Computing systems II

Luca Gammaitoni, NiPS Laboratory, Università di Perugia, IT

The act of computing

Computing = messing up with quantities

Better done with some device help

Counting with fingers

The act of computing

Computing = messing up with quantities

Better done with some device help

Counting with stones

Calculus = small stone

The act of computing

Computing = messing up with quantities

Better done with some device help

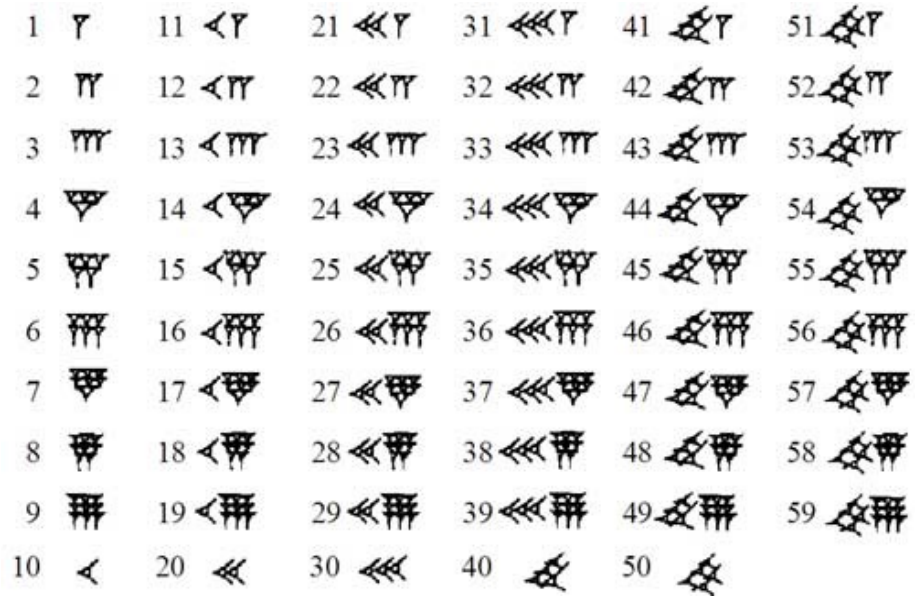
Counting with stones

The roman’s hand abacus

The act of computing

Computing = messing up with quantities

Symbolic computation
(counting with writing symbols)



Babylonian symbols

The act of computing

Computing = messing up with quantities

Symbolic computation
(counting with writing symbols)

	.	10	20	30	40	50	60	70	80	90
0	.	X	XX	XXX	XL	L	LX	LXX	LXXX	XC
1	I	XI	XXI	XXXI	XLI	LI	LXI	LXXI	LXXXI	XCI
2	II	XII	XXII	XXXII	XLII	LII	LXII	LXXII	LXXXII	XCII
3	III	XIII	XXIII	XXXIII	XLIII	LIII	LXIII	LXXIII	LXXXIII	XCIII
4	IV	XIV	XXIV	XXXIV	XLIV	LIV	LXIV	LXXIV	LXXXIV	XCIV
5	V	XV	XXV	XXXV	XLV	LV	LXV	LXXV	LXXXV	XCV
6	VI	XVI	XXVI	XXXVI	XLVI	LVI	LXVI	LXXVI	LXXXVI	XCVI
7	VII	XVII	XXVII	XXXVII	XLVII	LVII	LXVII	LXXVII	LXXXVII	XCVII
8	VIII	XVIII	XXVIII	XXXVIII	XLVIII	LVIII	LXVIII	LXXVIII	LXXXVIII	XCVIII
9	IX	XIX	XXIX	XXXIX	XLIX	LIX	LXIX	LXXIX	LXXXIX	XCIX

Roman symbols

Non positional system

The act of computing

Computing = messing up with quantities

Symbolic computation

Positional systems with different bases	0,1,2,3,4,5,6,7,8,9	Base ten
	0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F	Base sixteen
	0,1	Base two

examples	thirteen	13	D	1101

Number = (rightmost-zero) digit * $base^{zero}$ + (rightmost-one) digit * $base^{one}$ + (rightmost-two) digit * $base^{two}$ +…

The act of computing

Computing = messing up with quantities

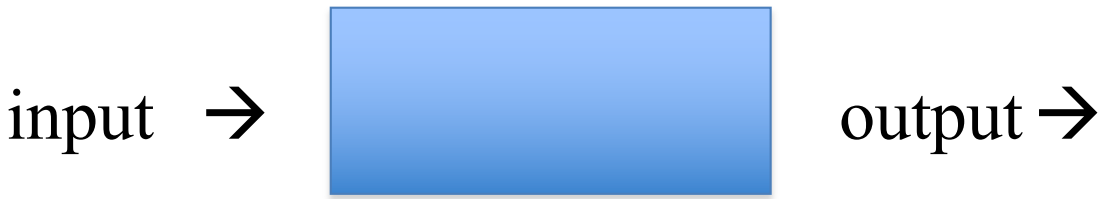
Computing = manipulating symbols that represent quantities according to certain rules

Positional systems vs non-positional systems have the advantage of simpler rules

Can we do this with a machine?

Computing device

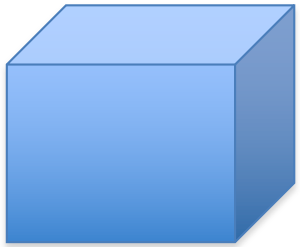
input – output model



Numerical data in the forms of symbols are transformed during computation

Physical systems as computing devices

A physical system that can be in a number of different physical states.



S0….SN

If we associate to each state a different symbolic value, by changing state we can operate (process) on the different symbols.

