Digital Logic Design

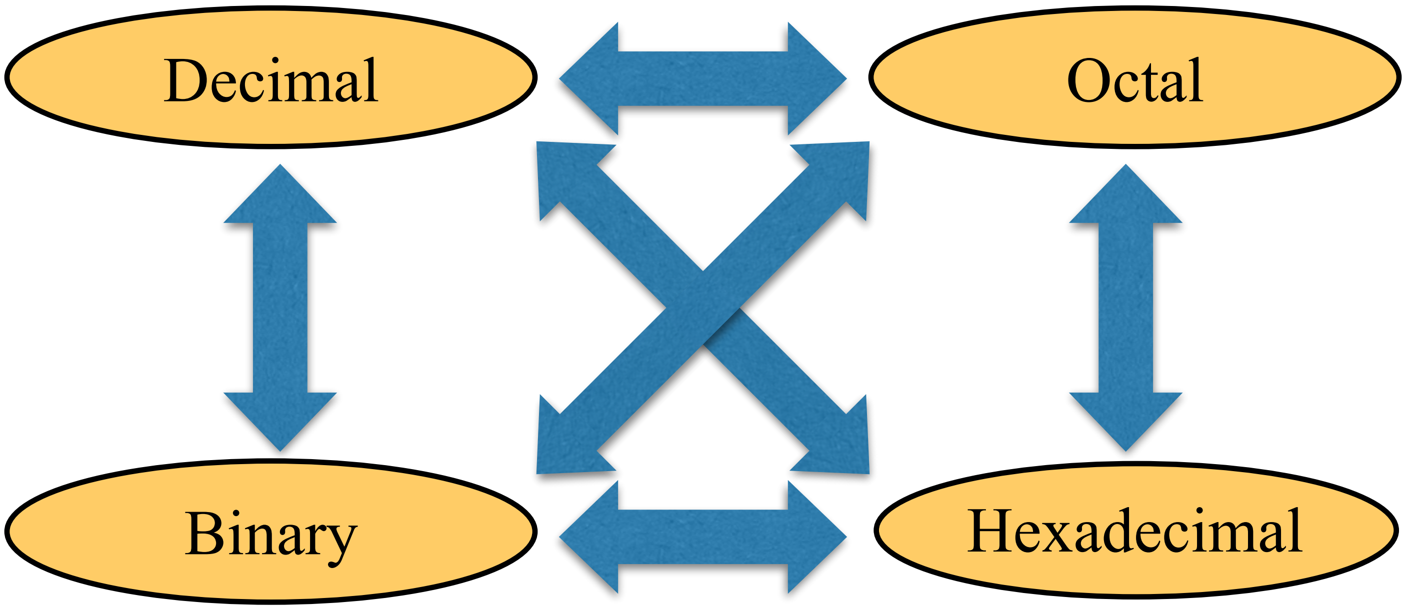
Sung-Soo Lim

KESL
Kookmin Univ. Embedded System Lab

Number Systems

Kookmin Univ. Embedded System Lab

$$25_{10} = 11001_2 = 31_8 = 19_{16}$$


Decimal
Octal
Binary
Hexadecimal

$$125_{10} \Rightarrow \begin{array}{lcr} 5 \times 10^0 & = & 5 \\ 2 \times 10^1 & = & 20 \\ 1 \times 10^2 & = & 100 \\ & & 125 \end{array}$$

$$101011_2 \Rightarrow \begin{array}{lcr} 1 \times 2^0 & = & 1 \\ 1 \times 2^1 & = & 2 \\ 0 \times 2^2 & = & 0 \\ 1 \times 2^3 & = & 8 \\ 0 \times 2^4 & = & 0 \\ 1 \times 2^5 & = & 32 \\ & & 43_{10} \end{array}$$

$$ABC_{16} \Rightarrow \begin{array}{lclcr} C \times 16^0 & = & 12 \times 1 & = & 12 \\ B \times 16^1 & = & 11 \times 16 & = & 176 \\ A \times 16^2 & = & 10 \times 256 & = & 2560 \\ & & & & 2748_{10} \end{array}$$

$125_{10} = ?_2$

2	125	
2	62	1
2	31	0
2	15	1
2	7	1
2	3	1
2	1	1
	0	1

$125_{10} = 1111101_2$

$10AF_{16} = ?_2$

1 0 A F

0001 0000 1010 1111

$10AF_{16} = 0001000010101111_2$

$1234_{10} = ?_{16}$

16	1234	
16	77	2
16	4	13 = D
	0	4

$1234_{10} = 4D2_{16}$

$1010111011_2 = ?_{16}$

$$\underbrace{10}_{2}\ \underbrace{1011}_{B}\ \underbrace{1011}_{B}$$

$1010111011_2 = 2BB_{16}$

$1F0C_{16} = ?_8$

Exercise

Decimal	Binary	Octal	Hexa-
33			
	1110101		
		703	
			1AF

KESL
Kookmin Univ. Embedded Systems Lab

Powers

Power	Preface	Symbol	Value
10^{-12}	pico	p	.000000000001
10^{-9}	nano	n	.000000001
10^{-6}	micro	µ	.000001
10^{-3}	milli	m	.001
10^{3}	kilo	k	1000
10^{6}	mega	M	1000000
10^{9}	giga	G	1000000000
10^{12}	tera	T	1000000000000

Power	Preface	Symbol	Value
2^{10}	kilo	k	1024
2^{20}	mega	M	1048576
2^{30}	Giga	G	1073741824

Estimating Powers of Two

- What is the value of 2^{24}?

  $2^4 \times 2^{20} \approx 16$ **million**

- How many values can a 32-bit variable represent?

  $2^2 \times 2^{30} \approx 4$ **billion**

KESL
Kookmin Univ. Embedded Systems Lab

이진수 덧셈

A	B	A + B
0	0	0
0	1	1
1	0	1
1	1	10

이진수 덧셈

- Add the following 4-bit binary numbers

$$\begin{array}{r} 1001 \\ +\ \ 0101 \\ \hline \end{array}$$

- Add the following 4-bit binary numbers

$$\begin{array}{r} 1011 \\ +\ \ 0110 \\ \hline \end{array}$$

이진수 덧셈

- Add the following 4-bit binary numbers

$$\begin{array}{r} 1 \\ 1001 \\ +\;\;0101 \\ \hline 1110 \end{array}$$

- Add the following 4-bit binary numbers

$$\begin{array}{r} 111 \\ 1011 \\ +\;\;0110 \\ \hline 10001 \end{array}$$

Overflow!

Overflow

- Digital systems operate on a **fixed number of bits**
- Overflow: when result is too big to fit in the available number of bits
- See previous example of 11 + 6

Exercise

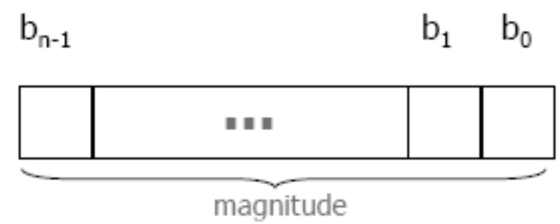
Decimal	Binary	Octal	Hexa-
29.8			
	101.1101		
		3.07	
			C.82

부호 있는 수의 표현

- Decimal
  - Represented by “+” or “-” sign

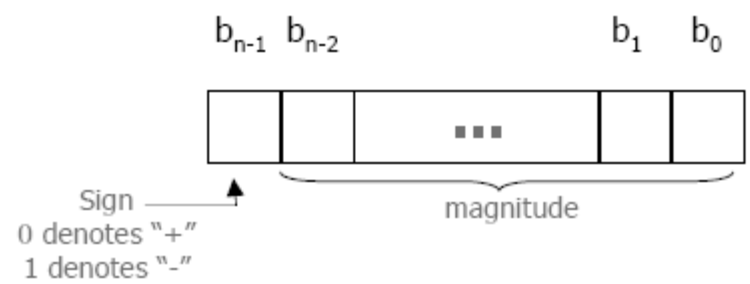
- 100 523 + 3

- Binary
  - Unsigned binary number
    - All bits determine magnitude of number



Unsigned number

  - Signed binary number
    - n-1 bits determine magnitude of number
    - Sign denoted by left most bit
      - 0 indicates positive number
      - 1 indicates negative number



