COMP2611: Computer Organization

Introduction to Digital Logic

- **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**

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.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

  NOT A = 10100

  A+B = 11011

  A . 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**

Basic Laws of Boolean Algebra

- **Identity laws**:

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

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

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

- **Complement laws**:

  $A + \bar{A} = 1$ $\qquad$ $A \times \bar{A} = 0$

- **Commutativity laws**:

  $A + B = B + A$ $\qquad$ $A \times B = B \times A$

- **Associativity laws**:

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

- **Distributivity laws**:

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

Additional Laws of Boolean Algebra

- **Idempotence**:

  $A + A = A \qquad A \times A = A$

- **Absorption laws**:

  $A + (A.B) = A \qquad A \times (A + B) = A$

- **De Morgan Laws**:

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

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

- Using all these laws, we can express and simplify all other logic functions as well as define arithmetic functions for computers

- **Example**:
  - Consider a logic function with two inputs (A, B) and two outputs (C, D). Show the logic equations for the logic function that has the following properties and give the truth table:
    - C is true if exactly one input is true;
    - D is true if exactly two inputs are true.
- **Answer**:

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

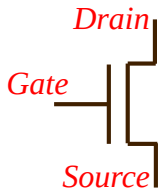
$$D = A \cdot B$$

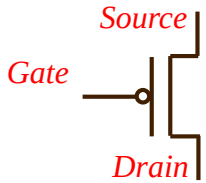
A	B	D	C
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

- **Question:** What operation do functions C and D implement?

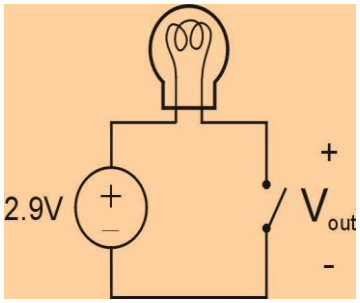
Computer Arithmetic and Logic operation can be specified via logic functions

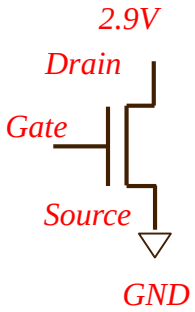
- At the base of modern computers is the MOS (Metal Oxide Semiconductor) Transistor
- Transistor:
  - Three terminals: Gate, Source and Drain
  - Two types of transistors N (left) and P(right)

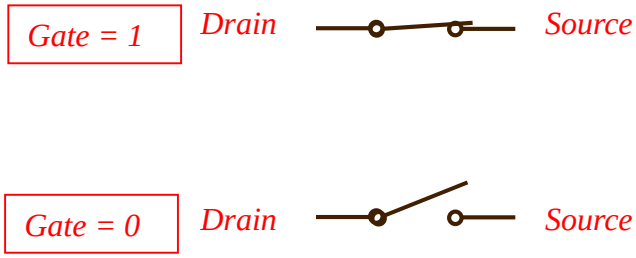




  - Operates as an Electronic switch
  - For the N-type: if the Gate is powered, then the path from source to drain acts like a wire otherwise the path is broken.

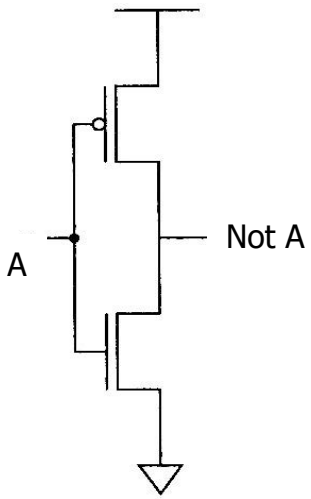






Logic gates

- 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)

