COM1006 Devices and Networks (Autumn)
COM1090 Computer Architectures

Lecture #4

▶ Gates and Logic

Dr Dirk Sudholt
Department of Computer Science
University of Sheffield

d.sudholt@sheffield.ac.uk
http://staffwww.dcs.shef.ac.uk/~dirk/campus_only/com1006/

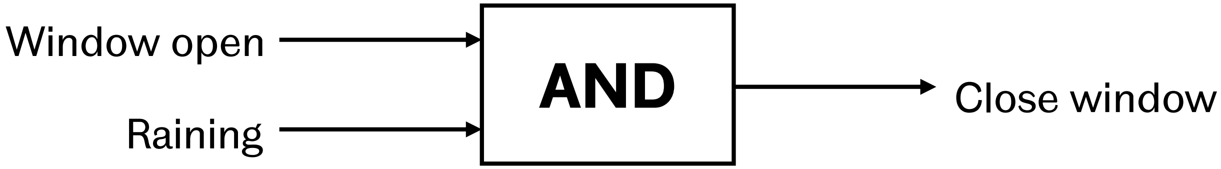
Based on parts of Chapter 2 in Clements, Principles of Computer Hardware

Aims of this lecture

- To introduce combinatorial logic
- To introduce fundamental gates: AND, OR, NOT
- To show how truth tables can be used to describe digital circuits
- To show how a Boolean equation consisting of a sum of products can be used to describe a digital circuit
- To introduce NAND, NOR and EOR gates

Combinatorial and sequential logic

- Digital computers are constructed from two kinds of logic elements:
- **Combinatorial logic**. Generates output based solely on its current input, e.g. if window open AND raining then close window:



- **Sequential logic** (see later lectures). The output from a sequential logic element depends on its past history as well as its current input (i.e., a sequential element remembers its previous inputs).

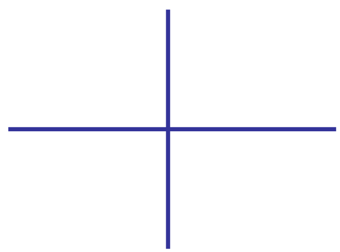
Gates

- A **gate** is a simple processing element with one or more input terminals and an output terminal.
- The output of a gate is a function of its inputs only.
- Gates can be regarded as transmission elements that control the flow of information through a computer.
- Digital computers consist of nothing more than the interconnection of **AND**, **OR** and **NOT** gates.
- Other gates (**NAND**, **NOR**, **EOR**) can be derived from them.
- Later we'll see that any digital circuit can be designed from interconnected NAND (or NOR) gates alone.
- At the chip level, gates are implemented by transistors.

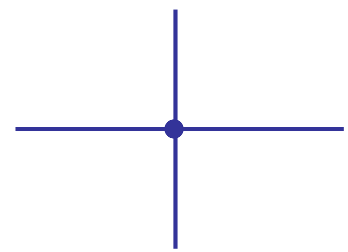
Circuit conventions

- Logic circuits are generally read from left to right (i.e., inputs on left, outputs on right).
- Circuits can contain many signal paths (lines) and sometimes these must be drawn as crossing one another:

Two lines that cross but are not connected



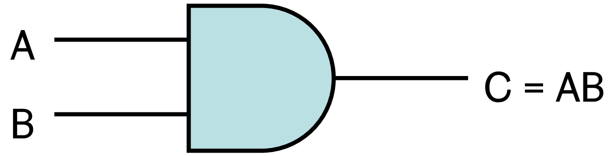
Two lines are connected (indicated by dot at intersection)



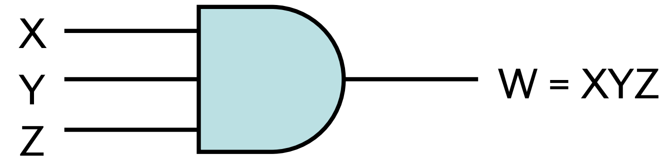
- The voltage at any point along a conductor is constant, so the logical state is the same everywhere on a line.

