constructing AND gates

- This diagram shows how an NPN transistor might be connected to give an AND gate.
- If there is a high signal coming into both transistors, then the source signal will pass through to the output and it will be high (1).
- If either transistor receives a low signal then the output signal is low (0).



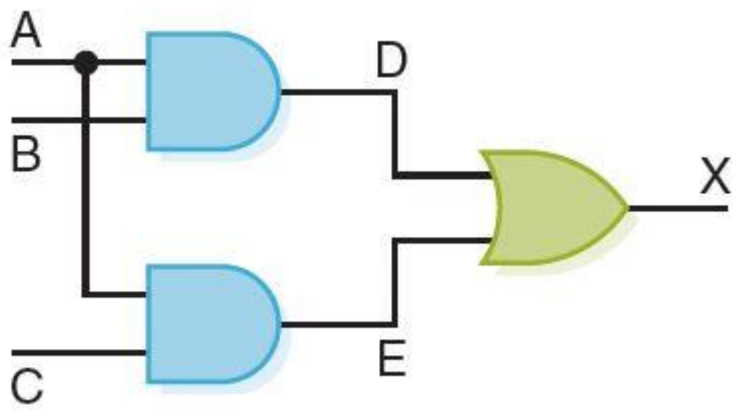
Circuits

- We can combine individual gates together into more complex **circuits**

- Circuits can be described by:
  - Boolean expressions: Same as for gates.
  - Truth tables: Same as for gates.
  - Logic diagrams: A graphical representation combining gate symbols.

Combinational Circuits

- Gates are combined into circuits by using the output of one gate as the input for another.



$$A \cdot B + A \cdot C$$

A	B	C	X
0	0	0	0
1	0	0	0
0	1	0	0
1	1	0	1
0	0	1	0
1	0	1	1
0	1	1	0
1	1	1	1

- Lecture 2:
  - Discuss circuit equivalence and Boolean algebra
  - Discuss several common circuits in computing

Circuit equivalence

- Circuits which produce the same output when provided identical inputs are call **equivalent**

- For example:

$$\mathbf{A \cdot (B+C) = A \cdot B + A \cdot C}$$

The truth tables match. Therefore, these expressions are equivalent.

$A \cdot (B + C)$

A	B	C	X
0	0	0	0
1	0	0	0
0	1	0	0
1	1	0	1
0	0	1	0
1	0	1	1
0	1	1	0
1	1	1	1

$A \cdot B + A \cdot C$

A	B	C	X
0	0	0	0
1	0	0	0
0	1	0	0
1	1	0	1
0	0	1	0
1	0	1	1
0	1	1	0
1	1	1	1

Circuit equivalence

- **Boolean algebra** allows us to apply provable mathematical principles to help design circuits and identify equivalence.

PROPERTY	AND	OR
Commutative	$AB = BA$	$A + B = B + A$
Associative	$(AB)\,C = A\,(BC)$	$(A + B) + C = A + (B + C)$
Distributive	$A\,(B + C) = (AB) + (AC)$	$A + (BC) = (A + B)\,(A + C)$
Identity	$A1 = A$	$A + 0 = A$
Complement	$A(A') = 0$	$A + (A') = 1$
De Morgan's law	$(AB)' = A'$ OR B'	$(A + B)' = A'B'$

AND (·) and OR (+)

- Why are we using the multiplication and addition operators here?
- Remember the Binary arithmetic section in the Number Systems lecture?

Binary Addition Table		
+	**0**	**1**
0	0	1
1	1	10

Binary Multiplication Table		
·	**0**	**1**
0	0	0
1	0	1

AND (·) and OR (+)

- We are applying operators on Boolean values.
- Let’s compare binary addition table against the OR truth table:

Binary Addition Table	
0 + 0	**0**
0 + 1	1
1 + 0	1
1 + 1	10

A	B	X
0	0	0
1	0	1
0	1	1
1	1	1

Using circuits to do stuff

- In the previous topic we discussed how our information is being represented by binary values.
- Given that our gates perform operations on binary values, can we design circuits which allow us to work with this underlying data/binary information?
- We could even use transistors (physical implementations of gates) to build this behaviour in hardware.

Adders

- Logical circuit designed to perform addition of binary values.
- Remember that an addition in binary can result in a **carry out**.

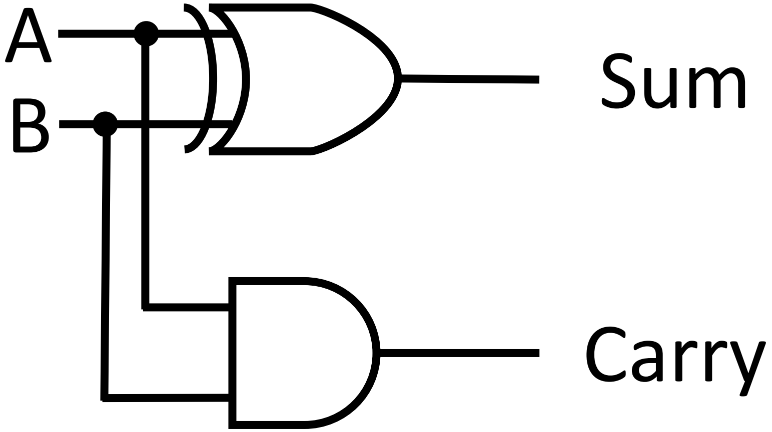
A	B	Sum	Carry Out
0	0	0	0
1	0	1	0
0	1	1	0
1	1	0	1

Let's build a circuit that reproduces this behaviour

Half Adder

- A half adder is a circuit that computes the **sum of two bits** and produces the correct **carry bit** as well.

A	B	Sum	Carry Out
0	0	0	0
1	0	1	0
0	1	1	0
1	1	0	1



$$sum = A \oplus B$$
$$carry = A \cdot B$$

Half Adder

- The half adder takes in two bits and computes the sum and carry.

- But when we add two bits in binary, we actually need **3 input values** to be considered!

