COMP2611: Computer Organization

Introduction to Digital Logic

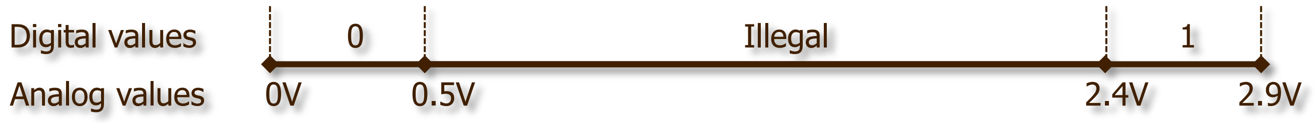
Combinational Logic

Basics of Logic Circuits

- **Bits** are the basis for **binary number representation** in digital computers
- **Combining bits** into patterns **following some conventions or rules (defined in the ISA)** allow for:
  - number representations
    - Integers,
    - Fractions and Real numbers, …
  - Instruction encoding
    - Operation
    - Operands
- How are bits represented at the low level and how are they handled in the hardware below the ISA?

Basic Operations on Bits

- The electronics inside modern computers are **digital**: they operate with only two voltage levels of interest - hence the use of **binary** numbers



- Two types of digital logic circuits inside a computer:
  - **Combinational logic circuits**:
    - Logic circuits that do not have memory.
    - The output depends only on the current input and the circuit.
  - **Sequential logic circuits**:
    - Logic circuits that have memory.
    - The output depends on both the current input and the value stored in memory (called **state**).
- Both rely on some basic logic circuits that implement some fundamental logic operations

Logical Operations on Bits

- Three fundamental logic functions defined by their truth table are at the center of all operations in modern computers:
  - **NOT**

Input values / Output values

Input	Output
A	NOT A
0	1
1	0

also written $\overline{A}$

  - **AND**

A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

also written $A \cdot B$

Logical Operations on Bits

- **OR**

A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

also written $A + B$

- **NOT, AND and OR can be applied to bit patterns** as well:
  - The rule applies to each bit position: **bit-wise operation**
  - **Example:** A=01011, B= 10010

$\bar{A}$ = 10100

$A + B$ = 11011

$A \cdot B$ = 00010

Logical Operations on Bits

- To simplify the design of circuits other logic functions can be defined based on the basic ones, notably the XOR (exclusive or)

  - **XOR (exclusive OR)**

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

also written
$A \oplus B$

  - All logic functions can be extended to apply on more than two operands following the basic rules of **Boolean Algebra**

Logic Function

- **Example**:
  - Consider a digital circuit with two single-bit inputs (A, B) and two single-bit outputs (C, D). The circuit implements:
    - C is true if exactly one input is true;
    - D is true if exactly two inputs are true.
  - Construct the truth table for the circuit
  - Construct the logic function for the circuit

A	B	D	C
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$$C = A \oplus B = A \cdot \overline{B} + \overline{A} \cdot B$$

$$D = A \cdot B$$

Computer Arithmetic and Logic operation can be specified via logic functions

Basic Laws of Boolean Algebra

Using all these laws, we can simplify logic and arithmetic functions

- **Identity laws**:

$$A + 0 = A \qquad A \cdot 1 = A$$

- **Annihilator (or Zero and one) laws**:

$$A + 1 = 1 \qquad A \cdot 0 = 0$$

- **Complement laws**:

$$A + \overline{A} = 1 \qquad A \cdot \overline{A} = 0$$

- **Commutativity laws**:

$$A + B = B + A \qquad A \cdot B = B \cdot A$$

- **Associativity laws**:

$$A + (B + C) = (A + B) + C \qquad A \cdot (B \cdot C) = (A \cdot B) \cdot C$$

- **Distributivity laws**:

$$A \cdot (B + C) = (A \cdot B) + (A \cdot C) \qquad A + (B \cdot C) = (A + B) \cdot (A + C)$$