Signed number

부호 있는 수의 표현

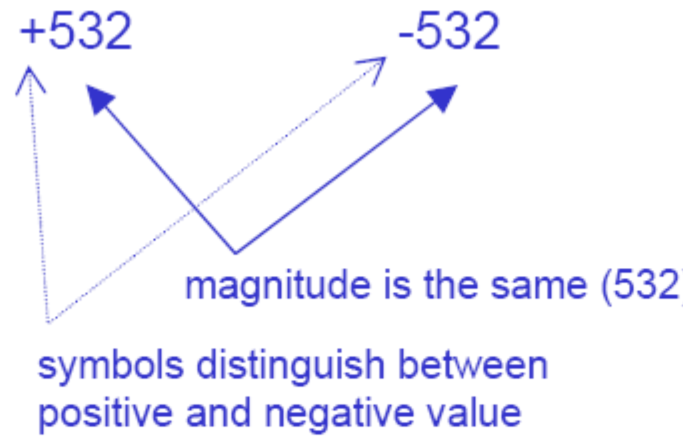
- How do we represent the magnitude?
  - Positive Binary Numbers
    - Represented by position numbering systems previously discussed
  - Negative Binary Numbers
    - Sign-and-Magnitude Representation
    - 1's Complement
    - 2's Complement

		1	0	1
2^4	2^3	2^2	2^1	2^0

$$
\begin{aligned}
101_2 &= (1 * 2^2) + (0 * 2^1) + (1 * 2^0) \\
&= (1 * 4) + (0 * 2) + (1 * 1) \\
&= 5_{10}
\end{aligned}
$$

Sign and Magnitude Representation

- Sign-and-Magnitude Representation
- Decimal representation
  - Magnitude of a number is expressed the same way
  - Symbol distinguishes positive or negative



- Binary can adopt same scheme
  - n-1 bits denote magnitude
  - Leftmost bit denotes positive (0) or negative value (1)
  - Intuitive representation for humans
  - Not well suited for computers

$0101_2 = 5_{10}$

$1101_2 = -5_{10}$

Addition of Sign and Magnitude Numbers

- Addition of sign-and-magnitude integers
- If signs are same
  - Add magnitude values
  - Copy sign
- If signs are different
  - Subtract smaller magnitude from larger magnitud
  - Copy sign of larger magnitude
- Circuitry required
  - Adder
  - Subtractor
  - Compare

$$\begin{array}{rll} & 0101 & (5_{10}) \\ + & 0010 & (2_{10}) \\ \hline & 0111 & (7_{10}) \end{array}$$

$$\begin{array}{rll} & 1011 & (-3_{10}) \\ + & 1011 & (-3_{10}) \\ \hline & 1110 & (-6_{10}) \end{array}$$

$$\begin{array}{rll} & 0111 & (7_{10}) \\ - & 1010 & (2_{10}) \\ \hline & 0101 & (5_{10}) \end{array}$$

1's Complement Representation of Binary Numbers (1의 보수)

- 1's Complement Representation
  - K = n-bit negative number
  - P = corresponding positive number
  - $K = (2^n - 1) - P$

n = 4

Convert +5 (0101_2) to a negative number, using 1's complement

$K = (2^4 - 1) - P$

$K = 15_{10} - P$

$K = 1111_2 - P$

$K = 1111_2 - 0101_2$

$K = 1010_2$

Convert +3 (0011_2) to a negative number, using 1's complement

$K = (2^4 - 1) - P$

$K = 15_{10} - P$

$K = 1111_2 - P$

$K = 1111_2 - 0011_2$

$K = 1100_2$

What we are actually doing is just complementing each of the bits (including sign bit)

Addition of 1's Complement Numbers (1의 보수 덧셈)

- Addition of 1's complement integers
- Consider four possible combination of signs
  - Top two are correct
  - Bottom two are incorrect
    - Carry produced by sign bit
  - If carry produced by sign bit, add it to the LSB
    - New result correct
- Drawback - signed addition may require twice as long as unsigned addition

	0101	$(+5_{10})$
+	0010	$(+2_{10})$
	0111	$(+7_{10})$

	1010	(-5_{10})
+	0010	$(+2_{10})$
	1100	(-3_{10})

	0101	$(+5_{10})$
+	1101	(-2_{10})
	10010	(-13_{10}) x
+	1	
	0011	$(+3_{10})$

	1010	(-5_{10})
+	1101	(-2_{10})
	10111	(-8_{10}) x
+	1	
	1000	(-7_{10})

2's Complement Representation of Binary Numbers (2의 보수)

- 2's Complement Representation
  - K = n-bit negative number
  - P = corresponding positive number
  - $K = 2^n - P$
- Notice value plus it's 2's complement result in 0 (ignoring carry)

$n = 4$

Convert +5 (0101_2) to a negative number, using 2's complement

$K = 2^4 - P$

$K = 16_{10} - P$

$K = 10000_2 - P$

$K = 10000_2 - 0101_2$

$K = 1011_2$

Convert +3 (0011_2) to a negative number, using 2's complement

$K = 2^4 - P$

$K = 16_{10} - P$

$K = 10000_2 - P$

$K = 10000_2 - 0011_2$

$K = 1101_2$

$$\begin{array}{rl} 0101 & (+5_{10}) \\ +\ 1011 & (-5_{10}) \\ \hline 1\ 0000 & \end{array}$$

$$\begin{array}{rl} 0011 & (+3_{10}) \\ +\ 1101 & (-5_{10}) \\ \hline 1\ 0000 & \end{array}$$

KESL Kookmin Univ. Embedded Systems Lab

2's Complement Representation of Signed Binary Numbers (부호 있는 수의 2의 보수)

- 2's Complement conversion shortcut
  - Given signed number, $B = b_{n-1}, b_{n-2}, \ldots, b_1, b_0$
  - Start from right to left, copy all bits that are 0 and the first bit that is 1
  - Complement remaining bits

Convert +6 (0110_2) to a negative number, using 2's complement shortcut

0110

Start from right, copy all 0's until we hit first 1

Copy first bit that is 1

Complement remaining bits

Result: 1 0 1 0

Convert +180 ($0\ 1011\ 0100_2$) to a negative number, using 2's complement shortcut

0 1011 0100

Start from right, copy all 0's until we hit first 1

Copy first bit that is 1

Complement remaining bits

Result: 1 0 1 0 0 1 1 0 0

부호 있는 수 표현의 비교

$b_3b_2b_1b_0$	Sign and Magnitude	1's complement	2's complement
0111	+7	+7	+7
0110	+6	+6	+6
0101	+5	+5	+5
0100	+4	+4	+4
0011	+3	+3	+3
0010	+2	+2	+2
0001	+1	+1	+1
0000	+0	+0	+0
1000	-0	-7	-8
1001	-1	-6	-7
1010	-2	-5	-6
1011	-3	-4	-5
1100	-4	-3	-4
1101	-5	-2	-3
1110	-6	-1	-2
1111	-7	-0	-1

- Multiple zero representations
  - Sign-and-magnitude
  - 1's Complement
- Single zero representation
  - 2's Complement
- Represent numbers from -7 to + 7
  - Sign-and-magnitude
  - 1's Complement
- Represent numbers from -8 to +7
  - 2's Complement

