ECE 341
Introduction to Computer Hardware

Instructor: Zeshan Chishti
zeshan@pdx.edu

Fall 2014

Portland State University

When and Where?

- When: Mondays and Wednesdays 4:40 - 6:30 PM
- Where: EB 103
- Office hours: Mondays and Wednesdays after class, or by appointment
- Webpage: **http://ece.pdx.edu/~zeshan/ece341.htm**
- TA: Leela Kamalesh Yadlapalli
- TA office hours: TBA
- Go to the course webpage for:
  - Class slides
  - Course syllabus and schedule
  - Grading policies
  - Homework assignments and solutions

Course Information

- Textbook: Computer Organization and Embedded Systems, 6th Edition, *Hamacher, Vranesic, Zaky and Manjikian*. McGraw-Hill, 2011. ISBN 9780073380650 / 0073380652

- Expected background: CS201 or equivalent
  - Basic knowledge of computer organization
  - Data representation in binary, decimal, and hexadecimal notation
  - Basic knowledge of instruction set architecture
  - Assembly language programming

Grading Policy

- Homeworks 35%
  - 8 homework assignments, top 7 chosen for each student to contribute towards the 35%
- Midterm Exam 25%
- Final Exam 40%
- Grading scale (tentative):
  - A: 92-100% A-: 87-91.5%
  - B+: 83-86.5% B: 79-82.5% B-: 75-78.5%
  - C+: 71-74.5% C: 67-70.5% C-: 63-66.5%
  - D+: 59-62.5% D: 55-58.5% D-: 50-54.5%
  - F: Below 50%

Other Policies

- All homeworks due in class. No extensions given
- Submit homework to the instructor soon after entering the class
- You can also submit homeworks via email before the start of class

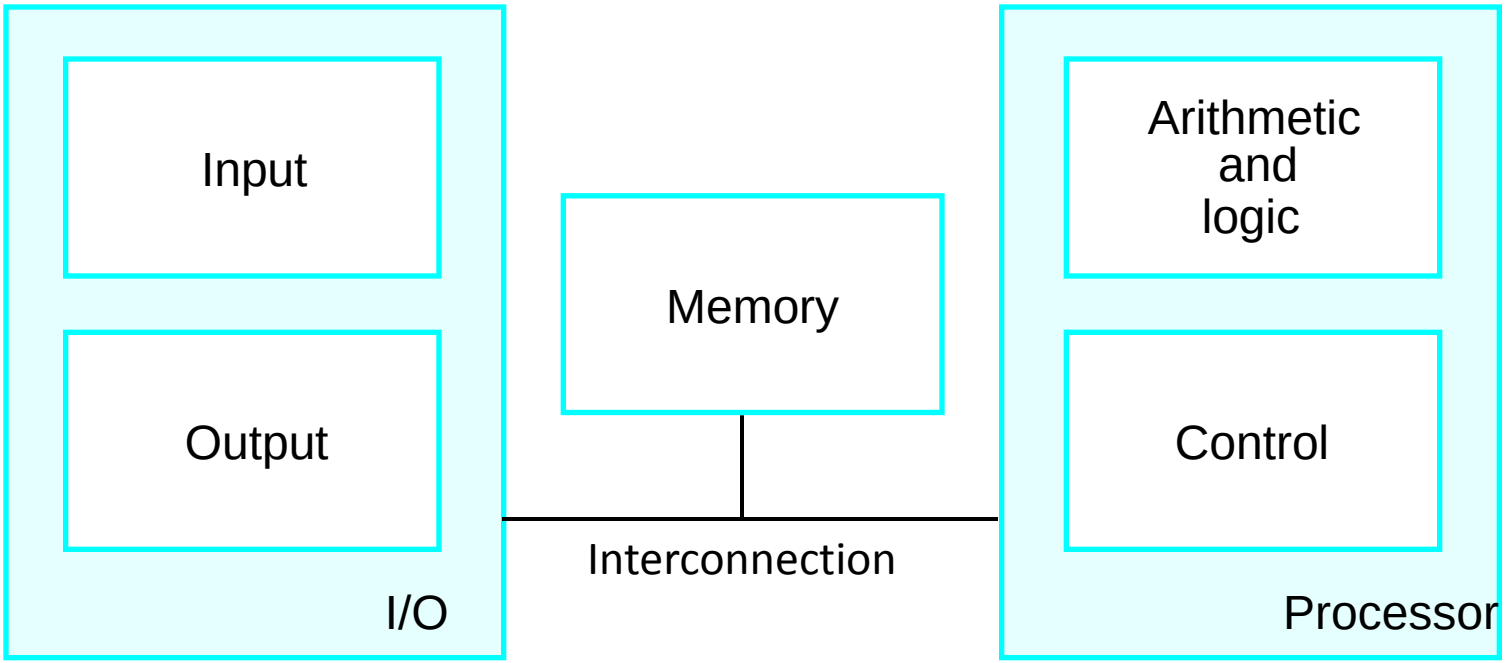
- Midterm exam in class during week 6
  - Wednesday, November 5, 4:40 – 6:30 PM