Additional Laws of Boolean Algebra

- **Idempotence**:

$$A + A = A \qquad A \cdot A = A$$

- **Absorption laws**:

$$A + (A \cdot B) = A \qquad A \cdot (A + B) = A$$

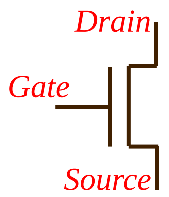
- **De Morgan Laws**:

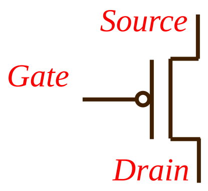
$$\overline{A + B} = \overline{A} \cdot \overline{B}$$

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

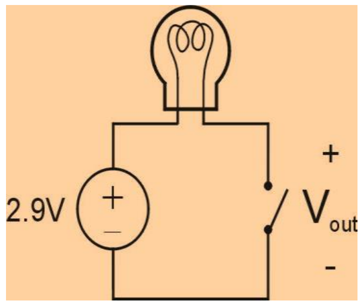
Digital Logic

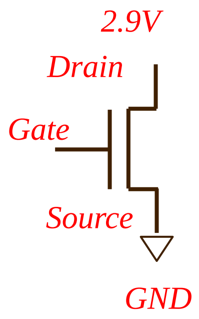
- At the base of modern computers is the MOS (Metal Oxide Semiconductor) Transistor
- Transistor:
  - Three terminals: Gate, Source and Drain
  - Two types of transistors N (left) and P(right)

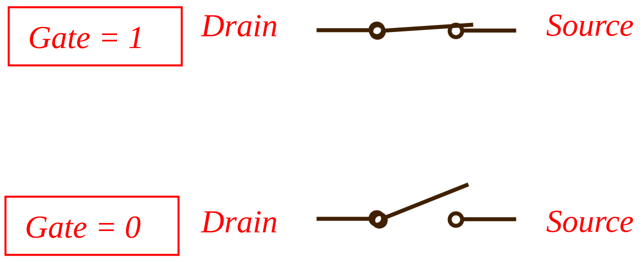




  - Operates as an Electronic switch
  - For the N-type: if the Gate is powered, then the path from source to drain acts like a wire otherwise the path is broken.







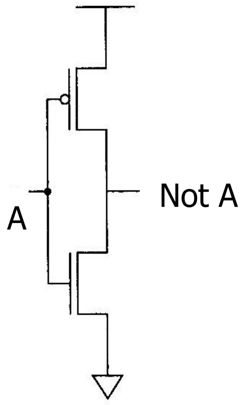
Logic gates

- We can build basic logic circuits (logic gates) for the three basic logic functions (NOT, AND, OR) by using several transistors

- **NOT Gate** (CMOS Inverter)

