SWITCHING THEORY & LOGIC DESIGN
BIT-11

Contact Hours Lecture : 3, Tutorial : 1 , Practical: 2 (online 4 lectures/week)
Number of Credits :5

B.Tech (IT)   **Semester: III**

By
JAY PRAKASH
(Asst. Prof)

Department of information technology & Computer Application
Madan Mohan Malaviya University of Technology
Gorakhpur, India

SWITCHING THEORY & LOGIC DESIGN

BIT-11	SWITCHING THEORY & LOGIC DESIGN		
Contact Hours/Week		:	Lecture : 3, Tutorial : 1 , Practical: 0
UNIT-I			
Number system, Binary Codes - Weighted and Non-Weighted - Binary Arithmetic Conversion Algorithms - Error Detecting and Error Correcting Codes - Canonical and Standard Boolean Expressions - Truth Tables.			9
UNIT-II			
K-Map Reduction - Don't Care Conditions - Adders / Subtractors- Carry Look-Ahead Adder - Code Conversion Algorithms - Design of Code Converters - Equivalence Functions. Binary/Decimal Parallel Adder/Subtractorfor Signed Numbers - Magnitude Comparator - Decoders / Encoders - Multiplexers / Demultiplexers- Boolean Function Implementation using Multiplexers.			9
UNIT-III			
Sequential Logic - Basic Latch - Flip-Flops (SR, D, JK, T and Master-Slave) - Triggering of Flip-Flops - Counters - Design Procedure - Ripple Counters - BCD and Binary - Synchronous Counters.			9
UNIT-IV			
Registers - Shift Registers - Registers with Parallel Load - Memory Unit - Examples of RAM, ROM, PROM, EPROM - Reduction of State and Flow Tables - Race-Free State Assignment - Hazards.			9

Text Books

" **Digital Design**" By M. Morris Mano, , Prentice Hall of India

Digital Systems

- It is a system that manipulates (processes) discrete elements of information.

- A digital system is a discrete information processing system.

- Examples of digital systems: – the calculator, – digital voltmeters, – and general-purpose computers.

- In such systems, all quantities are represented using two values.

Discrete elements of information are represented in a digital system by physical quantities called signals.

Signal

Signal can be defined as a physical quantity, which contains some information. It is a function of one or more than one independent variables. Signals are of two types.

- Analog Signal
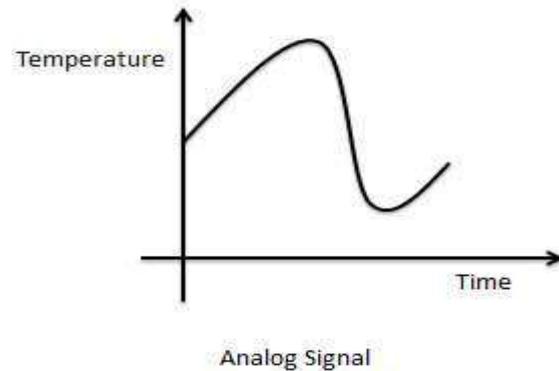- Digital Signal

Analog Signal

An **analog signal** is defined as the signal having continuous values. Analog signal can have infinite number of different values.

Examples of the analog signals:

Temperature, Pressure, Sound, Voltage, Current, Power

Graphical representation of Analog Signal (Temperature)

Analog Signal

The circuits that process the analog signals are called as analog circuits or system. Examples of the analog system:

- Filter
- Amplifiers
- Television receiver
- Motor speed controller

Disadvantage of Analog Systems

- Less accuracy
- More noise effect
- More distortion
- More effect of weather

Digital Signal

- A **digital signal** is defined as the signal which has only a finite number of distinct values.

- In the digital electronic calculator, the input is given with the help of switches. This input is converted into electrical signal which have two discrete values or levels.

- One of these may be called low level and another is called high level. The signal will always be one of the two levels. This type of signal is called digital signal.

  Examples of the digital signal:

    - Binary Signal

    - Octal Signal

    - Hexadecimal Signal

Graphical representation of the Digital Signal (Binary)
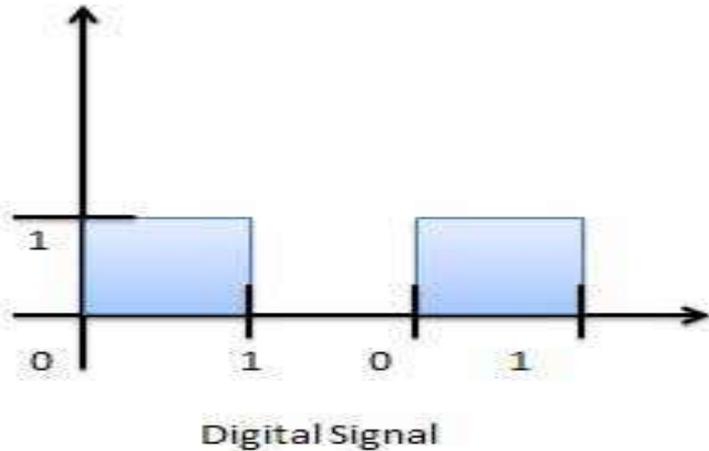
Digital Signal

The circuits that process the digital signals are called digital systems or digital circuits.

Examples of the digital systems:

Registers, Flip-flop, Counters, Microprocessors

Advantage of Digital Systems

- More accuracy
- Less distortion
- Easy communicate

Comparison of Analog and Digital Signal

S.No.	Analog Signal	Digital Signal
1	Analog signal has infinite values.	Digital signal has a finite number of values.
2	Analog signal has a continuous nature.	Digital signal has a discrete nature.
3	Analog signal is generated by transducers and signal generators.	Digital signal is generated by A to D converter.
4	Example of analog signal − sine wave, triangular waves.	Example of digital signal − binary signal.

Merits/Advantages of Digital Systems

- Digital Systems are generally easier to Design.

- Digital Information storage is easy.

- Digital circuits are less affected by the noise.

- More Circuitry can be fabricated on integrated Circuits (ICs) chips.

- Display of data and other information is very convenient, accurate

    and elegant using digital techniques.

Limitation of Digital Systems

- Only Analog Signal is available in the real world..

- An extra hardware is required to convert analog signal to digital signal.

By Using

- ❖ **Digital to Analog converter (DAC)**

- ❖ **Analog to digital converter (ADC)**

A digital system can understand positional number system only where there are a few symbols called digits and these symbols represent different values depending on the position they occupy in the number.

Number System

Number system is the language of digital system consisting a set of symbols called digits with rules defined for mathematical operations

> ❖ **Non Positional Number System**

> ❖ **Positional Number System**

- **A number is formed by collection of digits.**

- **It has two components Integer and Fraction**

- **Both are separated by decimal point**

> ❖ **Example:$(467.34)_{10}$**

Non-positional Number Systems

Characteristics

- In non-positional number system , each symbol represents the same value regardless of its position in the number

- Use symbols such as I for 1, II for 2, III for 3, IIII for 4, IIIII for 5, etc

- The symbols are simply added to find out the value of a particular number

Difficulty

- It is difficult to perform arithmetic with such a number system

Positional Number Systems

- In positional number system, each symbol represents different value depending on the position they occupy in a number.

- In positional number system, each system has a value that relates to the number directly next to it. The total value of a positional number is the total of the resultant value of all positions.

  Example: 12 can be 1 x 10 + 2 x 1, 10 + 2 = 12.

The value of each digit is determined by:

1. The digit itself
2. The position of the digit in the number
3. The base of the number system

(base = total number of digits in the number system)

The maximum value of a single digit is always equal to one less than the valueof the base

Decimal Number System

A positional number system

- Decimal number system has base 10 as it uses 10 symbols or digits (0,1,2,3,4,5,6,7,8,9).

- The maximum value of a single digit is 9 (one less than the value of the base)

- Each position of a digit represents a specific power of the base (10)

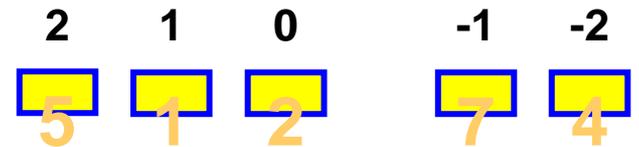- We use this number system in our day-to-day life

Example

$$2586_{10} = (2 \times 10^3) + (5 \times 10^2) + (8 \times 10^1) + (6 \times 10^0)$$

$$= 2000 + 500 + 80 + 6$$

Decimal Number System

- Base (also called radix) = 10
  - 10 digits { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }
- Digit Position
  - Integer & fraction
- Digit Weight
  - Weight = (*Base*) ^{Position}
- Magnitude
  - Sum of "*Digit* x *Weight*"
- Formal Notation

2	1	0	-1	-2
5	1	2	7	4

100	10	1	0.1	0.01

| 500 | 10 | 2 | 0.7 | 0.04 |

$$d_2*B^2+d_1*B^1+d_0*B^0+d_{-1}*B^{-1}+d_{-2}*B^{-2}$$

$$(512.74)_{10}$$

Octal Number System

A positional number system

- Octal number system has total 8 symbols or digits (0, 1, 2, 3, 4, 5, 6, 7). Hence, its base = 8