- Final exam will cover entire course with more emphasis on material taught after the midterm exam

What this course is all about?

This course is about designing a computer in hardware

End Goal: To understand the hardware implementation of different components of a modern computer



Basic functional units of a computer

Course Topics

- Digital logic – gates, flip flops, multiplexers, state machines
- Computer arithmetic
- Basic computer architecture – data path, control, and buses
- Pipelining hardware
- CISC vs RISC architectures
- Memory hierarchy and virtual memory
- Input/output techniques – polling, interrupts, and DMA

Lecture # 1

DIGITAL LOGIC DESIGN

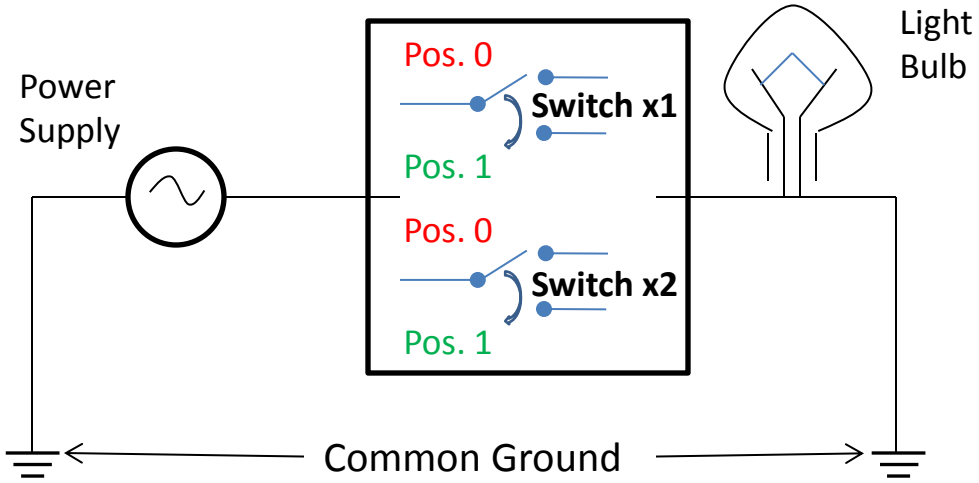
Lecture Topics

- Combinational Logic
  - Logic gates and logic circuits
  - Synthesis of logic functions
  - Logic expression minimization

- Reference: Appendix A of the textbook (pages 466-481), including sections A.1, A.2, A.3, and A.4, excluding A.3.1 and A.3.2

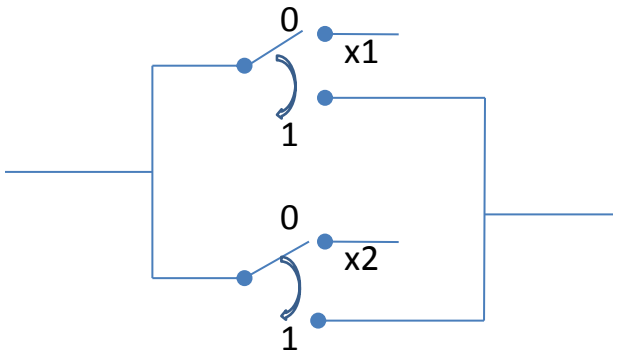
Logic Circuits

- Example: Light bulb controlled by two switches



- Input: Switch positions
  - Represented as binary variables **x1** and **x2** whose values can be either 0 or 1
- Output: Condition of light bulb
  - Represented as binary variable **f** with possible values 0 (off) or 1 (on)
- Logic circuit: Interconnection of switch terminals
  - Output=1 if a closed path exists from supply to the bulb through the switch network