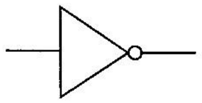
A	NOT A
0	1
1	0

Truth Table



Digital Inverter Using CMOS

A	NOT A
0V	2.9V
2.9V	0V



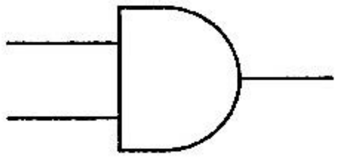
Symbolic representation of the NOT Gate

Logic Gates

- **AND Gate**

A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

Truth Table

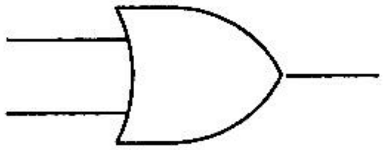


Symbolic representation of the AND Gate

- **OR Gate**

A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

Truth Table



Symbolic representation of the OR Gate

Construct a 1-bit Half Adder

- Continue the example on Page 7

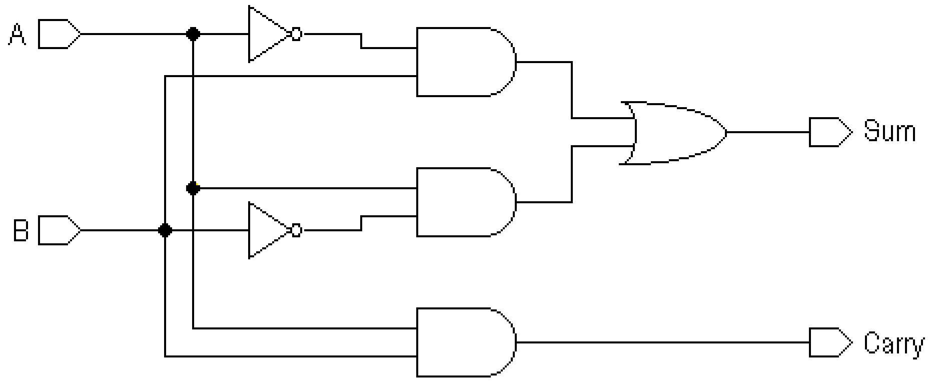
A	B	D	C
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Truth Table

$$C = A \oplus B = A \cdot \overline{B} + \overline{A} \cdot B$$

$$D = A \cdot B$$

Logic Function



Digital Circuit

Combinational Logic

- **Combinational logic circuits**:
  - Logic circuits that do not have memory.
  - The output depends only on the current input.
  - They can be specified fully with a truth table or a logic equation

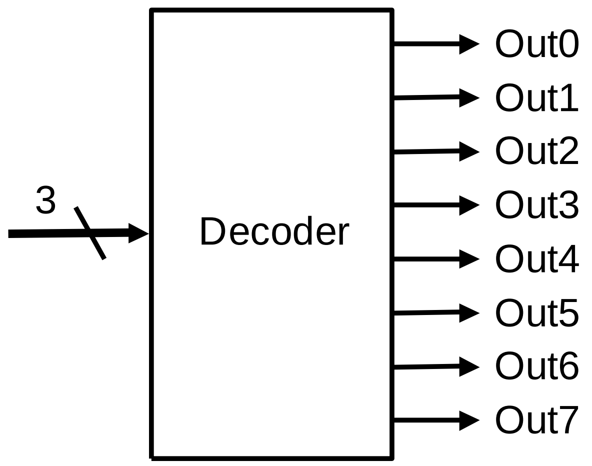
- Other than logic gates that are the most basic building blocks, there also exist some **higher-level basic building blocks** that are also commonly used:
  - **Decoders/encoders**
  - **Multiplexors**
  - **Two-level logic** and **PLAs**

- These building blocks can be implemented using AND, OR, and NOT gates only.

Decoder

- A **decoder** (**N-to-2^N decoder**) is a logical block with an N-bit input and 2^N 1-bit outputs. The output that corresponds to the input bit pattern is true while all other outputs are false.
- **Example** (3-to-8 decoder):

Inputs			Outputs							
X	Y	Z	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1



a. A 3-bit decoder

- An **encoder** performs the inverse function of a decoder, taking 2^N inputs and producing an N-bit output.

Multiplexor

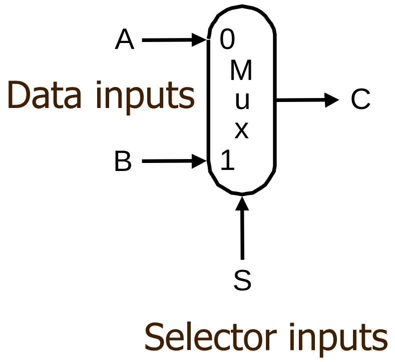
- A **multiplexor** (or **selector**) selects one of the data inputs as output by a control input value.

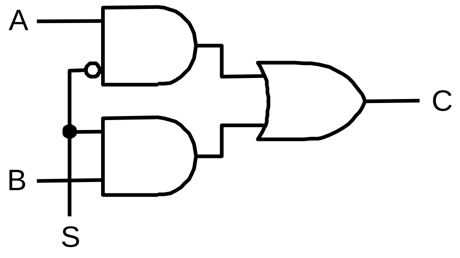
- A multiplexor can have an arbitrary number of data inputs:
  - Two data inputs require one selector input.
  - N data inputs require $\lceil \log_2 N \rceil$ selector inputs.

