Extension to Multiple Inputs

- All the gates shown except for the inverter and buffer can be extended to have more than two inputs.
- A gate can be extended to have multiple inputs if the binary operation it represents is commutative and associative.
- The AND and OR gates
  - Has both commutative and associative property
- The exclusive-OR and equivalence gates
  - are both commutative and associative and can be extended to more than two inputs.
  - multiple-input exclusive-OR gates are uncommon from the hardware standpoint.
    - In fact, even a two-input function is usually constructed with other types of gates.
  - when extended to more than two variables, exclusive-OR is an *odd function*(i.e., it is equal to 1 if the input variables have an odd number of 1’s).

Extension to Multiple Inputs

- The NAND and NOR gates
  - are commutative, and their gates can be extended to have more than two inputs, provided that the definition of the operation is modified slightly.
  - NAND and NOR operators are not associative

$$(x \downarrow y) \downarrow z \neq x \downarrow (y \downarrow z)$$

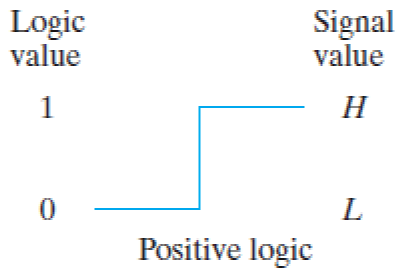
$$(x \downarrow y) \downarrow z = [(x + y)' + z]' = (x + y)z' = xz' + yz'$$
$$x \downarrow (y \downarrow z) = [x + (y + z)']' = x'(y + z) = x'y + x'z$$

  - To overcome this difficulty, we define the multiple NOR (or NAND) gate as a complemented OR (or AND) gate.

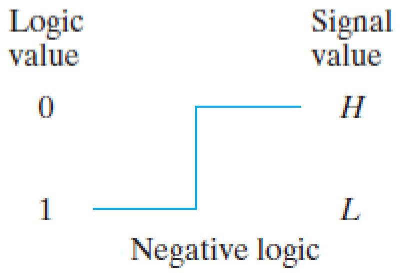
$$x \downarrow y \downarrow z = (x + y + z)'$$
$$x \uparrow y \uparrow z = (xyz)'$$

Positive and Negative Logic

- The binary signal at the inputs and outputs of any gate has one of two values, except during transition.
  - One signal value represents logic 1 and the other logic 0.
- **Choosing the high-level *H to represent logic 1 defines* a positive logic system.**
- **Choosing the low-level *L to represent logic 1 defines a negative* logic system.**
- It is not the actual values of the signals that determine the type of logic, but rather the assignment of logic values to the relative amplitudes of the two signal levels.
  - It is up to the user to decide on a positive or negative logic polarity.



Positive logic



Negative logic

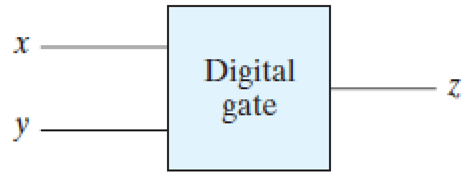
Positive and Negative Logic

- Example:
  - Consider a physical gate that specifies the physical behavior of the gate when *H is 3 V and L is 0 V*
- positive logic assignment, with *H = 1 and L = 0.*
  - *This truth* table is the same as the one for the AND operation.
- negative logic assignment with *L = 1* and *H = 0.*
  - This truth table represents the OR operation, even though the entries are reversed
- The same physical gate can operate either as a positive-logic AND gate or as a negative-logic OR gate.

- (The small triangles in the inputs and output designate a *polarity indicator, the presence of which along a terminal signifies that* negative logic is assumed for the signal)

x	y	z
L	L	L
L	H	L
H	L	L
H	H	H

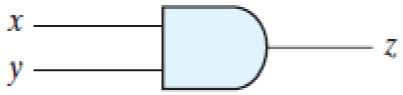
Truth table with H and L



Gate block diagram

x	y	z
0	0	0
0	1	0
1	0	0
1	1	1

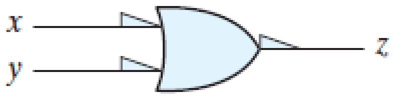
Truth table for positive logic



Positive logic AND gate

x	y	z
1	1	1
1	0	1
0	1	1
0	0	0

Truth table for negative logic



Negative logic OR gate

Positive and Negative Logic

- The conversion from positive logic to negative logic and vice versa is an operation that changes 1's to 0's and 0's to 1's in both the inputs and the output of a gate
- Since this operation produces the dual of a function, the change of all terminals from one polarity to the other results in taking the dual of the function.
- The AND operations are converted to OR operations (or graphic symbols) and vice versa.
- In addition, must not forget to include the polarity-indicator triangle in the graphic symbols when negative logic is assumed.