- The maximum value of a single digit is 7 (one less than the value of the base)

- Each position of a digit represents a specific power of the base (8)

- Since there are only 8 digits, 3 bits $(2^3=8)$ sufficient to represent any octal number in binary

Example

$$2057_8 = (2 \times 8^3) + (0 \times 8^2) + (5 \times 8^1) + (7 \times 8^0)$$

$$= 1024 + 0 + 40 + 7$$

$$= 1071_{10}$$

Octal Number System

- Base = 8
  - 8 digits { 0, 1, 2, 3, 4, 5, 6, 7 }

- Weights
  - Weight = (*Base*) ^{Position}

- Magnitude
  - Sum of "*Digit* x *Weight*"

- Formal Notation

64	8	1	1/8	1/64
5	1	2	7	4

2	1	0	-1	-2

$$5 * 8^2 + 1 * 8^1 + 2 * 8^0 + 7 * 8^{-1} + 4 * 8^{-2}$$

$$= (330.9375)_{10}$$

$$(512.74)_8$$

Binary Number System

A positional number system

- Binary number system has base 2 as it uses 2 digits from 0 to 1.
- The maximum value of a single digit is 1 (one less than the value of the base)
- Each position of a digit represents a specific power of the base (2)
- This number system is used in computers

Example

$10101_2 = (1 \times 2^4) + (0 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) \times (1 \times 2^0)$

$= 16 + 0 + 4 + 0 + 1$

$= 21_{10}$

Binary Number System

Binary number system has base 2 as it uses 2 digits from 0 to 1.

- Base = 2
  - 2 digits { 0, 1 }, called *b*inary dig*its* or "*bits*"
- Weights
  - Weight = (*Base*) ^{Position}
- Magnitude
  - Sum of "*Bit* x *Weight*"
- Formal Notation
- Groups of bits   4 bits = *Nibble*

    8 bits = *Byte*

4	2	1	1/2	1/4
1	0	1	0	1
2	1	0	-1	-2

$$1*2^2 + 0*2^1 + 1*2^0 + 0*2^{-1} + 1*2^{-2}$$

$$=(5.25)_{10}$$

$$(101.01)_2$$

1 0 1 1

1 1 0 0 0 1 0 1

Hexadecimal Number System

A positional number system

- Hexadecimal number system has total 16 symbols or digits (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F).   Hence its base = 16
- The symbols A, B, C, D, E and F represent the decimal values 10, 11, 12, 13, 14 and 15 respectively
- The maximum value of a single digit is 15 (one less than the value of the base)
- Each position of a digit represents a specific power of the base (16)
- Since there are only 16 digits, 4 bits (2^4 = 16) are   sufficient to represent any hexadecimal number in binary

Ex. $\quad$ **$1AF_{16}$** $\quad$ **= $(1 \times 16^2) + (A \times 16^1) + (F \times 16^0)$**

$\qquad\qquad\qquad$ **= $1 \times 256 + 10 \times 16 + 15 \times 1$**

$\qquad\qquad\qquad$ **= 256 + 160 + 15**

$\qquad\qquad\qquad$ **= 431_{10}**

Hexadecimal Number System

- Uses 10 digits and 6 letters, 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F.

- Base = 16
  - 16 digits { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F }
- Weights
  - Weight = (*Base*) *Position*
- Magnitude
  - Sum of "*Digit* x *Weight*"
- Formal Notation

256	16	1	1/16	1/256
1	E	5	7	A
2	1	0	-1	-2

$$1*16^2 + 14*16^1 + 5*16^0 + 7*16^{-1} + 10*16^{-2}$$

$$=(485.4765625)_{10}$$

$$(1E5.7A)_{16}$$

Addition

- Decimal Addition

$$
\begin{array}{ccc}
\textbf{1} & \textbf{1} & \\
& 5 & 5 \\
+ & 5 & 5 \\
\hline
\textbf{1} & \textbf{1} & \textbf{0}
\end{array}
$$

← **Carry**

= **Ten ≥ Base**

➔ **Subtract a Base**

Addition

➤ **Add and multiply the following numbers in the given base without converting to decimal.**

(a) $(1230)_4$ and $(23)_4$

```
        1
      1 2 3 0
  +       2 3
  ─────────────
      1 3 1 3
```

```
          1 1
          2 2
        1 2 3 0
    x       2 3
  ───────────────
      1 1 0 1 0
      3 1 2 0
  ───────────────
    1 0 2 2 1 0
```

(b) $(135.4)_6$ and $(43.2)_6$

(c) $(367)_8$ and $(715)_8$

Binary Addition

- Column Addition

$$
\begin{array}{cccccc}
\mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \\
 & 1 & 1 & 1 & 1 & 0 & 1 & = 61 \\
+ & & 1 & 0 & 1 & 1 & 1 & = 23 \\
\hline
1 & 0 & 1 & 0 & 1 & 0 & 0 & = 84 \\
\end{array}
$$

$\geq (2)_{10}$

Binary Subtraction

- Borrow a "Base" when needed

$$
\begin{array}{ccccccc}
 & 1 &   &   & 2 &   &   \\
0 & \cancel{2} & 2 & 0 & 0 & 2 & \\
\cancel{1} & 0 & 0 & \cancel{1} & \cancel{1} & 0 & 1 \\
- &   & 1 & 0 & 1 & 1 & 1 \\
\hline
0 & 1 & 1 & 0 & 1 & 1 & 0 \\
\end{array}
$$

$= (10)_2$

$= 77$

$= 23$

$= 54$

Binary Multiplication

- Bit by bit

```
                1   0   1   1   1
        x               1   0   1   0
        ─────────────────────────────
                    0   0   0   0   0
                1   0   1   1   1
            0   0   0   0   0
        1   0   1   1   1
        ─────────────────────────────
        1   1   1   0   0   1   1   0
```

Decimal (*Integer*) to Binary Conversion

- Divide the number by the 'Base' (=2)

- Take the remainder (either 0 or 1) as a coefficient

- Take the quotient and repeat the division

Example: $(13)_{10}$

	Quotient	Remainder	Coefficient
13 / 2 =	**6**	1	$a_0 = 1$
6 / 2 =	**3**	0	$a_1 = 0$
3 / 2 =	**1**	1	$a_2 = 1$
1 / 2 =	**0**	1	$a_3 = 1$

Answer: $(13)_{10} = (a_3\ a_2\ a_1\ a_0)_2 = (1101)_2$

MSB   **LSB**

Decimal (*Fraction*) to Binary Conversion

- Multiply the number by the 'Base' (=2)

- Take the integer (either 0 or 1) as a coefficient

- Take the resultant fraction and repeat the division

Example: (0.625)$_{10}$

		Integer	Fraction	Coefficient
0.625	$* \, 2 =$	1	. 25	$a_{-1} = 1$
0.25	$* \, 2 =$	0	. 5	$a_{-2} = 0$
0.5	$* \, 2 =$	1	. 0	$a_{-3} = 1$

Answer: (0.625)$_{10}$ = (0.$a_{-1} \, a_{-2} \, a_{-3}$)$_2$ = (0.101)$_2$

MSB   **LSB**

Decimal to Octal Conversion

Example: $(175)_{10}$

	Quotient	Remainder	Coefficient
175 / 8 =	21	7	$a_0 = 7$
21 / 8 =	2	5	$a_1 = 5$
2 / 8 =	0	2	$a_2 = 2$

Answer: $(175)_{10} = (a_2\,a_1\,a_0)_8 = (257)_8$

Example: $(0.3125)_{10}$

	Integer	Fraction	Coefficient
0.3125 * 8 =	2 .	5	$a_{-1} = 2$
0.5 * 8 =	4 .	0	$a_{-2} = 4$

Answer: $(0.3125)_{10} = (0.a_{-1}\,a_{-2}\,a_{-3})_8 = (0.24)_8$

Binary − Octal Conversion

- $8 = 2^3$

- Each group of 3 bits represents an octal digit
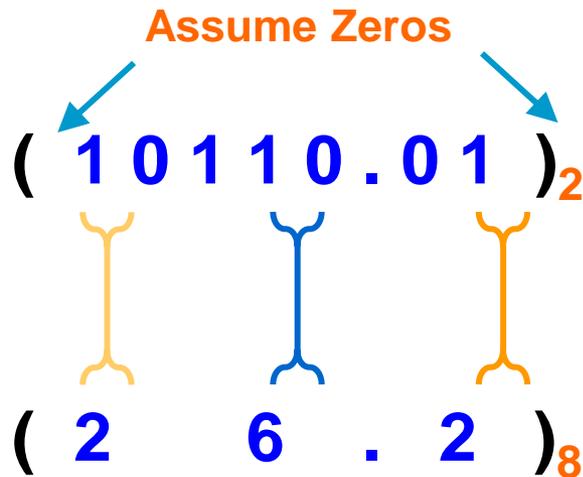
Octal	Binary
0	0 0 0
1	0 0 1
2	0 1 0
3	0 1 1
4	1 0 0
5	1 0 1
6	1 1 0
7	1 1 1

Example:

Assume Zeros

$(\; 1\,0\,1\,1\,0\,.\,0\,1 \;)_2$

$(\; 2 \qquad 6 \quad . \quad 2 \;)_8$

Works both ways (Binary to Octal & Octal to Binary)