OR function (Parallel Connection)



If either x1 or x2 is in the 1 position, the light is on (output = 1)

x1	x2	f(x1,x2) = x1+x2
0	0	0
0	1	1
1	0	1
1	1	1

- OR function is represented by the "+" symbol:

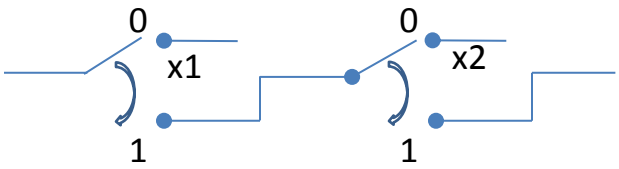
$$f = x1 + x2$$

- Basic properties of OR operation:

$$x1 + x2 = x2 + x1$$
$$(x1 + x2) + x3 = x1 + (x2 + x3)$$
$$1 + x = 1$$
$$0 + x = x$$
$$x + x = x$$

- Extendible to n variables: f = x1+x2+…+xn is equal to 1 if any variable xi = 1

AND function (Series Connection)



Light is on (output = 1) only if both x1 and x2 are in the "1" position

x1	x2	f(x1,x2) = x1.x2
0	0	0
0	1	0
1	0	0
1	1	1

- AND function is represented by the "." symbol:

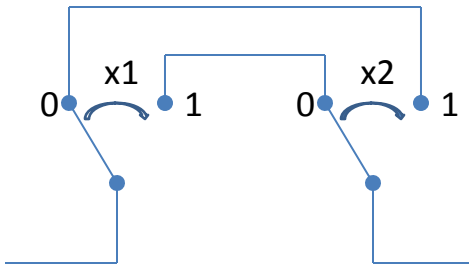
$$f = x1 \cdot x2$$

- Basic properties of AND operation:

$$x1 \cdot x2 = x2 \cdot x1$$
$$(x1 \cdot x2) \cdot x3 = x1 \cdot (x2 \cdot x3)$$
$$1 \cdot x = x$$
$$0 \cdot x = 0$$
$$x \cdot x = x$$

- Extendible to n variables: f = x1.x2…..xn is equal to 1 if for all i, xi = 1

XOR function (Exclusive-OR Connection)



If x1 = x2 = 0, light is off (output = 0).
Flipping any switch flips the output

x1	x2	f(x1,x2) = x1 ⊕ x2
0	0	0
0	1	1
1	0	1
1	1	0

- XOR function is represented by the "$\oplus$" symbol:

$$f = x1 \oplus x2$$

- Basic properties of XOR operation:

$$x1 \oplus x2 = x2 \oplus x1$$
$$(x1 \oplus x2) \oplus x3 = x1 \oplus (x2 \oplus x3)$$
$$1 \oplus x = \bar{x}$$
$$0 \oplus x = x$$
$$x \oplus x = 0$$

- Extendible to n variables: f = x1$\oplus$x2$\oplus$…$\oplus$ xn is equal to 1 if there is an odd number of input variables with values of 1, otherwise f is equal to zero