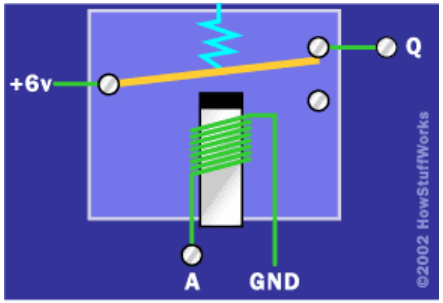


Digital Inverter Using CMOS

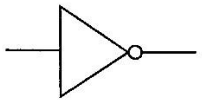
A	NOT A
0	1
1	0

Truth Table

A	NOT A
0V	2.9V
2.9V	0V

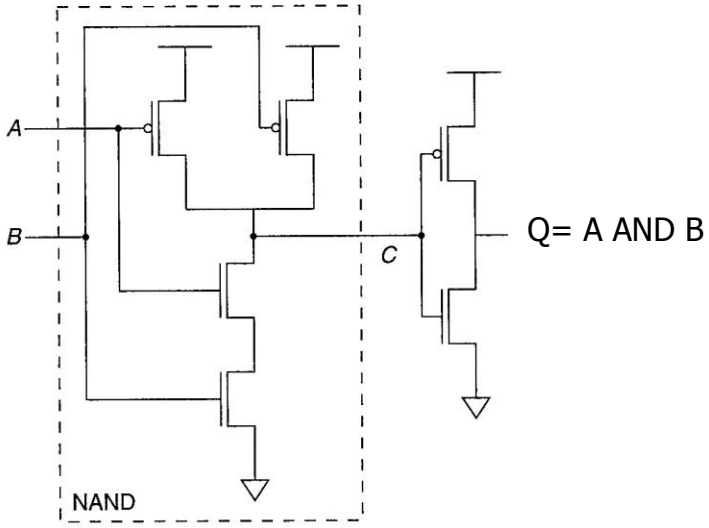


Digital Inverter Using Electromagnetic Relays



Symbolic representation of the NOT Gate

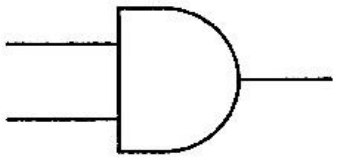
- **AND Gate**
- The AND gate is built from a NAND gate and an Inverter



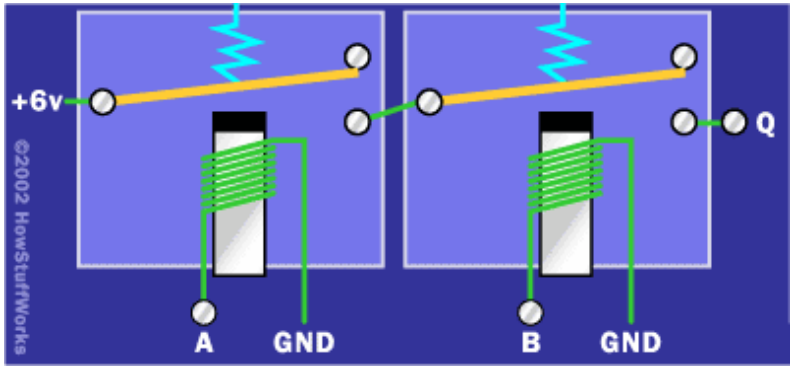
AND Gate Using CMOS

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

Truth Table

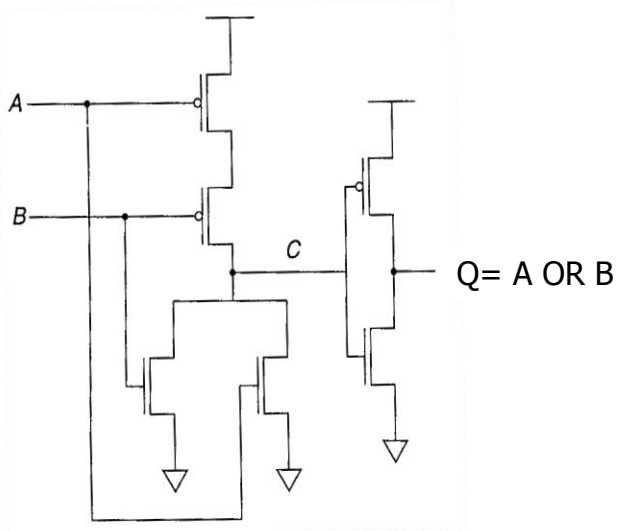


Symbolic representation of the AND Gate



AND Gate Using Electromagnetic Relays

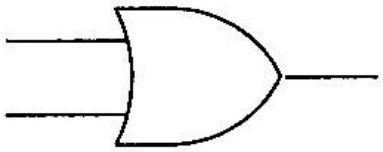
- **OR Gate**
- The OR gate is built from a NOR gate and an Inverter



OR Gate Using CMOS

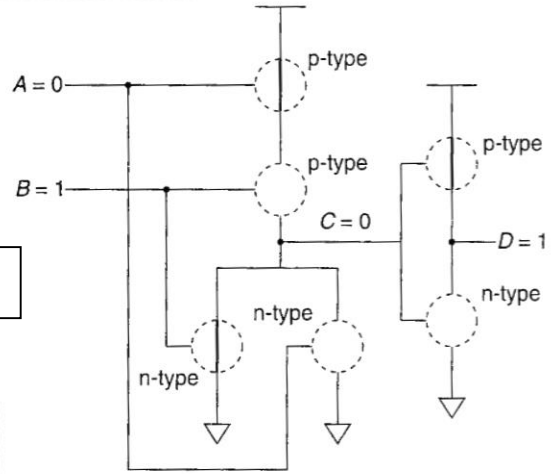
A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