- Why?
  - We have two digits to add at position ***n***, and the carry from position ***n-1***

- To handle this, we extend our **half adder** to a **full adder…**

Full Adder

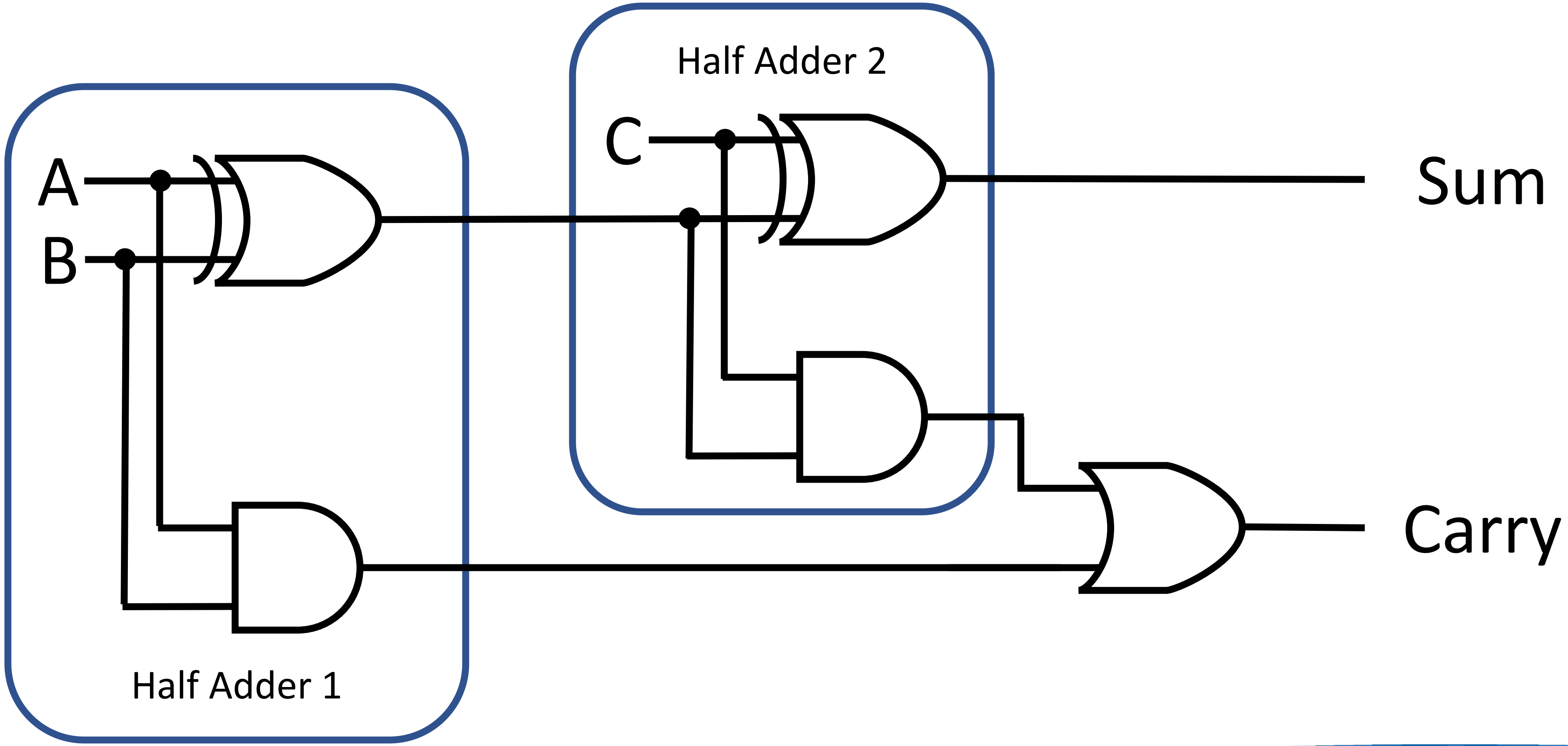
- Circuit which takes a carry-in value as well the two digits to add

$$\text{sum} = A \oplus B \oplus C$$

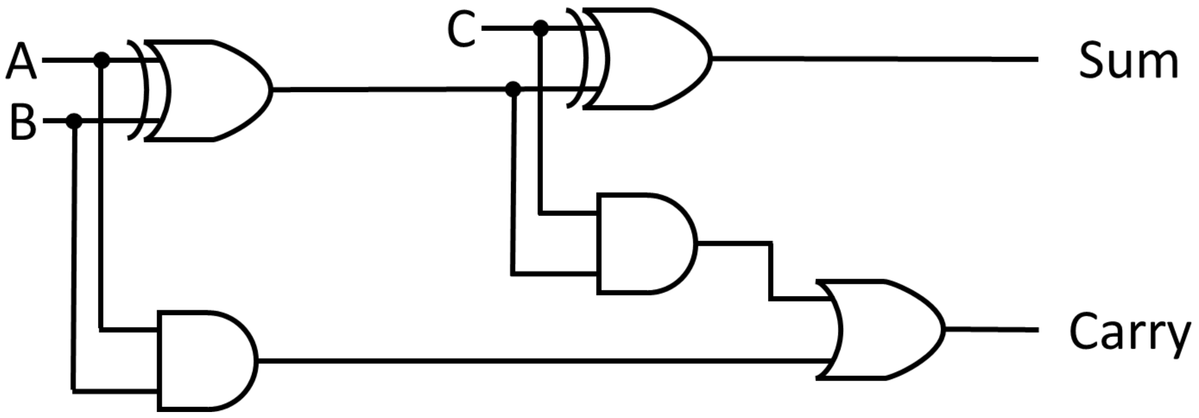
$$\text{carry} = (A.B) + (C.(A \oplus B))$$