2의 보수의 덧셈

- Addition of 2's complement integers
- Consider four possible combination of signs
  - All are correct
  - If carry produced by sign bit, ignore it
- Circuitry required
  - Adder
- Highly suitable for implementation of addition

$$\begin{array}{rrl} & 0101 & (+5_{10}) \\ + & 0010 & (+2_{10}) \\ \hline & 0111 & (+7_{10}) \end{array} \qquad \begin{array}{rrl} & 1011 & (-5_{10}) \\ + & 0010 & (+2_{10}) \\ \hline & 1100 & (-3_{10}) \end{array}$$

$$\begin{array}{rrl} & 0101 & (+5_{10}) \\ + & 1110 & (-2_{10}) \\ \hline & 10011 & (-3_{10}) \end{array} \qquad \begin{array}{rrl} & 1011 & (-5_{10}) \\ + & 1110 & (-2_{10}) \\ \hline & 11001 & (-7_{10}) \end{array}$$

ignore carry out bit

덧셈을 이용한 뺄셈

- What about subtraction of 2's complement integers?
- We can use an adder to perform subtraction
  - Negate subtrahend, add to the minuend
    - A – B = A + (-B)
    - A – (-B) = A + (+B)

$$D = X - Y$$

subtrahend

minuend

덧셈을 이용한 뺄셈

- Consider four possible combination of signs
  - All are correct
  - If carry produced by sign bit, ignore it
- Using 2's complement representation of negative numbers
  - Only 1 adder required to perform addition and subtraction

= 5 – 2
= 5 + (-2)
= 3

	0101	$(+5_{10})$
+	1110	(-2_{10})
	10011	$(+3_{10})$

ignore

= (-5) – 2
= (-5) + (-2)
= -7

	1011	(-5_{10})
+	1110	(-2_{10})
	11001	(-7_{10})

ignore

= 5 – (-2)
= 5 + 2
= 7

	0101	$(+5_{10})$
+	0010	$(+2_{10})$
	0111	$(+7_{10})$

= (-5) – (-2)
= (-5) + 2
= - 3

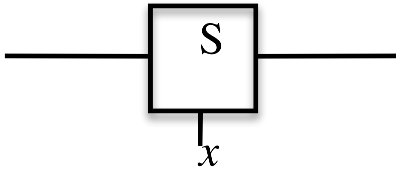
	1011	(-5_{10})
+	0010	$(+2_{10})$
	1100	(-3_{10})

Today's Objective

Logic의 완전 기초

Binary Switch

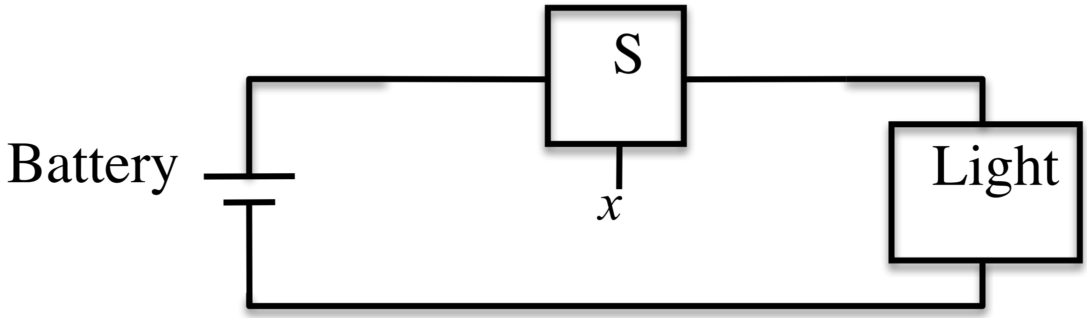
(a) Two states of a switch



(b) Symbol for a switch

Binary Switch Example

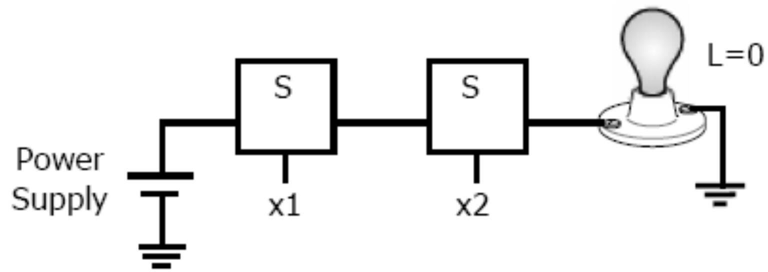
- Output
  - State of the light : L =1 if the light is on, L=0 (off)
  - The state of the light can be described as a function of the input variable x
- L(x) = x : logic function

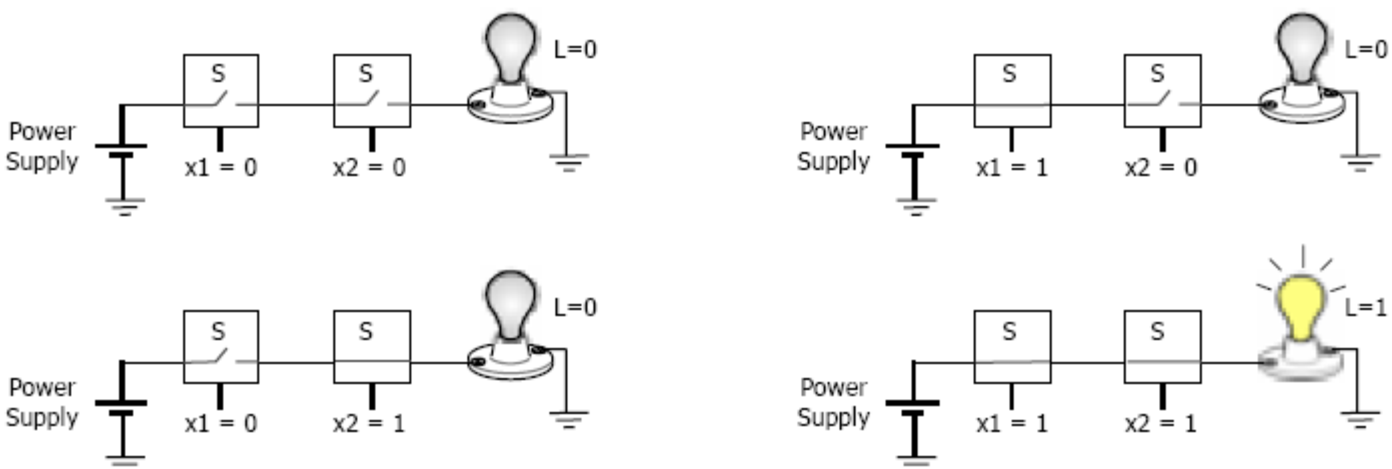


(a) Simple connection to a battery

Two Switch Examples

- When will the light be on?





KESL
Kookmin Univ. Embedded Systems Lab

Logical AND Function

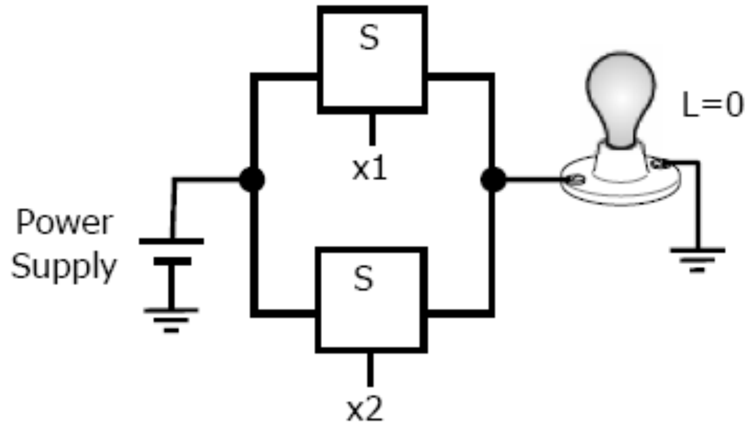
- We observed
  - Light on only when both switches are closed
  - If either switch open, light is off
- Logical expression to describe behavior
  - $L(x1, x2) = x1 \cdot x2$
- Function evaluates as follows
  - $L = 1$, if $x1 = 1$ and $x2 = 1$
  - $L = 0$, otherwise

