

3. Logic Gates and Boolean Algebra

Digital circuits can be found in smart phones, computers, washing machines, cars, etc.

Logic gates are the basic building blocks of digital circuits.

Boolean algebra is used to describe, design and simplify digital circuits.

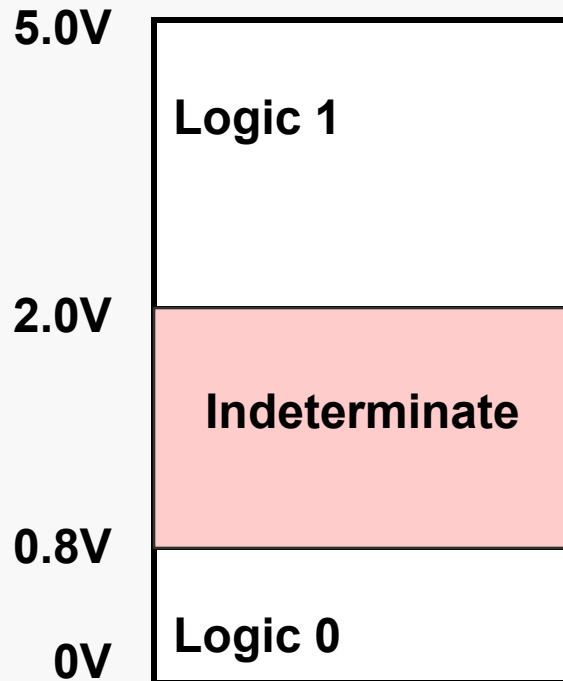
Boolean Constants – only 2 values

- **TRUE, FALSE**
- **Logic HIGH, Logic LOW**
- **HI, LO**
- **1, 0**

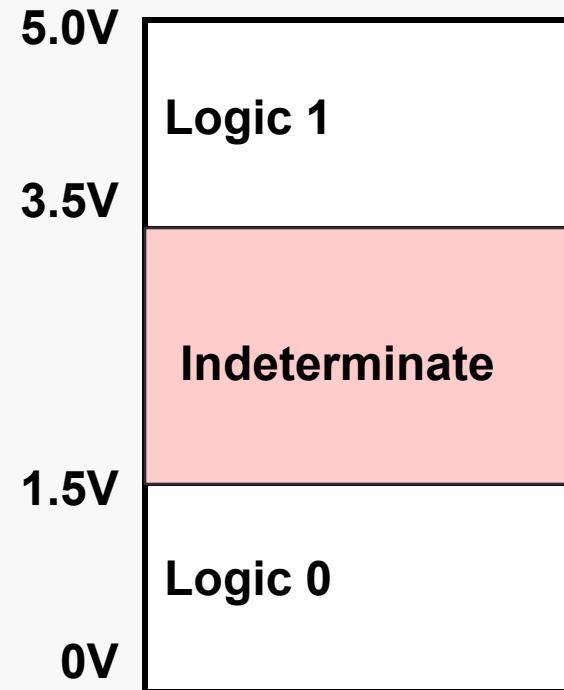
Boolean (logic) variables can only assume one of the two values

Common Logic-level voltage ranges

TTL



CMOS (74AC)

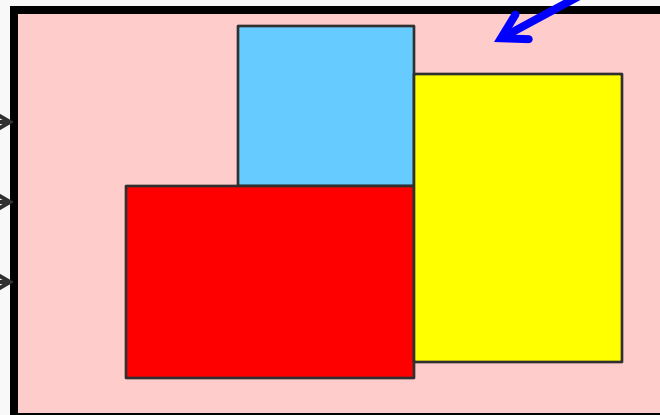


A typical logic circuit

- 1 or more logic inputs
- 1 or more logic outputs
- Outputs are related to inputs by logic functions

inputs:

X
Y
Z



Smaller logic
circuit blocks

output:

F

Truth table

- Logic function can be fully described by a **Truth Table**.
- The Truth table
 - shows how a logic circuit's output responds to various combinations of logic inputs
 - has 2^N number of input combinations for N inputs
 - lists all possible input combinations in the binary counting sequence

A typical N-input truth table

2^N
rows

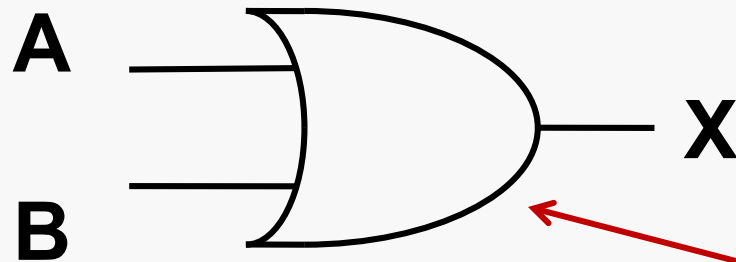
Input 1	Input 2	...	Input N-1	Input N	Output
0	0	...	0	0	1
0	0	...	0	1	0
0	1	0	0
0	1	1	.
.
.
1	0	0	.
1	0	1	.
1	1	...	1	0	.
1	1	...	1	1	.

3 Basic Logic Operations

- **Logical addition - OR**
- **Logical multiplication - AND**
- **Logical complement or inversion - NOT**

In digital circuits, these are realised by electronic devices called logic gates.

Logical OR Operation



Logic
symbol

- Logical OR operation

- $X = A + B$

- $X = A \text{ OR } B$

Logic
expression

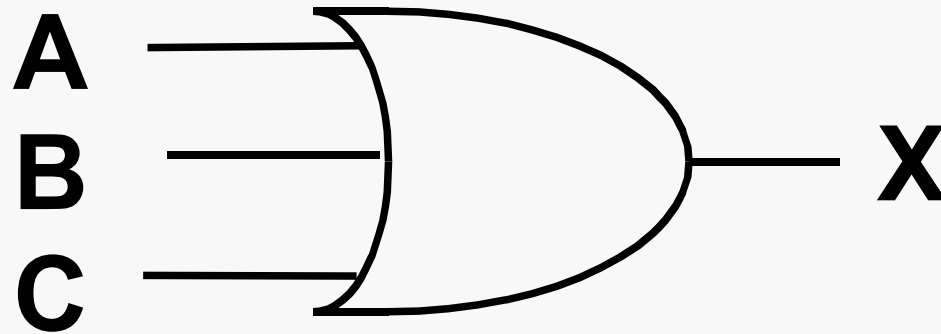
- 2-input OR gate: $X = A + B$

Truth table for a 2-input OR gate

$(X = A+B)$

Inputs		Output
A	B	X
0	0	0
0	1	1
1	0	1
1	1	1

X=1 if at least one input is = 1



3-input OR gate:

$$X = A + B + C$$

OR operation result will be 1 if at least one input is 1

Truth Table for a 3-input OR gate

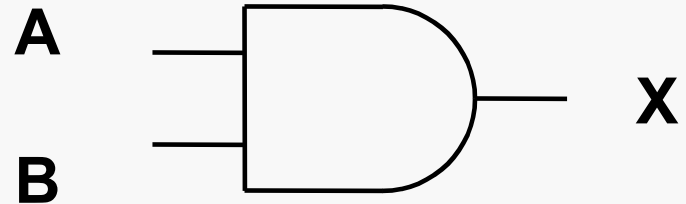
$(X = A+B+C)$

Inputs			Output
A	B	C	X
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

Logical AND operation

Logical AND operation

- $X = A \bullet B$
- $X = A \text{ AND } B$
- $X = AB$

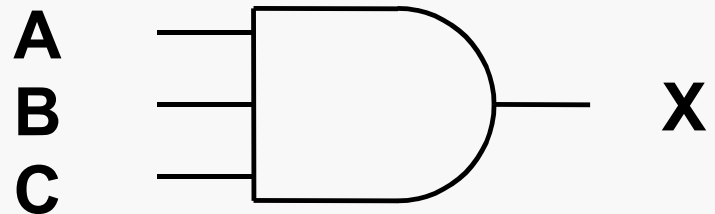


2-input AND gate:

- $X = AB$

3-input AND gate:

- $X = ABC$



AND operation result will be 1 only if all inputs are 1

Truth table for 2-input AND gate

$(X = AB)$

Inputs		Output
A	B	X
0	0	0
0	1	0
1	0	0
1	1	1

Output X=1 only if all inputs are 1

The Truth Table for a 3-input AND gate ($X = ABC$)

Inputs			Output
A	B	C	X
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

Logical NOT operation

- NOT gate only has 1 input and it is commonly known as an **inverter**
- the output is the complement/inverse of the input

- $X = \overline{A}$



- $X = A'$ ← we will use this notation

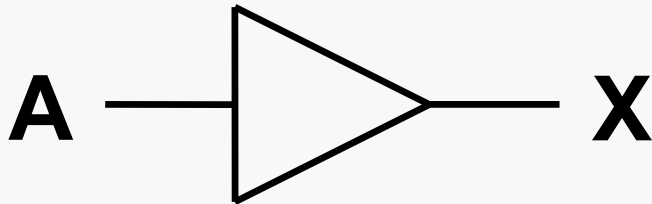
Logical NOT operation

- Its truth table is very simple

input	output
A	$X = A'$
0	1
1	0

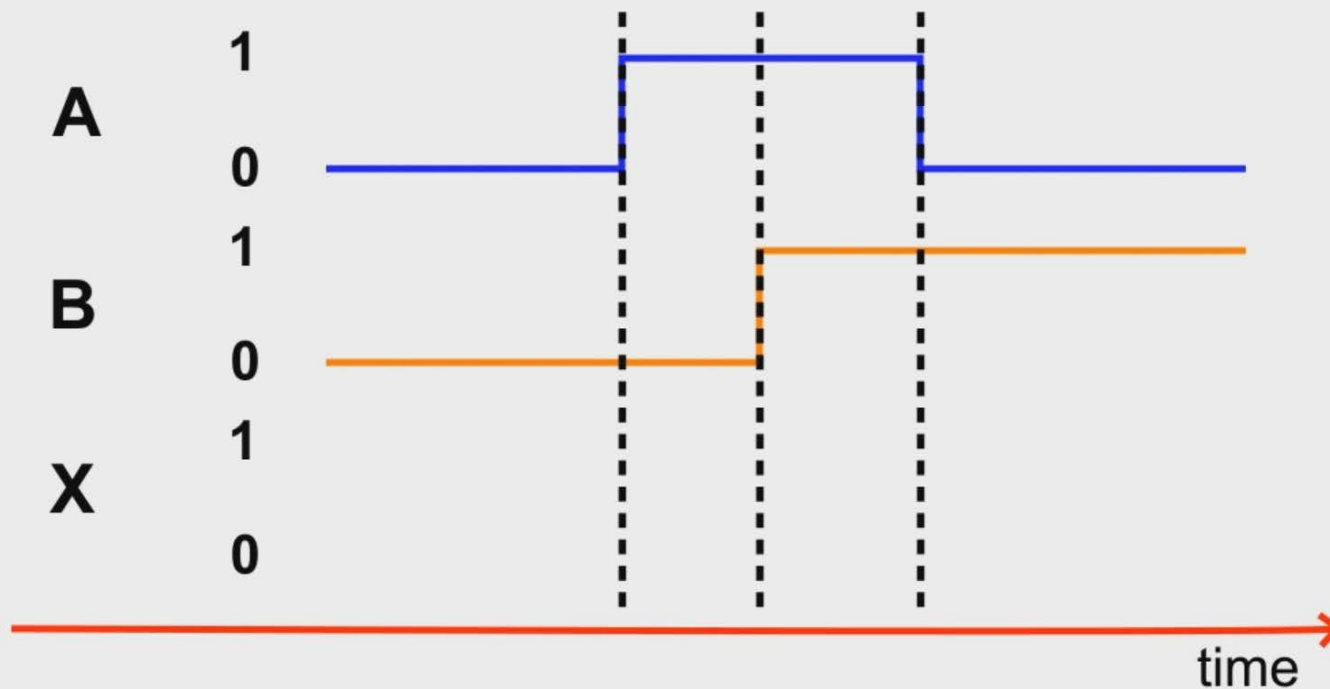
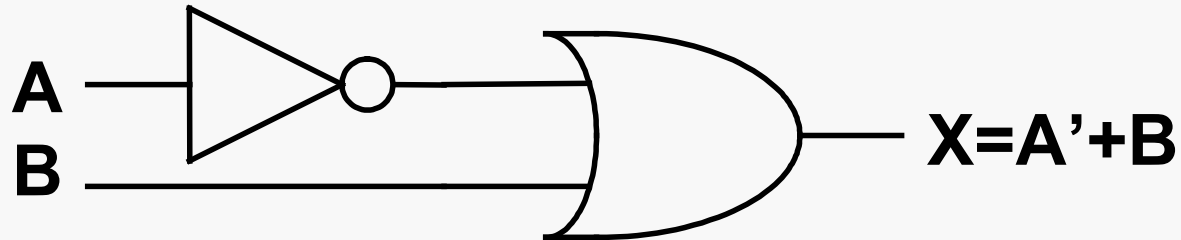
Buffer