where A and B are the digits in that position, and C is the carry in

Full Adder



Full Adder

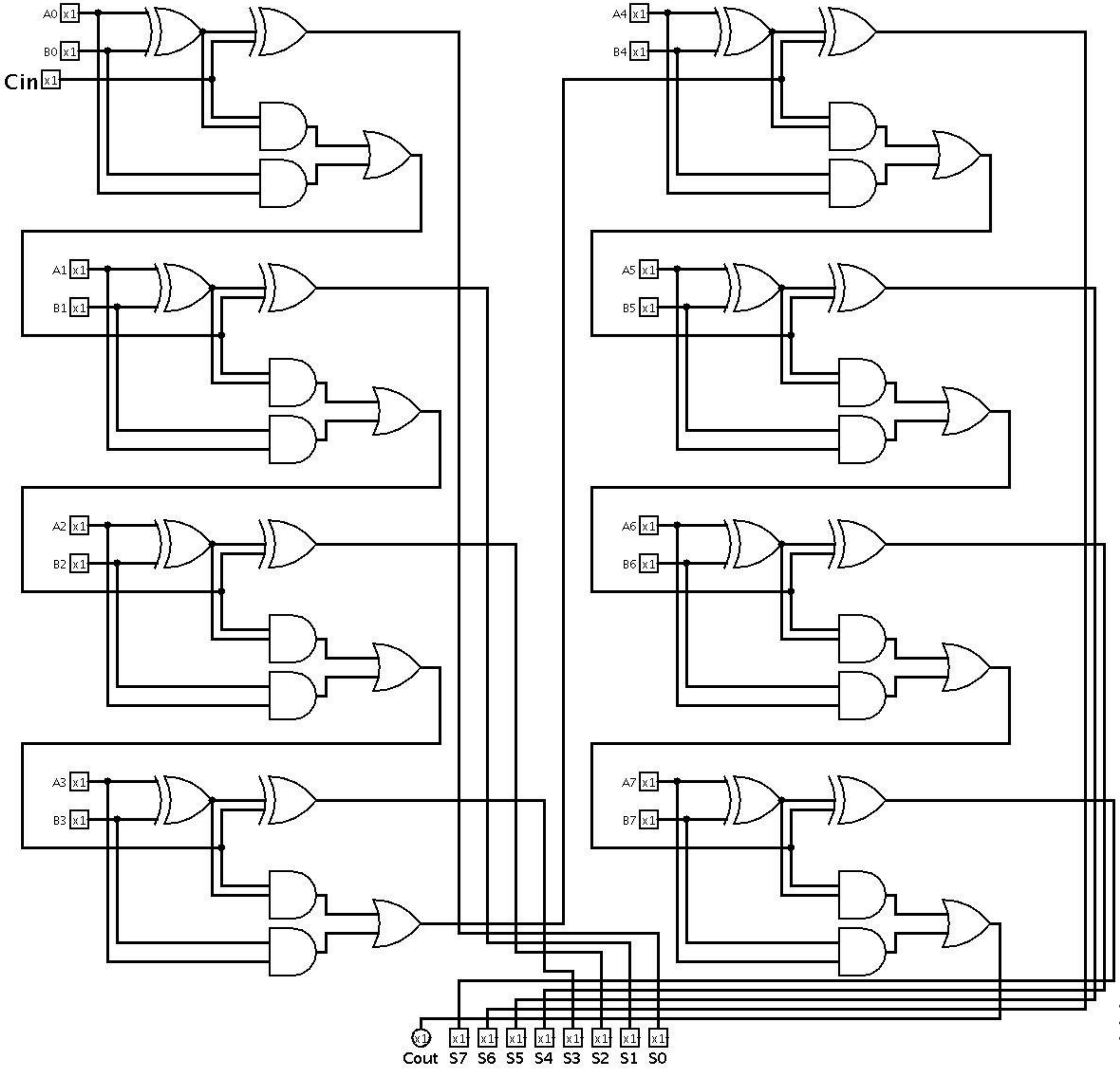


A	B	C	Sum	Carry Out
0	0	0	0	0
1	0	0	1	0
0	1	0	1	0
1	1	0	0	1
0	0	1	1	0
1	0	1	0	1
0	1	1	0	1
1	1	1	1	1

Full Adder

- The full adder adds two bits (and the carry in)
- But often our representations are multiples of 8 bits (a byte)
- We can therefore combine 8 full adders together to create a single 8-bit adder. This allows us to add two 8-bit values together using logical circuitry and electrical signals

8-bit Adder


A0
B0
Cin
A1
B1
A2
B2
A3
B3
A4
B4
A5
B5
A6
B6
A7
B7
Cout
S7
S6
S5
S4
S3
S2
S1
S0

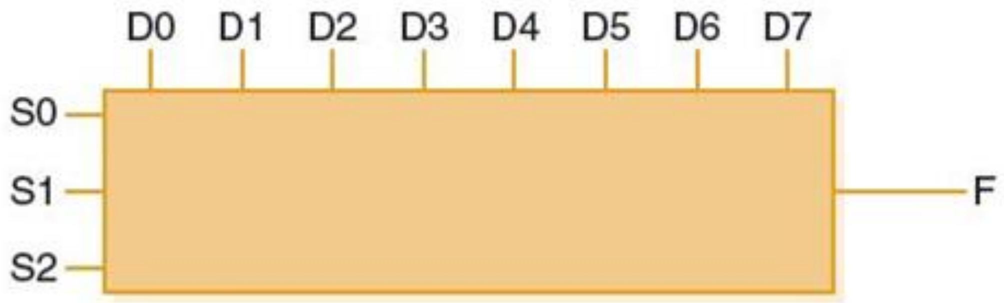
Multiplexers

- Often, we want to move values around our computer:
  - Passing them to and from storage
  - Pass them to the processor to perform calculations
  - Get values to and from auxiliary input/output devices
- We pass electrical signals down wiring to their destination, but we don’t want to have loads of unnecessary wiring.
- However, we need to make sure that signals are routed correctly. We don’t want signals overlapping or going to the wrong destination.

Multiplexers

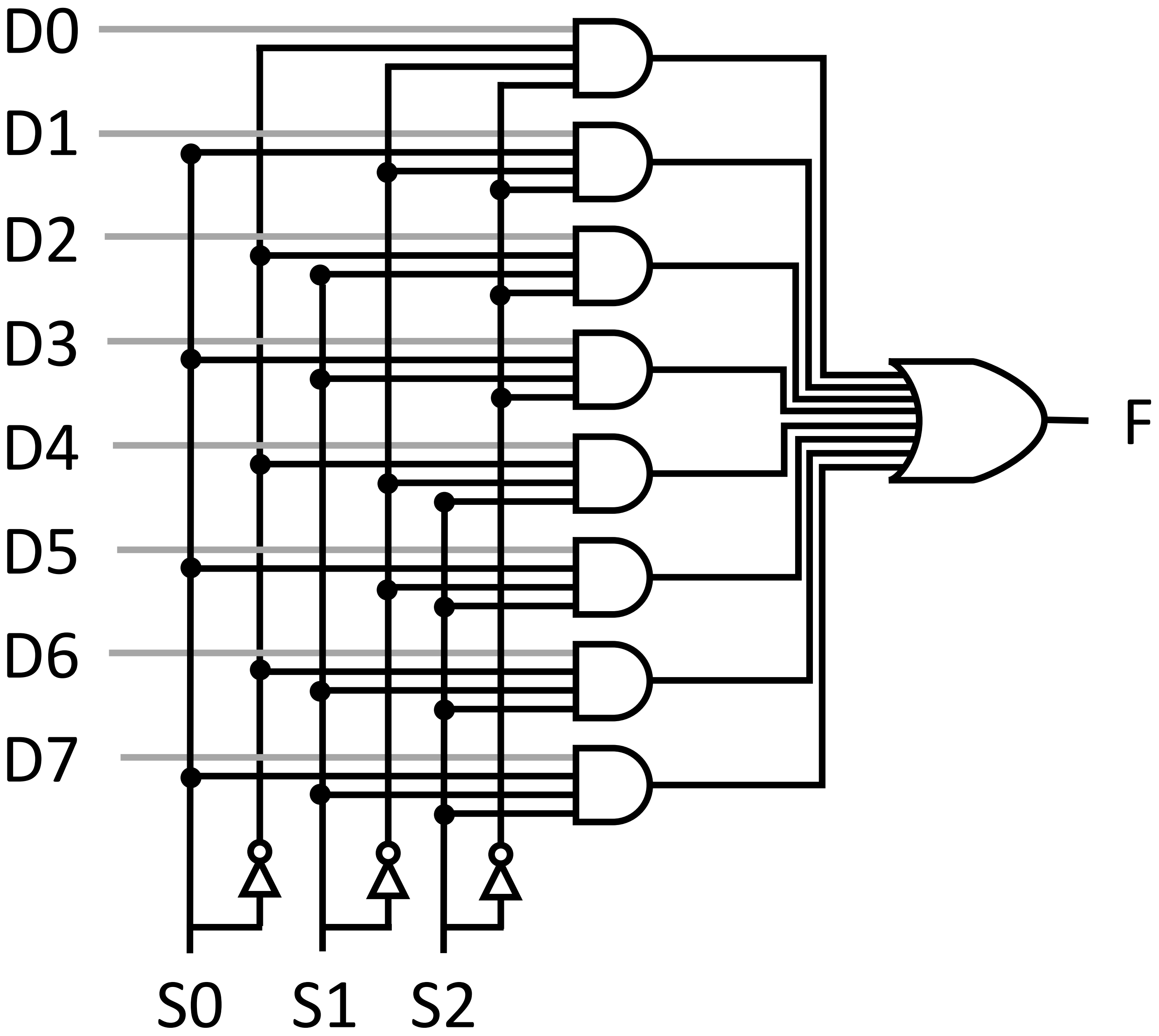
- A multiplexer (MUX) is a circuit which uses **input control signals (S)** to determine which of the **input data signals (D)** is routed to the **output signal (F)**
- E.g. we have 8 possible data signals, and we use 3 control signals to determine which one is routed to the output.
- Why do we need 3 control signals in this example?

