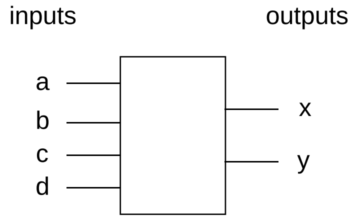
Basic Combinational Circuit

- A *combinational circuit*
  - Maps a set of inputs to a set of outputs
    - These inputs / outputs are called *signals*
    - Each signal has the value 1 or 0



  - Whenever the input values change, the output values change after a short delay (called the *propagation delay*)
  - Is purely functional
    - Output values depend only on inputs
    - There is no concept of *state* in a combinational circuit

Truth Tables

- A truth table specifies *what* a logic circuit does, but not *how* it does it
  - It lists, for every possible combination of input values, the output value(s)

a	b	c	d	x
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

- Our goal is to build the digital circuit that implements this truth table

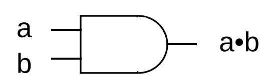
Boolean Operations

- In Boolean algebra, variables assume one of two logical values:
  - True = 1
  - False = 0
- Boolean algebra has three basic operators:
  - *and*, denoted as • (sometimes omitted)
  - *or*, denoted as +
  - *not* (or *complement*, or *inversion*), denoted as ' or ‾
- A *gate* is an electronic device that implements a simple Boolean operation
  - Gates are the basic building blocks of a digital circuit

4 Basic Boolean Operations & Gates

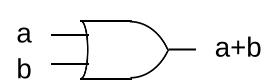
and

a	b	a•b (or ab)
0	0	0
0	1	0
1	0	0
1	1	1



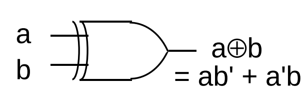
or

a	b	a+b
0	0	0
0	1	1
1	0	1
1	1	1



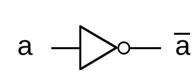
xor

a	b	a⊕b
0	0	0
0	1	1
1	0	1
1	1	0



not

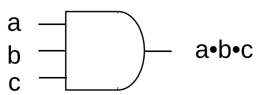
a	$\overline{a}$ (or a')
0	1
1	0



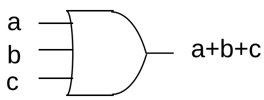
Generalized Gates

- These gates can also be generalized to an arbitrary number of inputs

a	b	c	a•b•c
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



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

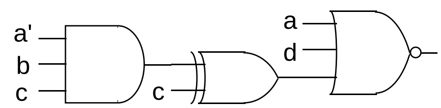


Building Combinational Circuits

- Consider the 4-input Boolean expression:

  z = ((a'bc ⊕ c) + a + d)'

- The equivalent circuit (logic) diagram would be:



Conventions in drawing logic diagrams:

- Inputs go on the left, outputs on the right
- Simplify inputs and outputs: don’t show inverters
- Signal lines are drawn in rectangular fashion (i.e., no diagonals, no curves)
- It’s OK for signal lines to cross; but if signal lines merge / split, indicate the junction with a black dot

Boolean Algebra

- The 10 fundamental properties are:
  - *commutative*

    $x + y = y + x$ $\qquad$ $x \bullet y = y \bullet x$
  - *associative*

    $x + (y + z) = (x + y) + z$

    $x \bullet (y \bullet z) = (x \bullet y) \bullet z$
  - *distributive*

    $x \bullet (y + z) = (x \bullet y) + (x \bullet z)$

    $x + (y \bullet z) = (x + y) \bullet (x + z)$
  - *identity*

    $x + 0 = x$ $\qquad$ $x \bullet 1 = x$
  - *complement*

    $x + x' = 1$ $\qquad$ $x \bullet x' = 0$
- Each property has a *dual* property, created by:
  - Exchanging + and •
  - Exchanging 1 and 0

Boolean Algebra (cont.)

- These postulates can be used to prove a number of useful theorems:
  - *idempotency*

    $x + x = x$

    $x \bullet x = x$
  - *null*

    $x + 1 = 1$

    $x \bullet 0 = 0$
  - *absorption*

    $x + (x \bullet y) = x$

    $x \bullet (x + y) = x$
  - *De Morgan’s law*

    $(x + y)' = x' \bullet y'$

    $(x \bullet y)' = x' + y'$
  - *other complement theorems*

    $(x')' = x$ $\qquad$ *involution*

    $1' = 0$

    $0' = 1$

2-Level Circuits

- ***Every*** Boolean expression can be written as ***a sum of products***, and as a ***product of sums***, and implemented as a 2-level circuit

$z = ((a'bc \oplus c) + a + d)'$	given
$= (a'bcc' + (a'bc)'c + a + d)'$	defn of $\oplus$
$= ((a'bc)'c + a + d)'$	complem.
$= ((a+b'+c')c + a + d)'$	De Morg.
$= (ac + b'c + a + d)'$	distributive
$= (ac)'(b'c)'a'd'$	De Morg.
$= (a'+c')(b+c')a'd'$	De Morg.
$= (a'+a'c')(b+c')d'$	distributive
$= (a'b + a'c' + a'bc' + a'c')d'$	distributive
$= (a'b + a'c' + a'bc')d'$	x+x=x
$= a'bd' + a'c'd' + a'bc'd'$	distributive

(this result is sum of products form)

Homework #1 — Due 9/14/98 (Part 3/3)

6. Draw the combinational circuit that directly implements the Boolean expression $a = bc'd + (b' + d)(cd')$

(This is the last question on Homework #1)