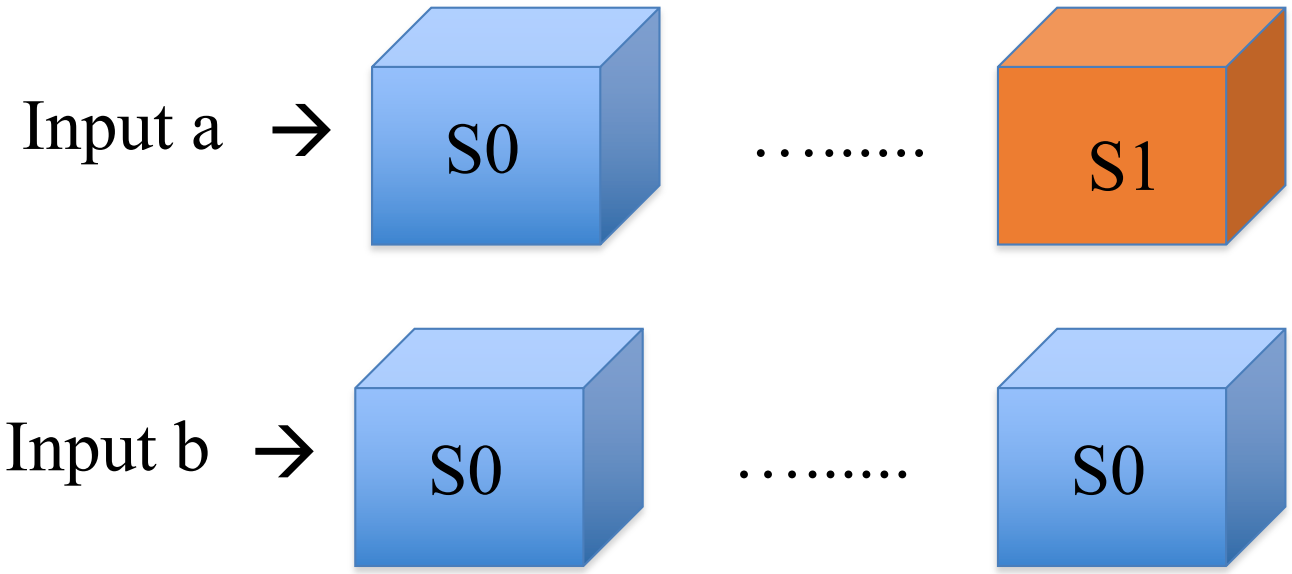
Physical systems as computing devices

In order to be able to produce an observable change in the system, the system has to admit at least 2 different states.

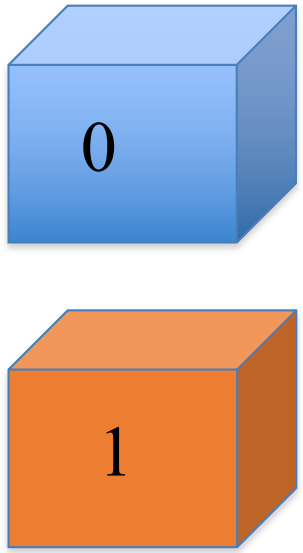
A computation is associated with system transformation. This requires a force acting on the system.

Physical systems as computing devices

The force represents the input and the state of the system the output.



A simple system to do computation



the physical system, made by a pebble* and two bowls.

a) The two states are represented by the measurable quantity “position of the pebble”: state “0” = pebble in left bowl; state “1” = pebble in right bowl;
b) the way to induce state changes represented by a force that brings around the pebble.

* “Calculus” is the Latin word for pebble

Base 2 systems

Binary representation of numerical quantities

A system that admits 2 distinct states is called a binary switch

Example:

Logic switches

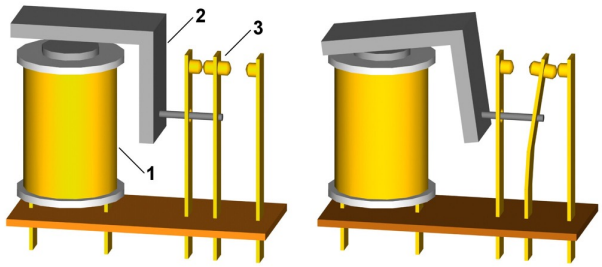
A *logic switch* is a device that can assume physically distinct states as a result of external inputs.

Usually the output of a physical system assume a continuous value (e.g. a voltage) and a threshold is used to partite the output space in two ore more states.

If the states are in the number of two we have binary logic switches.

Binary switches

There exist at least two classes of devices that can satisfy the rules a) and b). We call them ***combinational*** and ***sequential*** devices.

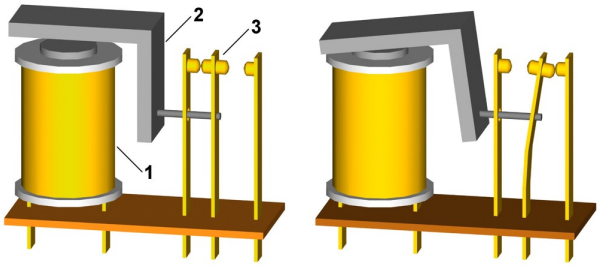


Conbinational:
in the absence of any external force, under equilibrium conditions, they are in the state S_0. When an external force F_{01} is applied, they switch to the state S_1 and remain in that state as long as the force is present. Once the force is removed they go back to the state S_0.

Sequential:
They can be changed from S_0 to S_1 by applying an external force F_{01}. Once they are in the state S_1 they remain in this state even when the force is removed. They go from S_1 to S_0 by applying a new force F_{10}. Once they are in S_0 they remain in this state even when the force is removed.

Computing with binary switches

Combinational switches can be easily employed do computation.
Sequential switches can be easily employed to store information.

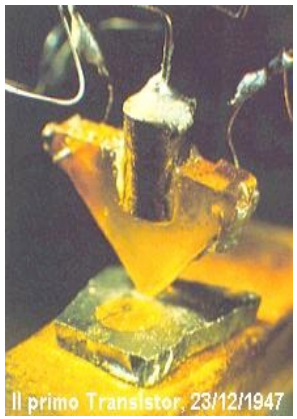


Conbinational:
in the absence of any external force, under equilibrium conditions, they are in the state S_0. When an external force F_{01} is applied, they switch to the state S_1 and remain in that state as long as the force is present. Once the force is removed they go back to the state S_0.