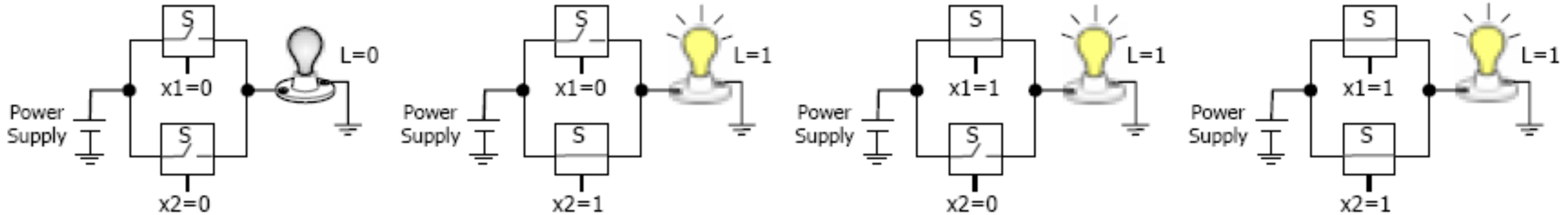
$L(x1, x2) = x1 \cdot x2$

"·" symbol is called *AND operator*
implements the *logical AND function*

Two Switch Example - Parallel

- When will the light be on?





Logical OR Function

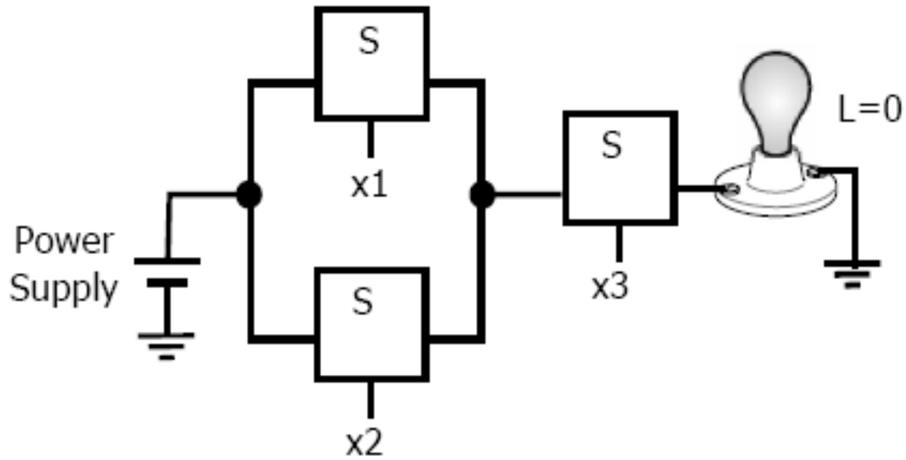
- Logical expression to describe behavior
  - $L(x1, x2) = x1 + x2$
  - Function evaluates as follows
    - L = 0, if x1 = 0 and x2 = 0
    - L = 1, otherwise

$$L(x1, x2) = x1 + x2$$

"+" symbol is called *OR operator*
implements the *logical OR function*

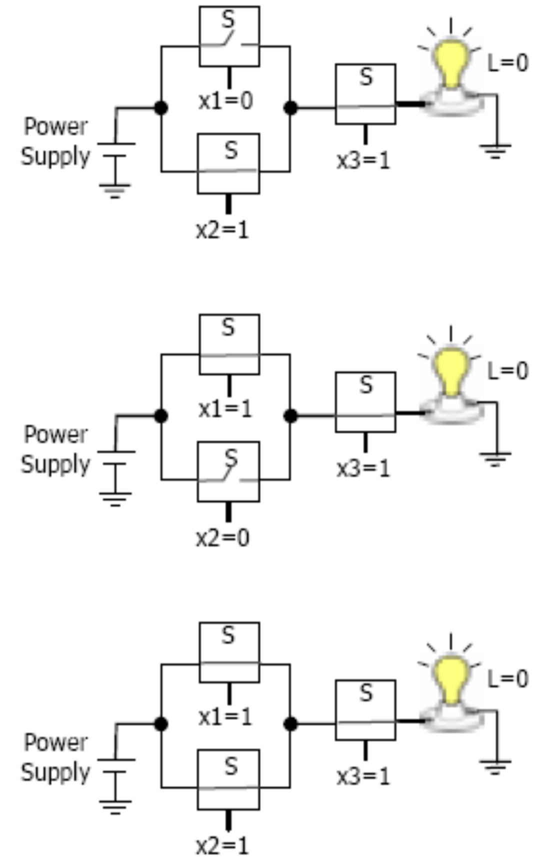
Three Switch Example

- AND and OR functions
  - Building blocks for larger circuits
- When will the light be on?



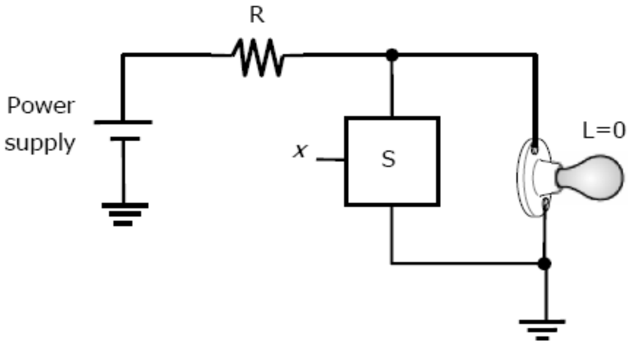
AND/OR Function

- Logical expression to describe behavior
  – L(x1, x2, x3) = (x1 + x2) · x3



Inversion

- What about light's turning on when the switch is closed?
- Logical expression to describe behavior
  - L(x) = ~x
  - Function evaluates as follows
    - L = 1, if x = 0
    - L = 0, if x = 1



$L(x) = \overline{x}$

overbar indicates complement
implements the *NOT function*

Inversion – NOT Operation

- Different names for NOT operation
  - Complement, Invert, Inverse
- Different notations for NOT operation
  - $\overline{x} = x' = !x = \sim x$
- NOT operation can be applied to a single variable or multiple variables
  - $F(x) = x'$
  - $F(x) = x' + a$
  - $F(x) = \overline{x1 + x2} = (x1 + x2)' = !(x1 + x2) = \sim(x1 + x2)$
- NOT operation can be applied to a function
  - If $F(x) = x1 + x2 + x3$
  - Complement of $F(x)$ is $F'(x) = (x1 + x2 + x3)'$

수학 아닌 Logic!!

Truth Table (진리표)

- To show all the possible logical results for all inputs
- Same operations can be defined in the form of a truth table

x1	x2	x1·x2	x1+x2
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

Left Side:
All possible combinations for input values

Right Side:
Values for outputs

Truth table for AND Operation

Truth table for OR Operation

Truth Table (진리표)

x1	x2	x1·x2	x1+x2
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

Truth tables can have single or multiple outputs

Truth tables can be formatted in different ways

F = a · b

a	b	F
0	0	0
0	1	0
1	0	0
1	1	1

G = input1 + t

input1	t	G
0	0	0
0	1	1
1	0	1
1	1	1

진리표의 한계

- Advantages
  – Only one truth table
  – Intuitive to read
- Disadvantages
  – Size explosion

a	b	F
0	0	0
0	1	0
1	0	1
1	1	1

2-input truth table

a	b	c	F
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

3-input truth table

a	b	c	d	F
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
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

4-input truth table

5+-input truth table

진리표의 확장

- AND and OR functions can be extended to n-inputs
  - AND function with n-inputs is equal to 1 only if all n inputs are 1
  - OR function with n-input is equal to 1 if at least one, or more n inputs are 1