- Its truth table is also very simple
- No change in logic



input	output
A	$X = A$
0	0
1	1

- **Example: Sketch the logic waveform of X**



- **Timing diagram with more realistic appearance, showing propagation delays, rise time and fall time**

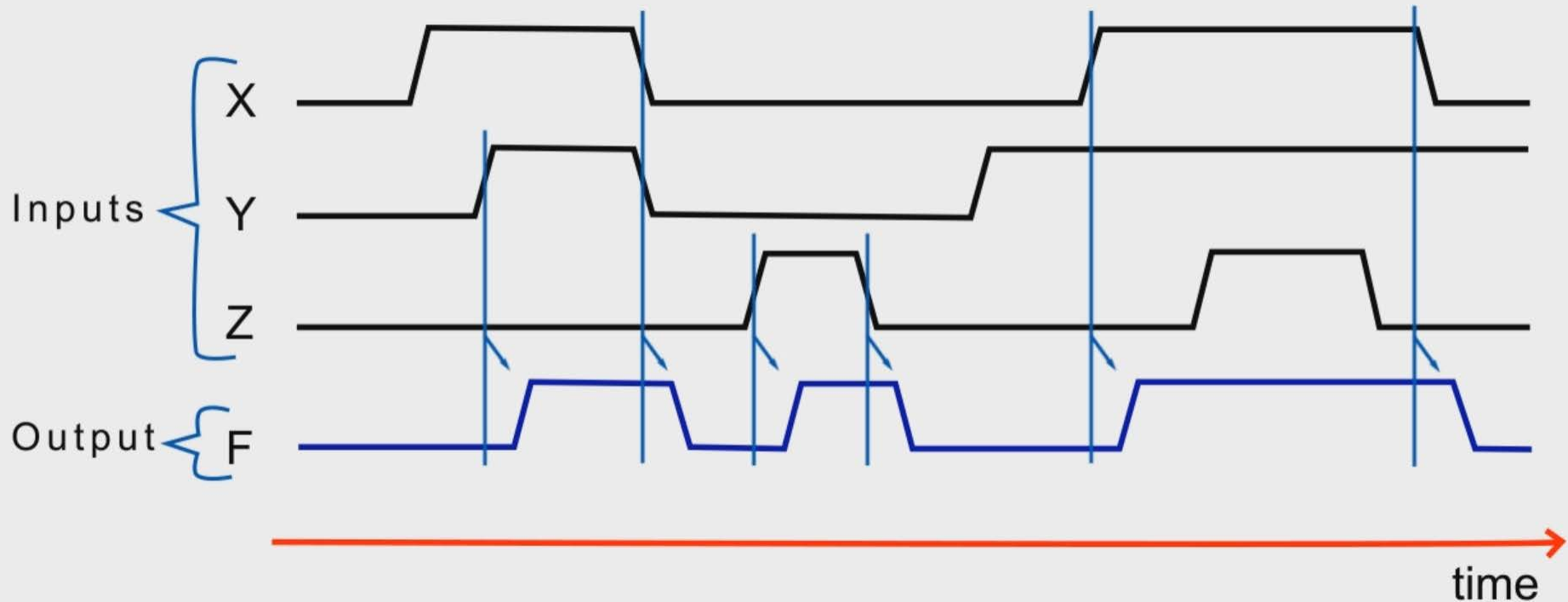


Fig. 3.17 (taken from Wakerly)

Boolean Algebra

- **Helps to analyse logic circuits.**
- **Express operations mathematically.**
- **Similar to normal algebra but much simpler.**
- **It does not have fraction or negative number.**

Order of Precedence in Boolean Algebra:

- Complement over a single variable (inversion)
- Expression within parentheses
- AND
- OR

Examples:

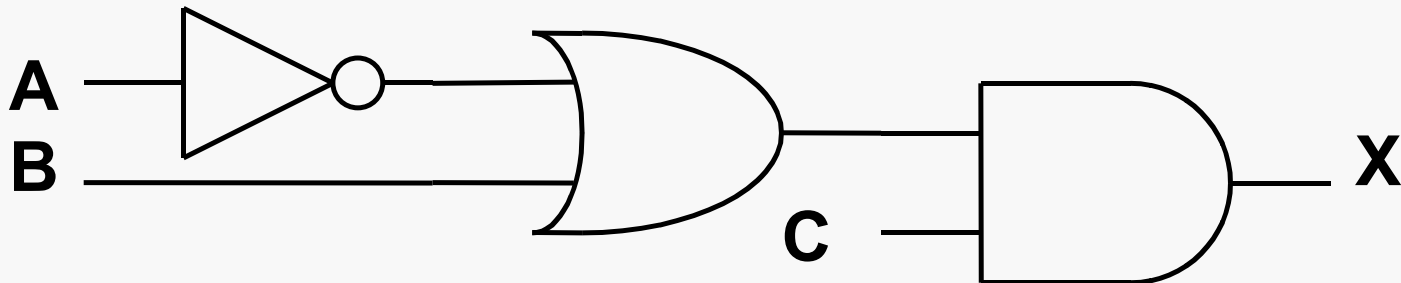
1. $Y = A + B C'$

2. $Y = (A + B) C'$

3. $Y = A + (B C)'$

Describing logic circuits algebraically

- **AND, OR and NOT operations**
 - are basic building blocks of digital system
 - can completely describe any logic circuit
- **Example: express output X in terms of inputs A, B and C**



Evaluating logic circuit outputs

From a Boolean expression, the logic level of an output can be determined for any values of the circuit inputs.

Example

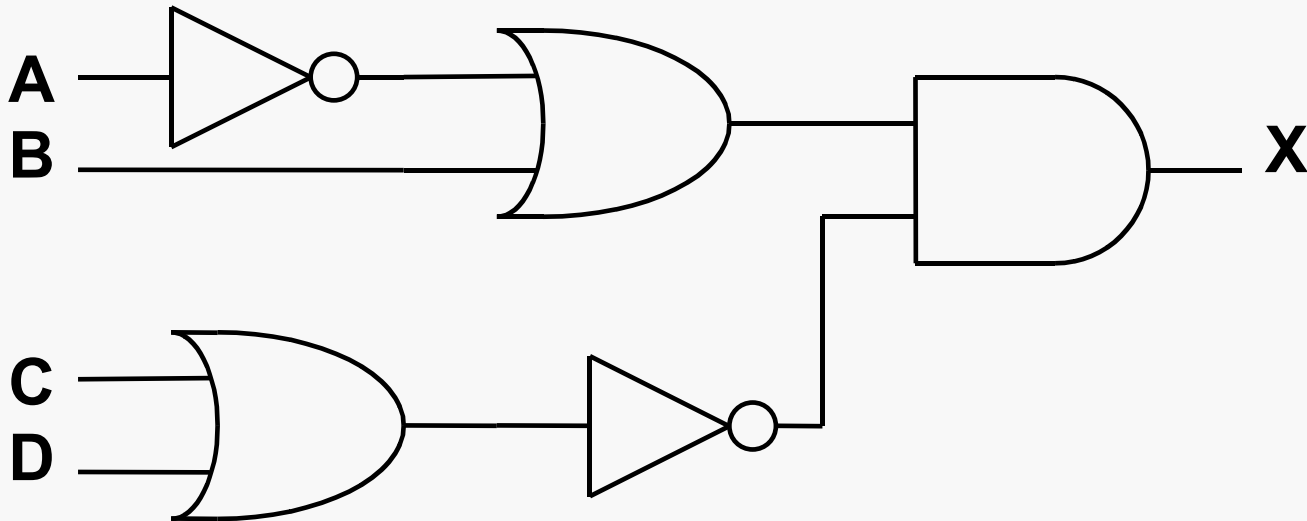
$$X = A'(B+C)(A+D)'$$

**If inputs A,B,C,D = 0,1,1,0
X = ?**

**If inputs A,B,C,D = 1,1,1,1
X = ?**

**If inputs A,B,C,D = 0,0,0,0
X = ?**

Determining Instantaneous Output Level from a Logic Circuit Diagram



e.g. if $A=1$, $B=1$, $C=0$, $D=0$, then $X=?$

Implementing Circuits from Boolean expressions

Example $Y = AC + BC' + A'BC$

Boolean Theorems

- Many of the theorems are similar to those in normal algebra.
- The theorems can be used to simplify logic expressions and therefore can help to simplify logic circuits.
- Simpler circuits cost less to build and are less prone to failure.

Boolean Theorems

Axioms:

$$X = 0 \text{ if } X \neq 1$$

$$X = 1 \text{ if } X \neq 0$$

$$0 \cdot 0 = 0$$

$$1 \cdot 1 = 1$$

$$0 \cdot 1 = 1 \cdot 0 = 0$$

$$1 + 1 = 1$$

$$0 + 0 = 0$$

$$1 + 0 = 0 + 1 = 1$$

Single variable theorems:

$$X \bullet 0 = 0$$

$$X \bullet 1 = X$$

$$X \bullet X = X$$

$$X \bullet X' = 0$$

$$X + 1 = 1$$

$$X + 0 = X$$

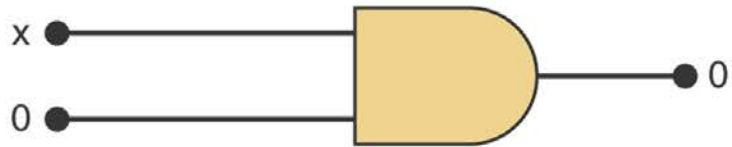
$$X + X = X$$

$$X + X' = 1$$

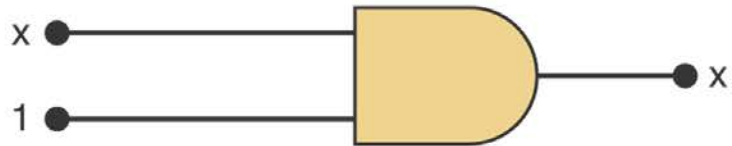
$$(X')' = X$$

Duality: any theorem or identity in switching algebra remains true if 0 and 1 are swapped and \cdot and $+$ are swapped throughout

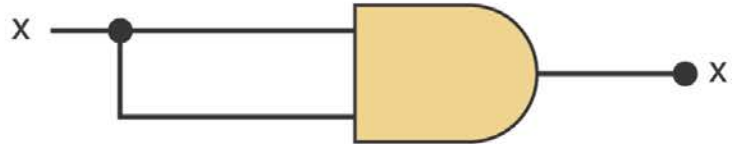
Single variable theorems:



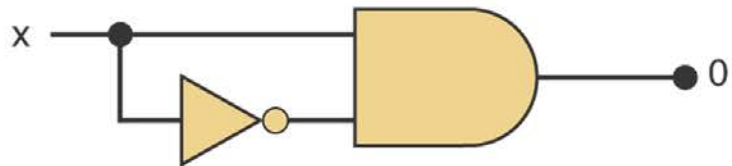
(1) $x \cdot 0 = 0$



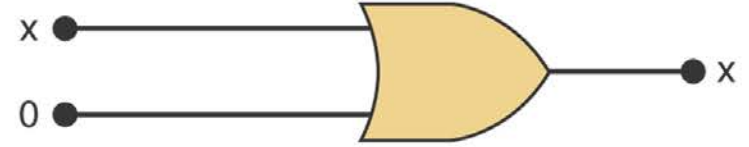
(2) $x \cdot 1 = x$



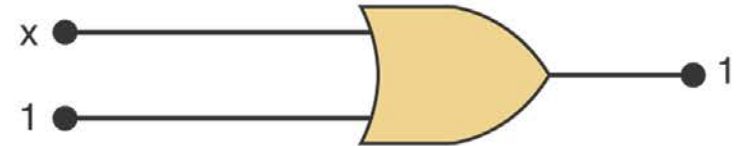
(3) $x \cdot x = x$



(4) $x \cdot \bar{x} = 0$



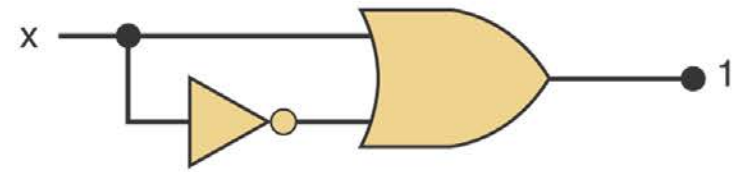
(5) $x + 0 = x$



(6) $x + 1 = 1$



(7) $x + x = x$



(8) $x + \bar{x} = 1$

Fig. 3-25 (Tocci, 10th ed. Pg. 77)