S0	S1	S2	F
0	0	0	D0
1	0	0	D1
0	1	0	D2
1	1	0	D3
0	0	1	D4
1	0	1	D5
0	1	1	D6
1	1	1	D7



Multiplexers

S0	S1	S2	F
0	0	0	D0
1	0	0	D1
0	1	0	D2
1	1	0	D3
0	0	1	D4
1	0	1	D5
0	1	1	D6
1	1	1	D7



Multiplexers

- At the other end of the output line (F) we may have a **demultiplexer** (or DEMUX), which would allow us to do the opposite.
- We can use control signals as a routing switch to say which of several output lines our signal will be broadcast to.

Gates as memory units

- Digital circuits can be used to store information

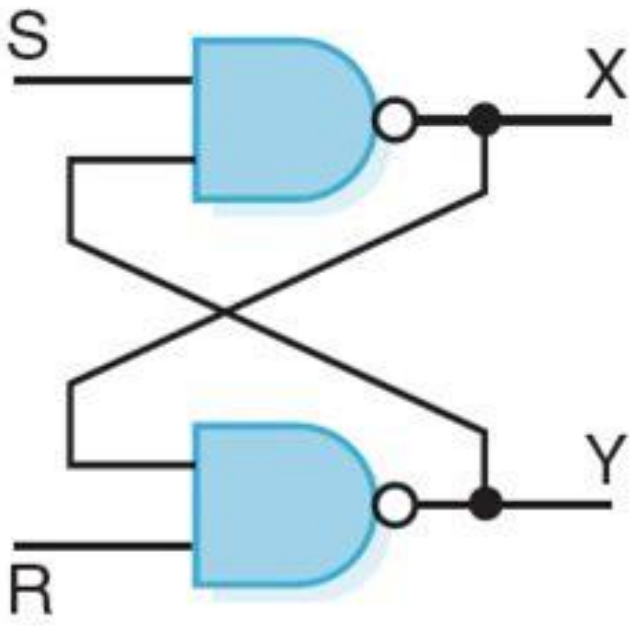
- These circuits form a **sequential circuit**, because the output of the circuit is also used as input to the circuit.

- By constructing a suitable circuit, we can **store a singular bit** of information (either 0 or 1).

- To do this we can use a circuit called the **S-R Latch**

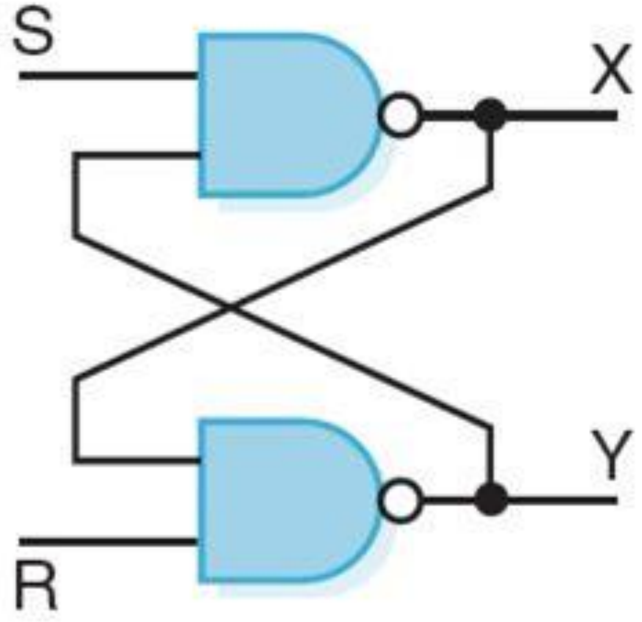
S-R Latch (Set/Reset Latch)

- An S-R Latch stores a single binary value
- It can be updated by changing the signal on S and R, which in turn affect X and Y
- If:
  - X = 1 and Y = 0, then the value stored is 1
  - X = 0 and Y = 1, then the value stored is 0
- We can design an S-R Latch in a variety of ways, depending on the kinds of gates we use



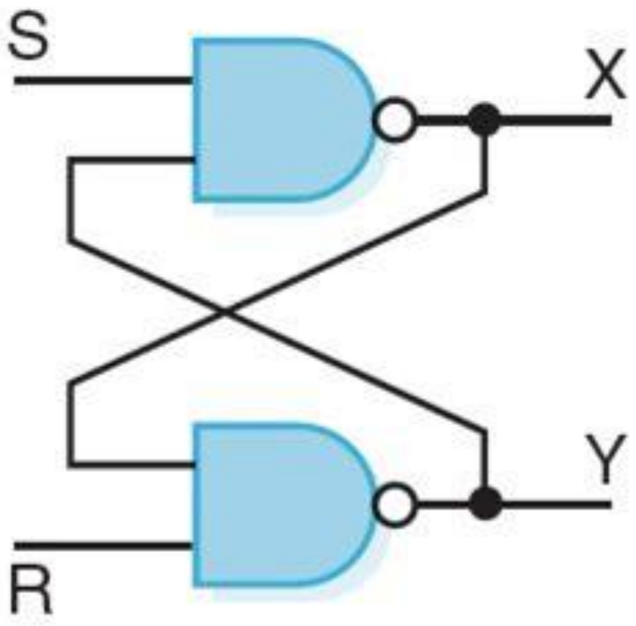
S-R Latch (Set/Reset Latch)

- Assume that S and R are never both 0 at the same time
- The design of this circuit guarantees that the two outputs X and Y are always complements of each other
- The value of X at **any point in time** is considered to be the **current state** of the circuit
- Therefore, if X is 1, the circuit is storing a 1; if X is 0, the circuit is storing a 0



S-R Latch (Set/Reset Latch)