Fundamental gates: AND

- An AND gate has two or more inputs and a single output.
- The output of an AND gate is true iff each of its inputs is true.



Two-input AND gate



Three-input AND gate

- The logical symbol for AND is a dot, so A AND B is written $A \cdot B$; in practice the dot is often omitted and we write AB.
- Alternative notations: $A \wedge B$, A.B
- Logical AND behaves like a multiplier in conventional algebra, e.g. A(B+C) = AB+AC in conventional and Boolean algebra.

Truth tables

- Truth tables are a useful way of describing the relationship between inputs and outputs of a gate.
- The value of each output is tabulated for every possible combination of the inputs.
- Inputs are binary so a circuit with *n* inputs has 2^n rows in its truth table.

Truth table
for the AND gate

A	B	F = AB
0	0	0
0	1	0
1	0	0
1	1	1

▶ Logical operations on words

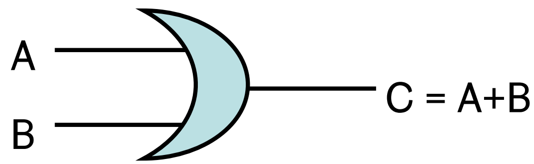
- Recall that computers process groups of bits called **words**. Typically a word consists of 8, 16, 32 or 64 bits.
- When logical operations such as AND are applied to words, the operation is applied to each pair of bits:

1	1	0	1	1	0	1	1	word A
0	0	0	0	1	1	1	1	word B
0	0	0	0	1	0	1	1	$C = A \cdot B$

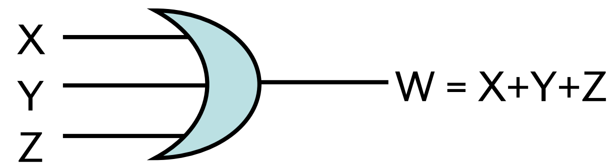
- This example shows a common use of AND; word B acts as a **mask** that clears some bits of word A.

▶Fundamental gates: OR

- An OR gate has two or more inputs and a single output.
- The output of an OR gate is true if any one (or more than one) of its inputs is true.



Two-input OR gate



Three-input OR gate

- The logical symbol for OR is +, so A OR B is written A+B.
- Alternative notation: $A \vee B$
- The logical OR operator behaves like addition in conventional algebra.

❓ **Does logical OR have the same meaning as "or" in English?**

Truth table and logical OR on words

- Computers also perform logical OR on words, e.g.:

1	0	0	1	1	1	0	0	word A
0	0	1	0	0	1	0	1	word B
1	0	1	1	1	1	0	1	C=A+B

- Logical OR is used to set one or more bits in a word to logical 1.
- So using AND and OR we can selectively set and clear the bits of a word.

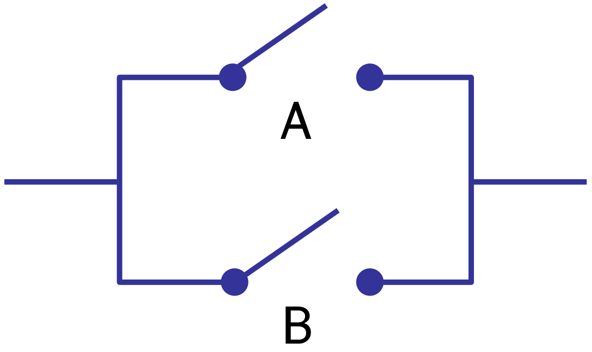
Truth table for the OR gate

A	B	F = A+B
0	0	0
0	1	1
1	0	1
1	1	1

AND and OR gates as circuits

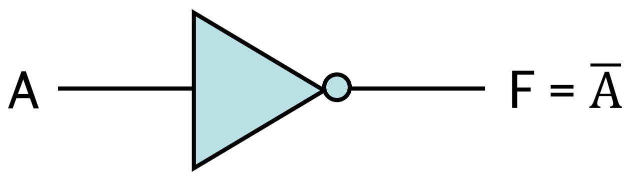
AND
Current will only flow
if A and B are closed

OR
Current will flow if
either A or B is closed



Fundamental gates: NOT

- The NOT gate is also called an **inverter** or **complementer**.
- The NOT gate has one input and one output. The output is the opposite (logical inverse) of the input.