Binary − Hexadecimal Conversion

Hex	Binary
0	0 0 0 0
1	0 0 0 1
2	0 0 1 0
3	0 0 1 1
4	0 1 0 0
5	0 1 0 1
6	0 1 1 0
7	0 1 1 1
8	1 0 0 0
9	1 0 0 1
A	1 0 1 0
B	1 0 1 1
C	1 1 0 0
D	1 1 0 1
E	1 1 1 0
F	1 1 1 1

- $16 = 2^4$

- Each group of 4 bits represents a hexadecimal digit

Assume Zeros

Example: $(\ 1\ 0\ 1\ 1\ 0\ .\ 0\ 1 \ )_2$

$(\ 1 \quad 6 \quad . \quad 4 \ )_{16}$
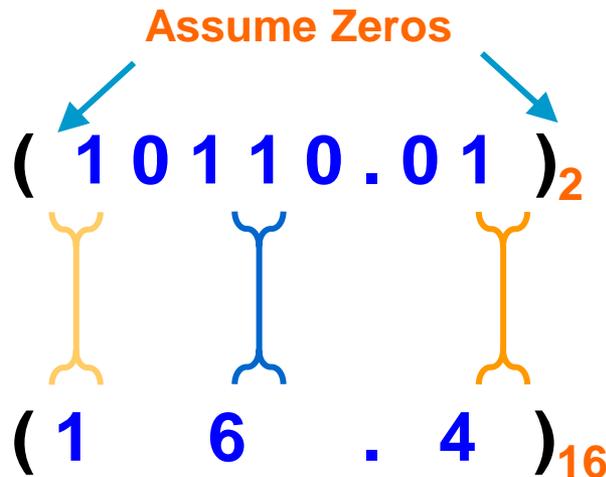
Works both ways (Binary to Hex & Hex to Binary)

Octal – Hexadecimal Conversion

- Convert to Binary as an intermediate step

Example:

$$(\quad 2 \qquad 6 \quad . \quad 2 \quad)_8$$

Assume Zeros →   **Assume Zeros**

$$(0\ 1\ 0\ 1\ 1\ 0\ .\ 0\ 1\ 0\ )_2$$

$$(1 \qquad 6 \quad . \quad 4 \quad)_{16}$$
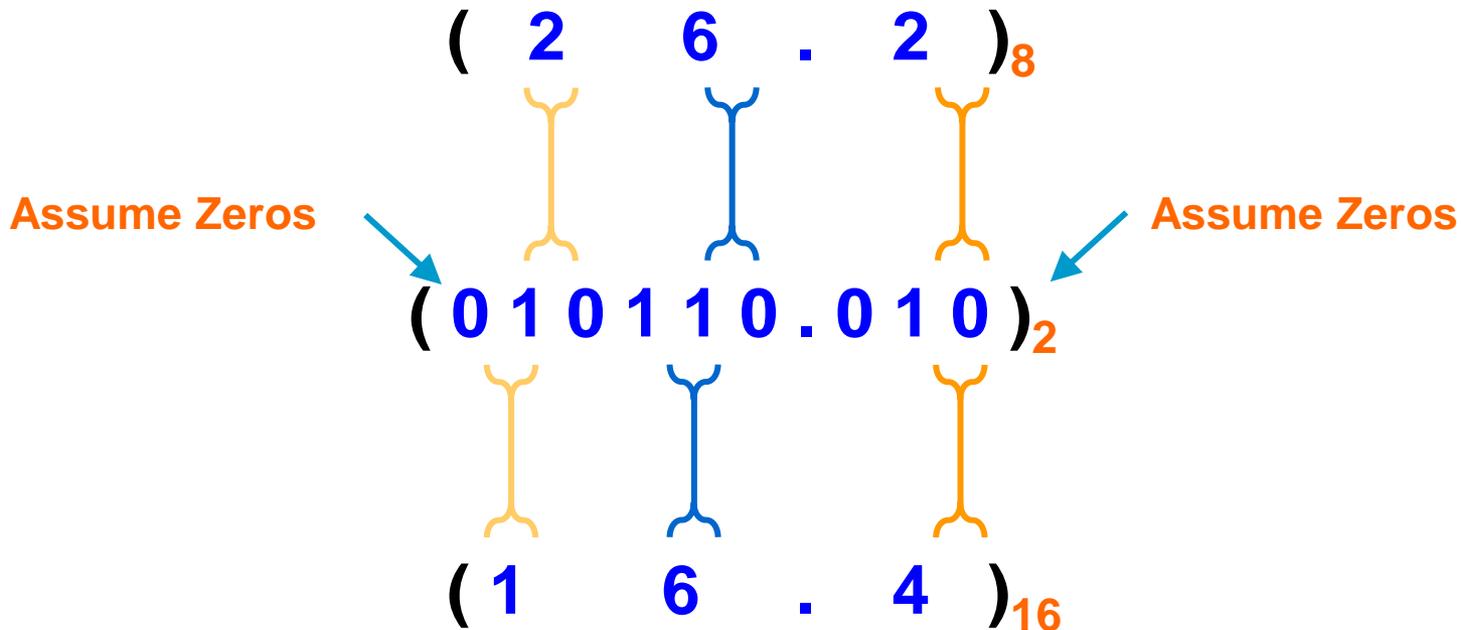
Works both ways (Octal to Hex & Hex to Octal)

Converting a Number of Some Base to a Number of Another Base

Method

Step 1: Convert the original number to a decimal number (base 10)

Step 2: Convert the decimal number so obtained to the new base number

Example

$$545_6 = ?_4$$

Solution:

Step 1: Convert from base 6 to base 10

$$545_6 = 5 \times 6^2 + 4 \times 6^1 + 5 \times 6^0$$
$$= 5 \times 36 + 4 \times 6 + 5 \times 1$$
$$= 180 + 24 + 5$$
$$= 209_{10}$$

Step 2: Convert 209_{10} to base 4

4	209	Remainders
	52	**1**
	13	**0**
	3	**1**
	0	**3**

Hence, $209_{10} = 3101_4$

So, $545_6 = 209_{10} = 3101_4$

Thus, $545_6 = 3101_4$

Convert the following numbers with the indicated bases to decimal :
$(4310)_5$, and $(198)_{12}$.

- $(4310)_5 = 0 \times 5^0 + 1 \times 5^1 + 3 \times 5^2 + 4 \times 5^3 = 0 + 5 + 75 + 500$

  $(4310)_5 = (580)_{10}$

- $(198)_{12} = 8 \times 12^0 + 9 \times 12^1 + 1 \times 12^2 = 8 + 108 + 144$

  $(198)_{12} = (260)_{10}$

Decimal, Binary, Octal and Hexadecimal

Decimal	Binary	Octal	Hex
00	0000	00	0
01	0001	01	1
02	0010	02	2
03	0011	03	3
04	0100	04	4
05	0101	05	5
06	0110	06	6
07	0111	07	7
08	1000	10	8
09	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

1.5   Complements

- There are two types of complements for each base-*r* system: the radix complement and diminished radix complement.

- **Diminished Radix Complement - (r-1)'s Complement**
  - Given a number *N* in base *r* having *n* digits, the (*r*–1)'s complement *of N* is defined as:

$$(r^n - 1) - N$$

- **Example for 6-digit <u>decimal</u> numbers**:
  - 9's complement is $(r^n - 1) - N = (10^6 - 1) - N = 999999 - N$
  - 9's complement of 546700 is $999999 - 546700 = 453299$

- **Example for 7-digit <u>binary</u> numbers:**
  - 1's complement is $(r^n - 1) - N = (2^7 - 1) - N = 1111111 - N$
  - 1's complement of 1011000 is $1111111 - 1011000 = 0100111$

- **Observation:**
  - Subtraction from $(r^n - 1)$ will never require a borrow
  - Diminished radix complement can be computed digit-by-digit
  - For binary: $1 - 0 = 1$ and $1 - 1 = 0$

Complements

- 1's Complement (*Diminished Radix* Complement)
  - All '0's become '1's
  - All '1's become '0's

  Example $(10110000)_2$

  $\Rightarrow (01001111)_2$

  If you add a number and its 1's complement …

$$1\ 0\ 1\ 1\ 0\ 0\ 0\ 0$$
$$+\ 0\ 1\ 0\ 0\ 1\ 1\ 1\ 1$$
$$\overline{1\ 1\ 1\ 1\ 1\ 1\ 1\ 1}$$

Complements

- Radix Complement

> **The r's complement of an n-digit number N in base r is defined as $r^n - N$ for $N \neq 0$ and as 0 for $N = 0$. Comparing with the $(r-1)$'s complement, we note that the r's complement is obtained by adding 1 to the $(r-1)$'s complement, since $r^n - N = [(r^n - 1) - N] + 1$.**

- Example: Base-10

> **The 10's complement of 012398 is 987602**
> **The 10's complement of 246700 is 753300**

- Example: Base-2

> **The 2's complement of 1101100 is 0010100**
> **The 2's complement of 0110111 is 1001001**

Complements

- 2's Complement (*Radix* Complement)
    - Take 1's complement then add 1

OR
    - Toggle all bits to the left of the first '1' from the right

Example:

Number:	1 0 1 1 0 0 0 0	1 0 1 1 0 0 0 0
1's Comp.:	0 1 0 0 1 1 1 1	
	+ 1	
	0 1 0 1 0 0 0 0	0 1 0 1 0 0 0 0

Complements

- Subtraction with Complements
  - The subtraction of two n-digit unsigned numbers $M - N$ in base r can be done as follows:

1. Add the minuend M to the r's complement of the subtrahend N. Mathematically, $M + (r^n - N) = M - N + r^n$.
2. If $M \geq N$, the sum will produce and end carry r^n, which can be discarded; what is left is the result $M - N$.
3. If $M < N$, the sum does not produce an end carry and is equal to $r^n - (N - M)$, which is the r's complement of $(N - M)$. To obtain the answer in a familiar form, take the r's complement of the sum and place a negative sign in front.