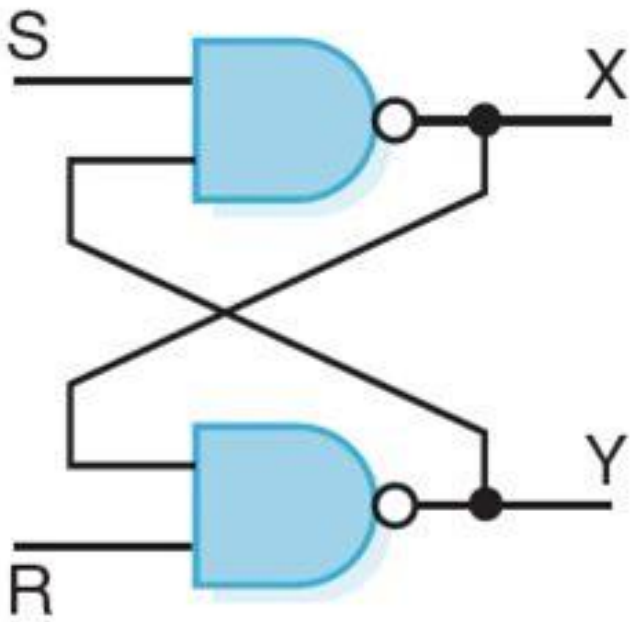
- If S and R are both 1, the output on X **will not change.**
- To set the value of X to 1, we set S to 0 and then change S back to 1 to stabilise.
- To set the value of X to 0 we set R to 0, and then change R back to 1 to stabilise.
- Setting both S and R to 0 at the same time is an **invalid** action.



S-R Latch (Set/Reset Latch)

- Truth Table:

S	R	X	Y	Notes
1	0	0	1	
1	1	0	1	(if following S=1,R=0)
0	1	1	0	
1	1	1	0	(if following S=0,R=1)
0	0	1	1	Invalid operation



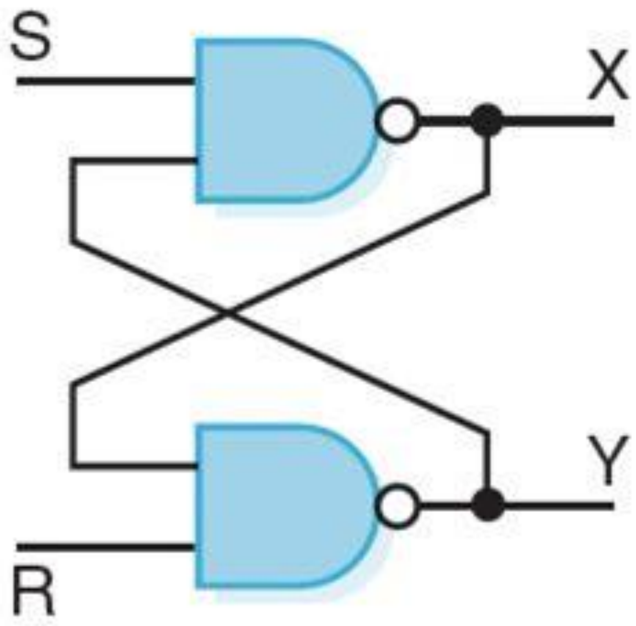
S-R Latch Worked Example

- But really, this is confusing without considering the **temporality** of the system.

- Changing a value in S or R creates a voltage change in the system that **travels through the circuit**, impacting on the outputs of other gates.

S-R Latch Worked Example

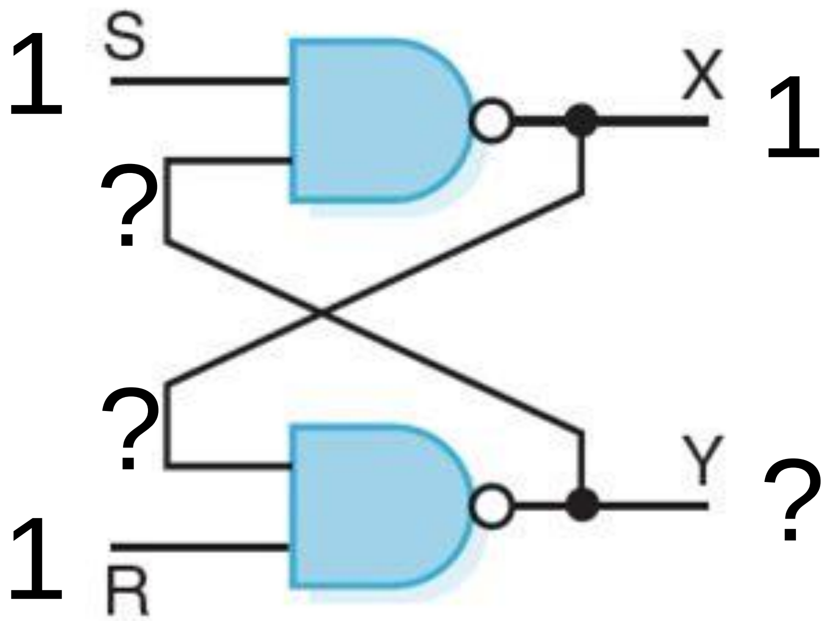
Consider and initial state of S = 1, R = 1, and X = 1:



What is Y?

S-R Latch Worked Example

Consider and initial state of S = 1, R = 1, and X = 1:



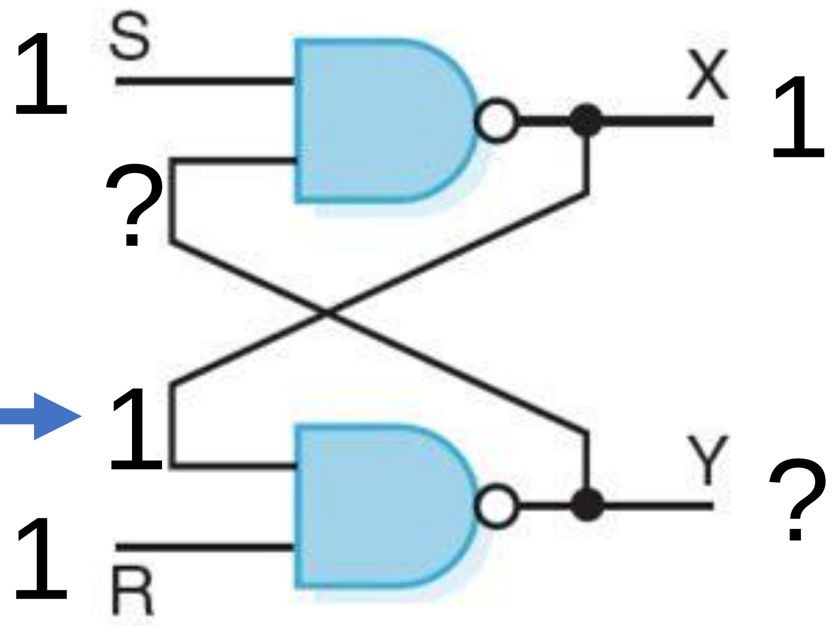
What are the values of the “?” ?