Gate level minimization

- *Gate-level minimization is the design task of finding an optimal gate-level implementation* of the Boolean functions describing a digital circuit.
- Methods:
  - **THE MAP METHOD**
  - **TABULATION METHOD**

Algebraic Simplification !!

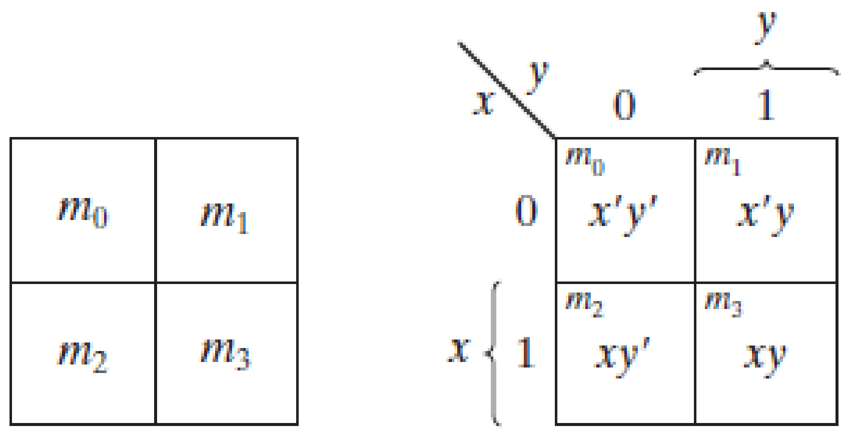
- Although the truth table representation of a function is unique, when it is expressed algebraically it can appear in many different, but equivalent forms.
- Boolean expressions may be simplified by algebraic means.
- This procedure of minimization is awkward because it lacks specific rules to predict each succeeding step in the manipulative process.

THE MAP METHOD/ *Karnaugh map or K-map .*

- The map method provides a simple, straightforward procedure for minimizing Boolean functions.
- This method may be regarded as a pictorial form of a truth table.
- The map method is also known as the *Karnaugh map or K-map .*
- A K-map is a diagram made up of squares, with each square representing one minterm of the function that is to be minimized.
- The map presents a visual diagram of all possible ways a function may be expressed in standard form.
- By recognizing various patterns, the user can derive alternative algebraic expressions for the same function, from which the simplest can be selected.
- The simplified expressions produced by the map are always in one of the two standard forms:
  - sum of products
  - product of sums.
- It will be assumed that the simplest algebraic expression is an algebraic expression with a minimum number of terms and with the smallest possible number of literals in each term.
- This expression produces a circuit diagram with a minimum number of gates and the minimum number of inputs to each gate.
- the simplest expression is not always unique:
  - It is sometimes possible to find two or more expressions that satisfy the minimization criteria.
  - In that case, either solution is satisfactory.

Two-Variable K-Map

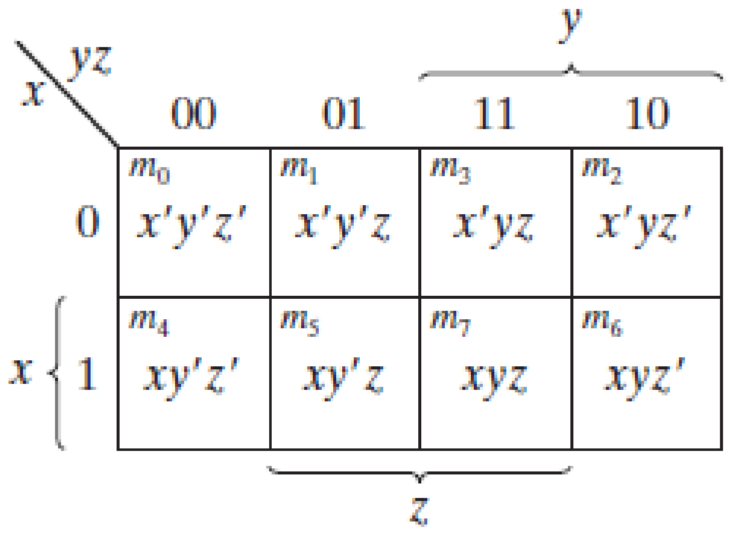
- There are four minterms for two variables.
- The map consists of four squares, one for each minterm.