Complements

- Example 1.5
  - Using 10's complement, subtract 72532 − 3250.

$$
\begin{array}{rrr}
 & M = & 72532 \\
\text{10's complement of} & N = & +\,96750 \\
\text{Sum} = & & 169282 \\
\text{Discard end carry } 10^5 = & & -\,100000 \\
\text{Answer} = & & 69282
\end{array}
$$

- Example 1.6
  - Using 10's complement, subtract 3250 − 72532.

$$
\begin{array}{rrr}
 & M = & 03250 \\
\text{10's complement of} & N = & +\,27468 \\
\text{Sum} = & & 30718
\end{array}
$$

There is no end carry.

Therefore, the answer is − (10's complement of 30718) = − 69282.

Complements

- Example 1.7
  - Given the two binary numbers X = 1010100 and Y = 1000011, perform the subtraction (a) $X - Y$; and (b) $Y - X$, by using 2's complement.

(a)
$$X = 1010100$$
$$\text{2's complement of } Y = +0111101$$
$$\text{Sum} = 10010001$$
$$\text{Discard end carry } 2^7 = -10000000$$
$$\text{Answer. } X - Y = 0010001$$

(b)
$$Y = 1000011$$
$$\text{2's complement of } X = +0101100$$
$$\text{Sum} = 1101111$$

There is no end carry. Therefore, the answer is Y – X = – (2's complement of 1101111) = – 0010001.

Complements

- Subtraction of unsigned numbers can also be done by means of the $(r-1)$'s complement. Remember that the $(r-1)$'s complement is one less then the r's complement.

- Example 1.8
    - Repeat Example 1.7, but this time using 1's complement.

(a) $X - Y = 1010100 - 1000011$

$$
\begin{aligned}
X &= 1010100 \\
\text{1's complement of } Y &= \pm\, 0111100 \\
\text{Sum} &= 10010000 \\
\text{End-around carry} &= +1 \\
\text{Answer. } X - Y &= 0010001
\end{aligned}
$$

(b) $Y - X = 1000011 - 1010100$

$$
\begin{aligned}
Y &= 1000011 \\
\text{1's complement of } X &= +\,0101011 \\
\text{Sum} &= 1101110
\end{aligned}
$$

There is no end carry, Therefore, the answer is Y − X = − (1's complement of 1101110) = − 0010001.

1.6 Signed Binary Numbers

- To represent negative integers, we need a notation for negative values.

- It is customary to represent the sign with a bit placed in the leftmost position of the number since binary digits.

- The convention is to make the sign bit 0 for positive and 1 for negative.

- Example:

Signed-magnitude representation:	10001001
Signed-1's-complement representation:	11110110
Signed-2's-complement representation:	11110111

- Table 1.3 lists all possible four-bit signed binary numbers in the three representations.

Signed Binary Numbers

Table 1.3
Signed Binary Numbers

Decimal	Signed-2's Complement	Signed-1's Complement	Signed Magnitude
+7	0111	0111	0111
+6	0110	0110	0110
+5	0101	0101	0101
+4	0100	0100	0100
+3	0011	0011	0011
+2	0010	0010	0010
+1	0001	0001	0001
+0	0000	0000	0000
−0	—	1111	1000
−1	1111	1110	1001
−2	1110	1101	1010
−3	1101	1100	1011
−4	1100	1011	1100
−5	1011	1010	1101
−6	1010	1001	1110
−7	1001	1000	1111
−8	1000	—	—

Signed Binary Numbers

- Arithmetic addition
    - The addition of two numbers in the signed-magnitude system follows the rules of ordinary arithmetic. **If the signs are the same**, we add the two magnitudes and give the sum the common sign. **If the signs are different**, we subtract the smaller magnitude from the larger and give the difference the sign if the larger magnitude.
    - The addition of two signed binary numbers with negative numbers represented in signed-2's-complement form is obtained from the addition of the two numbers, including their sign bits.
    - A carry out of the sign-bit position is discarded.
- Example:

+ 6	00000110	− 6	11111010
+13	00001101	+13	00001101
+ 19	00010011	+ 7	00000111
+ 6	00000110	− 6	11111010
−13	11110011	−13	11110011
− 7	11111001	− 19	11101101

Signed Binary Numbers

- Arithmetic Subtraction
  - In 2's-complement form:

> 1. **Take the 2's complement of the subtrahend (including the sign bit) and add it to the minuend (including sign bit).**
> 2. **A carry out of sign-bit position is discarded.**

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

- Example: $(-6) - (-13)$

$(11111010 - 11110011)$

$(11111010 + 00001101)$

$00000111 \ (+7)$

Binary Codes

Digital data is represented, stored and transmitted as groups of binary digits also known as binary code.

Binary codes are codes which are represented in binary system with modification from the original ones.

Classification of various Binary codes
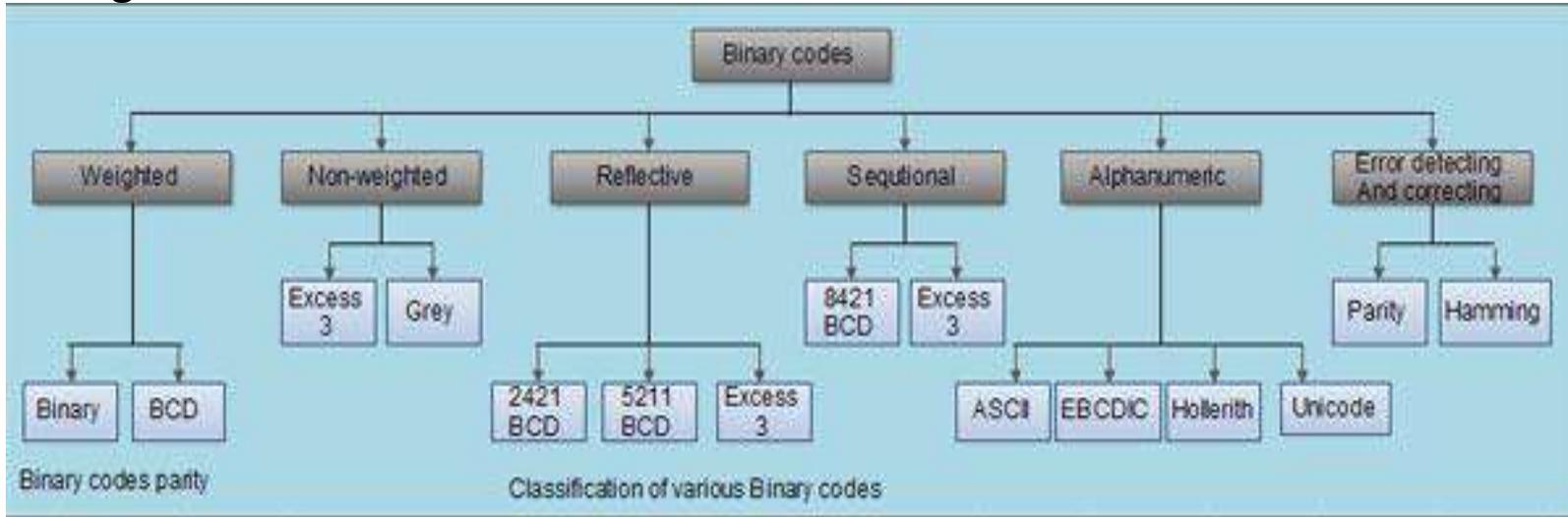
> **Weighted codes:** In weighted codes, each digit is assigned a specific weight according to its position.
> **Non-weighted codes:** Non weighted codes are codes that are not positionally weighted. That is, each position within the binary number is not assigned a fixed value..
> **Reflective codes:** A code is reflective when the code is self complementing. In other words, when the code for 9 is the complement the code for 0, 8 for 1, 7 for 2, 6 for 3 and 5 for 4.
> **Sequential codes:** In sequential codes, each succeeding 'code is one binary number greater than its preceding code.
> **Alphanumeric codes:** Codes used to represent numbers, alphabetic characters, symbols
> **Error defecting and correcting codes:** Codes which allow error defection and correction are called error detecting and' correcting codes.

Self-complementing codes   9's **complement** of a decimal number is obtained by interchanging all 1's and 0's by 0's and 1's respectively.