What is Y?

S-R Latch Worked Example

Consider and initial state of S = 1, R = 1, and X = 1:

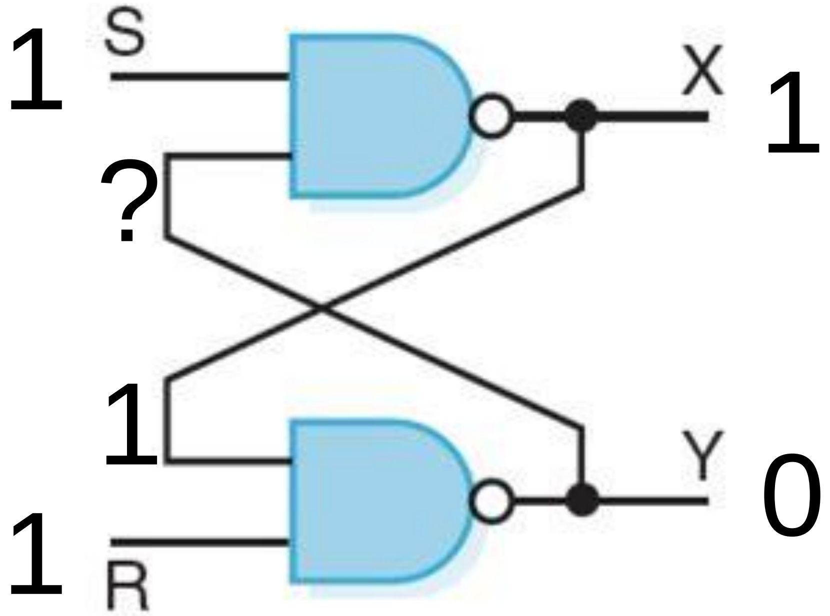
This is equal to X, it’s the same wire



What is Y?

S-R Latch Worked Example

Consider and initial state of S = 1, R = 1, and X = 1:



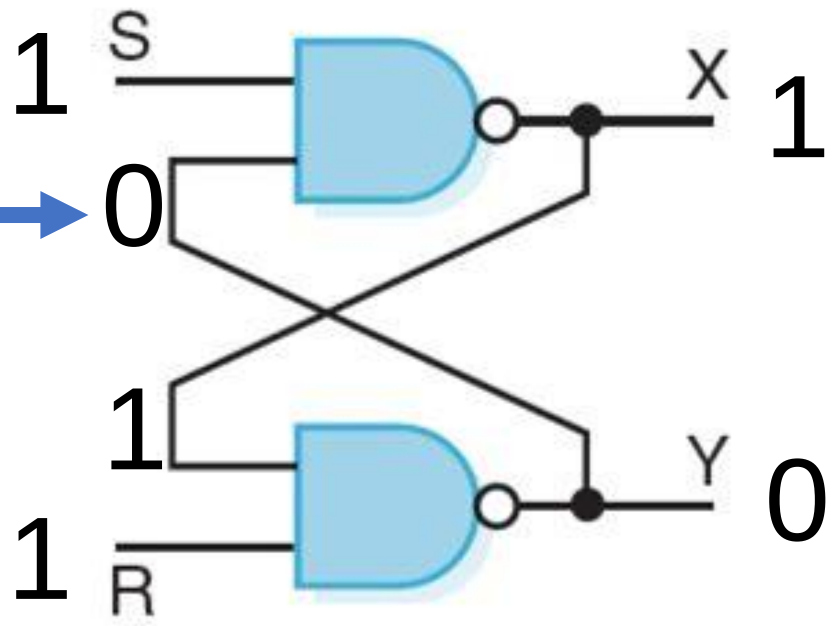
So then we can compute the output of the NAND

What is Y?

S-R Latch Worked Example

Consider and initial state of S = 1, R = 1, and X = 1:

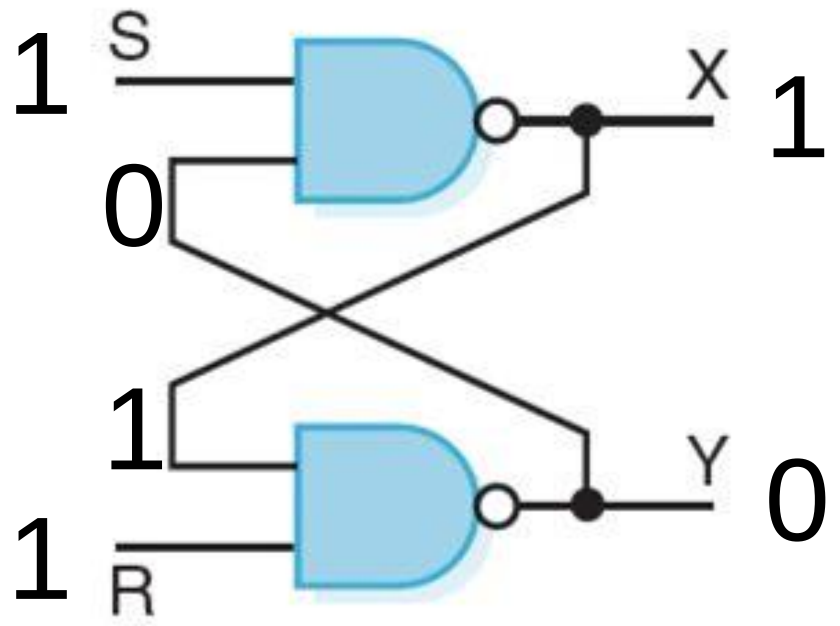
This is equal to Y, it’s the same wire



What is Y?

S-R Latch Worked Example

Consider and initial state of S = 1, R = 1, and X = 1:

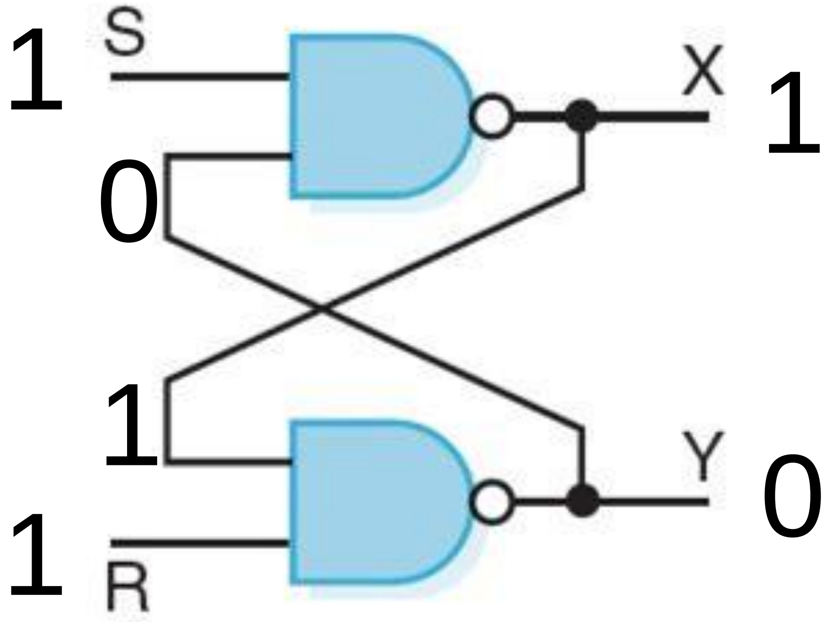


This output doesn't change, so our circuit is stable. This S-R Latch is storing the value 1 (the value in X)

What is Y?