NOT gate

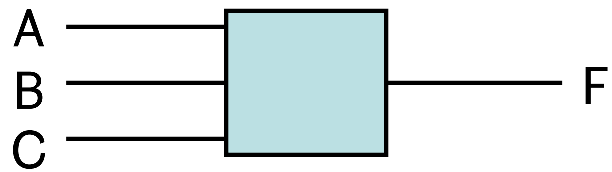
A	F = $\overline{A}$
0	1
1	0

Note that it's the small circle that indicates logical inversion, not the triangle

- Other conventions: A*, ¬A, /A
- NOT can also be applied to words (e.g., if A=`11011100` then $\overline{A}$ = `00100011`)

Example of AND-OR-INVERT logic

- Problem: construct a 3-input majority device. The output is to be TRUE when 2 or more of the inputs are TRUE.



- General approach:
  - Construct a **truth table** that defines the required behaviour;
  - Read a **sum of products** expression from the truth table
  - **Simplify** the expression in order to minimise the number of gates/lines used (just examples now; see later lectures)
  - **Construct** the device (on paper or in a simulator)

Step 1: Truth table

Note that when tabulating the possible values of the inputs the order is not important, but it is usual to list them in the natural binary sequence

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Step 2: Sum of products expression

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

- From the truth table, we can write down the set of inputs that cause the output to be true:

  $\overline{\mathrm{A}}\mathrm{BC}, \mathrm{A}\overline{\mathrm{B}}\mathrm{C}, \mathrm{AB}\overline{\mathrm{C}}, \mathrm{ABC}$

- The output F is the logical sum of these four cases:

  $\mathrm{F} = \overline{\mathrm{A}}\mathrm{BC} + \mathrm{A}\overline{\mathrm{B}}\mathrm{C} + \mathrm{AB}\overline{\mathrm{C}} + \mathrm{ABC}$

- This is called a **sum of products (S-of-P) expression** because it is the logical OR (sum) of a group of terms each composed of several variables ANDed together (products)
- Useful for drawing circuits in a systematic fashion (see later)

Sum of products expression (continued)

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

- The terms $\bar{A}BC, A\bar{B}C, AB\bar{C}, ABC$ represent **minterms**.
- A **minterm** is a product of all variables in true or complemented form.
- **Every** Boolean function F can be described as sum of products (i.e. ORing minterms where F is 1).
- This is called **canonical form**.
- So every Boolean function can be implemented with AND, OR, NOT.
- There is an alternative representation based on **products of sums (P-of-S)**, ANDing sums of variables where F is 0.

Step 3: Simplify the S-of-P expression

- We have $F = \overline{A}BC + A\overline{B}C + AB\overline{C} + ABC$
- There is a simpler form for this expression, using fewer and simpler terms.

Why do you think a simpler form is better?

- We'll spend the next lecture talking about how to simplify expressions.
- For now, I'll give the solution away – a simpler form is:

  F = AB + BC + AC

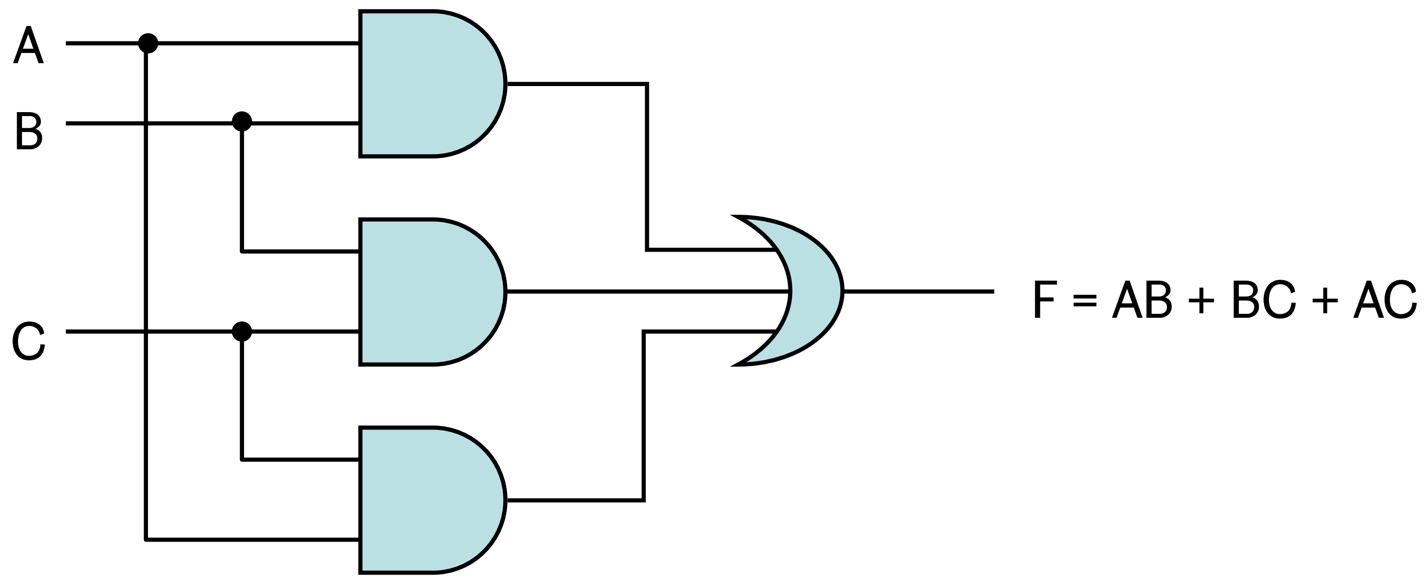
- How can you be sure that it's the same function?