$F = a \cdot b \cdot c$

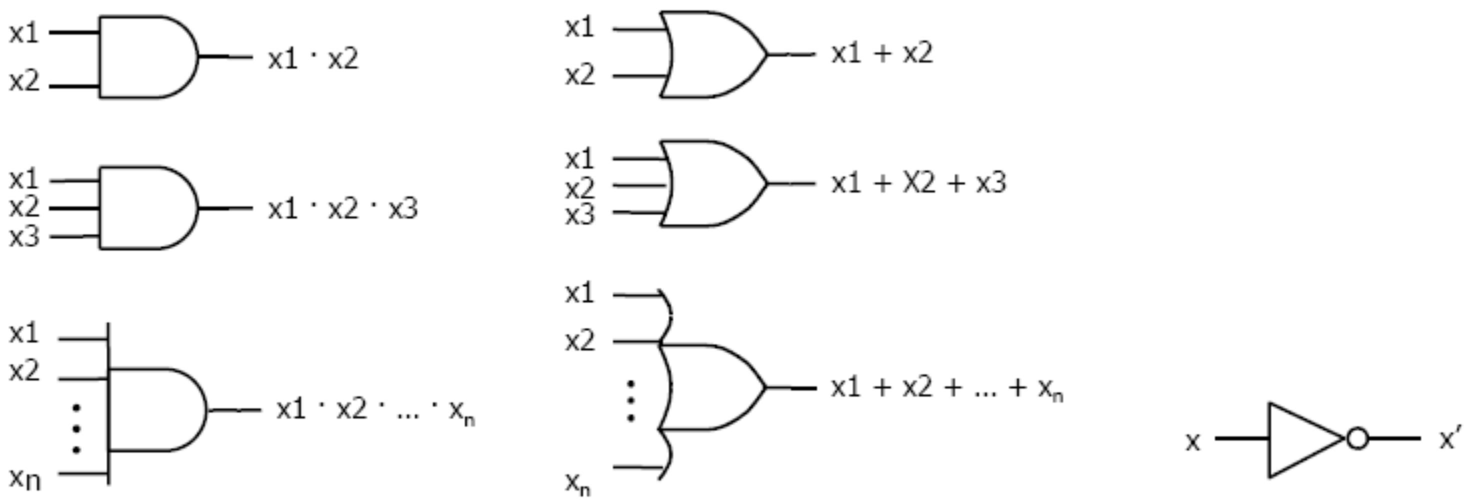
a	b	c	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

$F = a + b + c$

a	b	c	F
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

Logic Gates and Networks

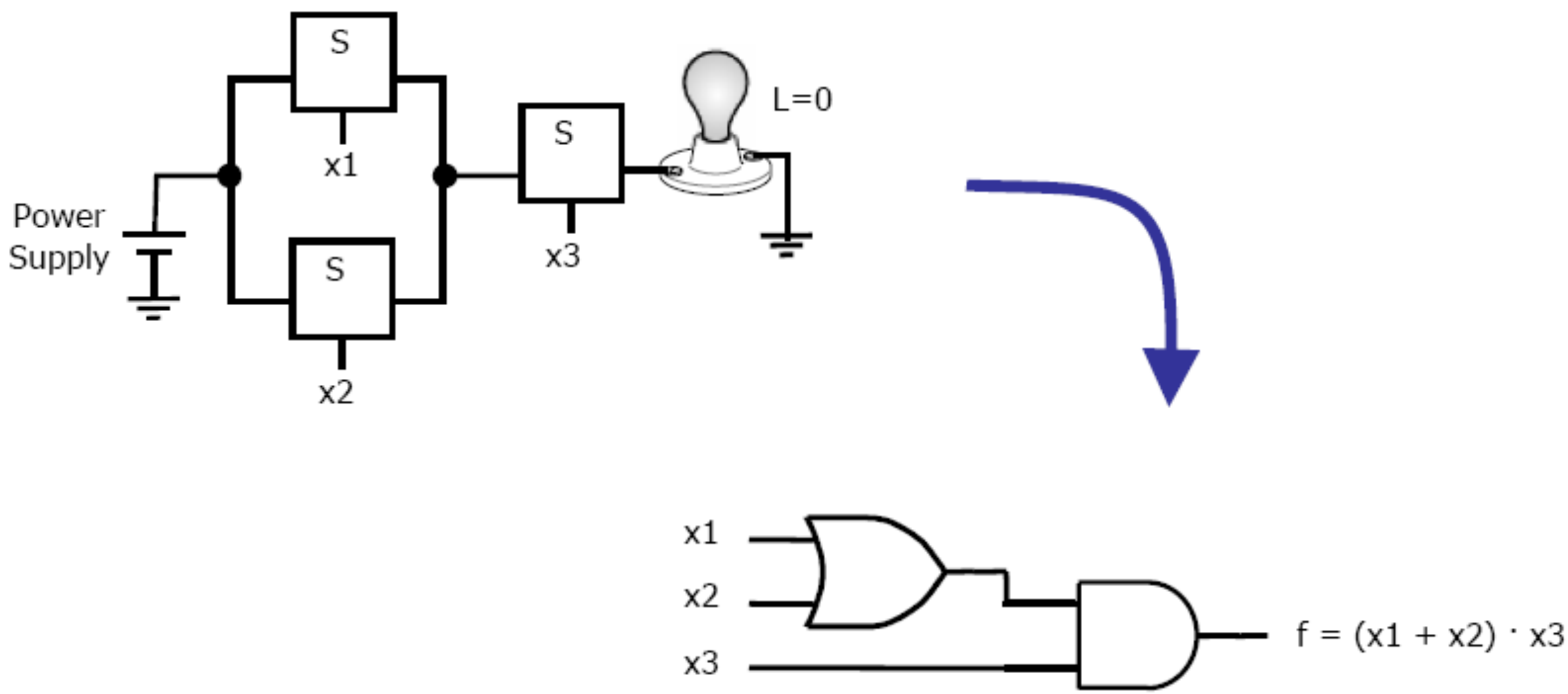
- AND, OR, and NOT operations can be implemented electronically with transistors
- Resulting circuit is a logic gate
  - One or more inputs
  - One output, function of its inputs
- Logic circuit often described graphically in a circuit diagram or schematic
  - Consists of graphical symbols for the AND, OR, and NOT gates



(a) AND gates (b) OR gates (c) NOT gate

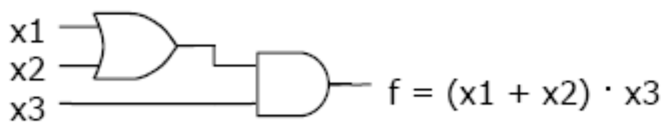
Logic Gates and Networks

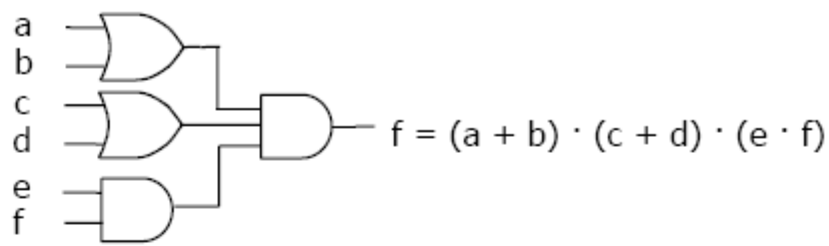
- Larger circuits are implemented by a network of gates
  - Logic function previously constructed using switches can be implemented by a network of gates

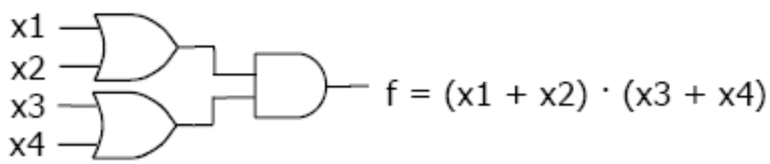


무엇이 중요한가?