Multivariable theorems:

- **Commutative laws:**

$$A + B = B + A$$

$$A \bullet B = B \bullet A$$

- **Associative laws:**

$$A + (B + C) = (A + B) + C = A + B + C$$

$$A(BC) = (AB)C = ABC$$

- **Distributive laws:**

$$A(B + C) = AB + AC$$

$$(A + B)(C + D) = AC + BC + AD + BD$$

Absorption laws

$$A + AB = A$$

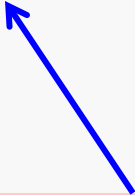
proof:

$$\begin{aligned} A + AB &= A(1 + B) \\ &= A \bullet 1 = A \end{aligned}$$

$$A + A'B = A + B$$

proof:

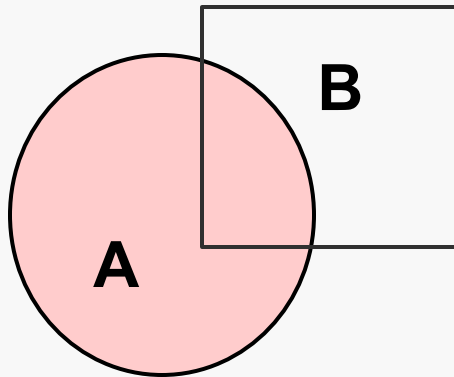
$$\begin{aligned} A + A'B &= A + AB + A'B \\ &= A + (A + A')B \\ &= A + (1) B \\ &= A + B \end{aligned}$$



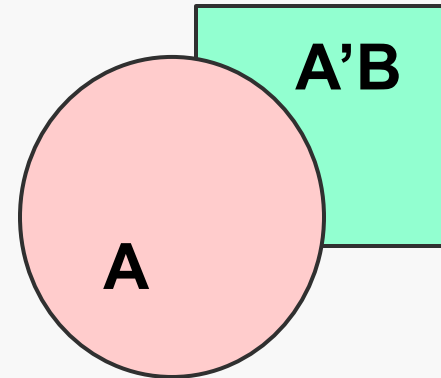
A technique commonly
used in algebraic
simplification

Absorption laws (Venn diagrams)

$$A + AB = A$$



$$A + A'B = A + B$$



Consensus

$$AB + A'C + BC = AB + A'C$$

proof:

$$BC = ABC + A'BC$$

$$\text{Thus } AB + A'C + BC = AB + A'C + ABC + A'BC$$

$$= AB + ABC + A'C + A'BC$$

$$= AB(1+C) + A'C(1+B)$$

$$= AB + A'C$$

DeMorgan's Theorems

- $(A + B)' = A' \bullet B'$

proof:

A	B	$(A+B)'$	$A' \bullet B'$
0	0	1	1
0	1	0	0
1	0	0	0
1	1	0	0

DeMorgan's Theorems

- $(AB)' = A' + B'$

proof:

A	B	$(AB)'$	$A' + B'$
0	0	1	1
0	1	1	1
1	0	1	1
1	1	0	0

DeMorgan's Theorems generalise to many variables

$$(A+B+C+D+\dots)' = A' \bullet B' \bullet C' \bullet D' \bullet \dots$$

$$(ABCD\dots)' = A'+B'+C'+D'+ \dots$$

- Add or remove inverter to each variable
- Interchange AND with OR

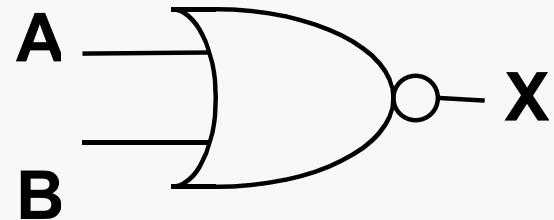
Example

Simplify $[A (B + C')' D]'$

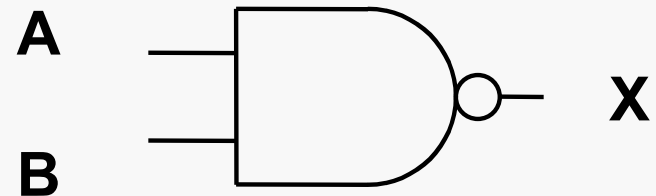
NOR gate & NAND gate

- Combines basic operations of OR & AND with NOT

- NOR: $X = (A+B)'$



- NAND: $X = (AB)'$



Truth table for 2-input NOR gate

$$X = A \text{ NOR } B$$

Inputs		Output
A	B	X
0	0	1
0	1	0
1	0	0
1	1	0

Output X=1 only when all inputs are 0

Truth table for 2-input NAND gate

$$X = A \text{ NAND } B$$

Inputs		Output
A	B	X
0	0	1
0	1	1
1	0	1
1	1	0

Output X=0 only when all inputs are 1

Truth table for 3-input NOR gate

$$X = (A+B+C)'$$

Inputs			Output
A	B	C	X
0	0	0	1
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	0

Truth table for 3-input NAND gate

$$X = (ABC)'$$

Inputs			Output
A	B	C	X
0	0	0	1
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	0

Summary

- **OR:** output is 1 when any of the inputs is 1
- **AND:** output is 1 when all the inputs are 1
- **NOT:** output is 0 when input is 1 and vice versa
- **NOR:** output is 0 when any of the inputs is 1
- **NAND:** output is 0 when all the inputs are 1

Universality of NAND gates and NOR gates

- **NAND gates can be used to form AND gate, OR gate and NOT gate**
- **Therefore, NAND gates can be used to implement any Boolean function**
- **Similarly for NOR gates**
- **Equivalence can be proved by DeMorgan's theorems**

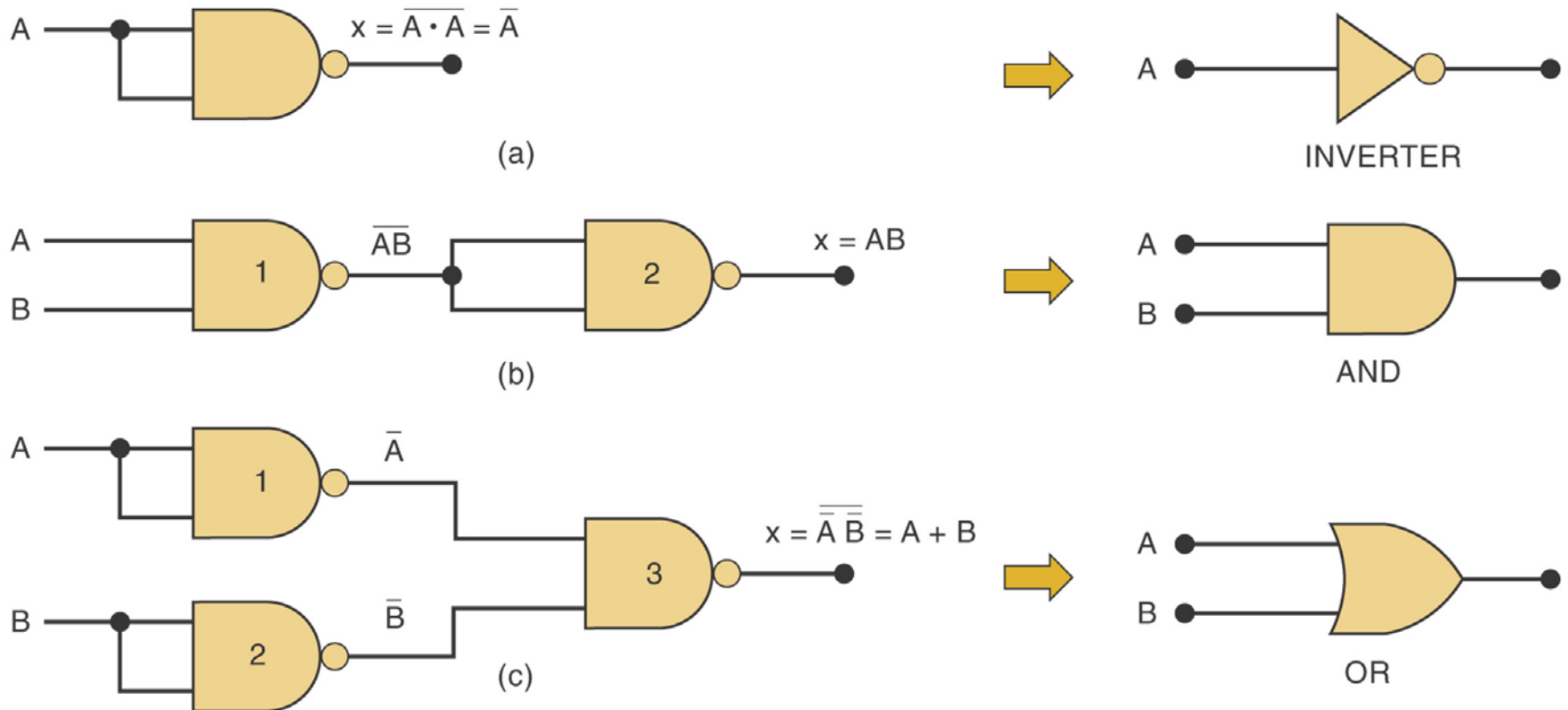
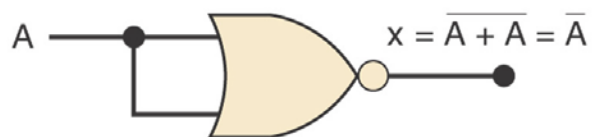
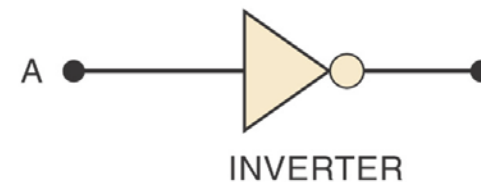


Fig. 3-29 Basic gates from NAND

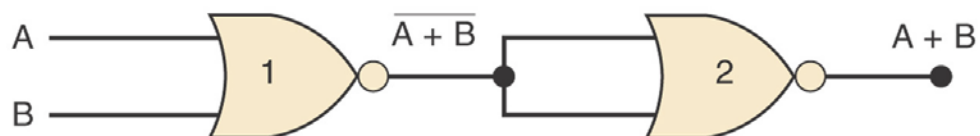
(Tocci, 10th ed. Pg. 84)



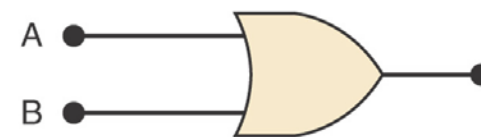
(a)



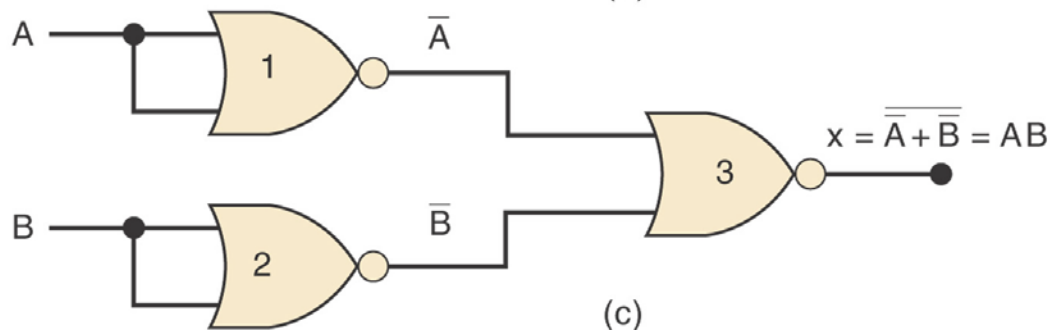
INVERTER



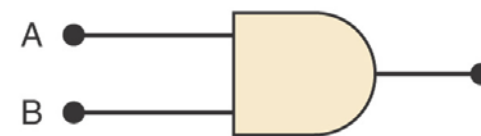
(b)



OR



(c)



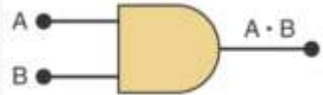
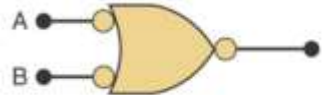
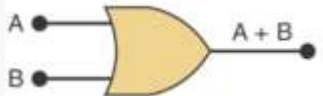
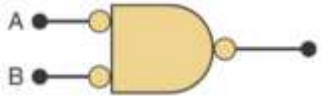
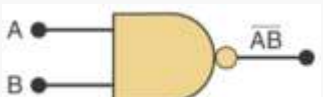

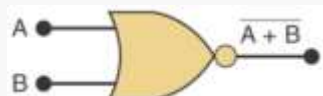
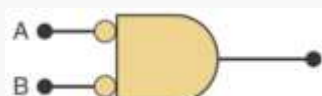
AND

Fig. 3-30 Basic gates from NOR


(Tocci, 10th ed. Pg. 84)

Alternate Logic gate representations

- The alternate symbol is obtained from the standard symbol by applying DeMorgan's theorems

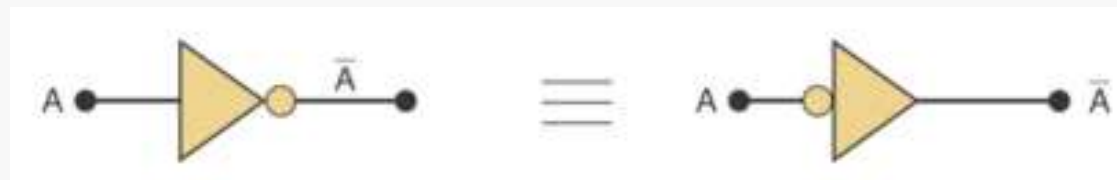
Logic operator	Standard symbol	Logic expression	Alternate Symbol
AND		$A.B = (A' + B')'$	
OR		$A+B = (A'.B')'$	
NAND		$(A.B)' = A' + B'$	
NOR		$(A+B)' = A'.B'$	

- Modification of standard to alternate symbol (and vice-versa) for the **same logic operator**:

	standard	 alternate
Bubble at input or output	No	Add
	Yes	Remove
Symbol shape	“and”	Replace with “or”
	“or”	Replace with “and”

Note that a pair of standard and alternate symbols describes the **same logic gate** with the same truth table.