S-R Latch Worked Example

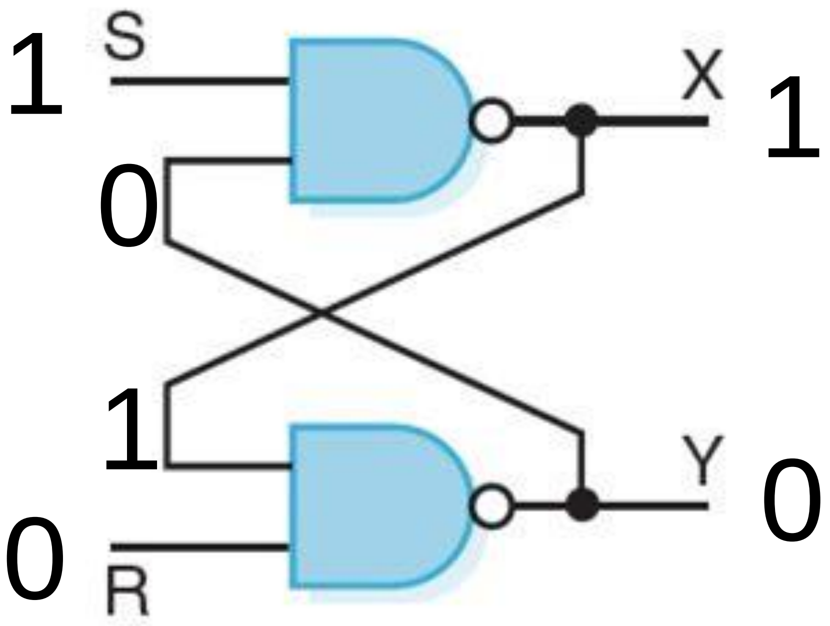
Now let’s change the signal going into R to 0:



S-R Latch Worked Example

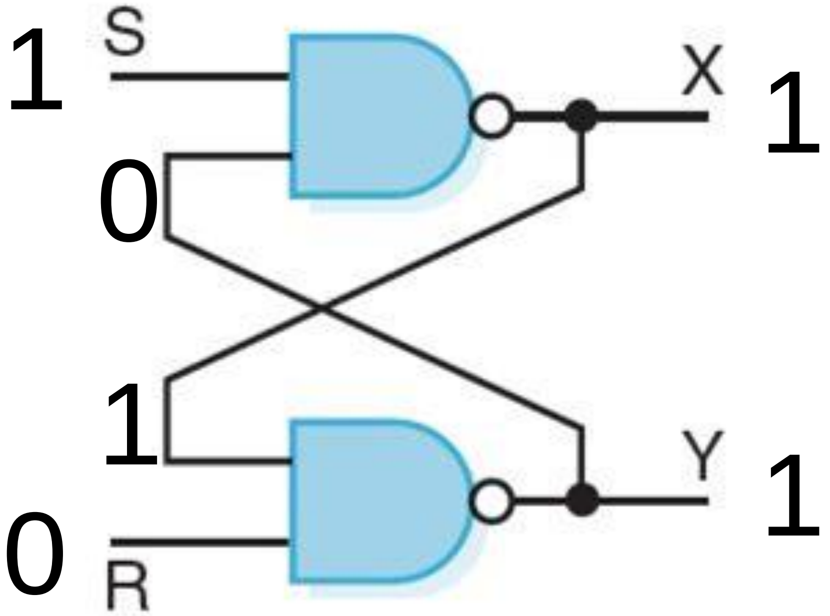
Now let's change the signal going into R to 0:

Let's update R to be 0



S-R Latch Worked Example

Now let's change the signal going into R to 0:

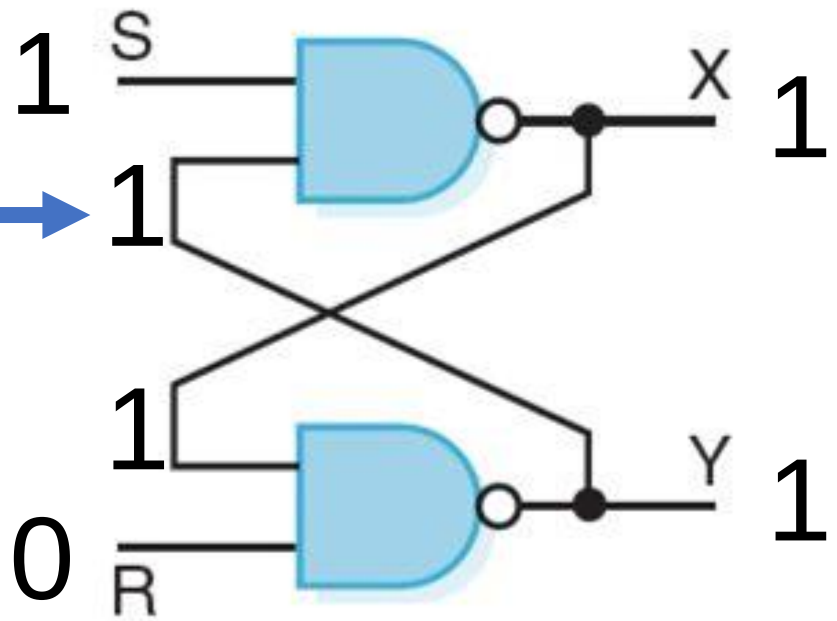


So then we can update the output of the NAND

S-R Latch Worked Example

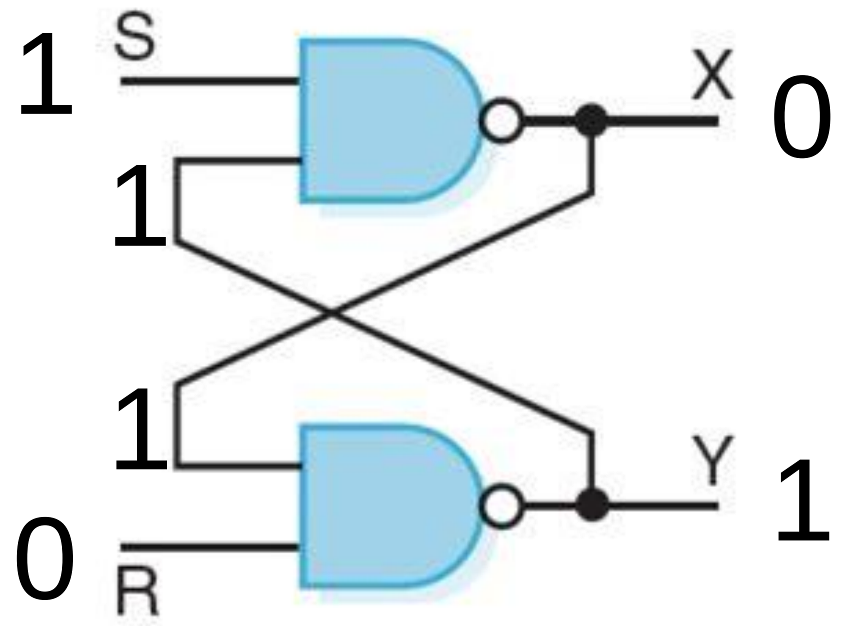
Now let's change the signal going into R to 0:

This is equal to Y, it's the same wire. So it updates as well



S-R Latch Worked Example

Now let's change the signal going into R to 0:

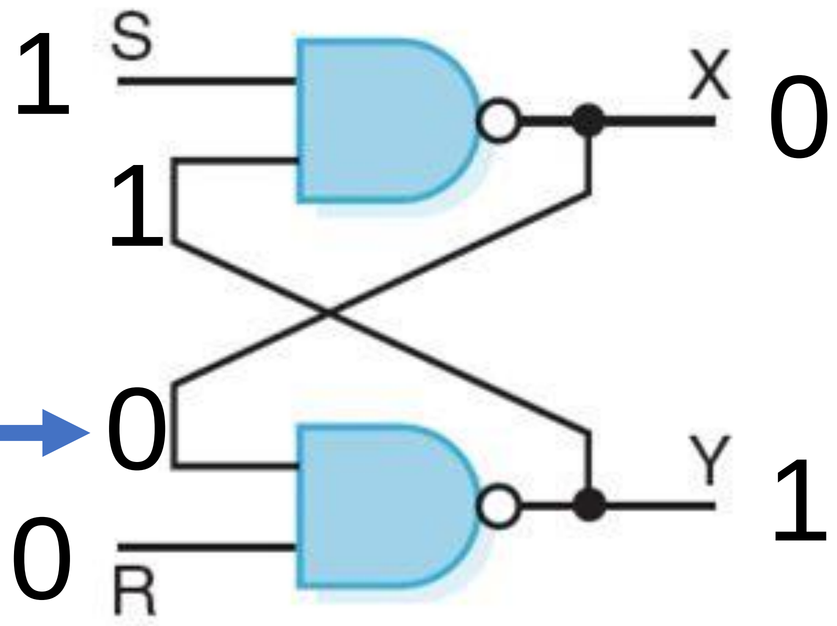


So then we can update the output of the NAND

S-R Latch Worked Example

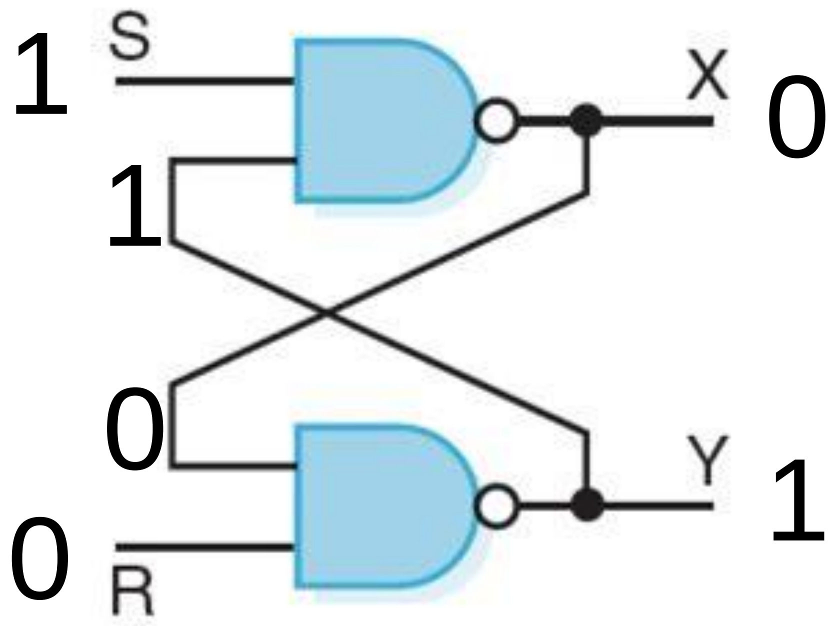
Now let's change the signal going into R to 0:

This is equal to X, it's the same wire. So it updates as well



S-R Latch Worked Example

Now let's change the signal going into R to 0:

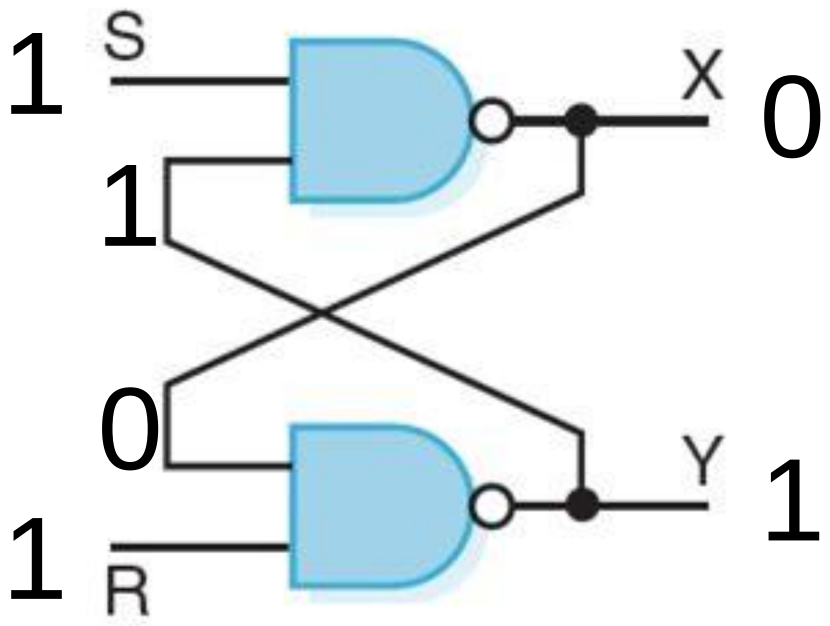


This output doesn't change, so our circuit is stable. This S-R Latch is storing the value 0 (the value in X)

S-R Latch Worked Example

Now let's change the signal going into R to 0:

We finally set R back to 1. This doesn't change the output of the NAND or have any effect on the values.



We do this final operation so that our S-R Latch is back to a state where we can update the signal on either S or R without it breaking the latch

S-R Latch

- This is why the temporality (behaviour over time) is important!
- Try working through the S-R latch yourself.
- What happens if we set both S and R to 0?
- How do we initialise the starting values of the S-R latch?