Decimal	2421 code	Excess code	5211 code
0	0000	0011	0000
1	0001	0100	0001
2	0010	0101	0011
3	0011	0110	0101
4	0100	0111	0111
5	1011	1000	1000
6	1100	1001	1010
7	1101	1010	1100
8	1110	1011	1110
9	1111	1100	1111

- ## BCD Code

  - A number with k decimal digits will require 4k bits in BCD.

  - Decimal 396 is represented in BCD with 12bits as 0011 1001 0110, with each group of 4 bits representing one decimal digit.

  - A decimal number in BCD is the same as its equivalent binary number only when the number is between 0 and 9.

  - The binary combinations 1010 through 1111 are not used and have no meaning in BCD.

Table 1.4
Binary-Coded Decimal (BCD)

Decimal Symbol	BCD Digit
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Example: **Consider decimal 185 and its corresponding value in BCD and binary:**

$$(185)_{10} = (0001\ 1000\ 0101)_{BCD} = (10111001)_2$$

BCD addition

```
  4     0100     4     0100     8     1000
+ 5   + 0101   + 8   + 1000   + 9   + 1001
  9     1001    12     1100    17    10001
                     + 0110         + 0110
                       10010          10111
```

Binary Codes

- Other Decimal Codes

Table 1.5
Four Different Binary Codes for the Decimal Digits

Decimal Digit	BCD 8421	2421	Excess-3	8, 4, −2, −1
0	0000	0000	0011	0000
1	0001	0001	0100	0111
2	0010	0010	0101	0110
3	0011	0011	0110	0101
4	0100	0100	0111	0100
5	0101	1011	1000	1011
6	0110	1100	1001	1010
7	0111	1101	1010	1001
8	1000	1110	1011	1000
9	1001	1111	1100	1111
	1010	0101	0000	0001
Unused	1011	0110	0001	0010
bit	1100	0111	0010	0011
combi-	1101	1000	1101	1100
nations	1110	1001	1110	1101
	1111	1010	1111	1110

Gray Code:

The gray code belongs to a class of codes called minimum change codes, in which only one bit in the code changes when moving from one code to the next.

The Gray code is non-weighted code, as the position of bit does not contain any weight.

The gray code is a reflective digital code which has the special property that any two subsequent numbers codes differ by only one bit. This is also called a unit-distance code. In digital Gray code has got a special place.

Binary Codes

- Gray Code
  - The advantage is that only bit in the code group changes in going from one number to the next.
    - Gray code is also known as **reflected binary code.**
    - We call it gray code after frank gray.
    - Unweighted code.
    - Two successive values differ in only 1 bit.
    - Binary number is converted to gray code to reduce switching operation.
    - **Unit distance code** & **min. error code.**
    - **Cyclic code**
    - Low power design.

Table 1.6
Gray Code

Gray Code	Decimal Equivalent
0000	0
0001	1
0011	2
0010	3
0110	4
0111	5
0101	6
0100	7
1100	8
1101	9
1111	10
1110	11
1010	12
1011	13
1001	14
1000	15

Binary Codes

Binary to Gray code Conversion

Step 1: Record the MSB as it is.

Step 2: Add the MSB to the next bit, record the sum and neglect
the carry. (X-OR)

Step 3: Repeat the process.

Exp. Convert 1011 to Gray code.

MSB   LSB

1 0 1 1   (Binary)

1 1 1 0   (Gray)

Binary Codes

Gray code to Binary Conversion

Step 1: Record the MSB as it is.

Step 2: Add the MSB to the next bit of Gray code, record the sum and neglect the carry. (X-OR)

Step 3: Repeat the process.

Exp. Convert 1011 to Gray code.

<div align="center">

MSB   LSB

1 0 1 1   (Gray)

1 1 0 1   (Binary)

</div>
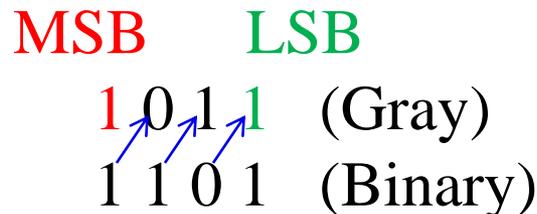
- American Standard Code for Information Interchange (ASCII) Character Code

Table 1.7
American Standard Code for Information Interchange (ASCII)

$b_4b_3b_2b_1$	$b_7b_6b_5$							
	000	**001**	**010**	**011**	**100**	**101**	**110**	**111**
0000	NUL	DLE	SP	0	@	P	`	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	''	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	'	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	l	|
1101	CR	GS	−	=	M	]	m	}
1110	SO	RS	.	>	N	∧	n	~
1111	SI	US	/	?	O	−	o	DEL

- ASCII Character Code

Control characters

NUL	Null	DLE	Data-link escape
SOH	Start of heading	DC1	Device control 1
STX	Start of text	DC2	Device control 2
ETX	End of text	DC3	Device control 3
EOT	End of transmission	DC4	Device control 4
ENQ	Enquiry	NAK	Negative acknowledge
ACK	Acknowledge	SYN	Synchronous idle
BEL	Bell	ETB	End-of-transmission block
BS	Backspace	CAN	Cancel
HT	Horizontal tab	EM	End of medium
LF	Line feed	SUB	Substitute
VT	Vertical tab	ESC	Escape
FF	Form feed	FS	File separator
CR	Carriage return	GS	Group separator
SO	Shift out	RS	Record separator
SI	Shift in	US	Unit separator
SP	Space	DEL	Delete

ASCII Character Codes and Properties

- American Standard Code for Information Interchange (Refer to Table 1.7)

- A popular code used to represent information sent as character-based data.

- It uses 7-bits to represent:
  - 94 Graphic printing characters.
  - 34 Non-printing characters.

- Some non-printing characters are used for text format (e.g. BS = Backspace, CR = carriage return).

- Other non-printing characters are used for record marking and flow control (e.g. STX and ETX start and end text areas).

- ASCII has some interesting properties:

  - Digits 0 to 9 span Hexadecimal values 30_{16} to 39_{16}

  - Upper case A-Z span 41_{16} to $5A_{16}$

  - Lower case a-z span 61_{16} to $7A_{16}$
    - Lower to upper case translation (and vice versa) occurs by flipping bit 6.

Error-Detecting Code

- Data can be corrupted during transmission. For reliable communication, errors must be detected and corrected.

- To detect errors in data communication and processing, an <u>eighth bit</u> is sometimes added to the ASCII character to indicate its parity.

- A parity bit is an extra bit included with a message to make the total number of 1's either even or odd.

- Example:

  - Consider the following two characters and their even and odd parity:

	With even parity	With odd parity
ASCII A = 1000001	01000001	11000001
ASCII T = 1010100	11010100	01010100

- **Error-Detecting Code**
  - Redundancy (e.g. extra information), in the form of extra bits, can be incorporated into binary code words to detect and correct errors.
  - A simple form of redundancy is parity, an extra bit appended onto the code word to make the number of 1's odd or even. Parity can detect all single-bit errors and some multiple-bit errors.
  - A code word has even parity if the number of 1's in the code word is even.
  - A code word has odd parity if the number of 1's in the code word is odd.
  - Example: **Message A: 10001001 1   (even parity)**

    Message B: 10001001 0   (odd parity)

Hamming Codes

- It is not only provide to detect an error but also find out where the error occurred and correct it.
- In this code the receiver can detect whether there is any error in the received signal or not and also the bit (position) at which error happened.
- The formula to calculate the number of bits of hamming code is given as (number of parity bits depend on the message)

$$m+k+1 \leq 2^k$$

Where m number of message bits

k number of parity bits required

Thus k gives the number of bits that are to transmitted along with message.

Hence we have to make the hamming code at the transmitted end.

Let us assume as a seven bit message: 1010110

 hence m = 7

So, putting m=7 in the formula, the k is satisfied by

k = 4

$$7+k+1 \leq 2^k$$
$$7+4+1 \leq 2^4$$
$$11+1 \leq 16$$

So, the number of hamming code bits required to be transmitted along with the message are 4.