NOT function

x	f(x) = NOT(x)
0	1
1	0

NOT function complements (inverts) its input

- NOT function is represented by the " $\overline{}$ " symbol

$$f = \bar{x}$$

- Important properties of NOT function:

$$\bar{\bar{x}} = x$$

$$x + \bar{x} = 1$$

$$x\bar{x} = 0$$

$$\overline{w + y} = \bar{w}\bar{y}$$

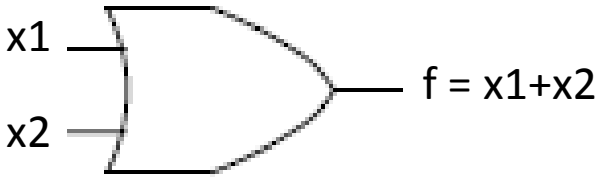
$$\overline{wy} = \bar{w} + \bar{y}$$

} De Morgan's Law

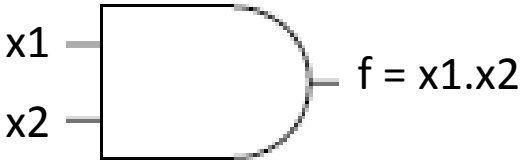
Logic Gates

Electronic circuits that perform basic logic operations are called **logic gates**

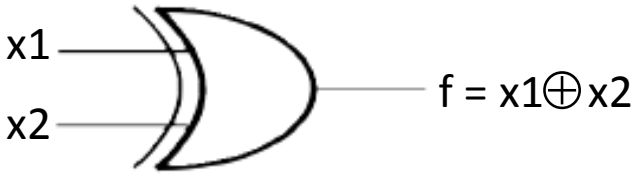
Logic gates are often represented by logic symbols



OR



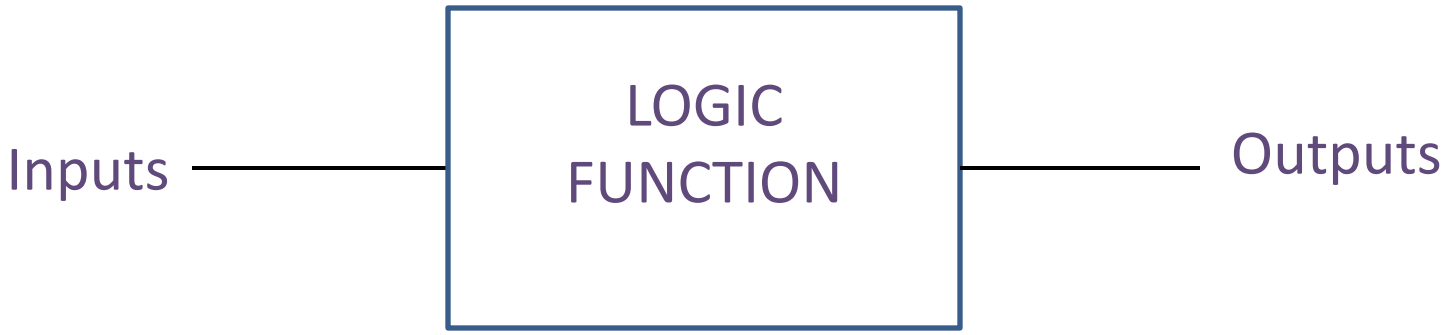
AND



XOR

NOT

Combinational Logic

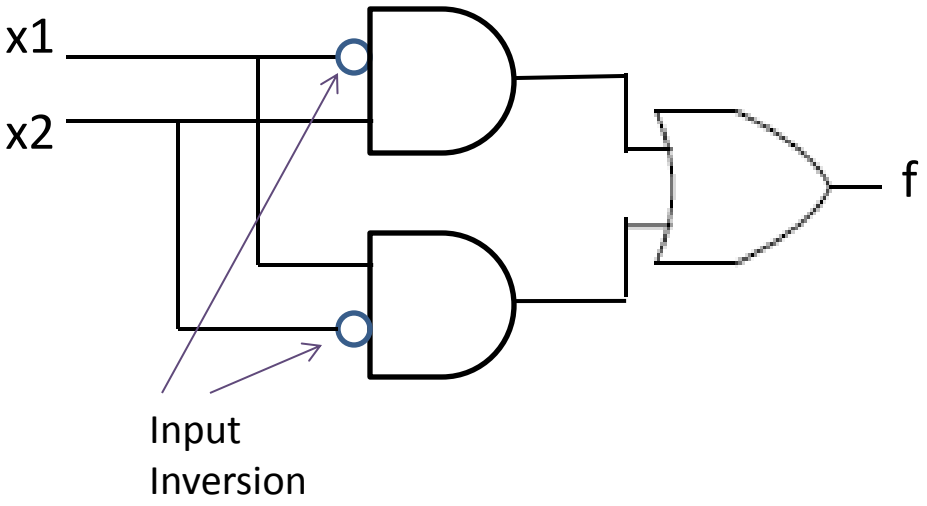


- Each input combination corresponds to a unique set of output values
- Logic function governs the relationship between inputs and outputs
- Logic function is implemented by a logic network
  - A logic network is a combination of logic gates interconnected according to the logic function specification