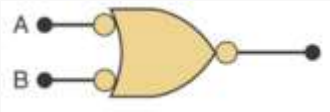
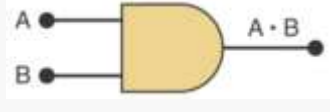
- The standard symbol of a NOT gate is similarly modified to its alternate symbol (and vice-versa) but there is no change in symbol shape:



We interpret the above symbol this way:
Output=0
when input=1

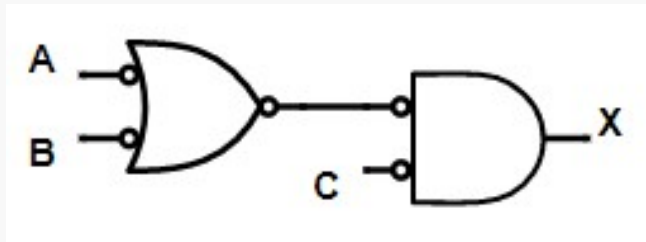
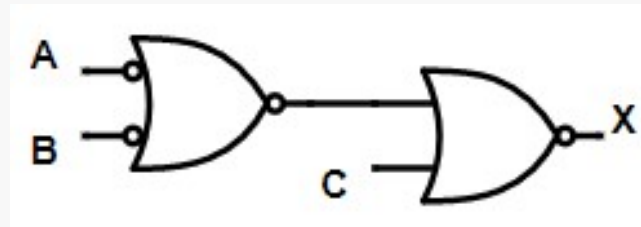
NOT gate truth table	
Input A	Output A'
0	1
1	0

We interpret the above symbol this way:
Output=1
when input=0

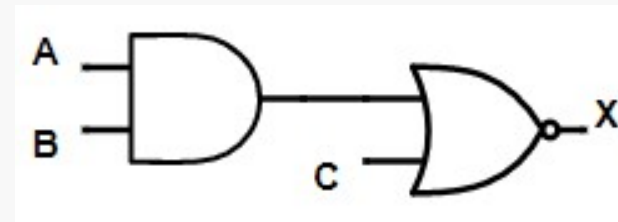
A	B	A AND B	symbol
0	0	0	
0	1	0	
1	0	0	
1	1	1	

- Both standard symbols and alternate symbols may be used in the same diagram to help describe logic flow
- Comply with bubble-to-bubble matching
- Say “0” when there is a bubble, otherwise say “1”

Example 1:

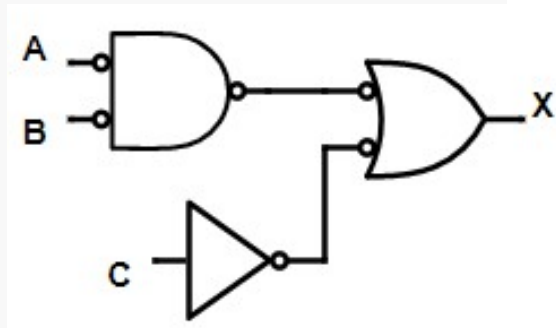
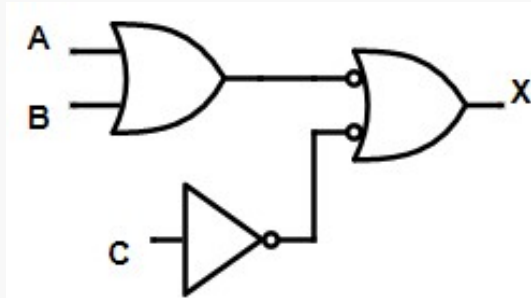


We say:
Output $X=1$
when inputs
 $C=0$ and at the
same time either
 $A=0$ or $B=0$

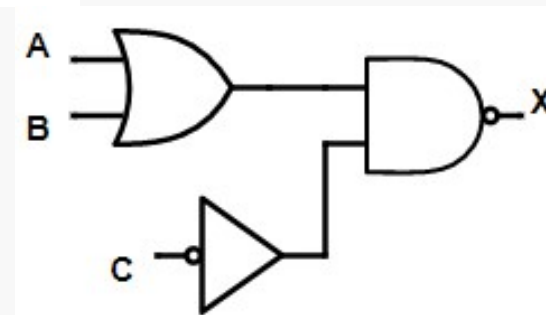


We say:
Output $X=0$
when inputs
 $C=1$ or
 A and B are
both $=1$

Example 2:



We say:
Output $X=1$
when inputs
 $C=1$ or both A and
 $B=0$



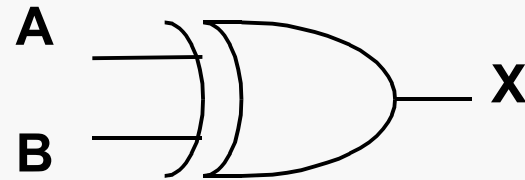
We say:
Output $X=0$
when inputs
 $C=0$ and at the same
time either $A=1$ or
 $B=1$

Exclusive-OR gate

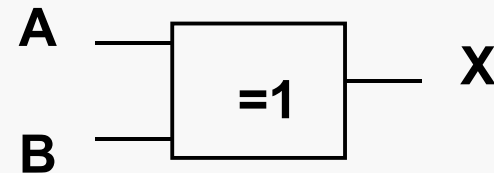
Ex-OR (XOR)

$$X = AB' + A'B$$

$$X = A \oplus B$$



A	B	X
0	0	0
0	1	1
1	0	1
1	1	0



**IEEE
symbol**

**Different
from OR**

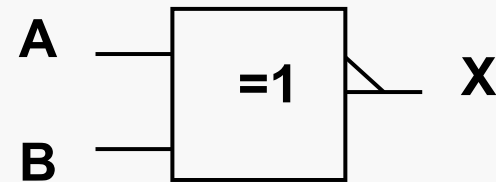
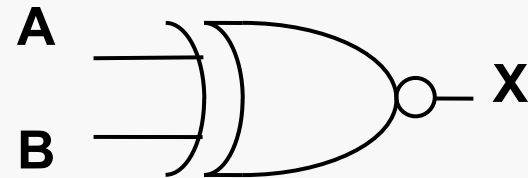
Exclusive-NOR gate

Ex-NOR (XNOR)

$$X = AB + A'B'$$

$$X = (A \oplus B)'$$

A	B	X
0	0	1
0	1	0
1	0	0
1	1	1



**IEEE
symbol**

Application of XOR

- **Bit-wise comparator**
 - **output is 1 if the two multi-bit inputs are different**

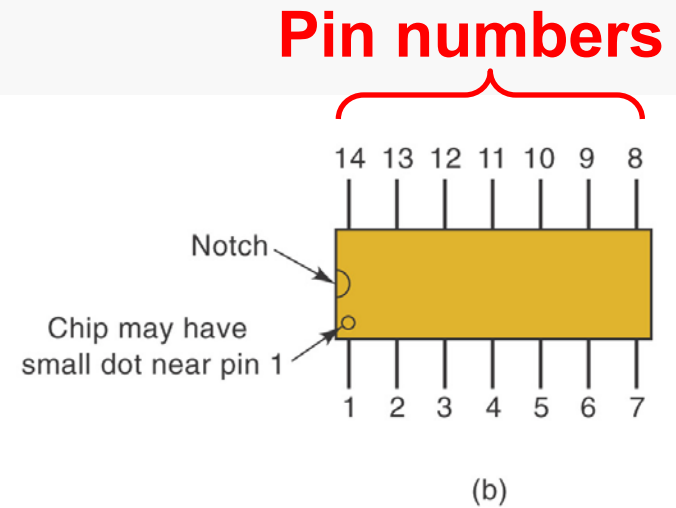
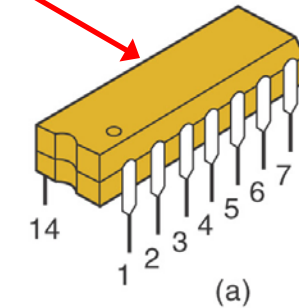
XOR with multiple inputs

- Essentially an odd-function generator
- Output is 1 if there is an odd number of 1's among all the inputs
- E.g. for 3-input XOR, the output is 1 if there are 1 or 3 bits of 1 among the inputs
- $A \oplus B \oplus C = (A \oplus B) \oplus C$
- $= A \oplus (B \oplus C)$

Logic devices

- Different ways to create a physically functioning logic circuit
- Examples: use standard logic integrated circuits (ICs), application-specific ICs (ASICs), programmable logic devices
- Small-scale integrated logic devices: AND, OR, NOT, NAND, NOR, XOR, XNOR
- We will use some of these in lab experiment 1

**Plastic or ceramic
protective casing**



Top view

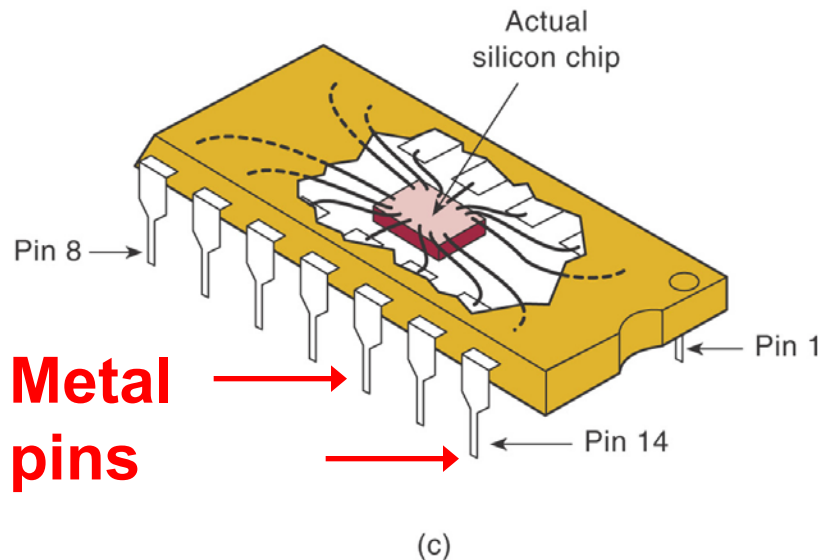
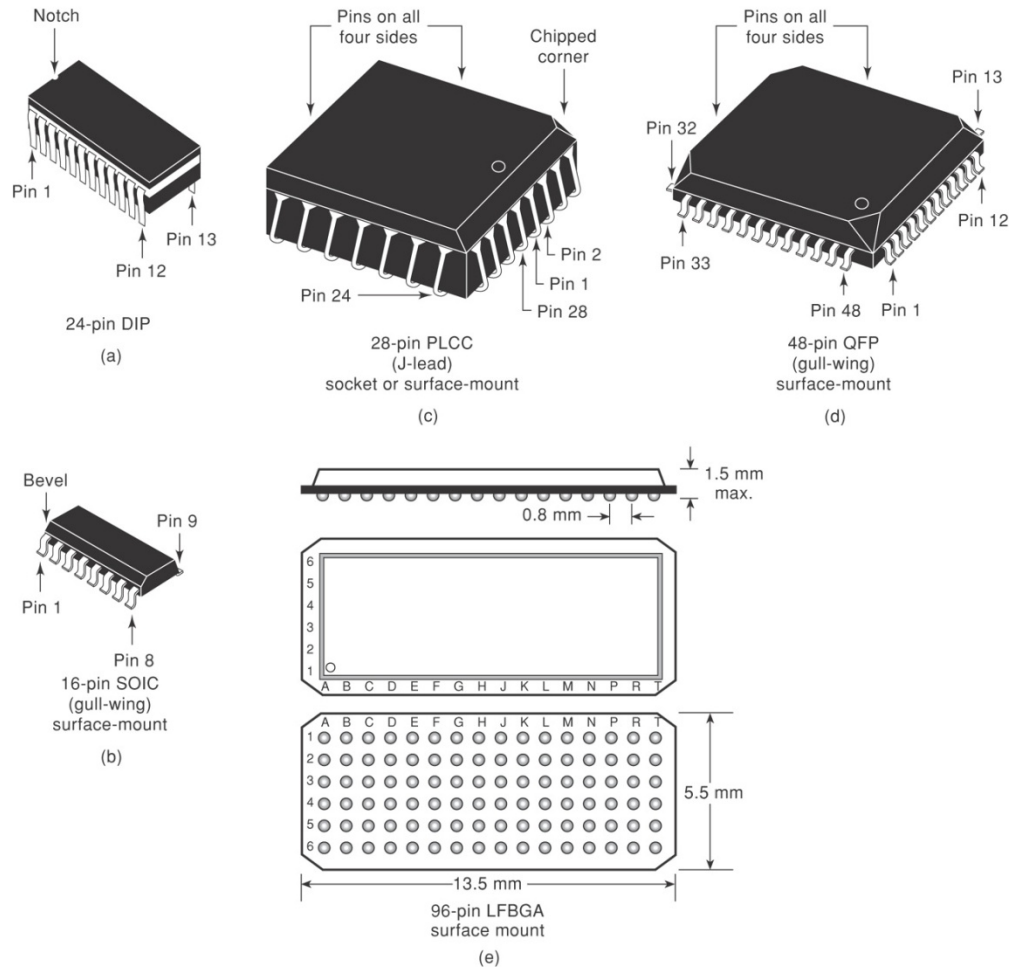