Truth Table



Symbolic representation of the OR Gate

Example



- **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

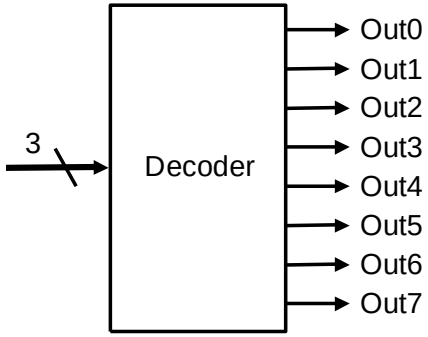
- Other than logic gates that are the most basic building blocks, there also exist some **higher-level basic building blocks** that are also commonly used:
  - **Decoders/encoders**
  - **Multiplexors**
  - **Two-level logic** and **PLAs**

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

Decoders

- 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):



a. A 3-bit decoder

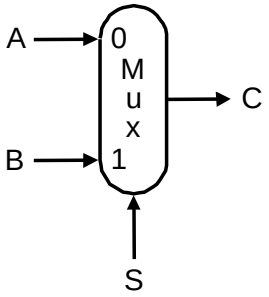
- An **encoder** performs the inverse function of a decoder, taking 2^N inputs and producing an N-bit output.

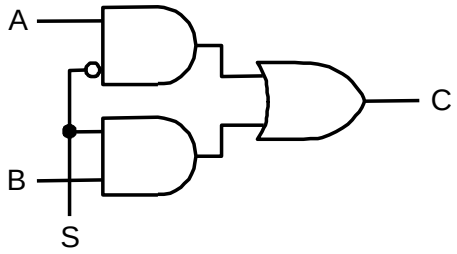
Multiplexors

- 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):





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

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	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

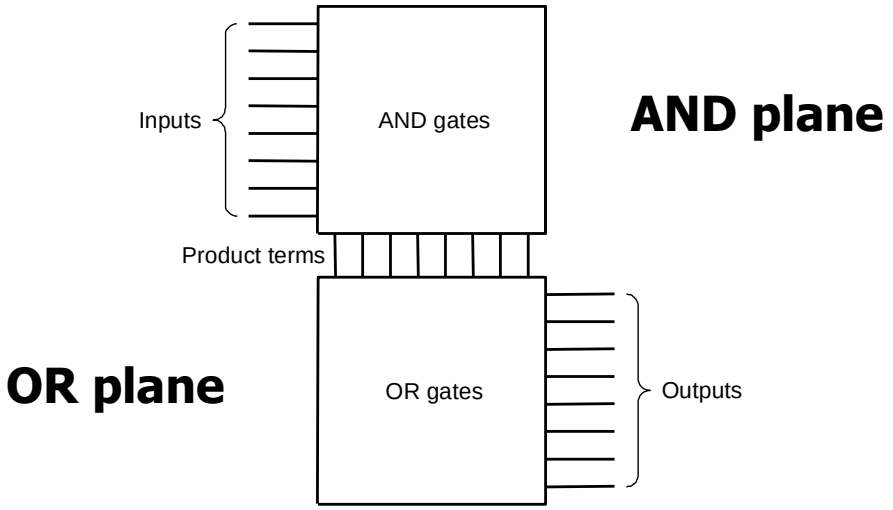
- **Answer**: $D = (\overline{A} \cdot \overline{B} \cdot C) + (\overline{A} \cdot B \cdot \overline{C}) + (A \cdot \overline{B} \cdot \overline{C}) + (A \cdot B \cdot C)$

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

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

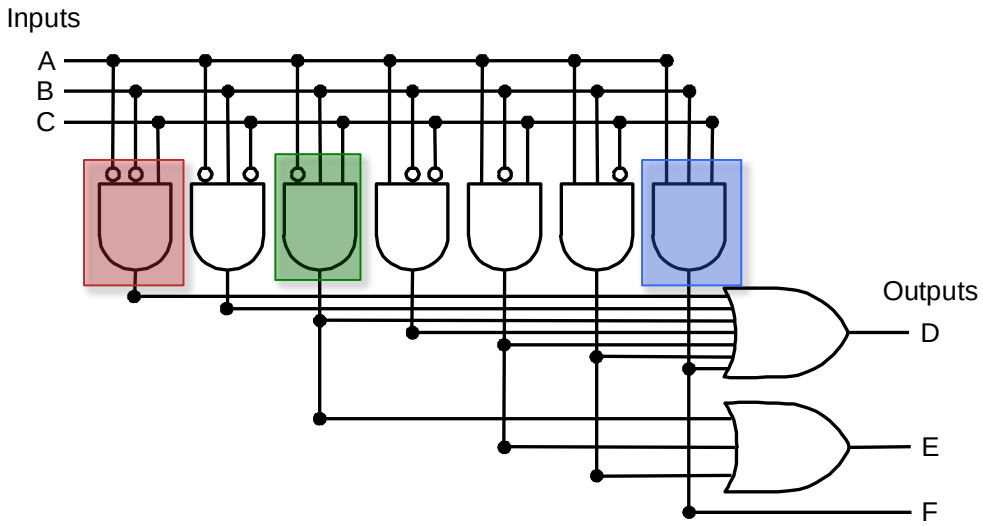
- Show a PLA implementation of this example:

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

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.

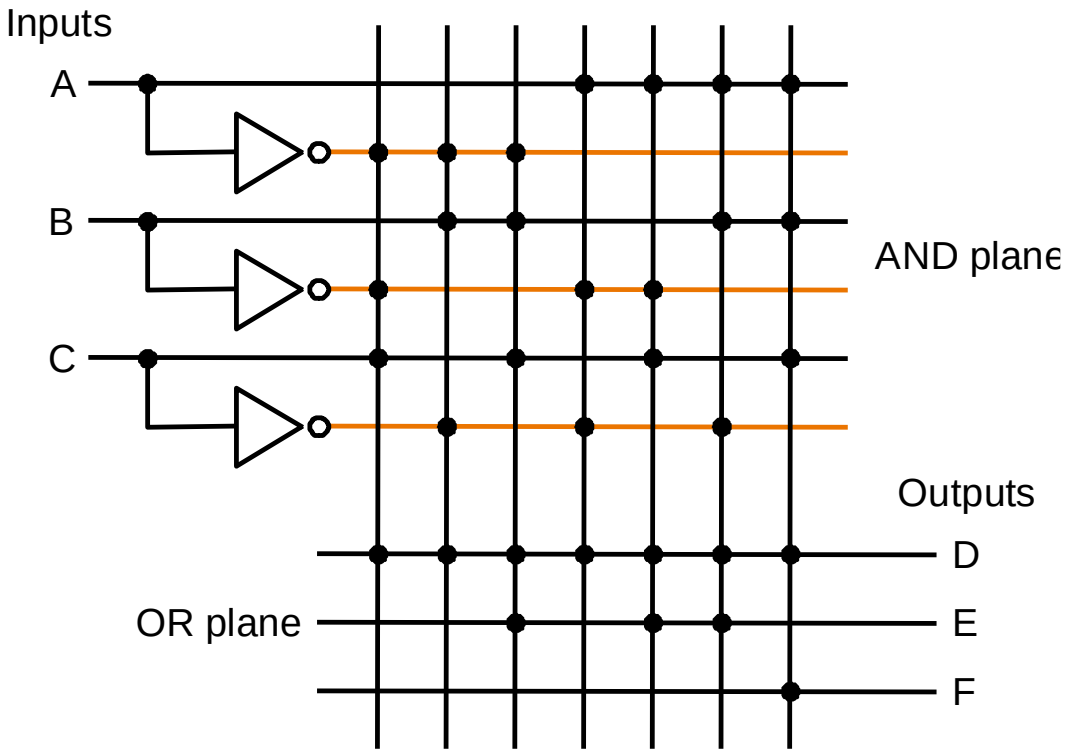
Inputs			Outputs		
A	B	C	D	E	F
0	0	0	0	0	0
0	0	1	1	0	0
0	1	0	1	0	0
0	1	1	1	1	0
1	0	0	1	0	0
1	0	1	1	1	0
1	1	0	1	1	0
1	1	1	1	0	1



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

Example - Answer (cont'd)

- 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

Example

- Three-person voting system

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

- **Answer**:

$$D = (\bar{A} \times B \times C) + (A \times B \times \bar{C}) + (A \times \bar{B} \times C) + (A \times B \times C)$$

- **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

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-Maps

- Simplify expression by finding largest size groups of adjacent cells at 1 in the K-Map
- Example: Simplify F = AB′ + AB + A′B

A \ B	0	1
0	A′B′	A′B
1	AB′	AB

A \ B	0	1
0	0	1
1	1	1

F = B + A

K-Maps

- Simplify the logic expression for the 3-person voting system

$$D = (\bar{A} \times B \times C) + (A \times B \times \bar{C}) + (A \times \bar{B} \times C) + (A \times B \times C)$$