Hence, the positions of parity bits are given as:

Positions (parity bits) Checks (bits to be checked)

$P_1 = 2^0$ (1st Column)   (1,3,5,7,9,11) bits

$P_2 = 2^1$ (2nd Column)   (2,3,6,7,10,11) bits

$P_4 = 2^2$ (3rd Column)   (4,5,6,7,9) bits

$P_8 = 2^3$ (4th Column)   (8,9,10,11) bits

Thus the message will now be 11 bits long

11	10	9	8	7	6	5	4	3	2	1
m_{11}	m_{10}	m_9	P_8	m_7	m_6	m_5	P_4	m_3	P_2	P_1
1	0	1	P_8	0	1	1	P_4	0	P_2	P_1

The value of parity bits can be calculated as:

Checks   Even Parity

$P_1 \rightarrow$(1, 3, 5, 7, 9, 11) $P_1 = 1$

P_1 0 1 0 1 1

$p_8 p_4 p_2 p_1$

0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	1010
11	1011

m_{11} m_{10} m_9 P_8 m_7 m_6 m_5 P_4 m_3 P_2 P_1

1   0   1 P_8 0   1   1 P_4 0 P_2 P_1

	$p_8p_4p_2p_1$
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	1010
11	1011

         Checks   Even Parity

$P_2 \rightarrow (2, 3, 6, 7, 10, 11)$ $P_2 = 0$

     P_2 0   1   0   0   1

$P_4 \rightarrow (4, 5, 6, 7, 9)$ $P_4 = 1$

     P_4 1   1   0   1

$P_8 \rightarrow (8,9,10,11)$ $P_8 = 0$

     P_4 1   0   1

The hamming's code is as

1 0 1 0 0 1 1 1 0 0 1

Reception of hamming's code and error detection

Let the received code for same message is as

  1   0   1   0   1   1   1   1   0   0   1

m_{11} m_{10} m_9 P_8 m_7 m_6 m_5 P_4 m_3 P_2 P_1

$$1 \quad 0 \quad 1 \quad 0 \quad 1 \quad 1 \quad 1 \quad 1 \quad 0 \quad 0 \quad 1$$
$$m_{11} \, m_{10} \, m_9 \, P_8 \, m_7 \, m_6 \, m_5 \, P_4 \, m_3 \, P_2 \, P_1$$

here   checks   Even Parity

P_1 ➡ $(1, 3, 5, 7, 9, 11) \rightarrow 1\,0\,1\,1\,1\,1 \rightarrow 1$

P_2 ➡ $(2, 3, 6, 7, 10, 11) \rightarrow 0\,0\,1\,1\,0\,1 \rightarrow 1$ Read from bottom to top side

P_4 ➡ $(4, 5, 6, 7, 9) \rightarrow 1\,1\,1\,1\,1 \rightarrow 1$

P_8 ➡ $(8,9,10,11) \rightarrow 0\,1\,0\,1 \rightarrow 0$

$P_8 \, P_4 \, P_2 \, P_1$ ➡ $0111 = 7^{th}$ bit

Hence the error is at 7th bit

To eliminate the error always complement the error bit.

$$1\,0\,1\,0\,0\,1\,1\,1\,0\,0\,1$$

If there is no error then $P_8 \, P_4 \, P_2 \, P_1 \rightarrow 0000$. which means there is no error.

Binary Storage and Registers

- Registers
  - A binary cell is a device that possesses two stable states and is capable of storing one of the two states.
  - A register is a group of binary cells. A register with *n* cells can store any discrete quantity of information that contains *n* bits.

  n cells ➡ **2^n possible states**

- A binary cell
  - Two stable state
  - Store one bit of information
  - Examples: flip-flop circuits, capacitor

- A register
  - A group of binary cells
  - AX in x86 CPU

- Register Transfer
  - A transfer of the information stored in one register to another.
  - One of the major operations in digital system.
  - An example in next slides.

General-Purpose Computers are an example of binary systems. A Digital Computer Example

```
┌─────────────────────────────────────────────┐
│                  Memory                       │
└─────────────────────────────────────────────┘
         ▲                        │
         │                        ▼
┌─────────────────────────────────────────────┐
│  CPU   ┌──────────┐        ┌──────────┐      │
│        │ Control  │  ───▶  │          │      │
│        │   unit   │        │ Datapath │      │
│        │          │  ◀───  │          │      │
│        └──────────┘        └──────────┘      │
└─────────────────────────────────────────────┘
         ▲                        │
         │                        ▼
┌─────────────────────────────────────────────┐
│                Input/Output                   │
└─────────────────────────────────────────────┘
```

Inputs: Keyboard, mouse, modem, microphone

Outputs: CRT, LCD, modem, speakers

Synchronous or Asynchronous?
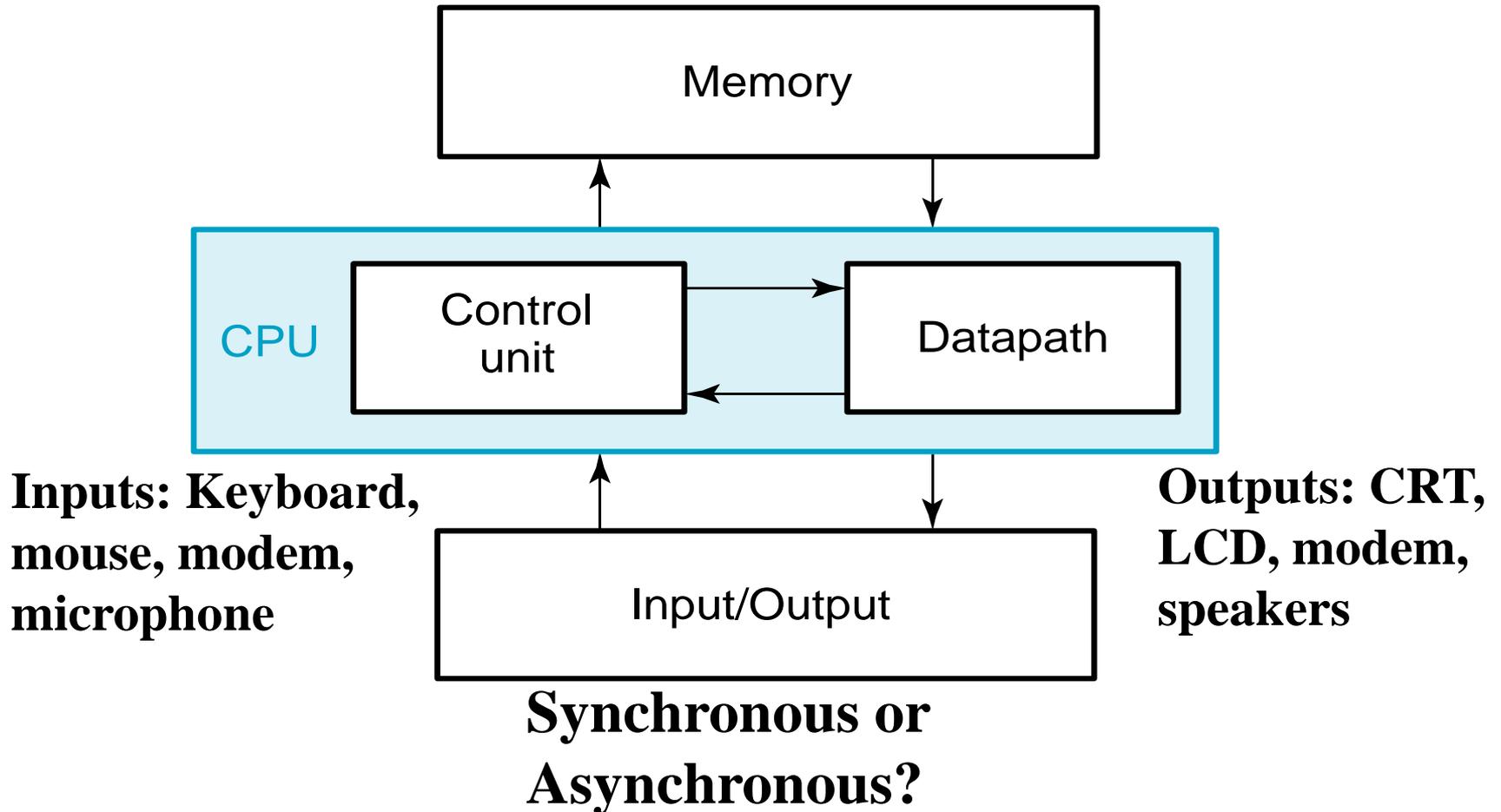
The Digital Computer

- The memory stores programs as well as input, output, ….
- The datapath (ALU) performs arithmetic and other data processing operations as specified by the program.
- The control unit supervises the flow of information between the various units.
- A datapath, when combined with the control unit, forms a component referred to as a central processing unit, or CPU.
- Input device: keyboard.
- Output device: LCD (liquid crystal display).
        -These devices use digital logic circuits

Transfer of information
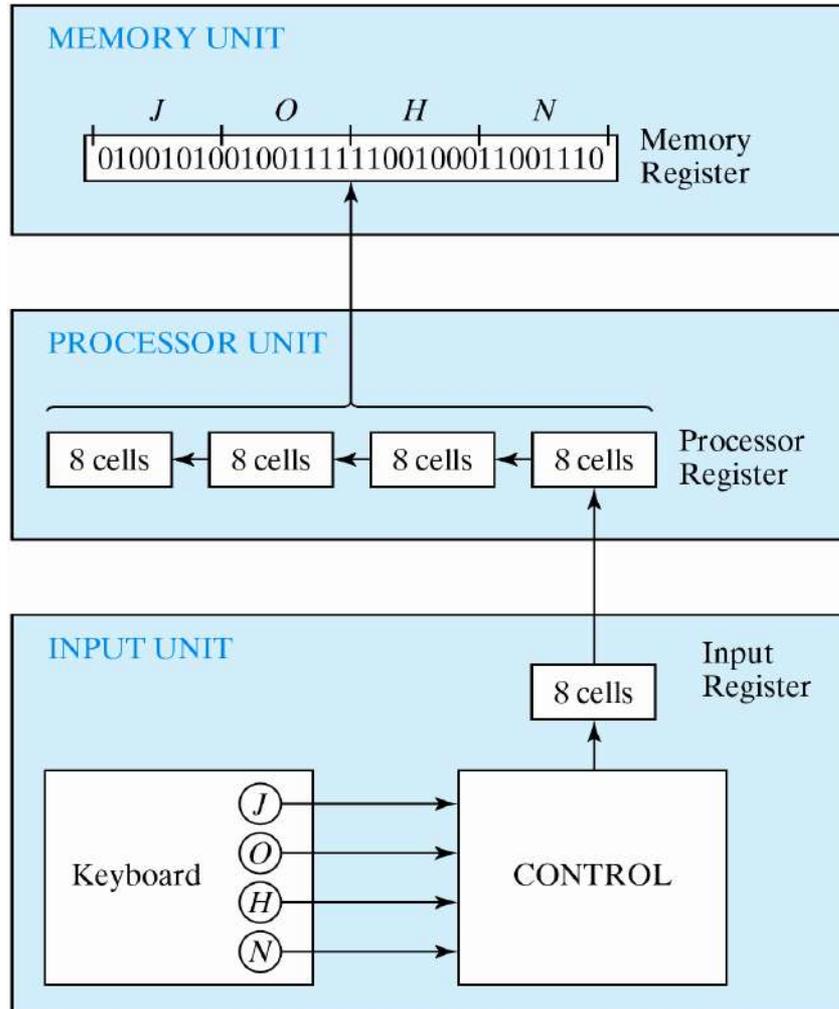
Figure 1.1 Transfer of information among register