Comparing Boolean Functions

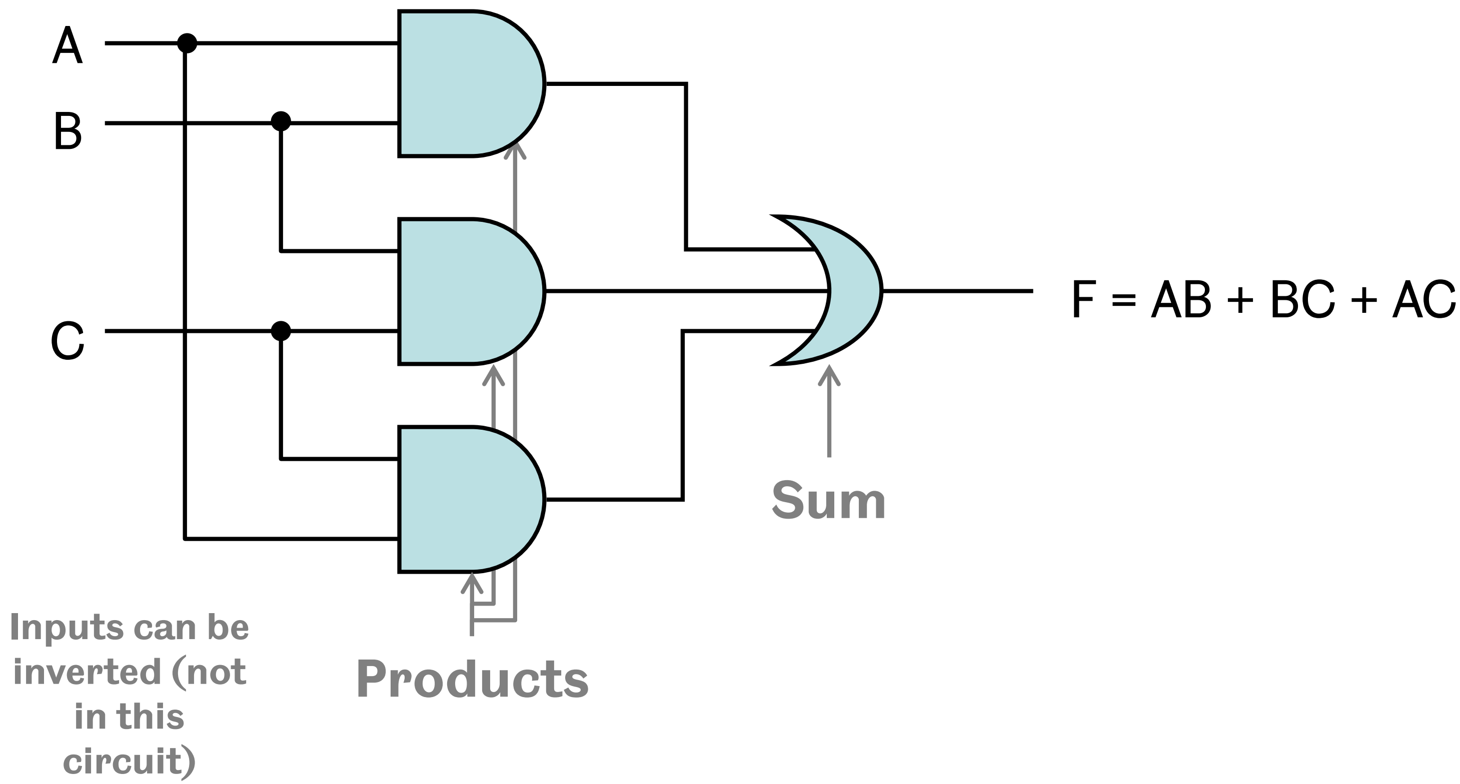
A	B	C	F	AB + BC + AC
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	1	1
1	0	0	0	0
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

- Two Boolean functions are equal if all entries in the truth table are the same.
- This method is not feasible for many variables.

Step 4: Construct the device



Step 4: Construct the device

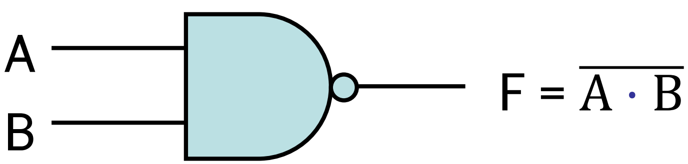


NAND and NOR gates

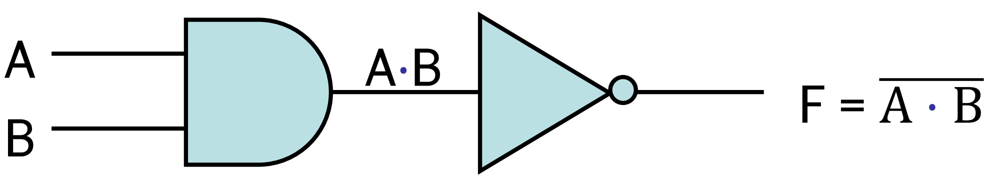
- NAND and NOR gates are the two most widely used gates in real circuits.
- They are not fundamental gates because they are derived from the other gates we've seen:
  - The NAND gate is derived from an AND gate followed by an inverter (Not AND)
  - The NOR gate is derived from an OR gate followed by an inverter (Not OR)
- Remember that the small circle indicates inversion.
- Another common gate is Exclusive OR (EOR) which is derived from AND, OR and NOT gates.