Figure 4.29: (Tocci 10th Ed) Dual-in-line Package

Common IC packaging

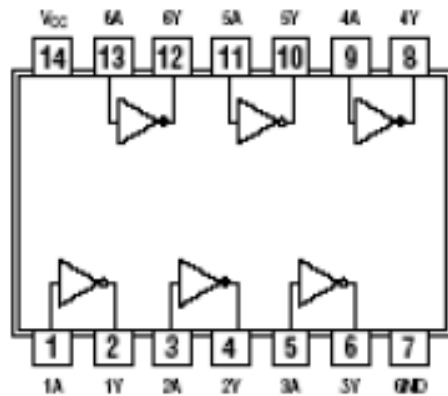


Source: Tocci, 10th ed., pg. 496

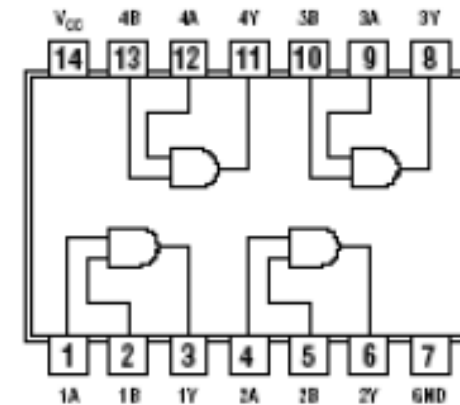
Logic circuit connections

- Implement $Y = AB'$

7404
Hex-NOT



7408
Quad-AND



Circuit connection diagram

4. Digital Arithmetic

In digital circuits, e.g. digital computers and electronic calculators, arithmetic operations are carried out on binary numbers:

- **Addition, subtraction, multiplication and division**

Binary Addition and Subtraction both begin with the **LSB (least significant bit)**

Decimal Addition and Subtraction both begin with the **LSD**:

carry

$$\begin{array}{r} \text{1} \text{ 7 6 4}_{10} \\ + \quad \text{5 8 3}_{10} \\ \hline \text{1} \text{ 3 4 7}_{10} \end{array}$$

borrow

$$\begin{array}{r} \text{6 } \cancel{\text{7}}^{\text{1}} \text{ 6 4}_{10} \\ - \quad \text{5 8 3}_{10} \\ \hline \text{1 8 1}_{10} \end{array}$$

Binary addition example:

$$10111_2 + 1010_2 = ?$$

$$(23_{10} + 10_{10} = 33_{10})$$

$$\begin{array}{r} 1\ 0\ 1\ 1\ 1_2 \\ +\ 1\ 0\ 1\ 0_2 \\ \hline \\ \hline \end{array}$$

Adder

Half adder (HA)

- combination logic circuit that performs addition of **2 bits**

$$\text{Carry} = A \cdot B$$

$$\text{Sum} = A \oplus B$$

Inputs		Outputs	
A	B	Carry	Sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Full adder (**FA**)

- combination logic circuit that performs addition of **3 bits**

Inputs			Outputs	
A	B	Cin	Cout	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Since the truth table is rather simple, we can obtain the Boolean expressions by observation,

$$\text{Sum} = \mathbf{A \oplus B \oplus Cin}$$

$$\text{Cout} = \mathbf{A \cdot B + B \cdot Cin + A \cdot Cin}$$

Fig 6.81 shows the circuit implementation.

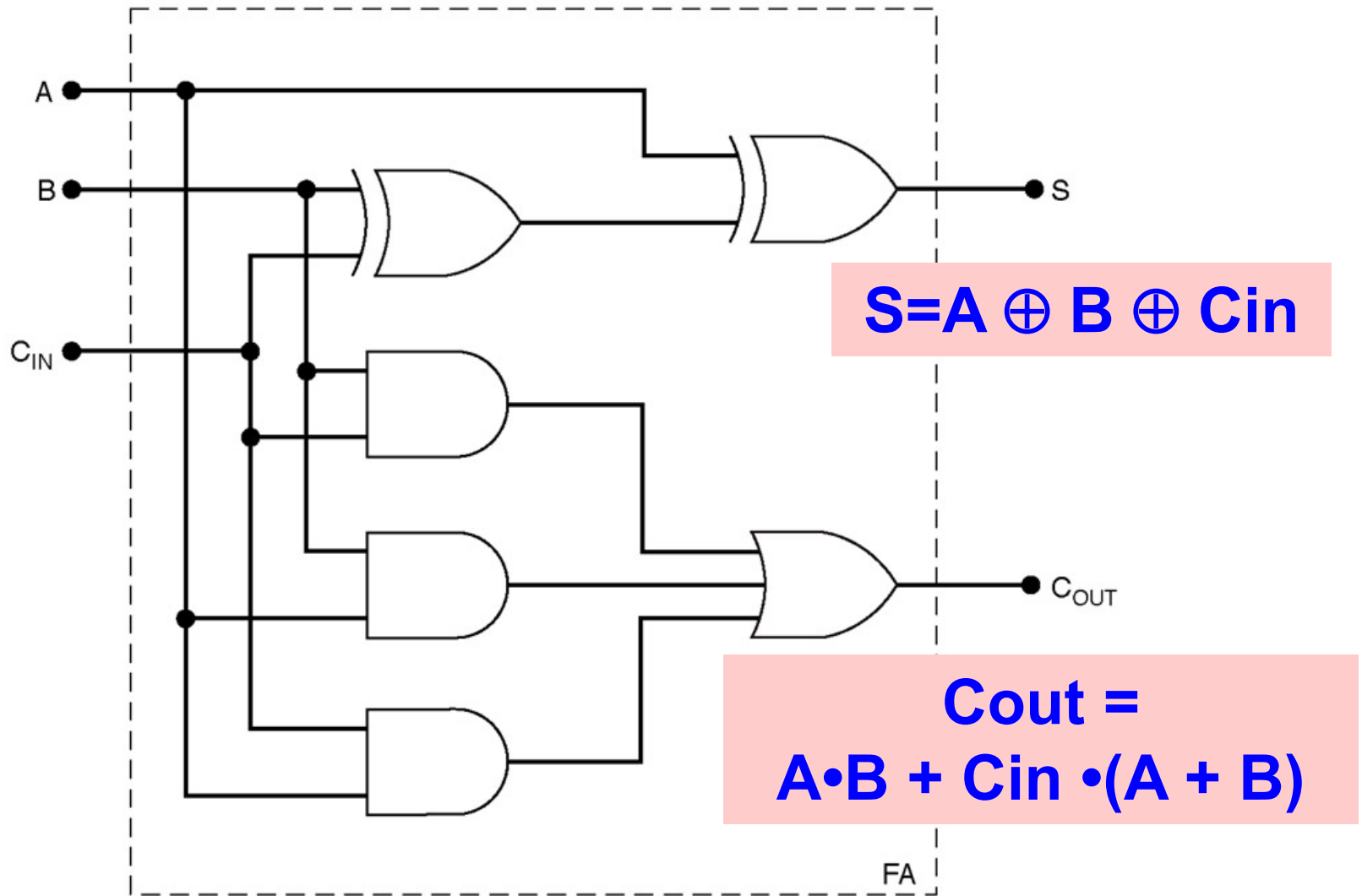


Fig. 6-81 Full adder circuit

If we re-arrange the Boolean expression,

$$\text{Cout} = A \cdot B + B \cdot \text{Cin} + A \cdot \text{Cin}$$

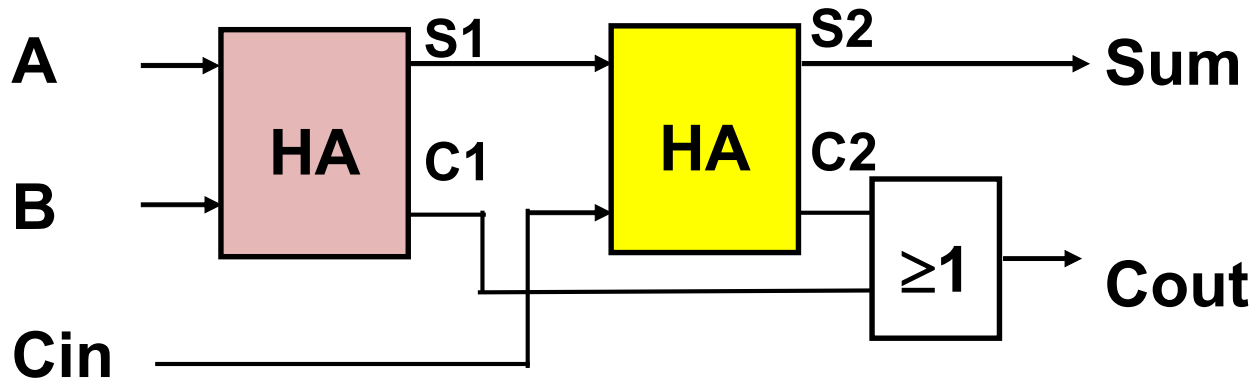
$$= A \cdot B + \text{Cin} (A' B + AB) + \text{Cin} (AB' + AB)$$

$$= A \cdot B + \text{Cin} (AB) + \text{Cin} (A' B + AB')$$

$$= \underbrace{A \cdot B}_{\text{red}} + \text{Cin} \cdot (A \oplus B)$$

- An alternate circuit implementation using Half-adders is obtained

Full adder circuit implemented using 2 half-adders and an OR gate



$$S1 = A \oplus B$$

$$C1 = A \cdot B$$

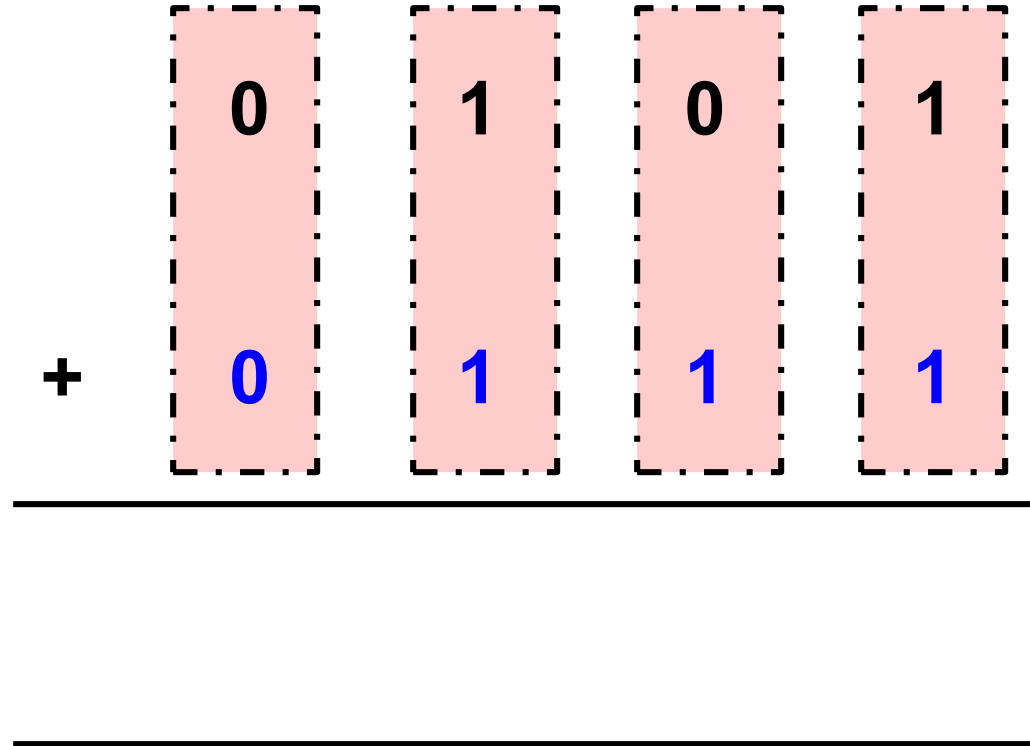
$$S2 = (A \oplus B) \oplus Cin$$

$$C2 = Cin \cdot (A \oplus B)$$

$$Cout = A \cdot B + Cin \cdot (A \oplus B)$$

Binary addition example: $0101_2 + 0111_2 = ?$

Need 4 FAs



Parallel Adder

- N full-adders can be **cascaded** to form an N-bit parallel adder
- also known as ripple adder
- all the bits of the augend and addend are fed into the adder circuits simultaneously
- addition is very fast
- addition speed is limited by propagation delays of FAs - **carry propagation**
- Fig. 6-82 shows a 4-bit adder