- **Example** (2-input multiplexor):

S	A	B	C
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

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





Two-Level Logic

- Any logic function can be expressed in a canonical form as a **two-level representation**:
  - Every input is either a variable or its negated form.
  - One level consists of **AND** gates only.
  - The other level consists of **OR** gates only.

- **Sum-of-products representation**:
  - E.g., $E = (A \cdot B \cdot \overline{C}) + (A \cdot C \cdot \overline{B}) + (B \cdot C \cdot \overline{A})$
  - More commonly used than product-of-sums representation.

- **Product-of-sums representation**:
  - E.g., $E = (\overline{A} + \overline{B} + C) \cdot (\overline{A} + \overline{C} + B) \cdot (\overline{B} + \overline{C} + A)$

Example

- Show the sum-of-products representation for the following truth table:

Inputs			Output
A	B	C	D
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

Only those table entries for which the output is 1 generate corresponding terms in the equation

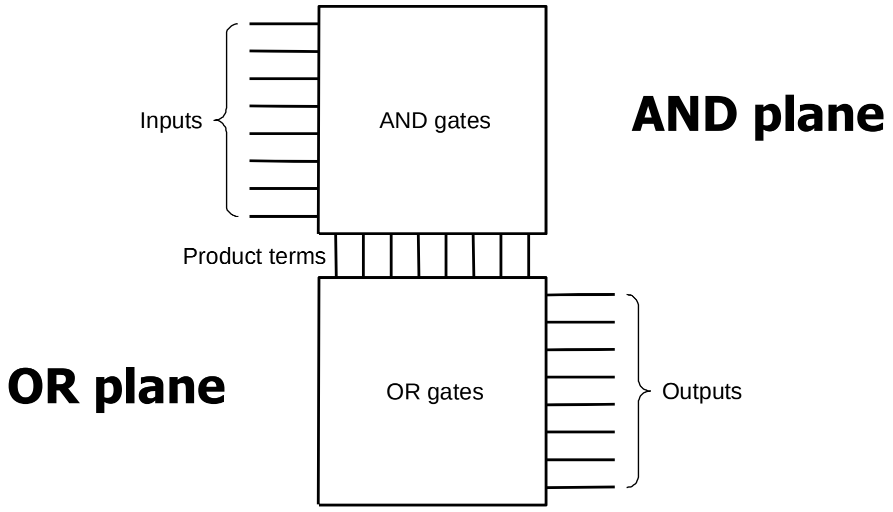
- Answer: $D = (\overline{A} \cdot B \cdot C) + (A \cdot \overline{B} \cdot C) + (A \cdot B \cdot \overline{C}) + (A \cdot B \cdot C)$
- It's a 3-person voting system

- **Minterm**: a group of variables ANDed where either the variable or its negation is represented
- Any logic function can be represented as a SUM of minterms

Programmable Logic Arrays

- A **programmable logic array** (**PLA**) is a gate-level implementation of the **two-level representation** for any set of logic functions, which corresponds to a truth table with multiple output columns.

- A PLA corresponds to the **sum-of-products representation**.



Example - Problem

- Show a PLA implementation of this example:

Inputs			Outputs		
A	**B**	**C**	**D**	**E**	**F**
0	0	0	0	0	0
0	0	1	1	0	0
0	1	0	1	0	0
0	1	1	1	1	0
1	0	0	1	0	0
1	0	1	1	1	0
1	1	0	1	1	0
1	1	1	1	0	1

- Sum-of-product representation

$$D = \overline{A}\cdot\overline{B}\cdot C + \overline{A}\cdot B\cdot\overline{C} + \overline{A}\cdot B\cdot C + A\cdot\overline{B}\cdot\overline{C} + A\cdot\overline{B}\cdot C + A\cdot B\cdot\overline{C} + A\cdot B\cdot C$$