NAND

NAND gate



AND gate followed by an inverter

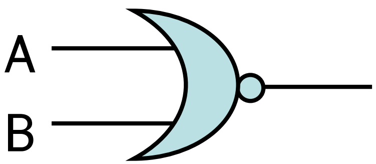


Truth table for the NAND gate

A	B	F = $\overline{A \cdot B}$
0	0	1
0	1	1
1	0	1
1	1	0

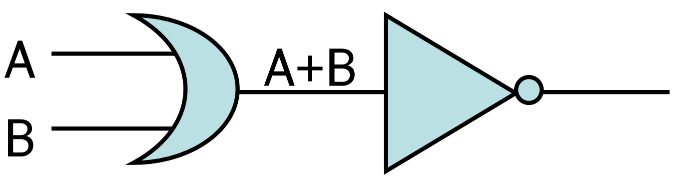
NOR

NOR gate



$F = \overline{A + B}$

OR gate followed by an inverter



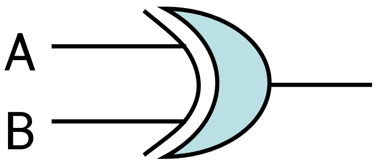
$F = \overline{A + B}$

Truth table for the NOR gate

A	B	$F = \overline{A + B}$
0	0	1
0	1	0
1	0	0
1	1	0

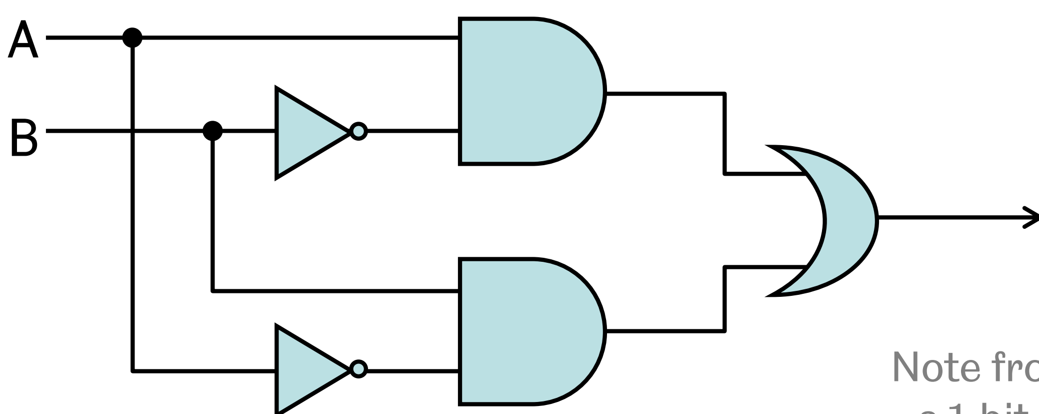
EOR

EOR gate



$F = \overline{A} \cdot B + A \cdot \overline{B}$

Circuit implementation of EOR gate



$F = \overline{A} \cdot B + A \cdot \overline{B}$

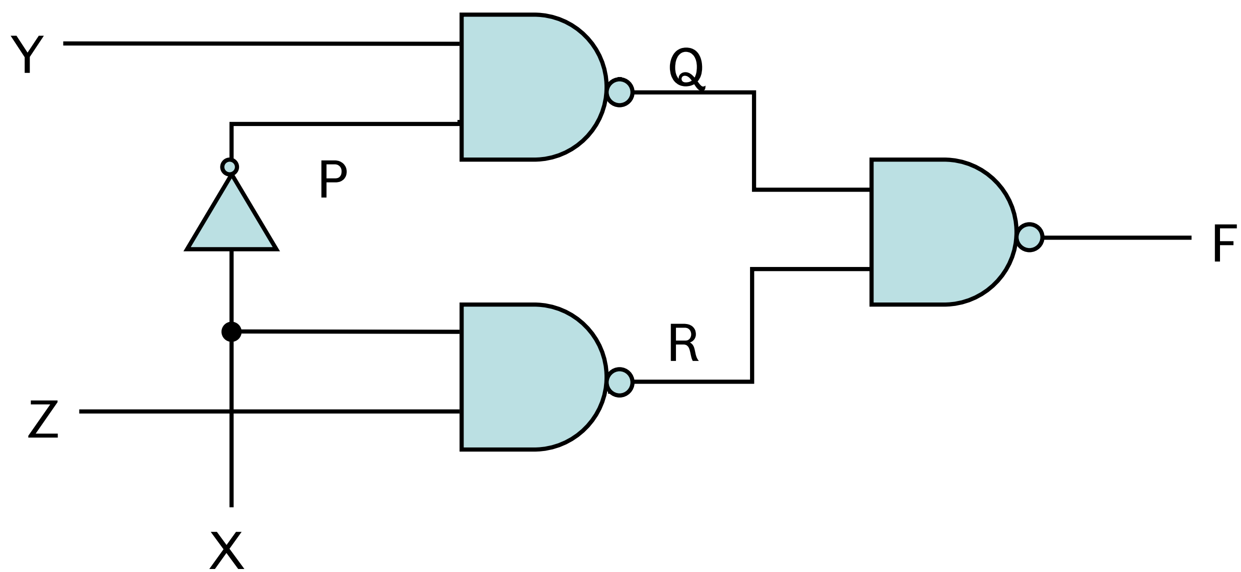
Truth table for the EOR gate

A	B	F
0	0	0
0	1	1
1	0	1
1	1	0

Note from the truth table that EOR is a 1-bit adder (excluding the carry)

Another example - what is it?

- Consider the following digital circuit which has three inputs, one output and three intermediate values.
- What does it do?



Truth table (incomplete)

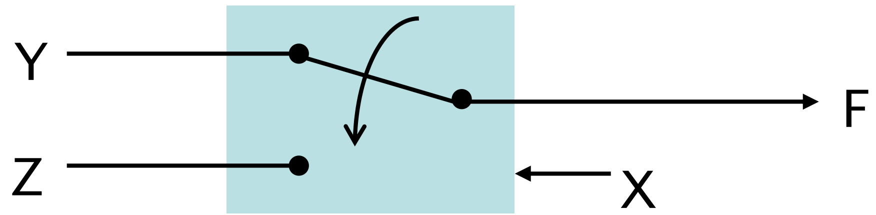
Inputs			Intermediate values			Output
X	Y	Z	P=$\overline{X}$	Q=$\overline{PY}$	R=$\overline{XZ}$	F=$\overline{QR}$
0	0	0				
0	0	1				
0	1	0				
0	1	1				
1	0	0				
1	0	1				
1	1	0				
1	1	1				

Truth table

Inputs			Intermediate values			Output
X	Y	Z	P=$\overline{X}$	Q=$\overline{PY}$	R=$\overline{XZ}$	F=$\overline{QR}$