Example: $0011_2 + 0001_2 = 0100_2$

(In decimal: $3 + 1 = 4$)

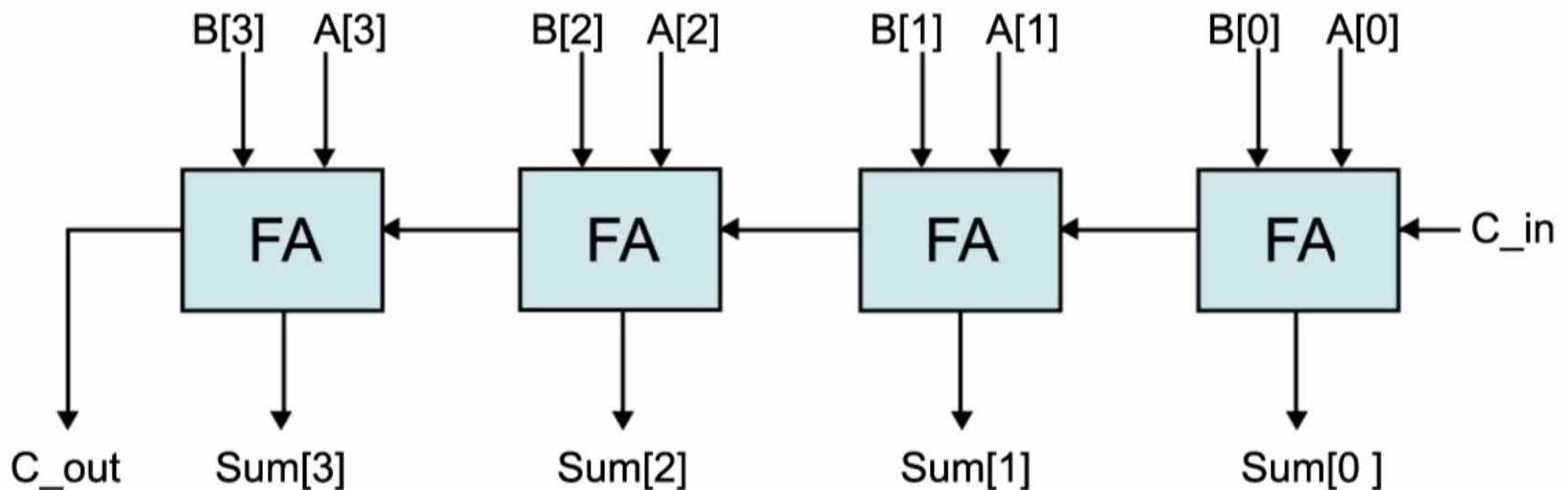


Fig. 6-82 : 4-bit ripple adder

Representing Signed Numbers

Signed-Magnitude system

- sign bit = MSB : 0 for positive numbers
- sign bit = MSB: 1 for negative numbers

$$14_{10} = 1\ 1\ 1\ 0_2$$

	sign	magnitude
+14	0	1 1 1 0
-14	1	1 1 1 0

Signed-Magnitude system

- There are equal numbers of positive and negative values
- An N-bit value lies in the range
 $\{ -(2^{N-1} - 1) \text{ to } +(2^{N-1} - 1) \}$

Examples:

$$0101\ 0101_2 = +85_{10}$$

$$1101\ 0101_2 = -85_{10}$$

$$0000\ 0000_2 = +0_{10}$$

$$1000\ 0000_2 = -0_{10}$$

2's complement system

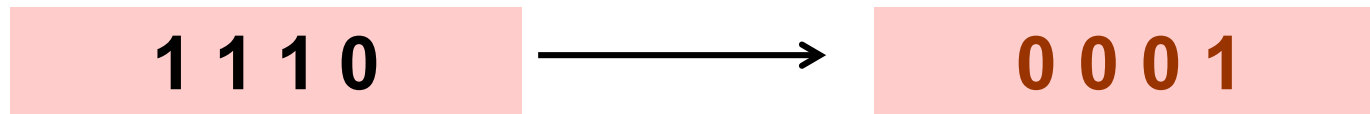
- sign bit = MSB: 0 for zero and positive numbers
- sign bit = MSB: 1 for negative numbers

$$14_{10} = 1\ 1\ 1\ 0_2$$

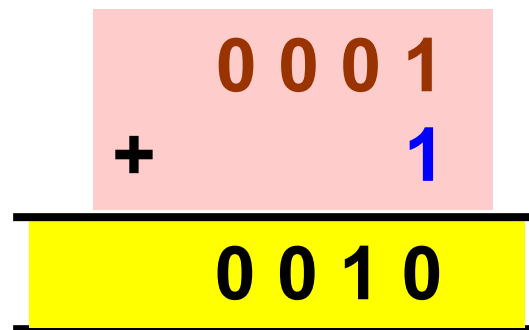
	sign	
+14	0	1 1 1 0
-14	1	???

2's complement operation:

Step 1: invert every bit of a binary number
(i.e. perform **1's complement**)



Step 2: add (arithmetic addition) **1** to it



The 2's complement of 1110 is 0010

2's complement system

$$14_{10} = 1\ 1\ 1\ 0_2$$

	sign	
+14	0	1 1 1 0
-14	1	0 0 1 0

A short-cut method for 2's complement operation:

- **starting from LSB, copy the bit if it is '0' and repeat process with remaining bits**
- **copy the bit if it is the first '1', then invert all the remaining bits**
- **a sequential process**

This method works on all binary bit patterns

Examples:

$$0\ 1\ 0\ 1\ 0\ 1\ 0\ 1_2 = +85_{10}$$



$$1\ 0\ 1\ 0\ 1\ 0\ 1\ 1_2 = -85_{10}$$

**Invert the
remaining
bits**

$$0\ 1\ 0\ 0\ 0\ 0\ 0\ 0_2 = +64_{10}$$



$$1\ 1\ 0\ 0\ 0\ 0\ 0\ 0_2 = -64_{10}$$

**Copy the
bits from
the right
until it
reaches
the first 1**

2's complement system

- There is 1 more negative value than positive ones
- An N-bit value lies in the range
 $\{ -(2^{N-1}) \text{ to } +(2^{N-1} - 1) \}$
- Eg. 4 bits: $\{-8, 7\}$
- 8 bits: $\{-128, 127\}$

Summary: representing signed numbers in 2's complement system:

If the number is **zero or positive**

- represent its magnitude in binary
- append a sign bit (0) in front of the MSB if the MSB is 1

If the number is **negative**

- represent its magnitude in binary
- obtain the 2's complement
- append a sign bit (1) in front of the MSB if the MSB is 0

Examples: Represent the following signed decimal numbers in 2's complement

(a) 5

- binary of 5 is 101
- Positive 5 is **0101**

(b) -7

- binary of 7 is 111
- 2's complement of 111 is 001
- Negative 7 is **1001**

E.g. A 4-bit 2's complement number system:

Decimal	Binary	
	magnitude	2's comp
-8	1000	1000
-7	0111	1001
-6	0110	1010
-5	0101	1011
-4	0100	1100
-3	0011	1101
-2	0010	1110
-1	0001	1111
0	0000	0000
1	0001	0001
2	0010	0010
3	0011	0011
4	0100	0100
5	0101	0101
6	0110	0110
7	0111	0111

Generalised to an N-bit 2's complement number system:

Decimal	2's comp	
$-(2^{N-1})$	$1000\dots00$	← Most negative
$-(2^{N-1}) + 1$	$1000\dots01$	
$-(2^{N-1}) + 2$	$1000\dots10$	
.	.	
.	.	
-1	$1111\dots11$	← -1
0	$0000\dots00$	← zero
1	$0000\dots01$	
2	$0000\dots10$	
.	.	
.	.	
$(2^{N-1}) - 2$	$0111\dots10$	
$(2^{N-1}) - 1$	$0111\dots11$	← Most positive

Some special bit patterns:

- **100....000 (with N-1 zeros)**

the decimal value is $-(2^{N-1})$

- **111....111 (all ones)**

the decimal value is -1

- **000....000 (all zeroes)**

the decimal value is 0

- **011....111 (with N-1 ones)**

the decimal value is $(2^{N-1}-1)$

Obtaining the decimal value of a 2's complement number

If the number is **positive**:

- perform a binary-to-decimal conversion on the number

If the number is **negative**:

- First perform 2's complement to convert it to **positive**
- Next perform a binary-to-decimal conversion on the **positive** number

Examples: Obtain the decimal equivalent of the following 2's complement numbers

(a) 01001

- The magnitude 1001 is 9 in decimal
- 01001 is **positive 9** in decimal

(b) 10011

- 2's complement of 10011 is 01101
- The magnitude 1101 is 13 in decimal
- 10011 is **negative 13** in decimal

Some observations on the 2's complement number system

- The 2's complement of an N -bit binary number is also of N bits.
- To represent a binary number with N significant bits in its magnitude using the 2's complement system, we need $(N+1)$ bits. The extra bit is the sign bit. *

*** note exception**

- If there are more bits than necessary to represent a binary number in the 2's complement system, the more significant bits are filled with the sign bit, which is 0 for positive and 1 for negative – known as **sign extension**

$$101 = 1101 = 11101$$

$$011 = 0011 = 00011$$

- A 2's complement operation will change a positive number to negative and vice-versa, with no change in the magnitude. *

*** note exception**

*** Exception arises when dealing with the most negative number that can be represented given a number of bits.**

e.g.