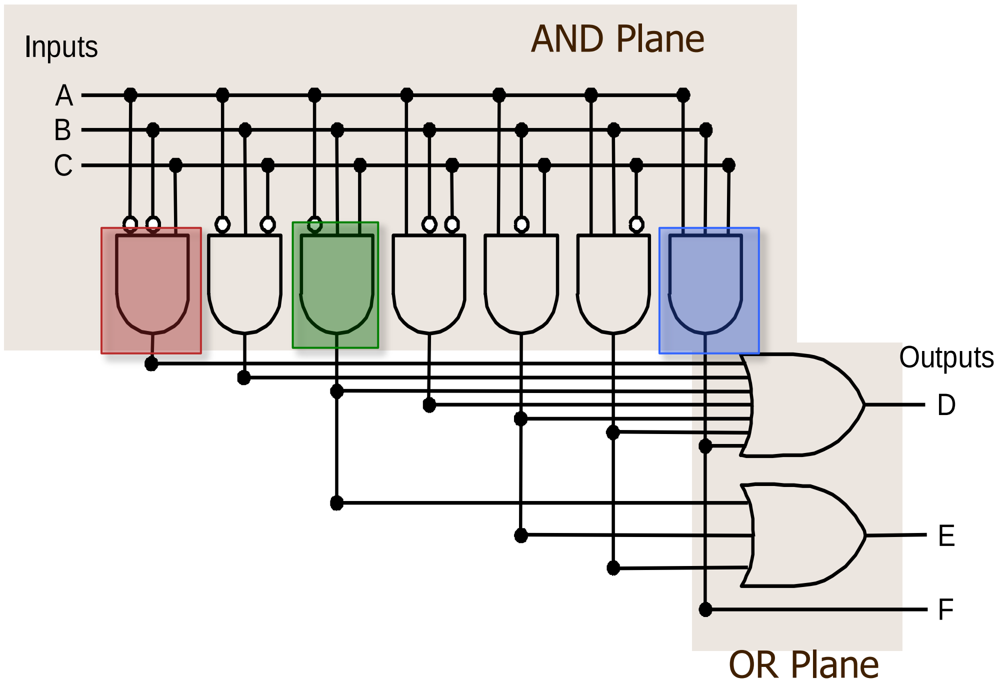
$$E = \overline{A}\cdot B\cdot C + A\cdot\overline{B}\cdot C + A\cdot B\cdot\overline{C}$$

$$F = A\cdot B\cdot C$$

Example - Answer

- There are seven unique product terms with at least one true value in the output section, and hence there are seven columns in the AND plane. There are three inputs and hence the number of rows in the AND plane is three.

Inputs			Outputs		
A	B	C	D	E	F
0	0	0	0	0	0
0	0	1	1	0	0
0	1	0	1	0	0
0	1	1	1	1	0
1	0	0	1	0	0
1	0	1	1	1	0
1	1	0	1	1	0
1	1	1	1	0	1