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

It's a multiplexer (switch)

- From the truth table, note that:
  - when X=0, F=Y
  - when X=1, F=Z
- So this circuit acts as an electronic switch, that connects the output to one of the two inputs (Y or Z) depending on the value of X:



- This circuit is called a **multiplexer**.

Two equivalent Boolean functions

- From the truth table, we get a S-of-P expression for F:

  $F = \overline{X}Y\overline{Z} + \overline{X}YZ + X\overline{Y}Z + XYZ$

- From the circuit we could also get an equation for F by writing the outputs of each gate in terms of its inputs:

  $F = \overline{QR}$, $Q = \overline{YP}$, $P = \overline{X}$

  Therefore $Q = \overline{Y\,\overline{X}}$ by substituting for P

  $R = \overline{XZ}$

  Therefore $F = \overline{\overline{Y\,\overline{X}}\;\overline{XZ}}$.

- So a given Boolean function can be written in more than one way; we return to this point in the next lecture.

Summary

- Fundamental gates are AND, OR and NOT
- Digital circuits can be conveniently described by
  - A truth table
  - A Boolean function in sum-of-products form
- Fundamental gates can be used to construct three other very important gates: NAND, NOR and EOR
- The Boolean function that describes a circuit can be written in many forms; some are more amenable to implementation than others