Logic Function Representation

Three different ways to represent a logic function

(a) Logic Network



(c) Truth Table

x1	x2	$\overline{x1}.x2$	$x1.\overline{x2}$	f
0	0	0	0	0
0	1	1	0	1
1	0	0	1	1
1	1	0	0	0

(b) Logic Expression $f = (\overline{x1}.x2) + (x1.\overline{x2})$

Sum-of-Products Form

- Logic expressions are often shown in **sum of products** form
- For example: $f = (\overline{x1}.x2) + (x1.\overline{x2})$
- To simplify appearance, parentheses and "." signs are removed, for example, $f = \overline{x1}x2 + x1\overline{x2}$
  - Each product terms represents an AND operation
  - Each addition represents an OR operation
  - Operations performed in the following order: NOT, AND, OR
- Sum of products form makes it easier to derive the logic network
  - For each product term, connect input variables to AND gate, use NOT gates wherever input needs to be inverted
  - Connect AND gate outputs to OR gate

Logic Function Synthesis

- A truth table is often used to specify a logic function
- Sum-of-products form can be derived from truth table
- Example:

x1	x2	x3	f1
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Required: Logic function for f1 in sum-of-products form

Steps:

1. Identify rows in the table for which f1 = 1
2. For each such row, write down a product term (AND) which includes all the input variables. NOT operator is applied to variables whose values are 0 in that row
3. Add the "+" operators (OR) between the product terms:

Result:

$$f1 = \overline{x1}.\overline{x2}.\overline{x3} + \overline{x1}.x2.x3 + x1.\overline{x2}.x3 + x1.x2.x3$$

Practice Exercise # 1

- Derive the logic equation in sum-of-products form and draw the logic network for output f2 in the following truth table:

x1	x2	x3	f2
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Practice Exercise # 1

- Derive the logic equation in sum-of-products form and draw the logic network for output f2 in the following truth table:

x1	x2	x3	f2
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

$$f2 = \overline{x1}\,\overline{x2}\,\overline{x3} + \overline{x1}\,x2\,\overline{x3} + x1\,\overline{x2}\,\overline{x3} + x1\,\overline{x2}\,x3 + x1\,x2\,\overline{x3}$$

Logic Function Equivalence

- Two logic functions are ***equivalent*** if they yield same outputs for each combination of inputs
- Equivalence can be determined by comparing truth tables
- Example: Compare $f1 = \overline{x1}.\overline{x2}.\overline{x3} + \overline{x1}.\overline{x2}.x3 + \overline{x1}.x2.x3 + x1.x2.x3$ with $f3 = \overline{x1}.\overline{x2} + x2.x3$

x1	x2	x3	f1
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

x1	x2	x3	$\overline{x1}.\overline{x2}$	$x2x3$	f3
0	0	0	1	0	1
0	0	1	1	0	1
0	1	0	0	0	0
0	1	1	0	1	1
1	0	0	0	0	0
1	0	1	0	0	0
1	1	0	0	0	0
1	1	1	0	1	1

Equivalent

Logic Expression Minimization

- Many equivalent logic functions exist for a particular truth table
- The cost of a logic function can be measured in terms of gate count
- The objective of minimization is to reduce the implementation cost
- An expression is ***minimal*** if there is *no* other equivalent expression of lower cost
- Minimization is performed as a series of algebraic manipulations
- Refer to properties summarized in Table A-4
- Most important rule used in minimization is **distributive rule**:

$$w(y + z) = wy + wz$$

Logic Expression Minimization

Example: Find the minimal logic expression for

$$f1 = \overline{x1}.\overline{x2}.\overline{x3} + \overline{x1}.\overline{x2}.x3 + \overline{x1}.x2.x3 + x1.x2.x3$$