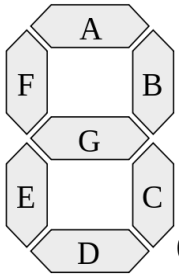
A \ BC	00	01	11	10
0	0	0	1	0
1	0	1	1	1

F = BC + AC + AB

- Can only group 8, 4, 2 or 1 adjacent cells
- Table is a toroid (i.e., rightmost cells are adjacent to the leftmost cells and topmost cells are adjacent to bottom cells)
- Guidelines:
  - Larger groups = fewer inputs to the AND gate
  - Fewer groups = fewer AND gates and fewer inputs to OR gate

K-Maps

- Example: Consider a 7-segment digital display. Each segment is represented by a logic function



- Assuming we only want to display one Hexadecimal digit:
  - Give the Truth table for segment G
  - Deduce the sum-of-products logic equation from the table (left as an exercise)
  - Use a K-Map to simplify the equation
- Answer:
  - Hexadecimal => 4 inputs i_3, i_2, i_1, i_0 to represent values 0, …, 9, a, b, c, d, e, f (to avoid confusion between 0, and 8 on one hand and D and B respectively, on the other hand, we use miniscule b and d representation)

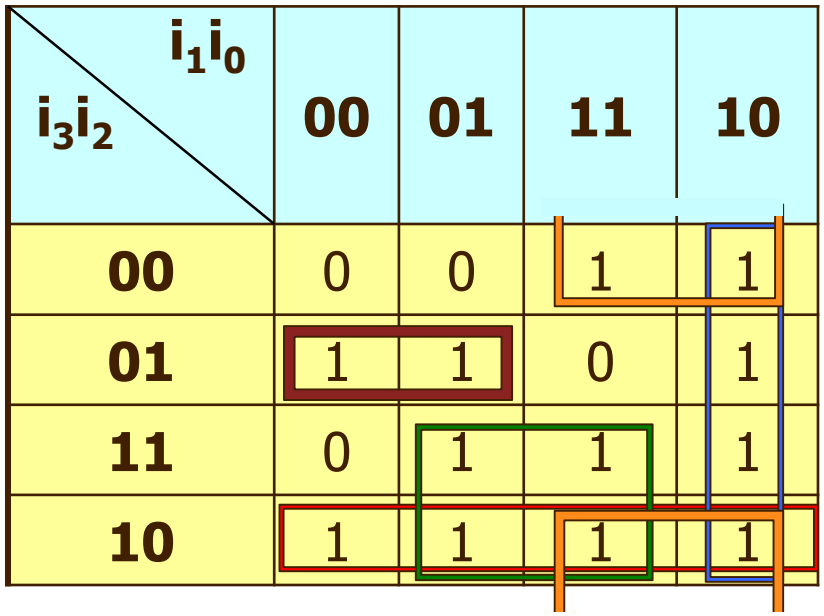
- Truth Table for segment G:

Inputs				Output
i_3	i_2	i_1	i_0	G
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 in the logic expression

K-Maps

- K-Map:



i_3i_2 \ 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

Sequential Logic Circuits

- Computers are designed using both combinational logic circuits and sequential logic circuits.

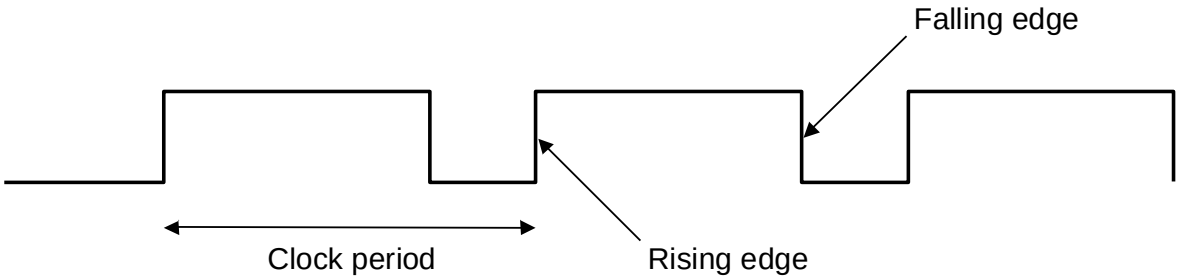
- **Sequential logic circuits** are circuits whose output depends on **both** the current input and the value stored in the memory of the circuit (called **state**)

- Sequential logic circuits often depend on a clock to determine when the memory or state element of the circuit is updated.

Clocks

- A **clock** is a free-running signal with a fixed **cycle time** (called **clock period**) or, equivalently, a fixed **clock frequency** (i.e., inverse of the cycle time).