Transfer of information
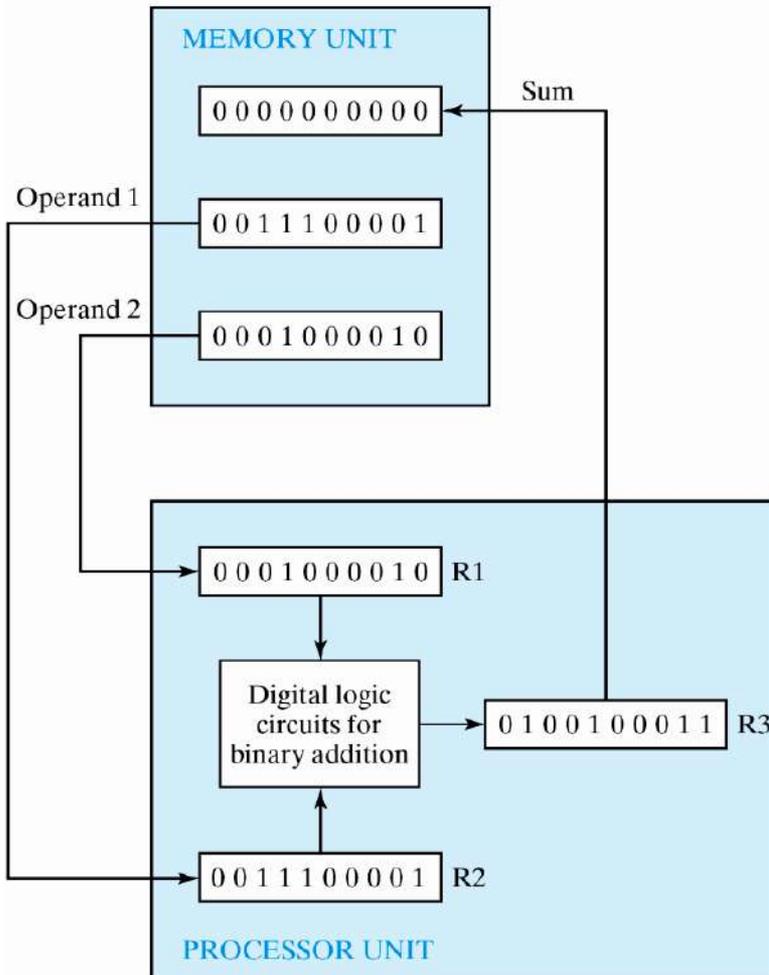
- • The other major component of a digital system
  - • Circuit elements to manipulate individual bits of information
  - • Load-store machine

LD	R1;
LD	R2;
ADD	R3, R2, R1;
SD	R3;

Figure 1.2 Example of binary information processing

Binary Logic

- Definition of Binary Logic
    - Binary logic consists of binary variables and a set of logical operations.
    - The variables are designated by letters of the alphabet, such as *A*, *B*, *C*, *x*, *y*, *z*, etc, with each variable having two and only two distinct possible values: 1 and 0,
    - Three basic logical operations: AND, OR, and NOT.

1. AND: This operation is represented by a dot or by the absence of an operator. For example, $x \cdot y = z$ or $xy = z$ is read "*x* AND *y* is equal to *z*," The logical operation AND is interpreted to mean that $z = 1$ if only $x = 1$ and $y = 1$; otherwise $z = 0$. (Remember that x, y, and z are binary variables and can be equal either to 1 or 0, and nothing else.)

2. OR: This operation is represented by a plus sign. For example, $x + y = z$ is read "*x* OR *y* is equal to *z*," meaning that $z = 1$ if $x = 1$ or $y = 1$ or if both $x = 1$ and $y = 1$. If both $x = 0$ and $y = 0$, then $z = 0$.

3. NOT: This operation is represented by a prime (sometimes by an overbar). For example, $x' = z$ (or $\bar{x} = z$) is read "not *x* is equal to *z*," meaning that *z* is what *z* is not. In other words, if $x = 1$, then $z = 0$, but if $x = 0$, then $z = 1$, The NOT operation is also referred to as the complement operation, since it changes a 1 to 0 and a 0 to 1.

Binary Logic Function

$F(vars) = expression$

Operators (+, •, ')

Variables

Constants (0, 1)

Groupings (parenthesis)

set of binary variables

Example: $F(a,b) = a' \bullet b + b'$

$G(x,y,z) = x \bullet (y+z')$

Binary Logic Function

- AND
- OR
- NOT

Binary

Unary
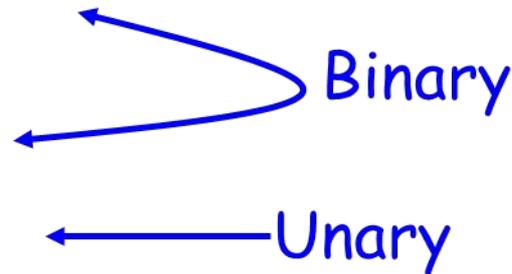
- $F(a,b) = a \bullet b$,   F is 1 <u>if and only if</u> a=b=1
- $G(a,b) = a+b$, G is 1 if either a=1 or b=1
- $H(a) = a'$,   H is 1 if a=0

Binary Logic gates

- Truth Tables, Boolean Expressions, and Logic Gates

- **Truth table**: tabular form that <u>uniquely</u> represents the relationship between the input variables of a function and its output

AND

x	y	z
0	0	0
0	1	0
1	0	0
1	1	1

$$\mathbf{z = x \bullet y = x\,y}$$

OR

x	y	z
0	0	0
0	1	1
1	0	1
1	1	1

$$\mathbf{z = x + y}$$

NOT

x	z
0	1
1	0

$$\mathbf{z = \overline{x} = x'}$$

Logic gates are abstractions of electronic circuit components that operate on one or more input signals to produce an output signal.

AND

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

$X = A \cdot B$

NAND

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

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

OR

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

$X = A + B$

NOR

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

$X = \overline{A + B}$

XOR

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

$X = A \oplus B$

Buffer

A	X
0	0
1	1

$X = A$

Inverter (NOT)

A	X
0	1
1	0

$X = \overline{A}$
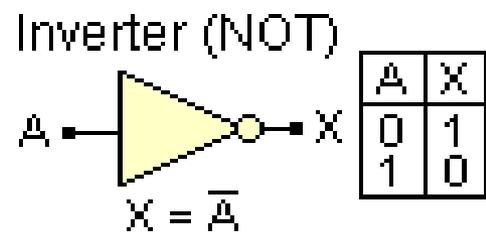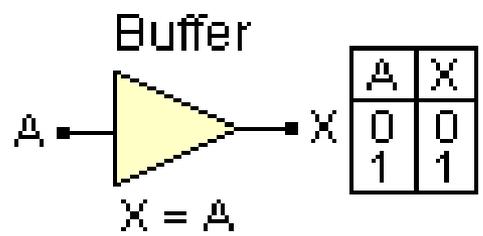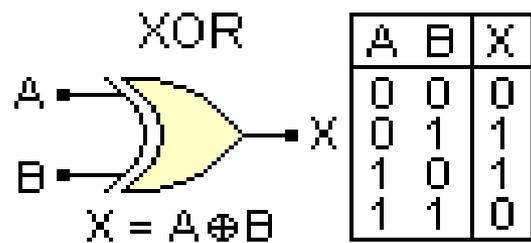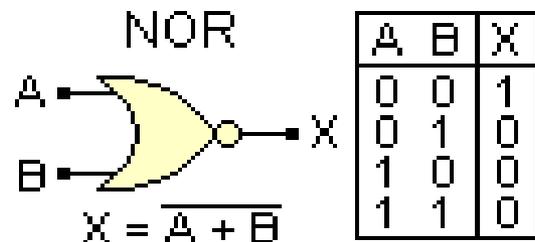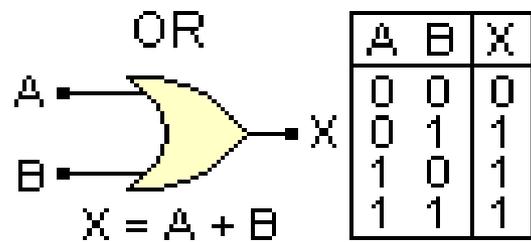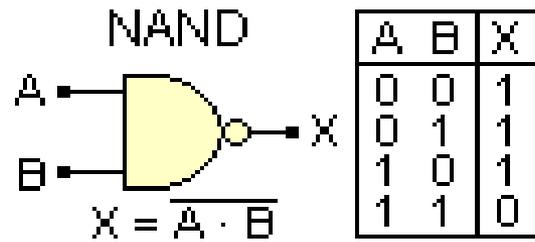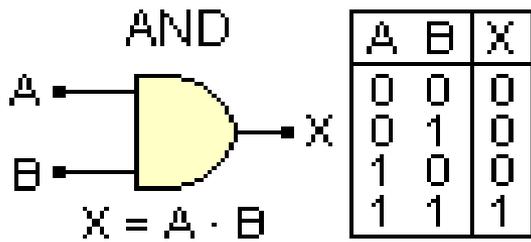
Logic Function	Boolean Notation
AND	A.B
OR	A+B
NOT	$\overline{A}$
NAND	$\overline{A \cdot B}$
NOR	$\overline{A+B}$
EX-OR	$(A.\overline{B}) + (\overline{A}.B)$ or $A \oplus B$
EX-NOR	$\overline{(A.B)} +$ or $\overline{A \oplus B}$