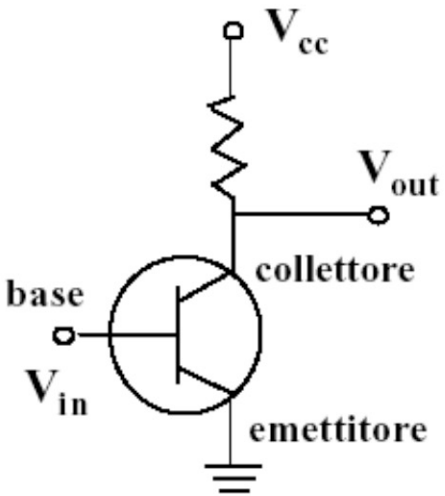
Sequential:
They can be changed from S_0 to S_1 by applying an external force F_{01}. Once they are in the state S_1 they remain in this state even when the force is removed. They go from S_1 to S_0 by applying a new force F_{10}. Once they are in S_0 they remain in this state even when the force is removed.

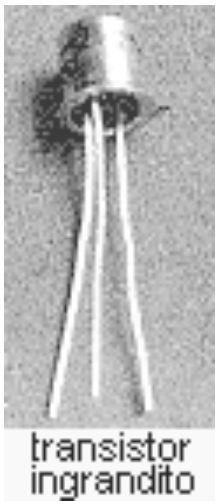
In modern computers binary switches are made with **transistors**. These are electronic devices that satisfy the two conditions required:

a) The two states are represented by the measurable quantity "electric voltage" at point V_{OUT}. As an example state "0" = $V_{OUT} < V_T$; state "1" = $V_{OUT} > V_T$; with V_T a given reference voltage.
b) The way to induce state changes represented by an electromotive force applied at point V_{IN}.



1944 (Bardeen, Brattley e Shotky)





If $V_{in} > 0.5V$ then $V_{out} = 0$ V
If $V_{in} < 0.5V$ then $V_{out} = 5$ V

Computing with binary switches

Binary switches can be used to perform computations.
In order to see this, let's start analysing elementary computations with binary numbers

Addition between two binary numbers:

$$\begin{array}{r} 10110101+ \\ 1000110 = \\ \hline 11111011 \end{array} \qquad \begin{array}{r} 00110011+ \\ 00111000 = \\ \hline 01101011 \end{array}$$

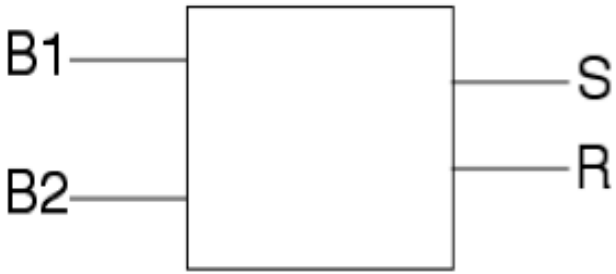
1 bit addition:

B1 +	0 +	0 +	1 +
B2 =	0 =	1 =	1 =
S	0	1	10

B1	B2	S	R
0	**0**	**0**	**0**
0	**1**	**1**	**0**
1	**0**	**1**	**0**
1	**1**	**0**	**1**

1 bit addition truth table

How to build an adder

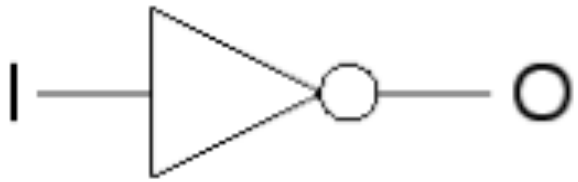


B1	B2	S	R
0	**0**	**0**	**0**
0	**1**	**1**	**0**
1	**0**	**1**	**0**
1	**1**	**0**	**1**

1 bit adder truth table

Addition is not the sole operation that we can do with bits.
Other than artithmetic operations, there are LOGIC operations

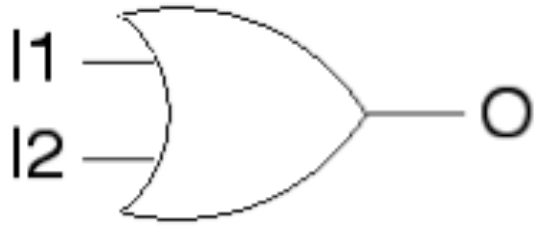
LOGIC operation NOT



Truth table

I	O
0	1
1	0

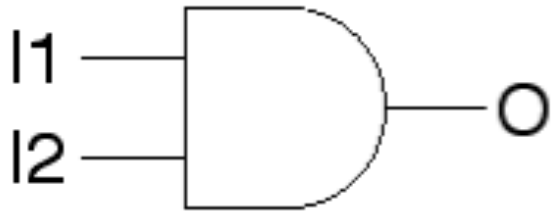
LOGIC operation OR



Truth table

I1	I2	O
0	**0**	**0**
0	**1**	**1**
1	**0**	**1**
1	**1**	**1**

LOGIC operation AND

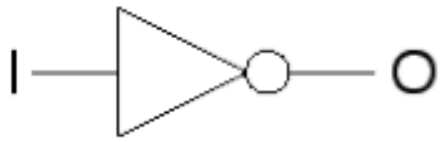


Truth table

I1	I2	O
0	0	0
0	1	0
1	0	0
1	1	1

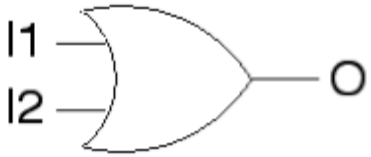
Boolean algebra: ….. George Boole (1815 - 1864)

NOT



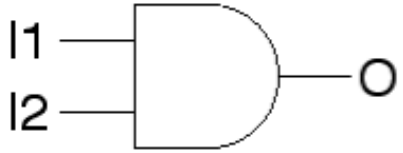
I	O
0	1
1	0

OR



I1	I2	O
0	0	0
0	1	1
1	0	1
1	1	1

AND



I1	I2	O
0	0	0
0	1	0
1	0	0
1	1	1

Let’s see some example….