-8 in a 4-bit system,

-16 in a 5-bit system

-128 in an 8-bit system

$-(2^{N-1})$ in an N-bit system

Reasons for using the 2's complement system:

- **Using the 2's complement representation for signed numbers, subtraction of numbers can be carried out in the same way as addition**
- **Therefore, the same set of hardware circuits can be used for both subtraction and addition**

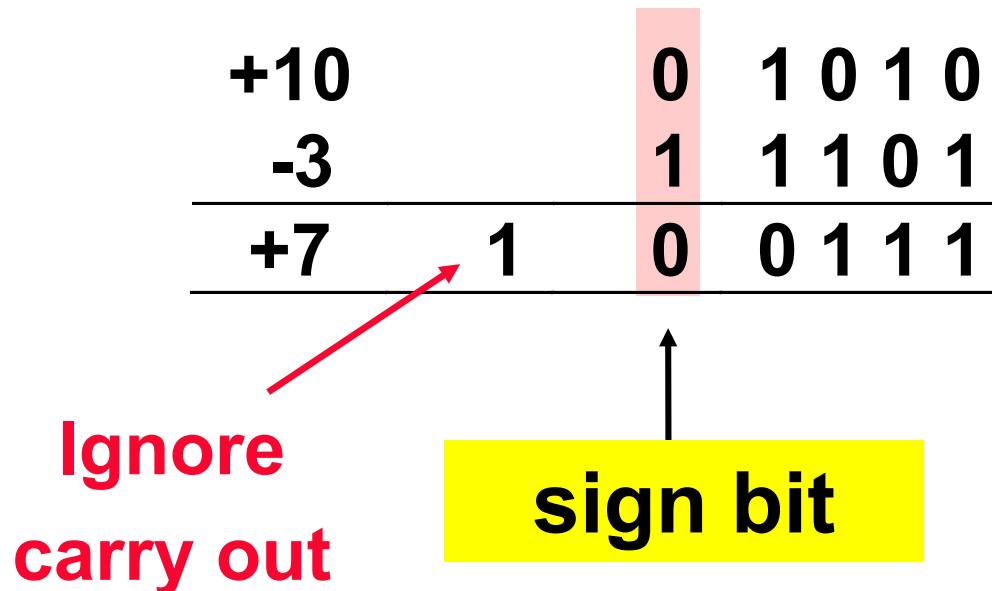
Addition in the 2's complement system

two positive numbers

+10	0	1	0	1	0	(augend)
+3	0	0	0	1	1	(addend)
<hr/>						
+13	0	1	1	0	1	(sum)
<hr/>						

↑
sign bit

The sign bits are added like the other bits
positive number and smaller
negative number



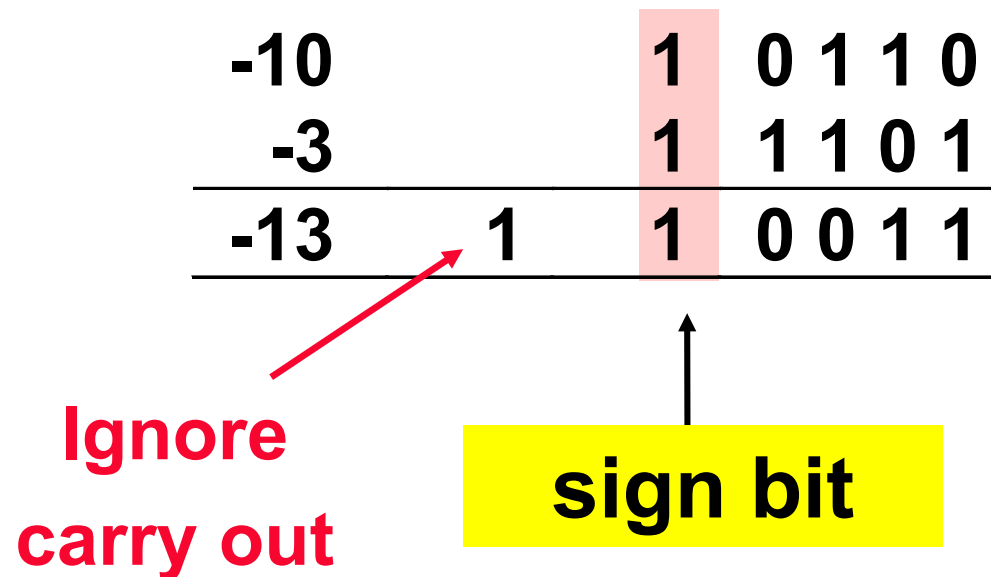
positive number and larger negative number

-10	1	0	1	1	0
+3	0	0	0	1	1
<hr/>					
-7	1	1	0	0	1
<hr/>					

↑

sign bit

two negative numbers



Subtraction in the 2's complement system

Subtraction can be carried out using 2's complement and addition

$$A - B = A + (-B)$$

-B = 2's complement of B

B may be positive or negative

Examples:

$$10 - 3 = 10 + (-3) = 7$$

-3 is 2's complement of 3

Ignore
carry out



	1	0	1	0	1	0
+	1	1	1	0	1	
<hr/>						
	1	0	0	1	1	1
<hr/>						

$$-10 - (-3) = -10 + 3 = -7$$

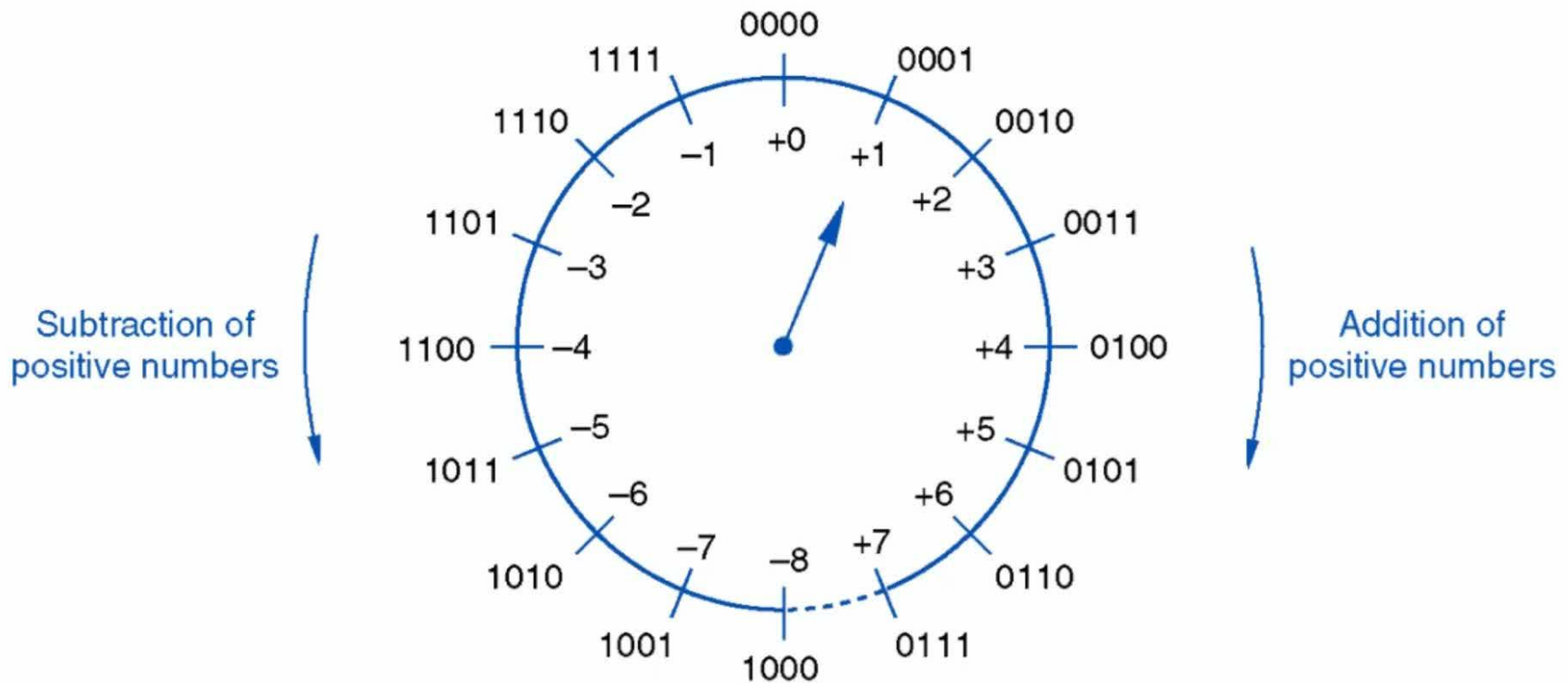
3 is 2's complement of -3

	1	1	0	1	1	0
+	0	0	0	1	1	
<hr/>						
	1	1	0	0	1	
<hr/>						

Remember:

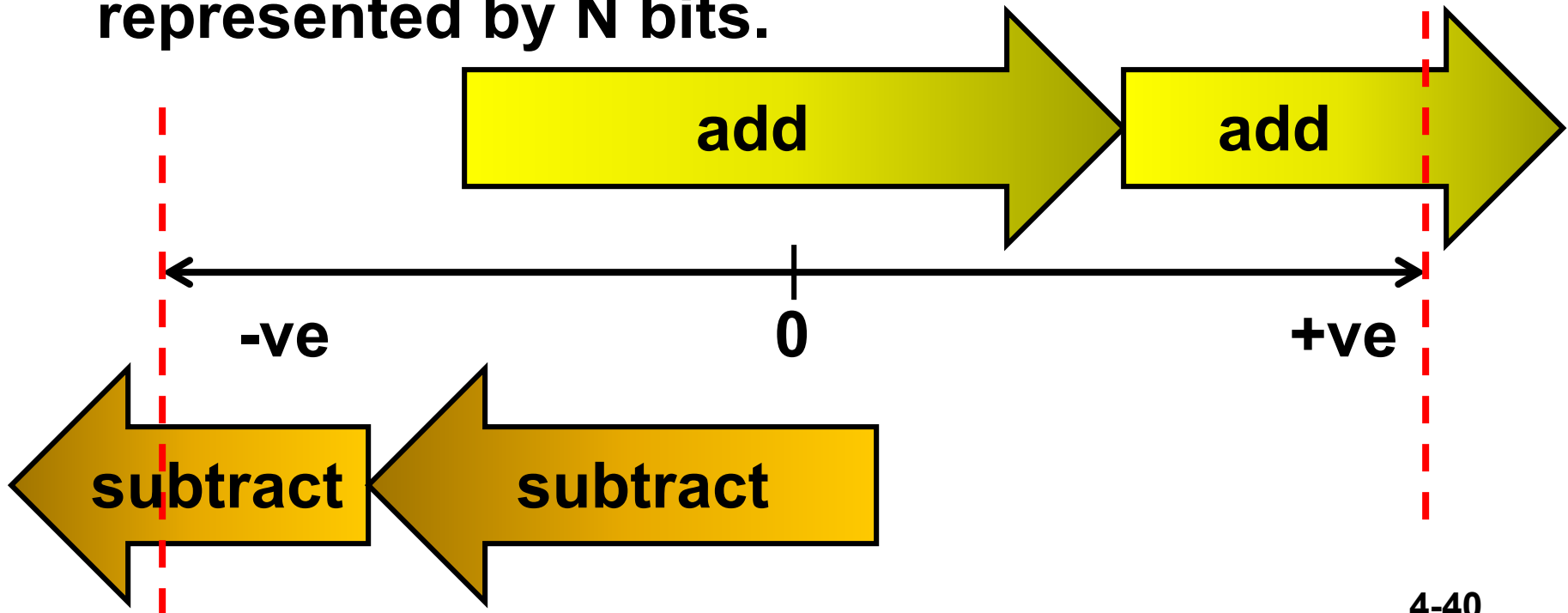
The pair of numbers to be added/subtracted must have the same size, i.e. same number of bits. This ensures that the sign bits are aligned. The resulting sum must also be of the same size.

Addition and Subtraction in the 2's complement system



Arithmetic Overflow

It occurs when an arithmetic operation between two N-bit operands produces a result that cannot be sufficiently represented by N bits.



Rules to detect overflow in 2's complement addition:

No overflow occurs if the operands have opposite signs.

Overflow is detected when operands have the **same sign**, but the arithmetic sum has an **opposite sign**.

¹ 0	1	0	1	0
+	0	1	0	0
<hr/>				
1	0	0	1	0
<hr/>				

10+8=-14?

For **subtraction**, no overflow occurs if the operands have the same sign.

Combined circuit for addition and subtraction

Fig 6-12 (**X0 and Y0 are LSB**)

- To perform the addition: $X + Y$
- Make inputs Add/Sub = 0, $C0 = 0$
- XOR gates do not invert Y
- Inputs to FAs are X and Y, with $C0 = 0$
- E.g. $0001 + 0011 = 0100$
- i.e. $1 + 3 = 4$ (in decimal)