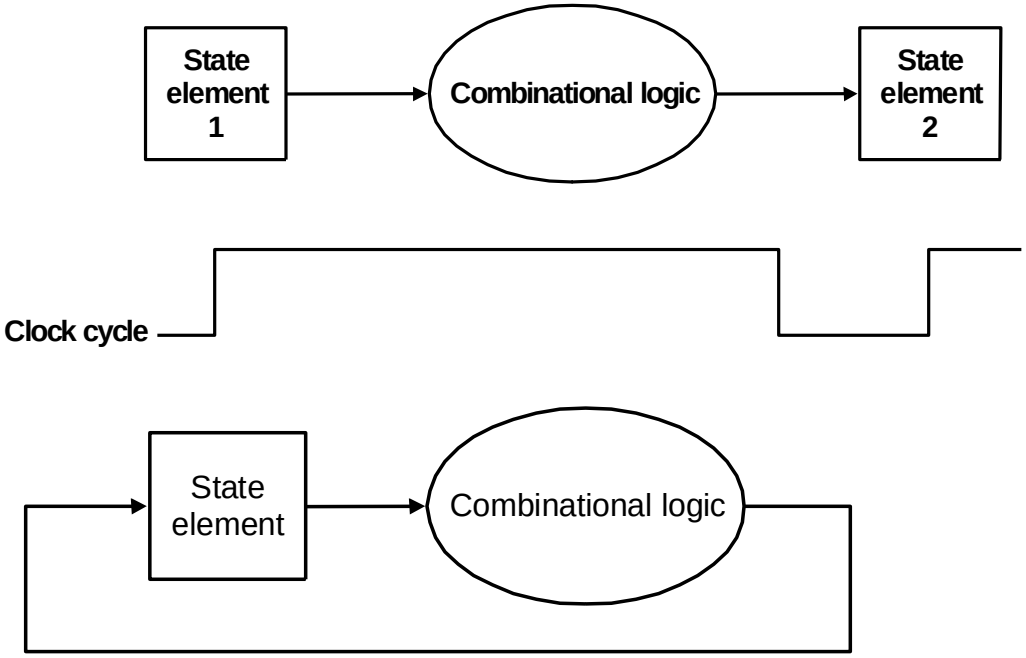
- **Edge-triggered clocking**:
  - Design methodology for sequential logic circuits in which all state changes occur on a clock edge (**rising edge** or **falling edge**).



Clocked Systems

- Clocked systems are also called **synchronous systems**.

- Sequential logic circuits are made of a combinational logic circuit and a state element that can store a state.
- Relationship among state elements and combinational logic blocks in a synchronous, sequential logic design:



Difference between Combinational and Sequential

Combinational Circuits

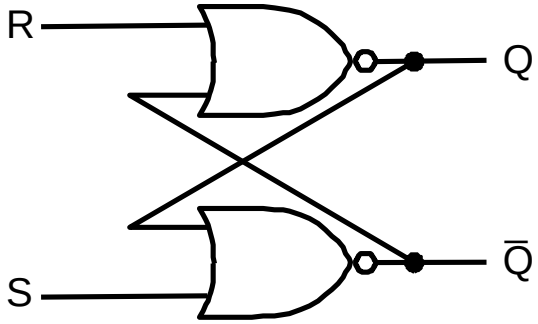
- For the same inputs, we always obtain the same output
  - Example: Logic functions D and C on slide 9 provide the result of the addition of A and B.

Sequential Circuits

- Have state: store information
- The output depends on the current state and the inputs. Depending on the current state, the same inputs may result in different outputs
  - *Examples:*
    - Counter
    - "memory" elements and "state machines"

Memory Elements

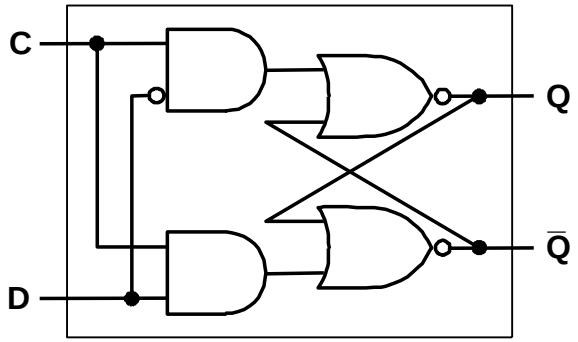
- **S-R latches** (set-reset latches):
  - **Unclocked** memory element built from a pair of cross-coupled NOR gates (i.e., OR gates with inverted outputs).