Procedure:

- Group product terms in pairs that differ only in that some variable appears complemented in one term and true in the other.
- Apply the distributive rule to factor out common subproduct terms from each group
- Apply other logic identities from Table A.4, whichever applicable

$$\begin{aligned} f1 &= \overline{x1}.\overline{x2}(\overline{x3} + x3) + x2.x3(\overline{x1} + x1) \\ &= \overline{x1}.\overline{x2}.1 + x2.x3.1 \\ &= \overline{x1}.\overline{x2} + x2.x3 \end{aligned}$$

This expression is minimal

Practice Exercise # 2

Find the minimal logic expression for:

$$f4 = \overline{x1}.\overline{x2}.\overline{x3} + \overline{x1}.\overline{x2}.x3 + \overline{x1}.x2.\overline{x3} + x1.\overline{x2}.\overline{x3} + x1.\overline{x2}.x3$$

Also, quantify the benefits of minimization for this example in terms of reduction in number of logic gates

Practice Exercise # 2

Find the minimal logic expression for:

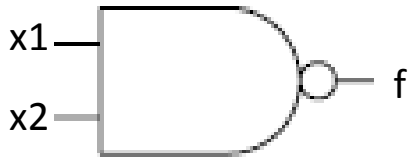
$$f4 = \overline{x1}.\overline{x2}.\overline{x3} + \overline{x1}.\overline{x2}.x3 + \overline{x1}.x2.\overline{x3} + x1.\overline{x2}.\overline{x3} + x1.\overline{x2}.x3$$

Also, quantify the benefits of minimization for this example in terms of reduction in number of logic gates

Solution: $f4 = \overline{x2} + \overline{x1}\,.\overline{x3}$

NAND function

Logic Symbol



NAND function is logically equivalent to NOT(AND)

x1	x2	f(x1,x2) = x1↑x2
0	0	1
0	1	1
1	0	1
1	1	0

- NAND function is represented by the "↑" symbol:

$$f = x1 \uparrow x2$$

- Basic properties of NAND operation:

$$x1 \uparrow x2 = \overline{x1.x2} = \overline{x1} + \overline{x2}$$

De Morgan's Law

$$1 \uparrow x = \overline{x}$$

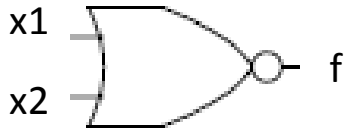
$$0 \uparrow x = 1$$

$$x \uparrow x = \overline{x}$$

- Extendible to n variables: $x1 \uparrow x2 \uparrow \ldots \uparrow xn = \overline{x1.x2 \ldots xn} = \overline{x1} + \overline{x2} + \ldots + \overline{xn}$

NOR function

Logic Symbol



NOR function is logically equivalent to NOT(OR)

x1	x2	f(x1,x2) = x1↓x2
0	0	1
0	1	0
1	0	0
1	1	0

- NOR function is represented by the "↓" symbol:

$$f = x1 \downarrow x2$$

- Basic properties of NOR operation:

$$x1 \downarrow x2 = \overline{x1 + x2} = \overline{x1}.\overline{x2}$$

De Morgan's Law

$$1 \downarrow x = 0$$
$$0 \downarrow x = \overline{x}$$
$$x \downarrow x = \overline{x}$$