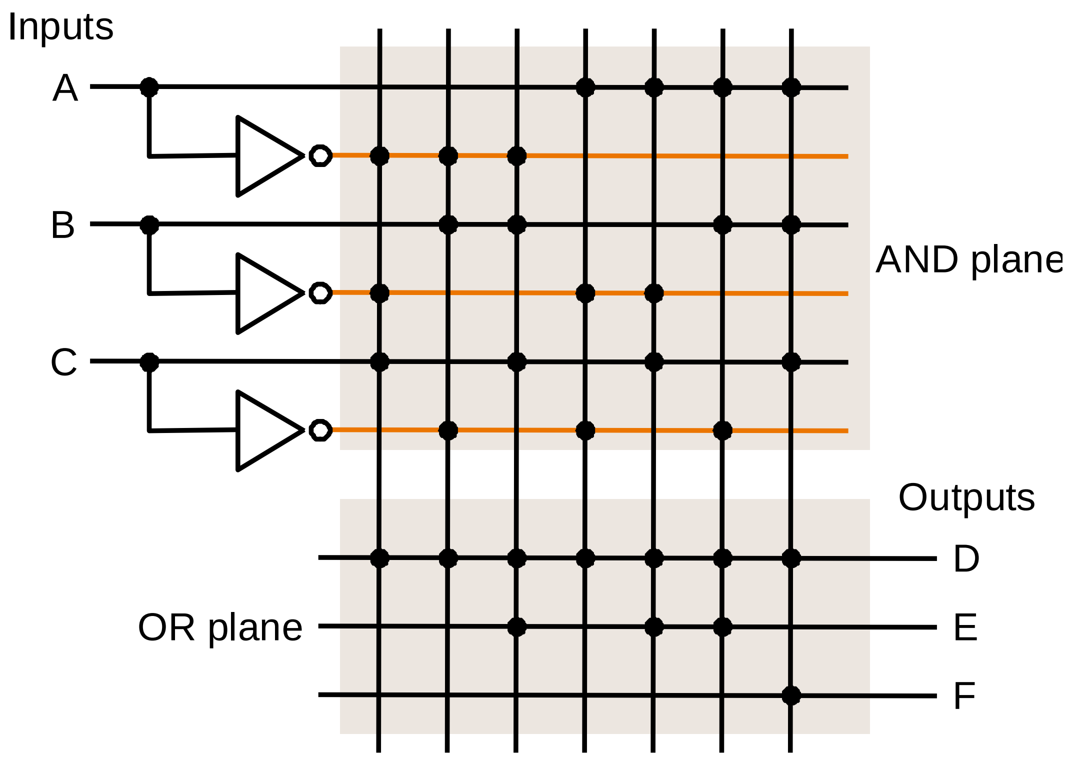


- There are three outputs and hence the number of rows in the OR plane is three.

Example - Answer (cont'd)

$$D=\overline{A}\cdot\overline{B}\cdot C+\overline{A}\cdot B\cdot\overline{C}+\overline{A}\cdot B\cdot C+A\cdot\overline{B}\cdot\overline{C}+A\cdot\overline{B}\cdot C+A\cdot B\cdot\overline{C}+A\cdot B\cdot C$$
$$E=\overline{A}\cdot B\cdot C+A\cdot\overline{B}\cdot C+A\cdot B\cdot\overline{C}$$
$$F=A\cdot B\cdot C$$

- An equivalent PLA representation:



Simplifying Logic Equations

- Truth tables can grow rapidly in size and become tedious.
- Logic equations are better in this case, however, there are many different ways of writing a Boolean expression, each of them will lead to a circuit implementation
- Simplifying Boolean expressions leads to simpler and cheaper circuits (lesser components)
- There are many formal methods, algorithms and software for simplifying Boolean expressions
- **Karnaugh-Maps (K-maps)** is one such method that can be run by hand for designing simple circuits

K-Maps

- K-Map is a graphical representation of the truth table or logic function
- In a K-map each cell represents one possible minterm
- Cells are arranged following a Gray code i.e., two adjacent cells are such that the corresponding minterms differ in only one variable
- Examples: K-Map Layouts

A \ B	0	1
0	m0	m1
1	m2	m3

A \ BC	00	01	11	10
0	m0	m1	m3	m2
1	m4	m5	m7	m6

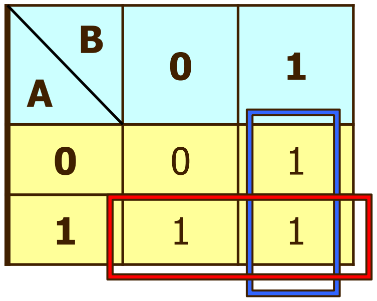
AB \ CD	00	01	11	10
00	m0	m1	m3	m2
01	m4	m5	m7	m6
11	m12	m13	m15	m14
10	m8	m9	m11	m10

K-map Simplification Rules

- Find largest size groups of adjacent cells at 1
- 2^N (i.e. 1, 2, 4, 8) adjacent cells in each group
- K-map is toroid(i.e., rightmost cells are adjacent to the leftmost cells and topmost cells are adjacent to bottom cells)
- Larger groups = fewer inputs to the AND gate
- Fewer groups = fewer AND gates and fewer inputs to OR gate
- Best group might not be unique