Three-Variable K-Map

- There are eight minterms for three binary variables
- The map consists of eight squares.
- Minterms are arranged, not in a binary sequence, but in a sequence similar to the Gray code.
  - characteristic of this sequence is that **only one bit changes in value from one adjacent column to the next.**

m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6

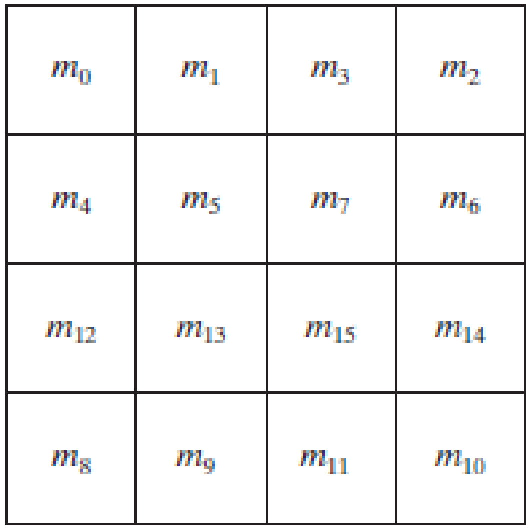


Three-Variable K-Map

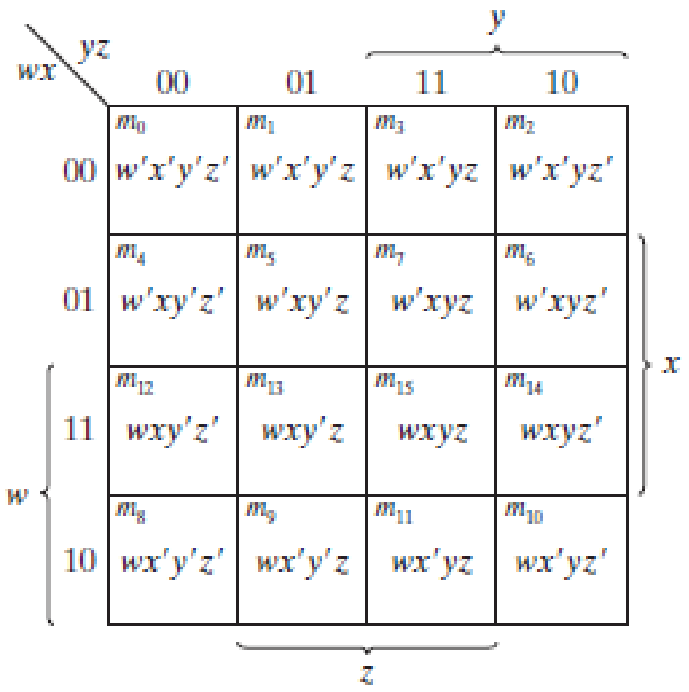
- As more adjacent squares are combined, we obtain a product term with fewer literals.
  - One square represents one minterm, giving a term with three literals.
  - Two adjacent squares represent a term with two literals.
  - Four adjacent squares represent a term with one literal.
  - Eight adjacent squares encompass the entire map and produce a function that is always equal to 1.

FOUR-VARIABLE K-MAP

- There are sixteen minterms for four binary variables
- The map consists of sixteen squares
- The rows and columns are numbered in a Gray code sequence, with only one digit changing value between two adjacent rows or columns.
- The minterm corresponding to each square can be obtained from the concatenation of the row number with the column number.



m_0 m_1 m_3 m_2
m_4 m_5 m_7 m_6
m_{12} m_{13} m_{15} m_{14}
m_8 m_9 m_{11} m_{10}



yz
wx
y
00 01 11 10
00: m_0 $w'x'y'z'$ | m_1 $w'x'y'z$ | m_3 $w'x'yz$ | m_2 $w'x'yz'$
01: m_4 $w'xy'z'$ | m_5 $w'xy'z$ | m_7 $w'xyz$ | m_6 $w'xyz'$
11: m_{12} $wxy'z'$ | m_{13} $wxy'z$ | m_{15} $wxyz$ | m_{14} $wxyz'$
10: m_8 $wx'y'z'$ | m_9 $wx'y'z$ | m_{11} $wx'yz$ | m_{10} $wx'yz'$
x
w
z

FOUR-VARIABLE K-MAP

- The combination of adjacent squares that is useful during the simplification process is easily determined from inspection of the four-variable map:
  - One square represents one minterm, giving a term with four literals.
  - Two adjacent squares represent a term with three literals.
  - Four adjacent squares represent a term with two literals.
  - Eight adjacent squares represent a term with one literal.
  - Sixteen adjacent squares produce a function that is always equal to 1.