- Cost and Complexity!!
  - Reduce the cost always.
- Network of gates
  - Also called logic network, logic circuit, circuit



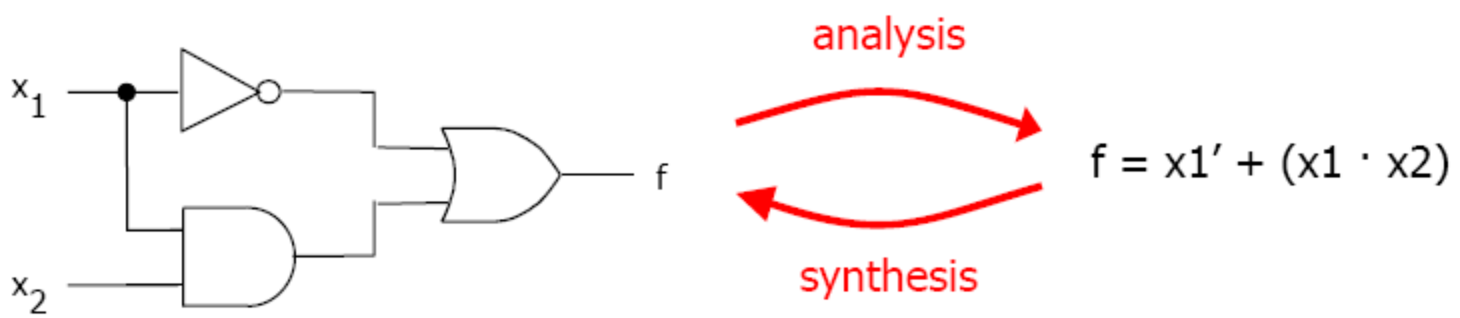




KESL
Kookmin Univ. Embedded Systems Lab

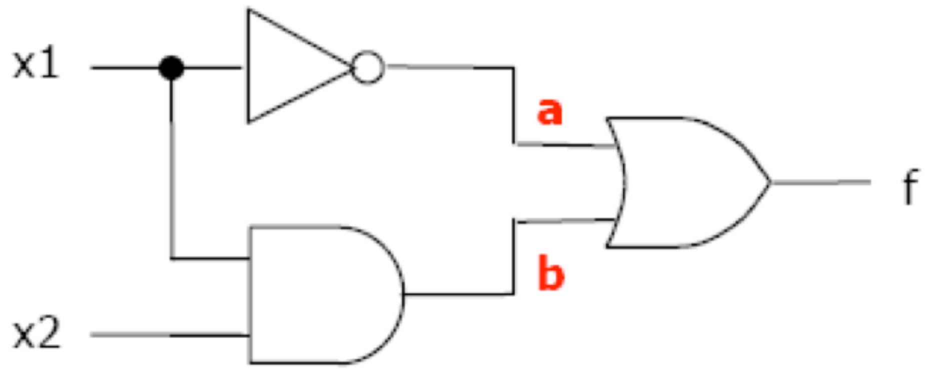
디지털 설계 전문가가 하는 일

- Digital system designer faced with two basic issues
  - Determine function of an existing logic network – Analysis
  - Designing a logic network to implement the desired function – Synthesis



Analysis of a Logic Network

- Logic circuit analysis
  - conversion to logical expression



f =

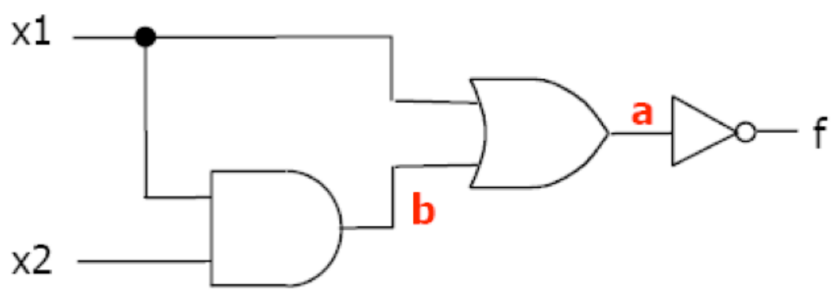
f = a + b

f = x1′ + b

f = x1′ + (x1 · x2)