NOT

I	O
0	**1**
1	**0**

OR

I1	I2	O
0	**0**	**0**
0	**1**	**1**
1	**0**	**1**
1	**1**	**1**

AND

I1	I2	O
0	**0**	**0**
0	**1**	**0**
1	**0**	**0**
1	**1**	**1**

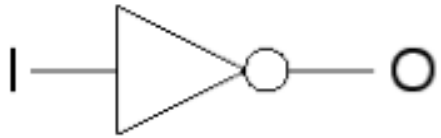
IF I1 = 1 ------> NOT (I1) = 0 IF I1 = 0 ------> NOT (I1) = 1

IF I1 = 0 e I2 = 1 ------> I1 AND I2 = 0

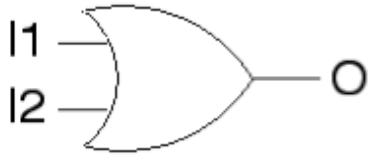
IF I1 = 1 e I2 = 1 ------> I1 AND I2 = 1

IF I1 = 0 e I2 = 1 ------> I1 OR I2 = 1

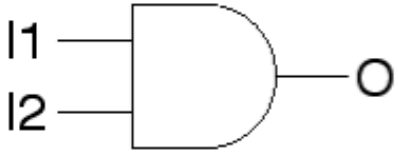
NOT



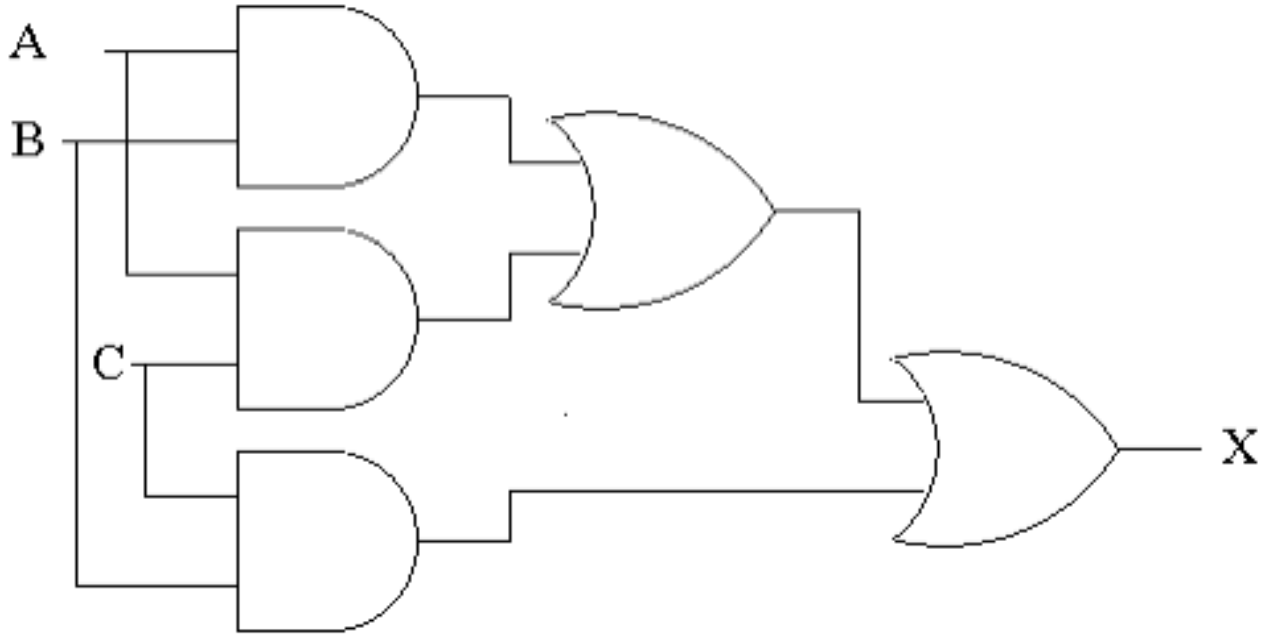
OR



AND



Combinational nets



Problem: given a truth table how can you build the logic expression that realizes it?

There are different techniques….

Exemple: half adder

B1	B2	S	R
0	**0**	**0**	**0**
0	**1**	**1**	**0**
1	**0**	**1**	**0**
1	**1**	**0**	**1**

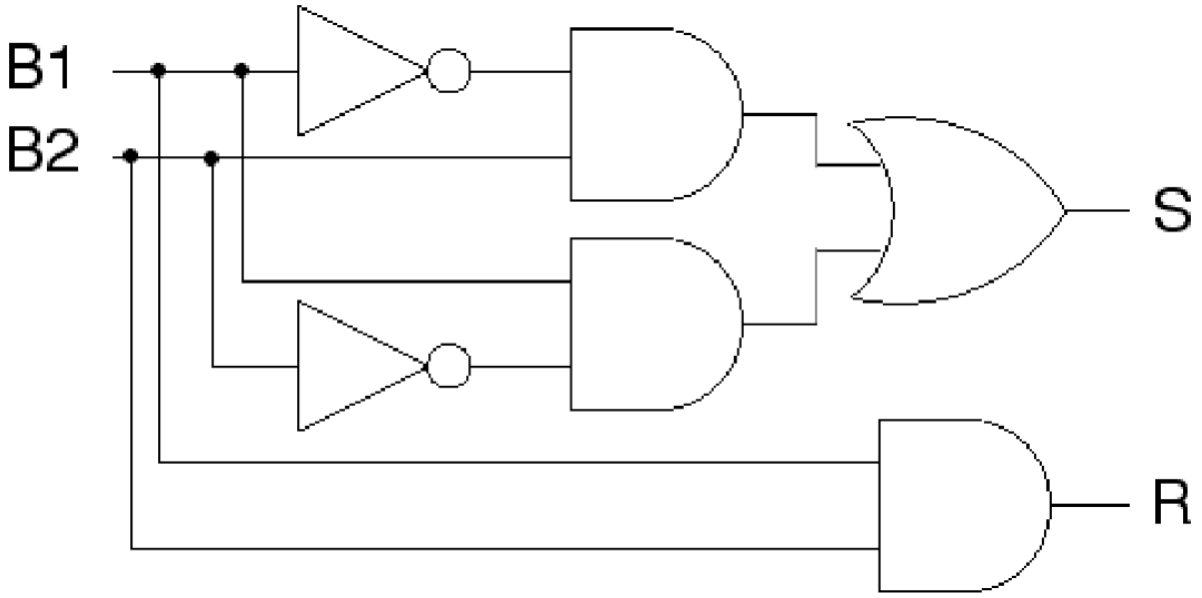
Sum

$S = (B1' \cdot B2) + (B1 \cdot B2')$

Carry…..