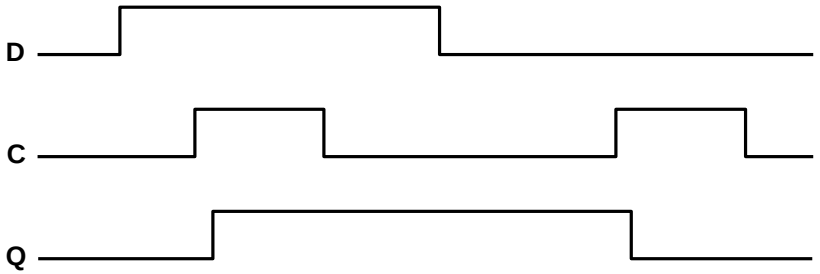


Inputs		Outputs	
R	**S**	**Q**	**Not Q**
0	0	Latch	
1	0	0	1
0	1	1	0
1	1	Invalid Input	

D Latches

- Unlike S-R latches which are unclocked, **D latches** are **clocked** (i.e., state changes are triggered by a clock).
- The output is equal to the value of the stored state inside it.





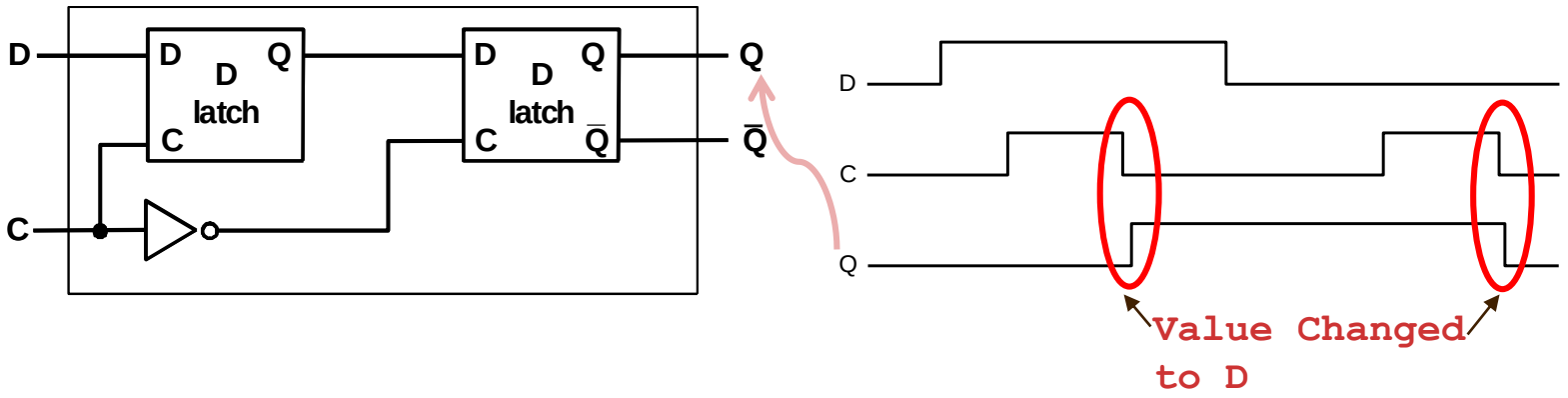
- **Operation**:
  - When the clock C is asserted, the latch is open and the Q output immediately assumes the value of the D input.

Inputs		Outputs	
C	**D**	**Q**	**Not Q**
0	0	Latch	
0	1	Latch	
1	0	0	1
1	1	1	0

- Clocked latches are used to build **flip-flops**.

D Flip-Flops

- **D flip-flops**, like D latches, are **clocked**.
- The outputs change only on the (rising or falling) clock edge.
- **D flip-flop with a falling-edge trigger**:



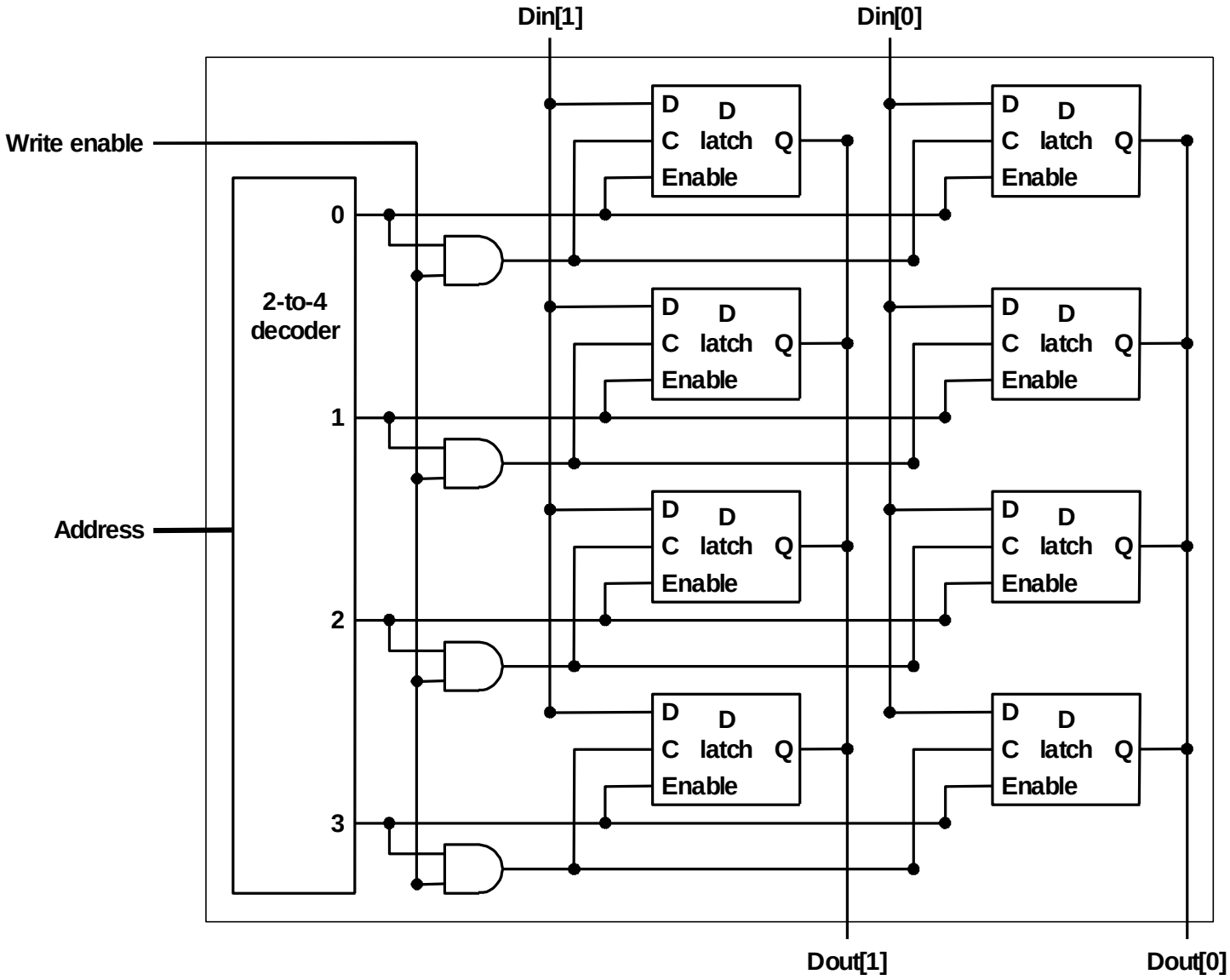
- **Operation**:
  - When the clock input C changes from asserted to deasserted, the Q output stores the value of the D input.

Flip-Flops and Memory elements

- D-Latches and flip-flops can be used to build memory elements used by digital computers to hold data temporarily.
  - **register file** is a structure in the datapath consisting of a set of registers that can be read and written by supplying a register number to be accessed. These represent the hardware "variables"
  - **Static random access memories** (**SRAMs**): relatively small memories used for cache and built with flip-flops

- Example: We want to build an SRAM chip that has the following characteristics:
  - Can store 4 words of data each word has 2 bits
  - We can read the content of one word at a time by supplying its address on 2 bits (i.e., word 00, or 01, or 10, or 11)
  - We can Write Data to one word by supplying 2 bits of Data and the 2-bit address of the word to be written

A 4 x 2 SRAM


Din[1]
Din[0]
Write enable
Address
2-to-4
decoder
0
1
2
3
D
C
Enable
latch
Q
Dout[1]
Dout[0]

Summary

- Computers use two reference levels of voltage to represent data as binary digits
- **Boolean Algebra** to express any operation on bits as a logic equation or a equivalently a truth table
  - Three basic operations: AND (•), OR (+), and NOT
  - Basic rules of Boolean Algebra
    - Commutativity, Associativity, Distributivity of + on • and • on +
    - Annihilator, Idempotence, Complement, De Morgan, Identity
- **The transistor (an electronic switch)** is the basis for building **logic gates**
  - AND, OR, NOT
  - NOR, NAND, XOR, …
- Logic Gates: basic building blocks of more complex logic circuits

Summary

- Two types of logic circuits:
  - **Combinational circuits**: process the inputs to produce the output
    - Multiplexers
    - Decoders
    - Arithmetic and logic operations
  - **Sequential circuits**: process the input and the **state element** (memory) of the circuit to produce the output
    - Memory, registers
    - Not possible to access (read) and update (write) the state element simultaneously, therefore a clock is necessary to control when operations are done
    - **Edge-triggered clocks** are more efficient and more accurate, than level-based clocks
    - Latches, Flip-flops