Analysis of a Logic Network

- Logic circuit analysis
  - conversion to logical expression



f =

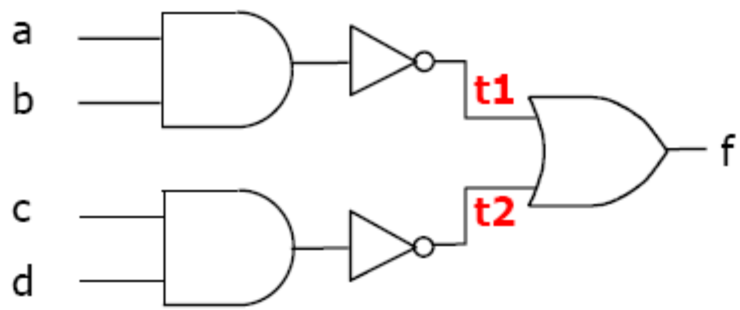
f = a′

f = (x1 + b)′

f = (x1 + (x1 · x2))′

Analysis of a Logic Network

- Convert logic circuit to logical expression



$f =$

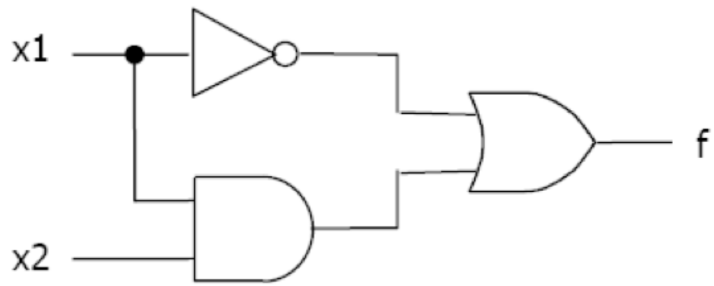
$f = t1 + t2$

$f = (a \cdot b)' + t2$

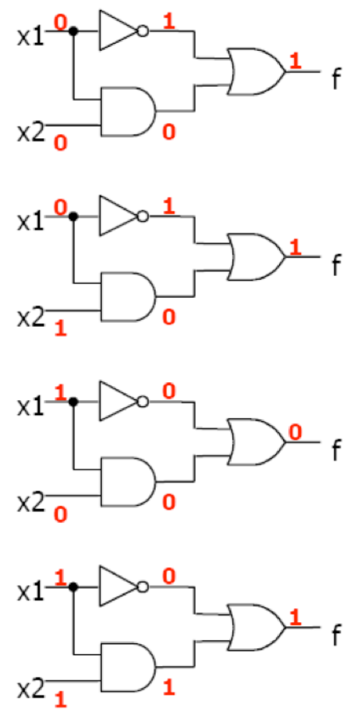
$f = (a \cdot b)' + (c \cdot d)'$

Analysis of a Logic Network

- Another way to specify circuit functionality – truth table



$$f = x1' + (x1 \cdot x2)$$



NOT

a	F
0	1
1	0

AND

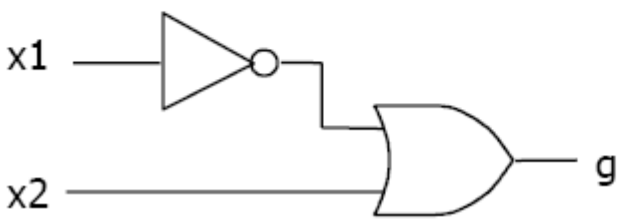
a	b	F
0	0	0
0	1	0
1	0	0
1	1	1

OR

a	b	F
0	0	0
0	1	1
1	0	1
1	1	1

x1	x2	F
0	0	1
0	1	1
1	0	0
1	1	1

Logic Circuit Analysis



$$g = x1' + x2$$

NOT

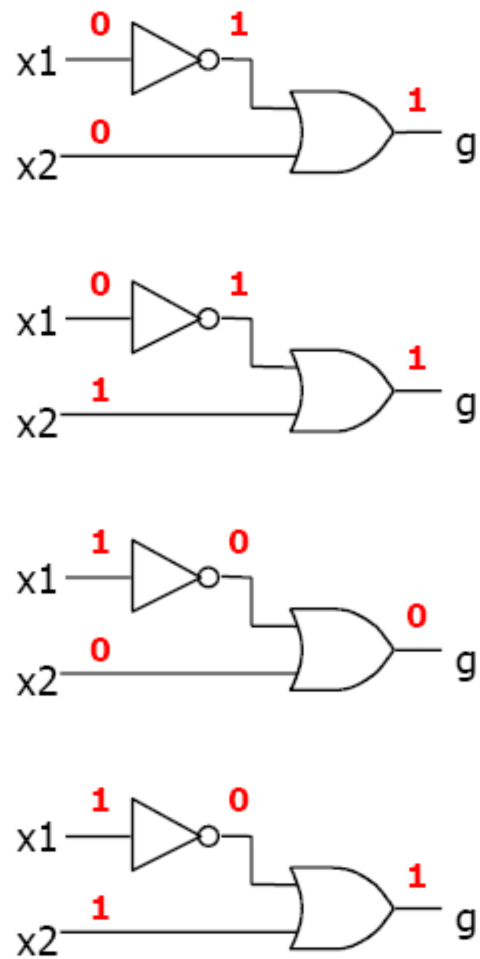
a	F
0	1
1	0

AND

a	b	F
0	0	0
0	1	0
1	0	0
1	1	1

OR

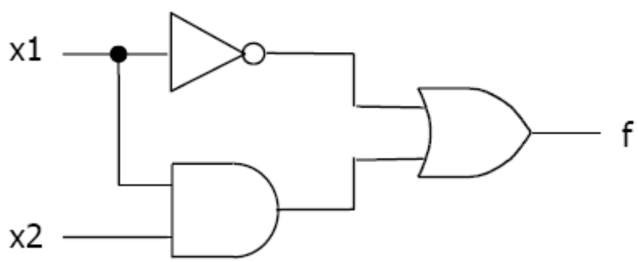
a	b	F
0	0	0
0	1	1
1	0	1
1	1	1



x1	x2	G
0	0	1
0	1	1
1	0	0
1	1	1

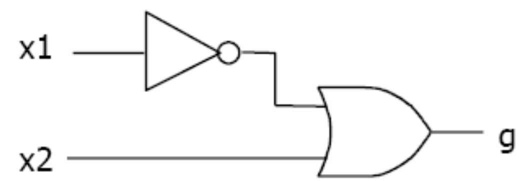
Functional Equivalence

- You may have noticed we got the same truth table for both examples
  - Logic circuits are functionally equivalent
- Logic function can be implemented in a variety of ways
  - Designer wants to use the simpler one – lower cost



$f = x1' + (x1 \cdot x2)$

x1	x2	F
0	0	1
0	1	1
1	0	0
1	1	1

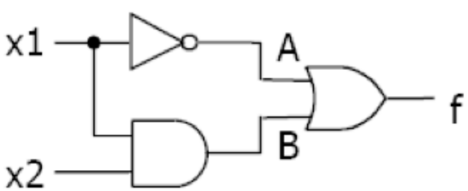


$g = x1' + x2$

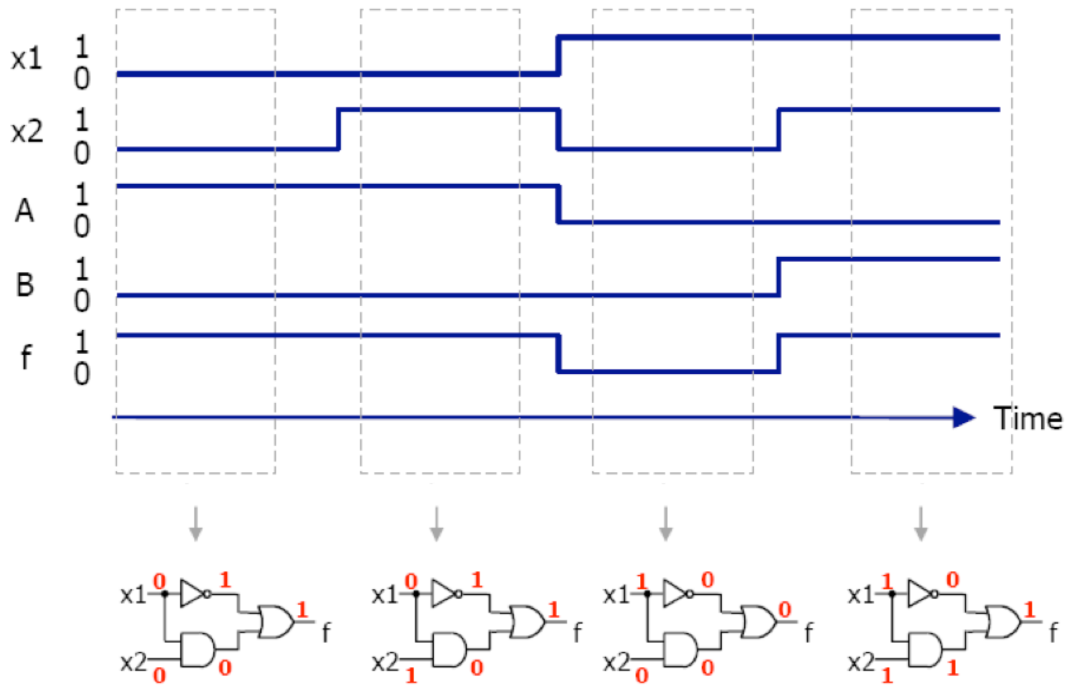
x1	x2	G
0	0	1
0	1	1
1	0	0
1	1	1

Timing Diagram

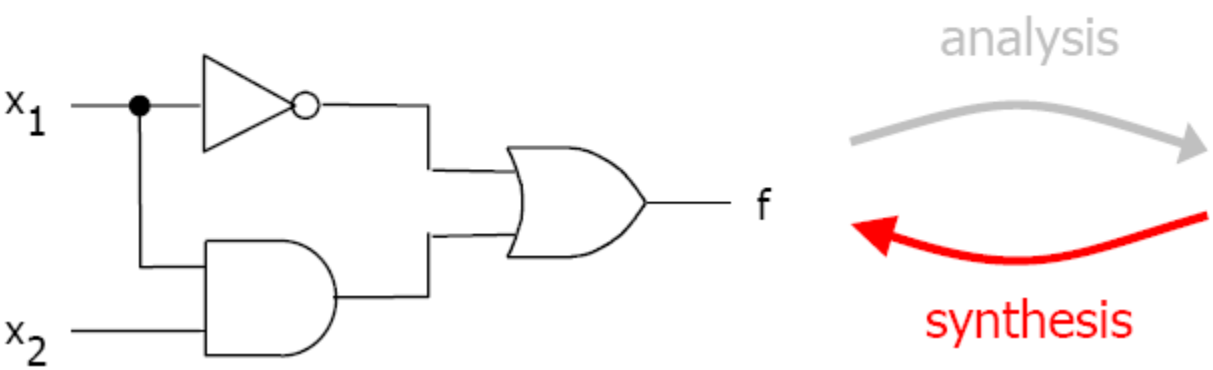
- Yet another way to describe logic circuit behavior -- timing diagram
  - Time runs from left to right
  - Waveform shown indicating inputs, output, and internal signal values