Combined circuit for addition and subtraction

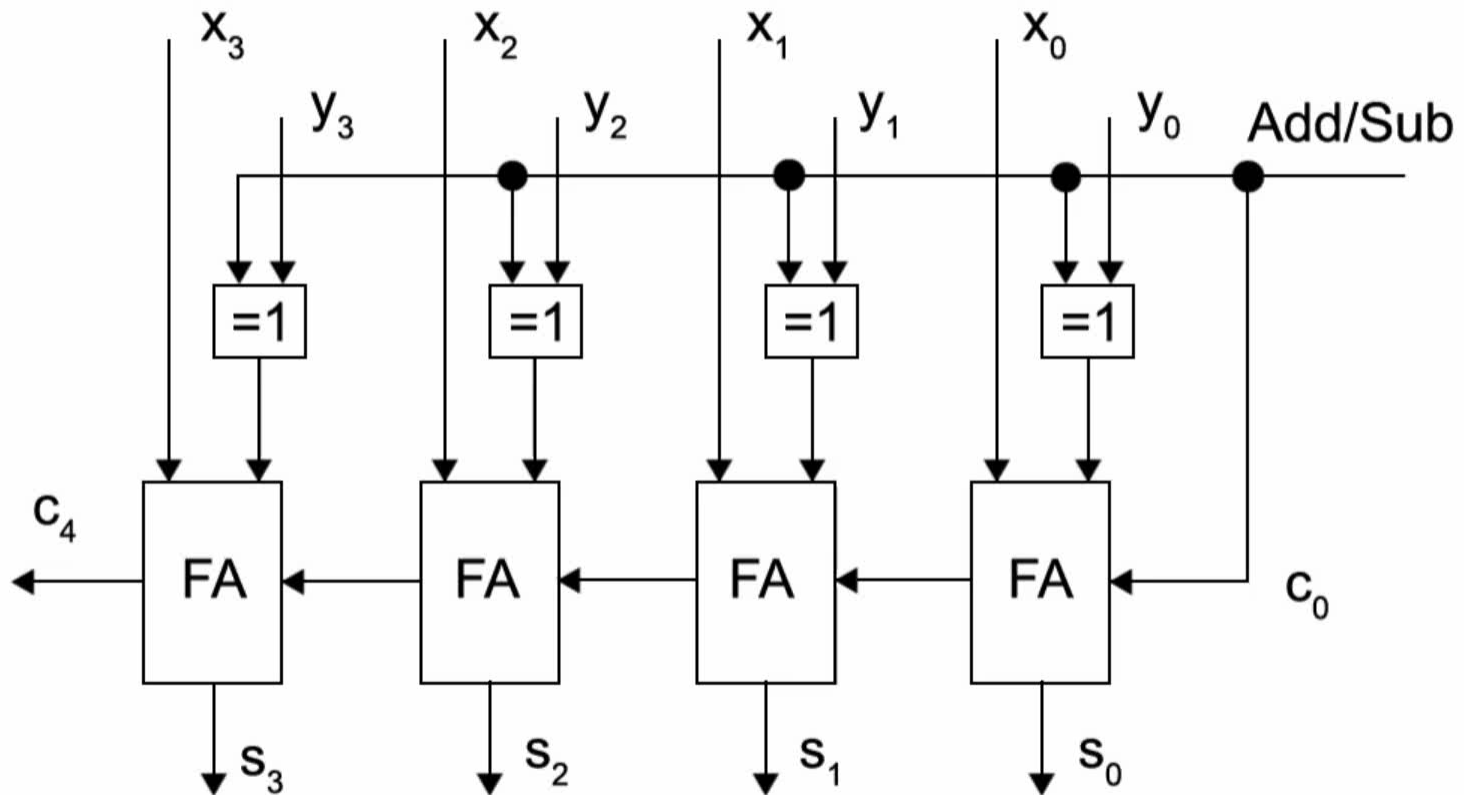


Fig 6-12 (add)

Fig 6-12 (subtraction)

- To perform the subtraction: $X - Y$
- Recall: $-Y$ is 2's complement of Y
- Make inputs Add/Sub = 1, $C0 = 1$
- XOR gates invert Y to give Y'
- Inputs to FAs are X and Y' , with $C0 = 1$
- E.g. $0001 - 0100 = 0001 + 1011 + 1 = 1101$
- i.e. $1 - 4 = -3$ (in decimal)

Combined circuit for addition and subtraction

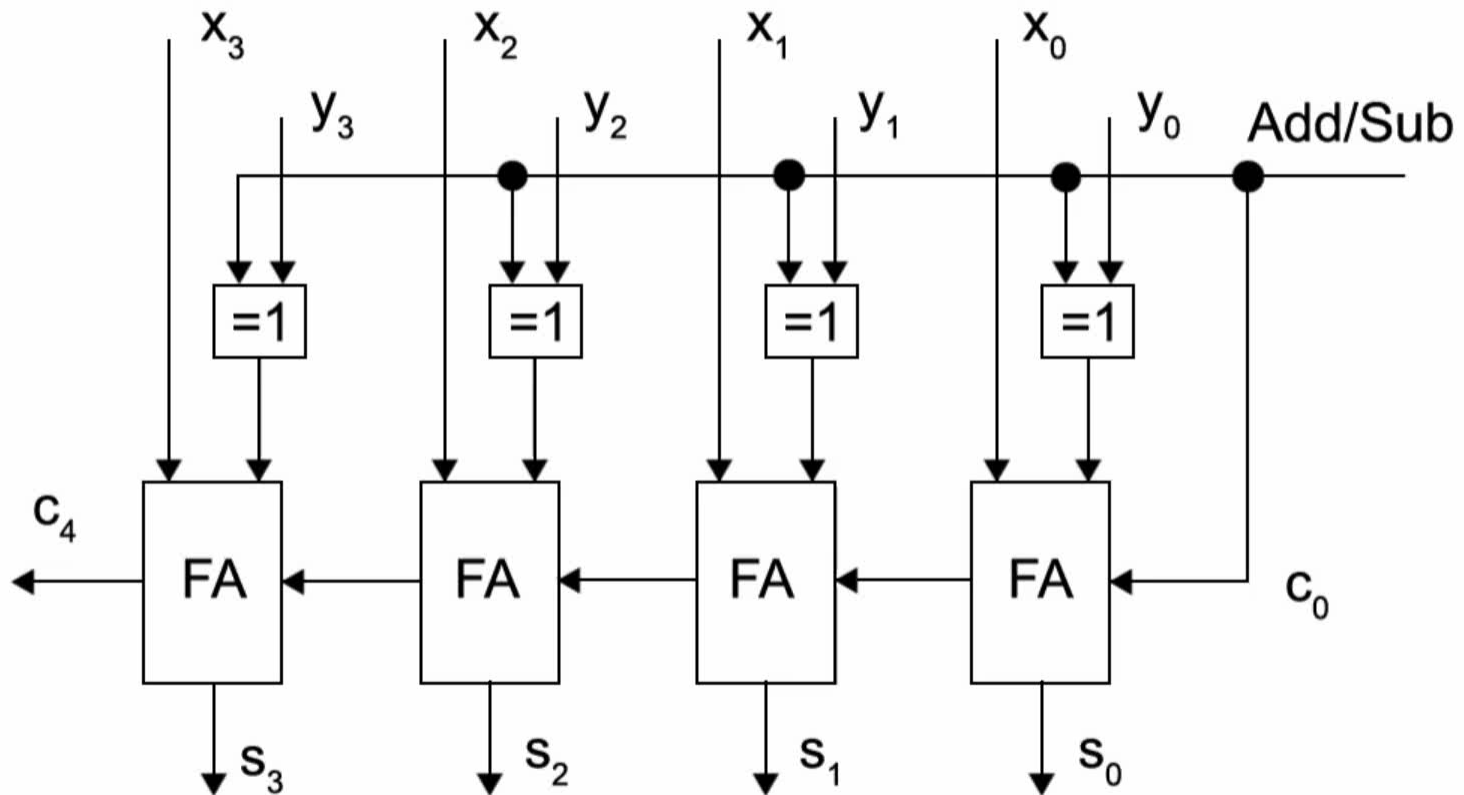


Fig 6-12 (subtract)

Fig. 6-9: Complete 4-bit Parallel Adder with Registers

t1: CLEAR* clears the contents of A register to 0's

t2: PGT of first LOAD pulse transfers operand X from memory into B register

t3: PGT of first TRANSFER pulse transfers FA output (=X) into A register

t4: PGT of second LOAD pulse transfers operand Y from memory into B register

t5: PGT of second TRANSFER pulse transfers FA output ($=X+Y$) into A register

- **Note that sufficient time (between load and transfer) must be given to FAs to complete addition**

- **E.g. $0001 + 0010 = 0011$**

Thus A register holds the result of $X+Y$

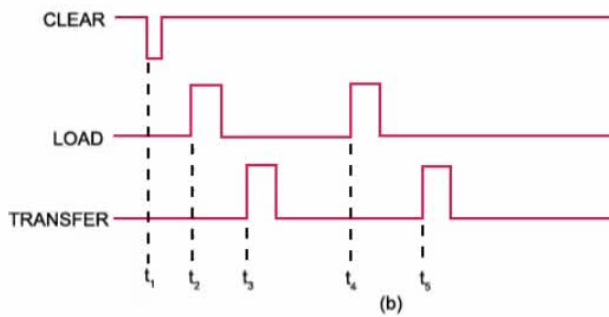
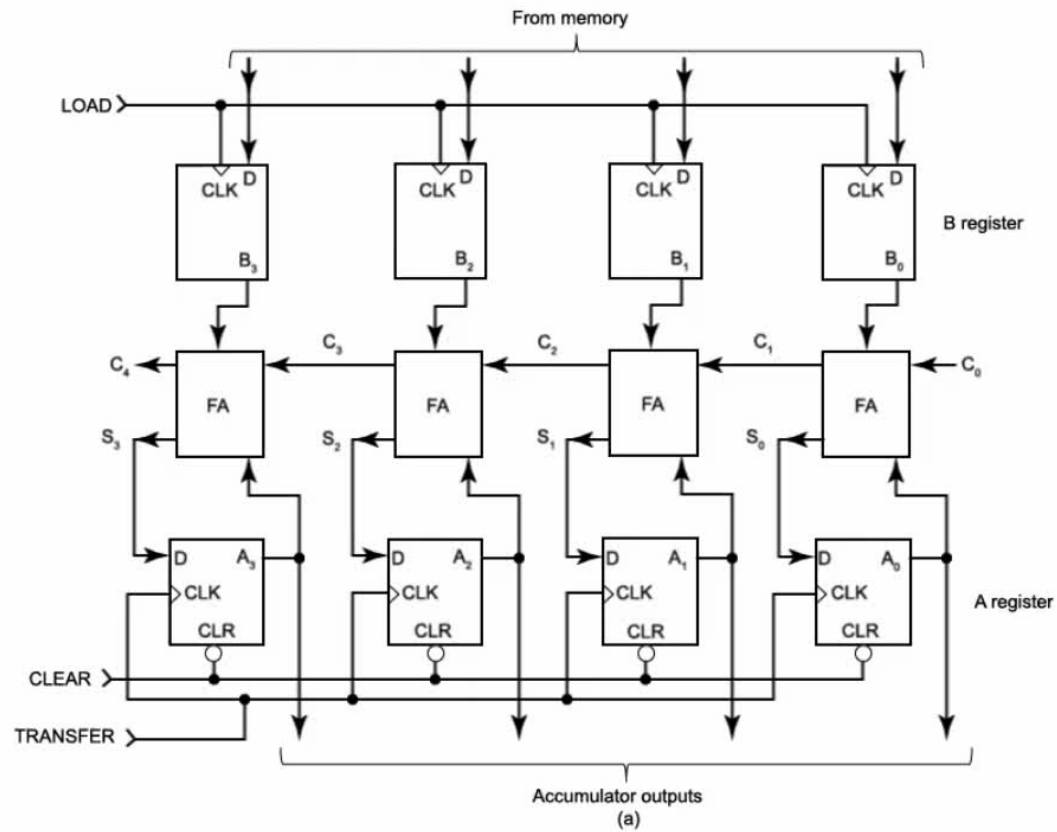


Fig.6-9

Carry propagation can be reduced (hence increasing speed of addition) by a special-purposed logic circuit

- **carry-look-ahead circuit**

Many IC parallel adders have built-in carry look-ahead circuit to speed up the addition.

Binary Multiplication

Unsigned multiplication:

similar to decimal multiplication

$$\begin{array}{r} 0111 \quad (7) \\ \times 1100 \quad (12) \\ \hline \end{array}$$

2's complement multiplication:

- **If the multiplier is positive, same as unsigned multiplication**
- **If the multiplier is negative, need to take care of negative weight of MSB (i.e. the sign bit)**
- **Treat multiplier as a sum of 2 parts:**
 - **MSB – negative part**
 - **Remaining bits – positive part**

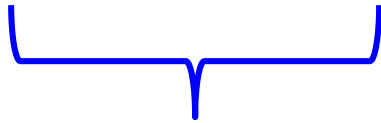
Example of 2's complement multiplication:

$$\begin{array}{r} -5 \\ x -3 \\ \hline 15 \\ \hline \end{array}$$

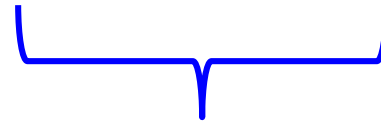
Explanation:

$$1011 \times 1101 = 1011 \times (1000 + 0101)$$

$$= (1011 \times 1000) + (1011 \times 0101)$$



This is done by
shift and
2's complement



This is done by
shift and sign-
extension

Binary Division

Unsigned division:

similar to the long division method in decimal arithmetic

$$9 \div 3 = 3$$

Signed division:

convert the signed numbers to unsigned, divide them as above, then convert the result using the appropriate sign representation

BCD addition

When the sum of two BCD digits does not exceed 9_{10} , the operation is the same as binary addition

If the sum of two BCD digits is more than 9_{10} , a **correction** needs to be made by adding 6 (0110) to skip over the six invalid codes

The correction involves two steps:

- **a carry of decimal value 1 is brought forward and added to the next higher digit**
- **the decimal value 6 is added to the sum to obtain the correct BCD digit**

Example:

$$24_{10} + 47_{10} = 71_{10}$$