$R = B1 \cdot B2$

B1	B2	S	R
0	**0**	**0**	**0**
0	**1**	**1**	**0**
1	**0**	**1**	**0**
1	**1**	**0**	**1**



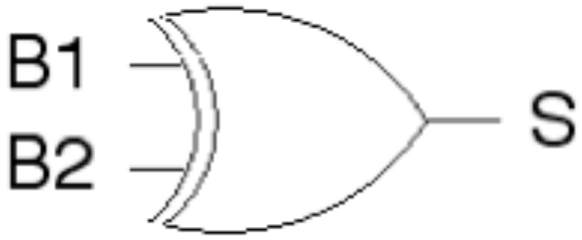
Half adder

$$S = (B1' \cdot B2) + (B1 \cdot B2')$$

A new logic gate:

XOR = $B1 \oplus B2$

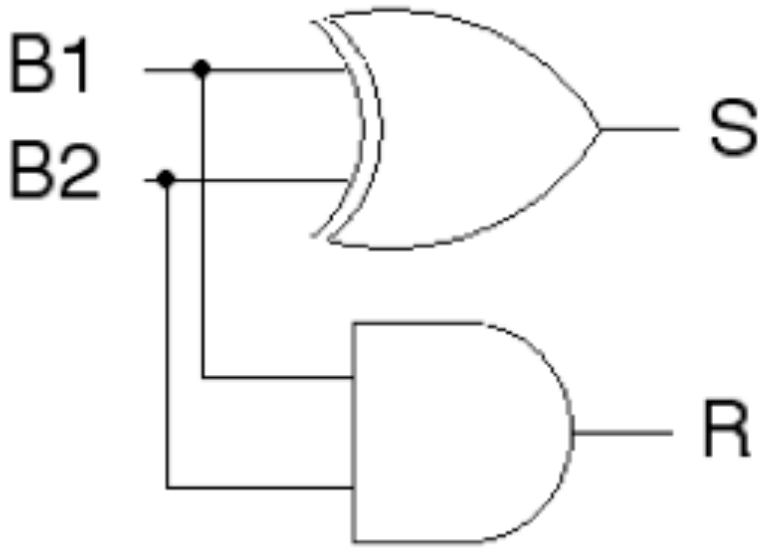
$$S = B1 \oplus B2 = (B1' \bullet B2) + (B1 \bullet B2')$$



Half adder

$S = B1 \oplus B2$

$R = B1 \cdot B2$



Full-adder

S = …?

R = …?

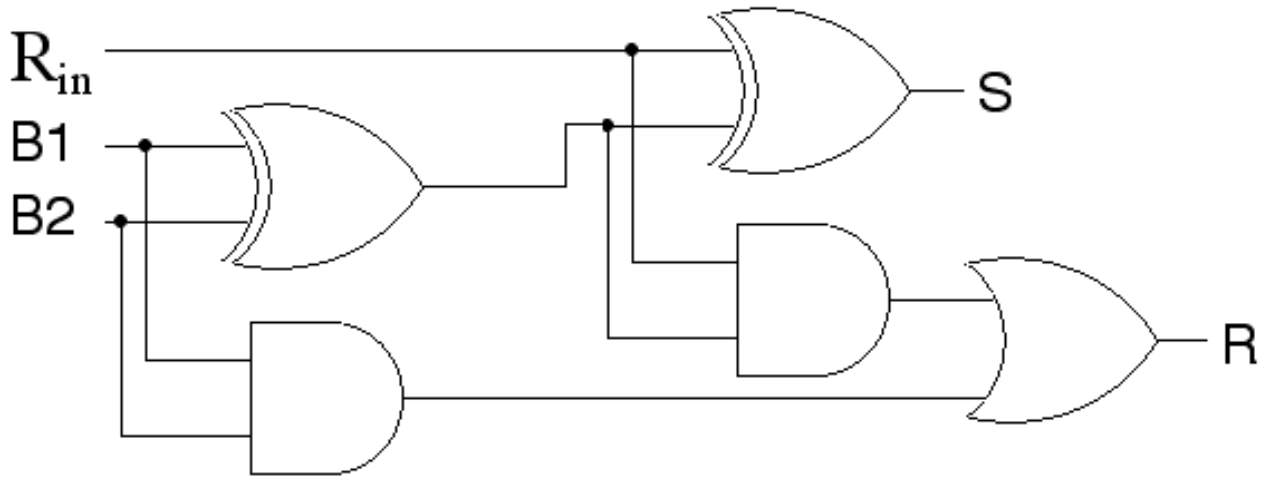
R_{in}	B1	B2	S	R
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$S = R_{in}'(B1' \cdot B2) + R_{in}'(B1 \cdot B2') + R_{in}(B1' \cdot B2') + R_{in}(B1 \cdot B2)$

$R = R_{in}'(B1' \cdot B2) + R_{in}(B1' \cdot B2) + R_{in}(B1 \cdot B2') + R_{in}(B1 \cdot B2)$

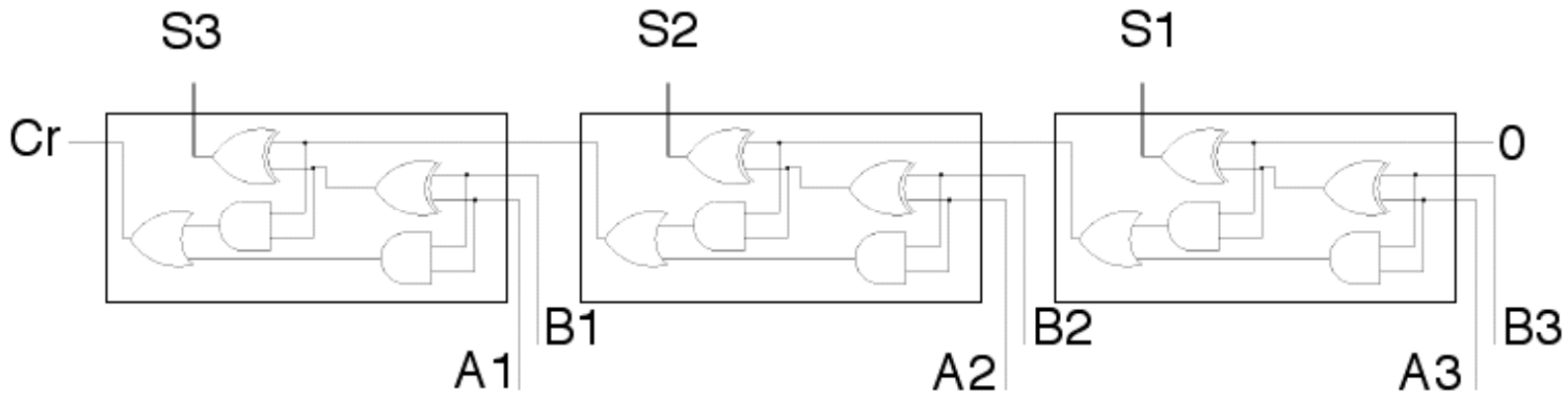
Full Adder:

$$S = R_{in} \oplus (B1 \oplus B2)$$
$$R = B1B2 + R_{in}(B1 \oplus B2)$$



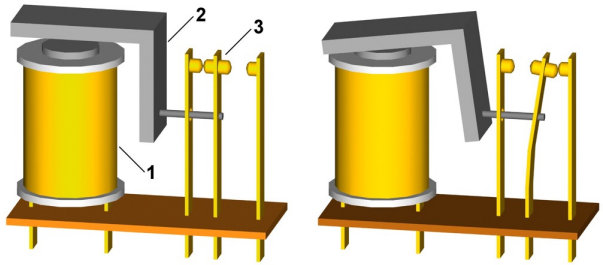
More than one bit

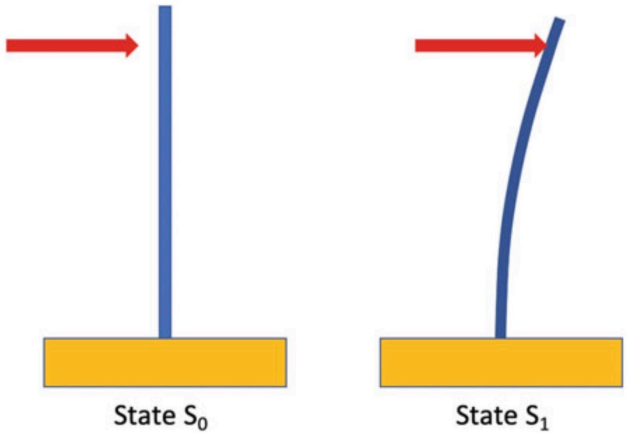
A1A2A3 + B1B2B3 (e.g. 011 + 100)



Back to binary switches

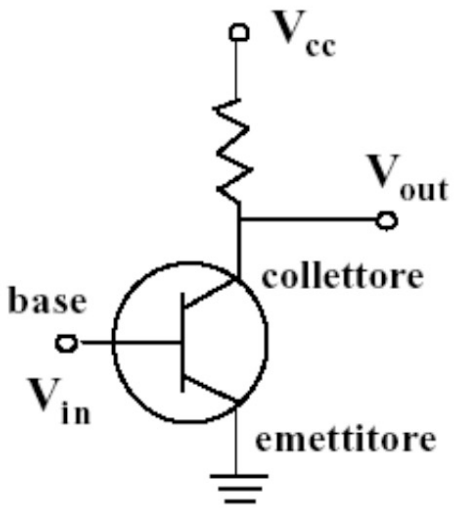
We can use combinational binary switches to make all the logic gates that we need.





Conbinational switch:
in the absence of any external force, under equilibrium conditions, they are in the state S_0. When an external force F_{01} is applied, they switch to the state S_1 and remain in that state as long as the force is present. Once the force is removed they go back to the state S_0.

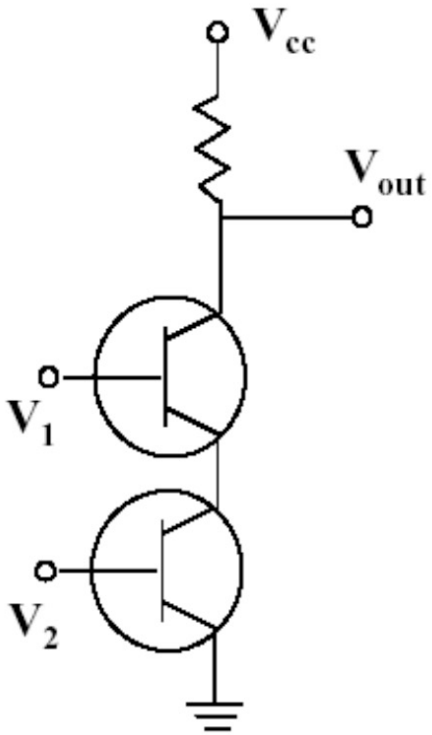
As an example let's cosider the transistor as a combinational binary switch and see how we can make logic gates by combining them



This is a NOT gate:

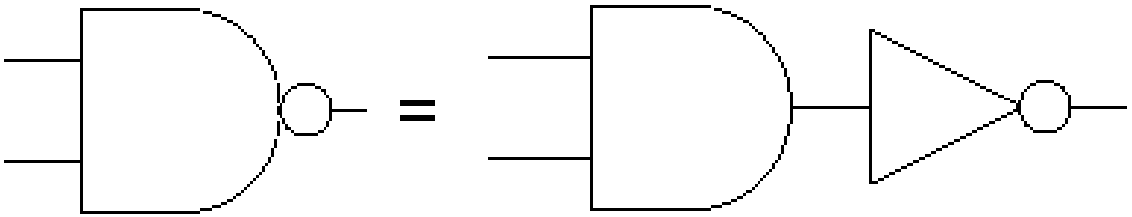
if $V_{in} > 0.5V$ (Logic Input = 1) then $V_{out} = 0$ V (Logic Output = 0)
if $V_{in} < 0.5V$ (Logic Input=0) then $V_{out} = 5$ V (Logic Output=1)

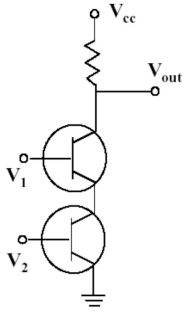
Let's see if we combine two of them

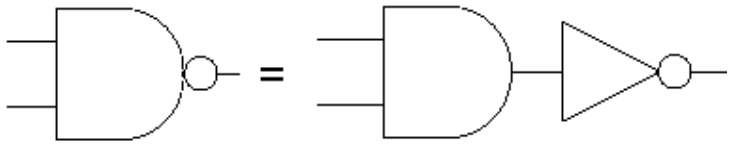


V_1	V_2	V_{out}
0	0	1
0	1	1
1	0	1
1	1	0

NAND gate:



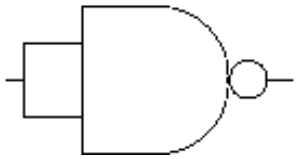




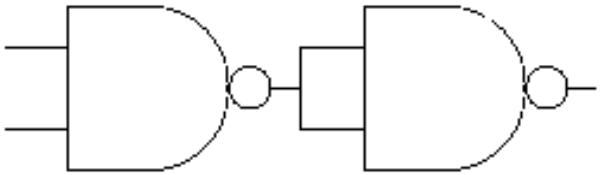
V_1	V_2	V_{out}
0	0	1
0	1	1
1	0	1
1	1	0

We can make all the gates by combining NANDS

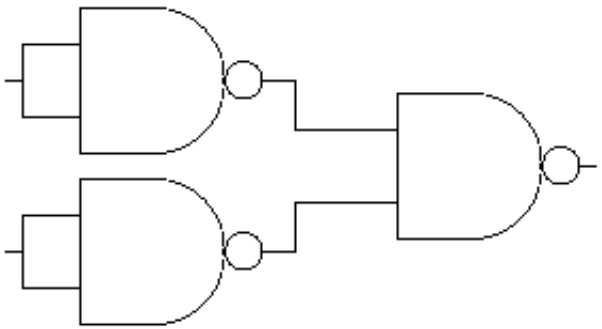
NOT gate:



AND gate:



OR gate:



Check ...

Summary

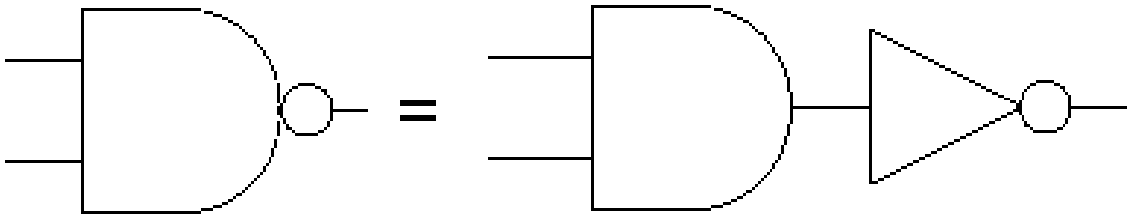
A *binary switch* is a device that:

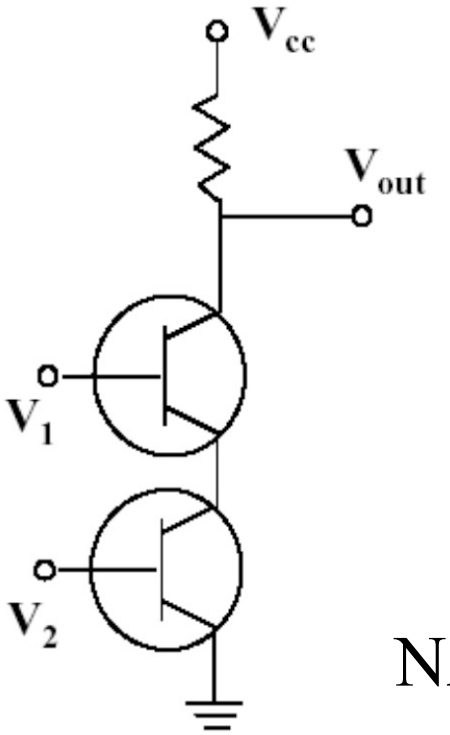
a)can assume two physically distinct states.

b)Can change state under the action of an external "force".

By combining 2 (combinatory) binary switches we can make a universal gate and, thus, any computing device.

NAND gate:



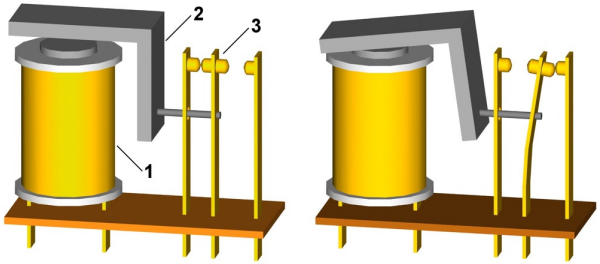


NAND gate made with transistors

Is that all?

What about memories?

Binary switches

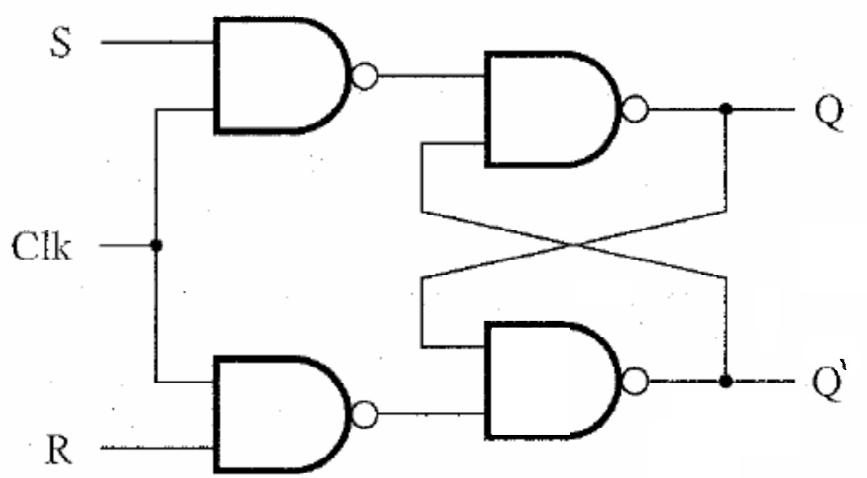


Conbinational:
in the absence of any external force, under equilibrium conditions, they are in the state S_0. When an external force F_{01} is applied, they switch to the state S_1 and remain in that state as long as the force is present. Once the force is removed they go back to the state S_0.