x1	x2	F
0	0	1
0	1	1
1	0	0
1	1	1



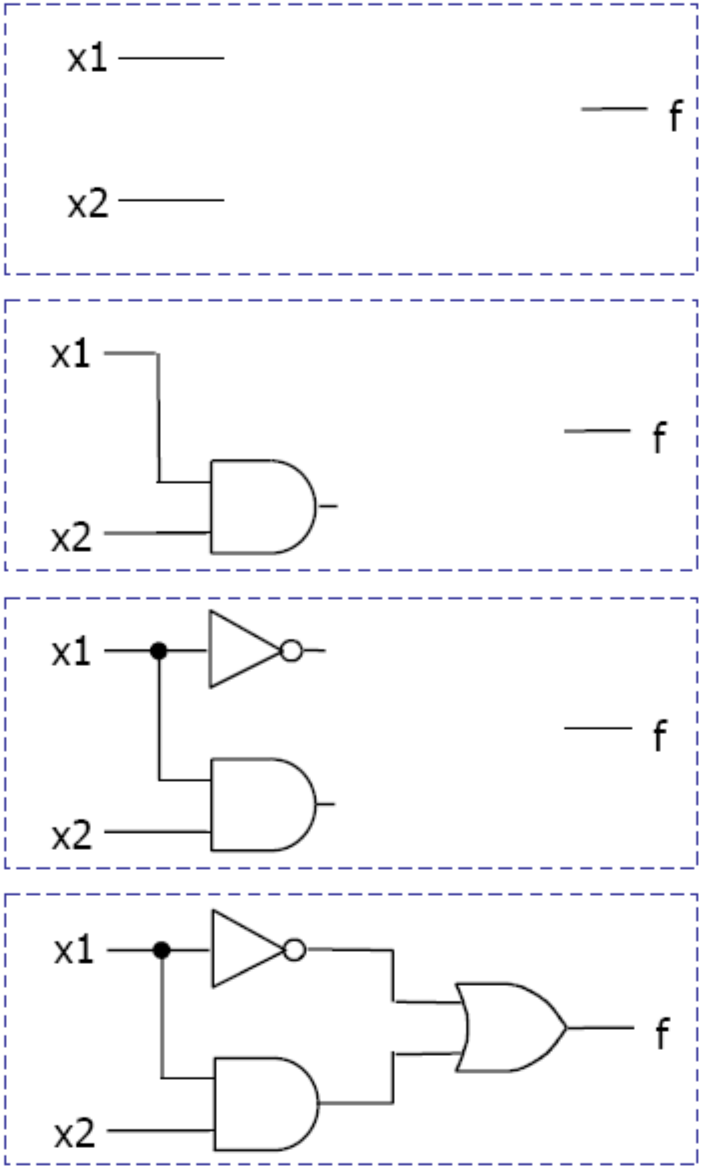
Synthesis of a Logic Network



$f = x1' + (x1 \cdot x2)$

Synthesis of a Logic Network

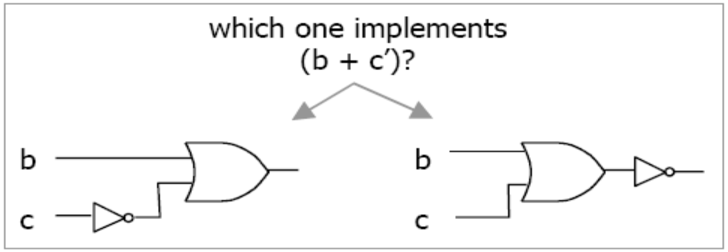
- Convert f= x1’ + (x1 · x2) to a logic circuit
  - What are your inputs?
    - x1, x2
  - What are is your output(s)?
    - f
  - How is the function evaluated?
    - Parentheses
    - NOT operation
    - OR operation



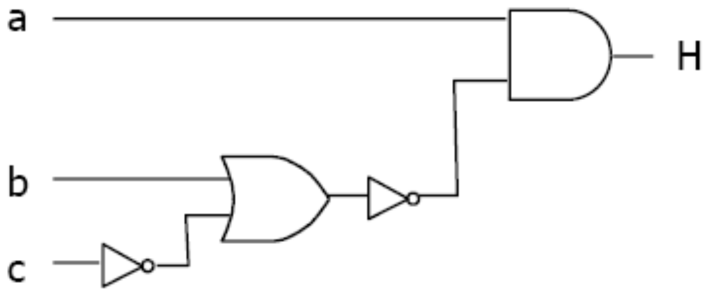
Symbol	Name	Description
( )	Parentheses	Evaluate expression nested in parentheses first
′	NOT	Evaluate from left to right
.	AND	Evaluate from left to right
+	OR	Evaluate from left to right

Synthesis of a Logic Network

- Convert H = a · (b + c')' to a logic circuit



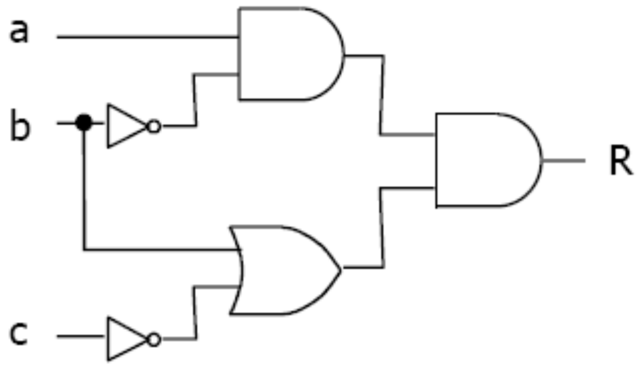
Symbol	Name	Description
( )	Parentheses	Evaluate expression nested in parentheses first
′	NOT	Evaluate from left to right
·	AND	Evaluate from left to right
+	OR	Evaluate from left to right



Synthesis of a Logic Network

- Convert $R = (a \cdot b') \cdot (b + c')$ to a logic circuit

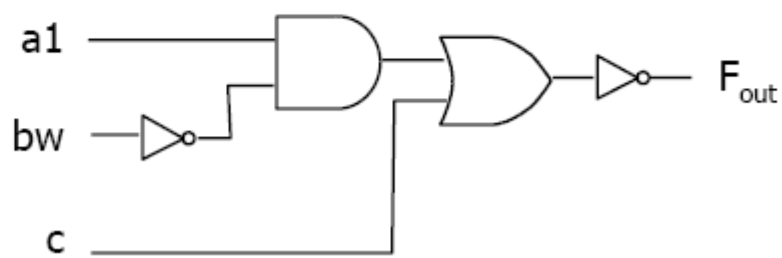
Symbol	Name	Description
( )	Parentheses	Evaluate expression nested in parentheses first
′	NOT	Evaluate from left to right
.	AND	Evaluate from left to right
+	OR	Evaluate from left to right



Synthesis of a Logic Network

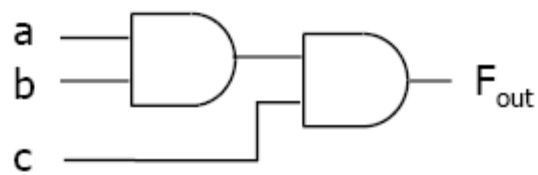
- Convert Fout = (a1 · bw' + c)' to a logic circuit
  - Expression inside parentheses first
  - NOT, AND, or OR operation?
  - NOT operation

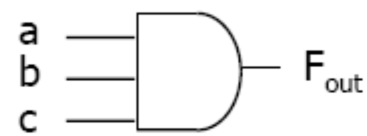
Symbol	Name	Description
( )	Parentheses	Evaluate expression nested in parentheses first
'	NOT	Evaluate from left to right
.	AND	Evaluate from left to right
+	OR	Evaluate from left to right



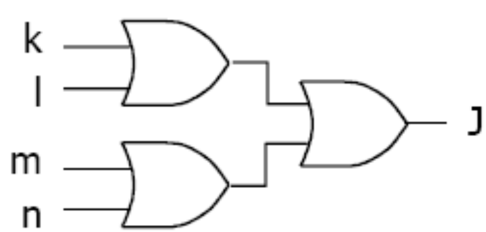
Synthesis of a Logic Network

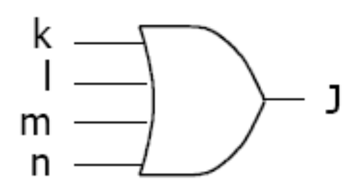
- Convert Fout = (a · b · c) to a logic circuit
  - How do we implement a 3-input gate?
  - Using 2-input AND gates
  - Using 3-input AND gates





- Same for OR gates
  - J = k + l + m + n





Circuit Drawing Conventions