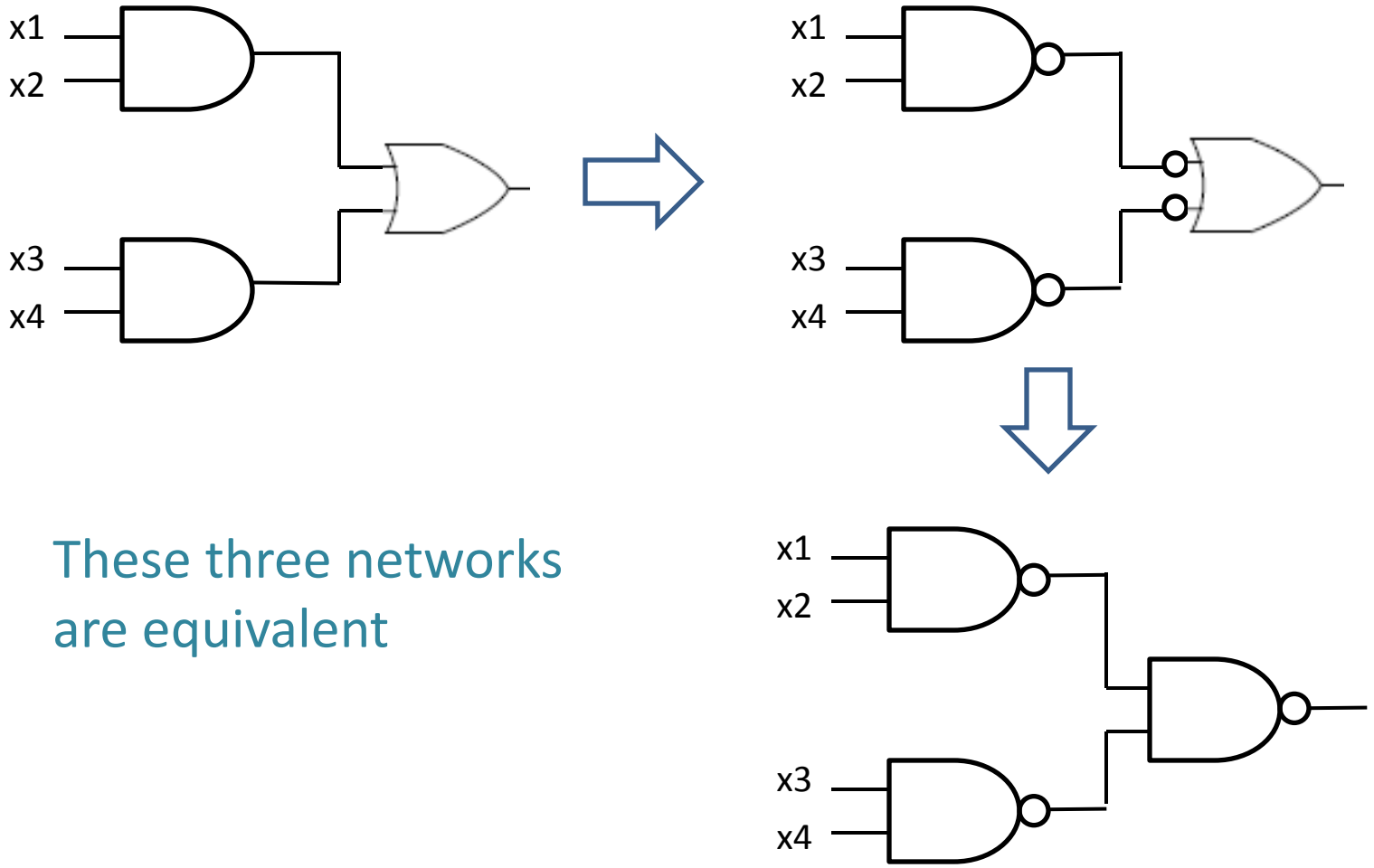
- Extendible to n variables: $x1 \downarrow x2 \downarrow \ldots \downarrow xn = \overline{x1 + x2 + \ldots + xn} = \overline{x1}.\overline{x2}\ldots\overline{xn}$

Logic Design with NAND/NOR gates

- Any logic function can by synthesized by using only NAND gates (or NOR gates)
- Procedure
  - Express the logic function in sum-of-products form
  - Apply involution and De Morgan's rules recursively
  - End result is a NAND-NAND network
- Example:

$$
\begin{aligned}
& x1x2 + x3x4 \\
&= \overline{\overline{x1x2}} + \overline{\overline{x3x4}} \\
&= \overline{(\overline{x1x2}).(\overline{x3x4})} \\
&= (x1 \uparrow x2) \uparrow (x3 \uparrow x4)
\end{aligned}
$$

Equivalence of NAND-NAND and AND-OR Networks



These three networks are equivalent