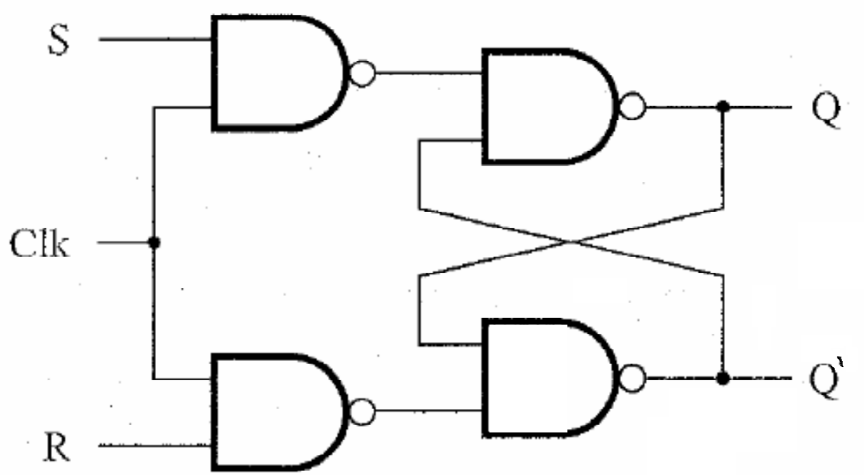
Sequential:
They can be changed from S_0 to S_1 by applying an external force F_{01}. Once they are in the state S_1 they remain in this state even when the force is removed. They go from S_1 to S_0 by applying a new force F_{10}. Once they are in S_0 they remain in this state even when the force is removed.

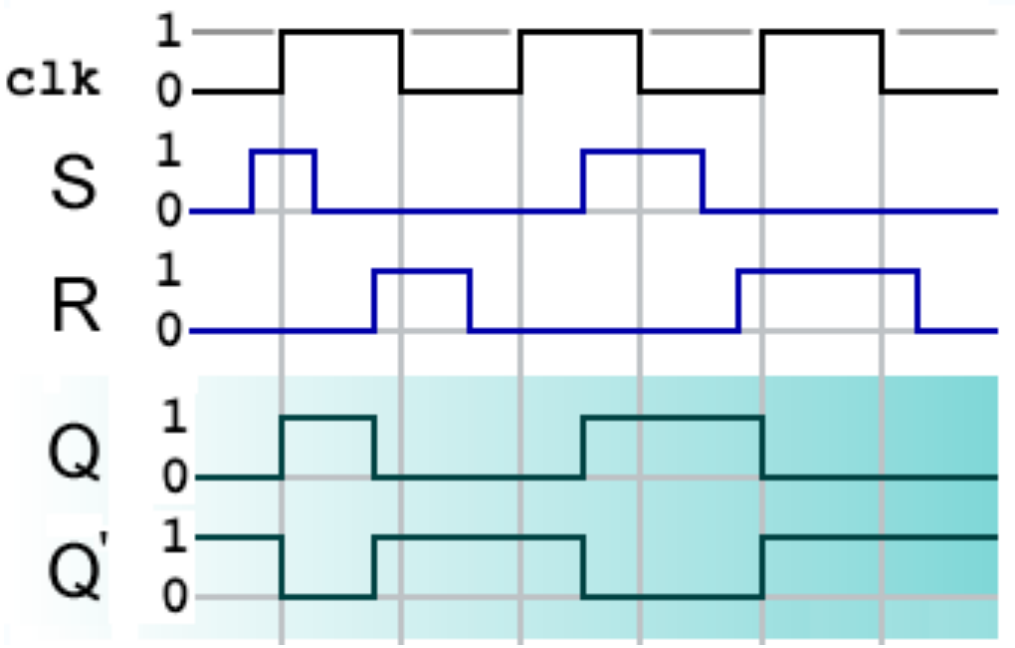
We can make memories also by combining NAND gates

We can use a “dirty trick”: the feedback



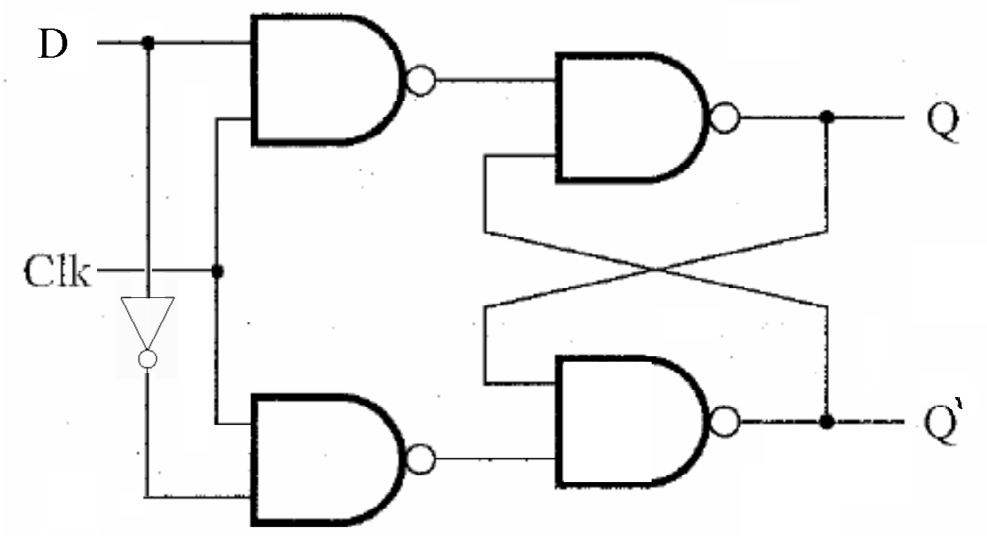
Flip-Flop SR





Flip-Flop SR

Clk	S	R	Q	Q’
0	x	x	Valore pre-esistente	Valore pre-esistente
1	1	0	1	0
1	0	1	0	1
1	1	1	X	x



Flip-Flop D type

Summary

We have seen that computing is messing up with quantities and that this can be done by manipulating symbols that represents these quantities according to some given rules.

We have seen that there is some convenience in using binary representation that employs only two symbols (bits).

We have seen that we can associate to each symbol a measurable state of a two state physical system, so that by acting on the physical system we can change its state.

The most elementary physical devices that realize this condition are called binary switches.

By combining binary switches we can do all the logic, arithmetic and memory operation of a computer.

To know more

Book The Physics of Computing, L. Gammaitoni, Springer, 2021. Chap. 2