- Example: Simplify $F = A\cdot\overline{B} + A\cdot B + \overline{A}\cdot B$

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



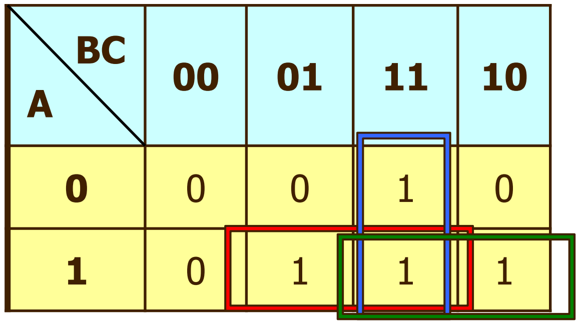
A \ B	0	1
0	0	1
1	1	1

$F = A + B$

K-Map Example

- Simplify the logic expression for the 3-person voting system on page 18

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

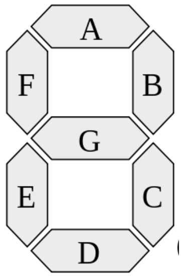


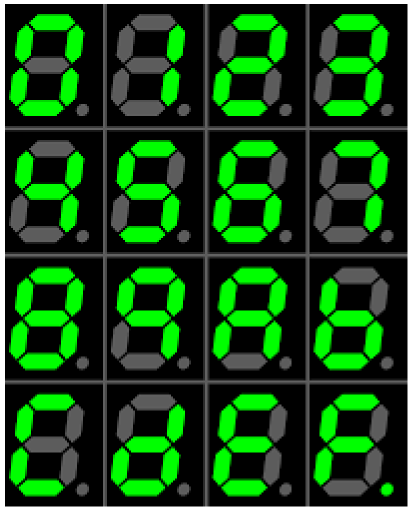
A \ BC	00	01	11	10
0	0	0	1	0
1	0	1	1	1

F = BC + AC + AB

7-segment Digital Display

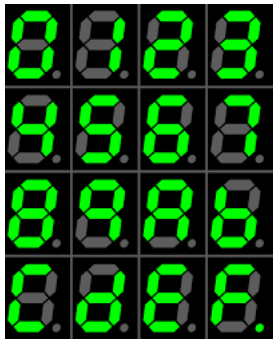
- Use 7-segment digital display to display one Hexadecimal digit
- Each segment is represented by a logic function





- **Tasks:**
  - Give the truth table for segment G
    - How many inputs needed?
  - Deduce the sum-of-products logic equation from the table
  - Use K-Map to simplify the logic equation

- Truth Table for segment G (try to fill segment A as exercise):



Inputs				Output	
i_3	i_2	i_1	i_0	G	A
0	0	0	0	0	
0	0	0	1	0	
0	0	1	0	1	
0	0	1	1	1	
0	1	0	0	1	
0	1	0	1	1	
0	1	1	0	1	
0	1	1	1	0	
1	0	0	0	1	
1	0	0	1	1	
1	0	1	0	1	
1	0	1	1	1	
1	1	0	0	0	
1	1	0	1	1	
1	1	1	0	1	
1	1	1	1	1	

- Using the table we have 12 minterms for segment G in the logic expression

- K-Map:

$i_3i_2 \backslash i_1i_0$	**00**	**01**	**11**	**10**
00	0	0	1	1
01	1	1	0	1
11	0	1	1	1
10	1	1	1	1

- $G = i_1i_0' + i_3i_2' + i_3i_0 + i_2'i_1 + i_3'i_2i_1'$
- Conclusion:
  - Before simplification we would need 12 AND gates with 4 inputs each and one OR gate with 12 inputs
  - After we only need 3 AND gates with 2 inputs one AND gate with 3 inputs and one OR gate with 4 inputs