Universal Gate

- **NAND and NOR** Gates are called *Universal Gates* because AND, OR and NOT gates can be implemented &created by using these gates.

NAND Gate Implementations
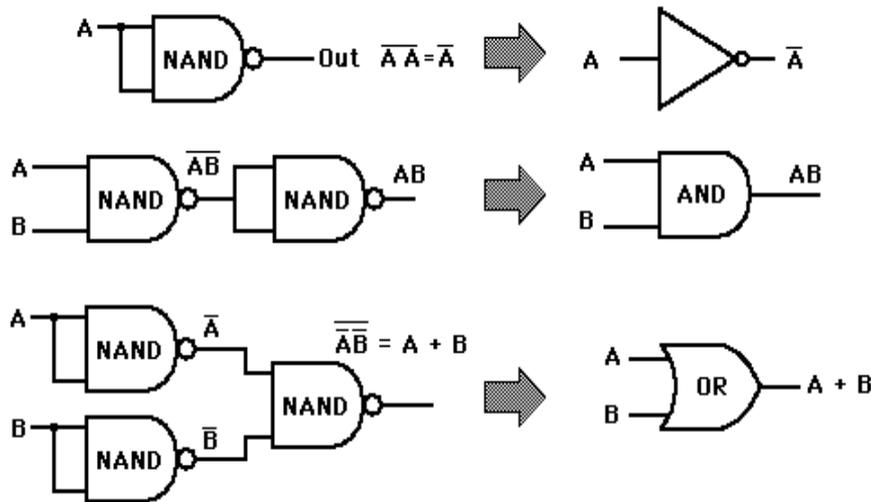
NOR Gate Implementations
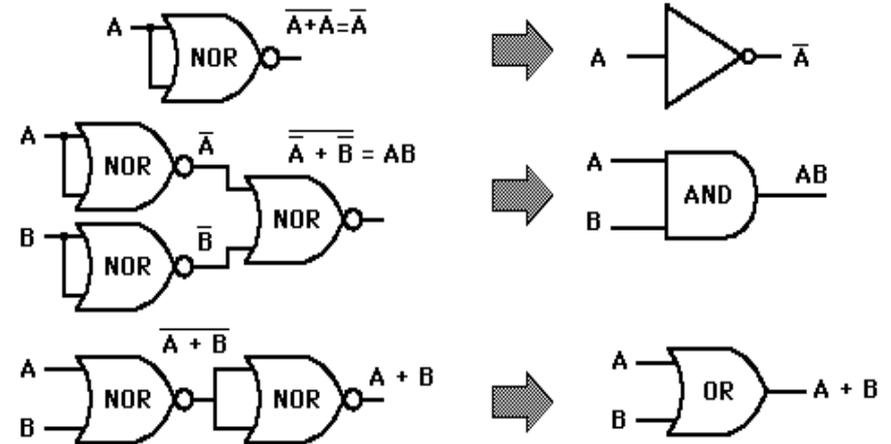
Binary Logic

- Logic gates
  - Graphic Symbols and Input-Output Signals for Logic gates:
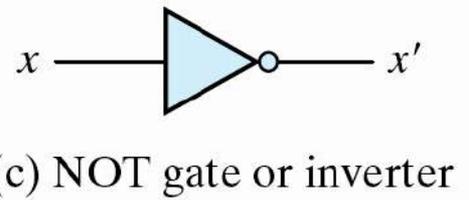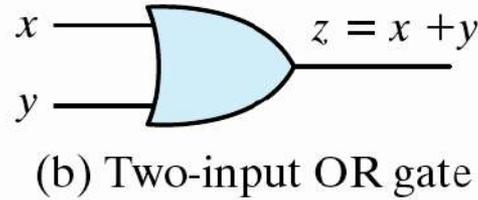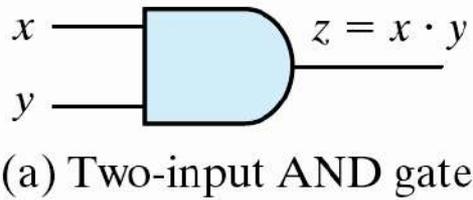
Fig. 1.4 Symbols for digital logic circuits
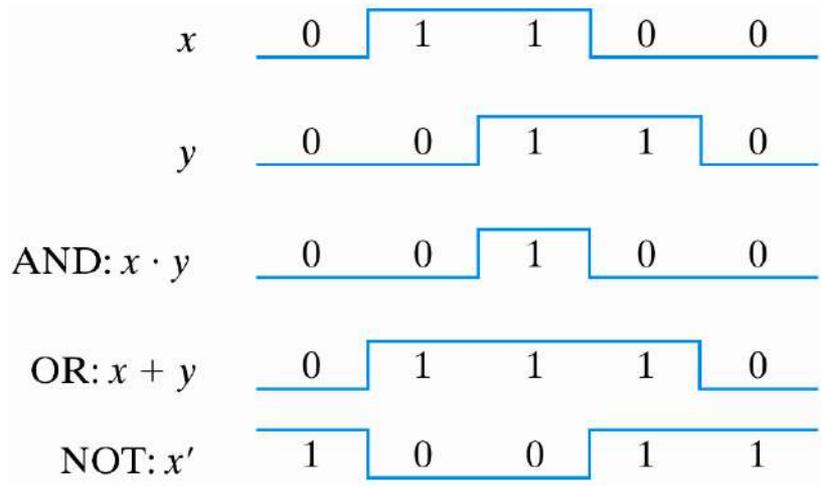
Fig. 1.5 Input-Output signals for gates

Binary Logic

- Logic gates
  - Graphic Symbols and Input-Output Signals for Logic gates:
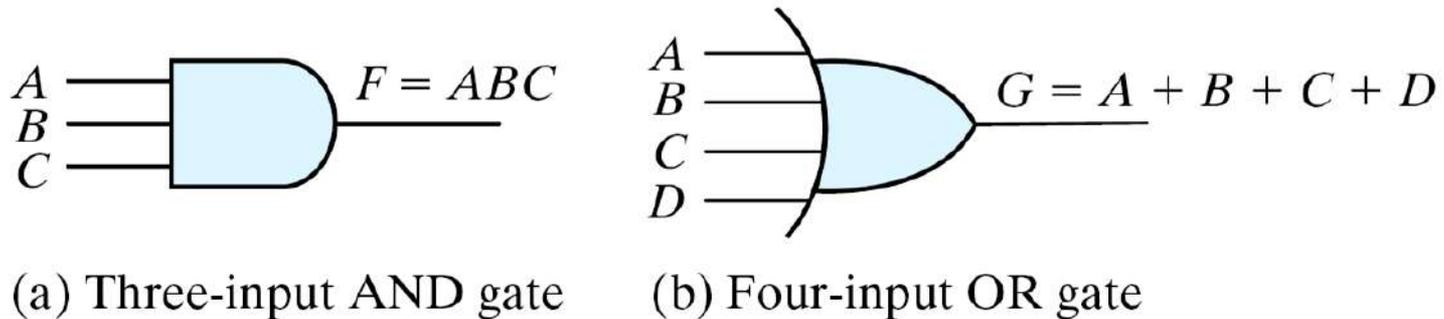
Fig. 1.6   Gates with multiple inputs

Combinational Logic Circuit from Logic Function

- Consider function F = A' + B•C' + A'•B'
- A combinational logic circuit can be constructed to implement F, by appropriately connecting input signals and logic gates:
  - Circuit input signals → from function variables (A, B, C)
  - Circuit output signal → function output (F)
  - Logic gates → from logic operations
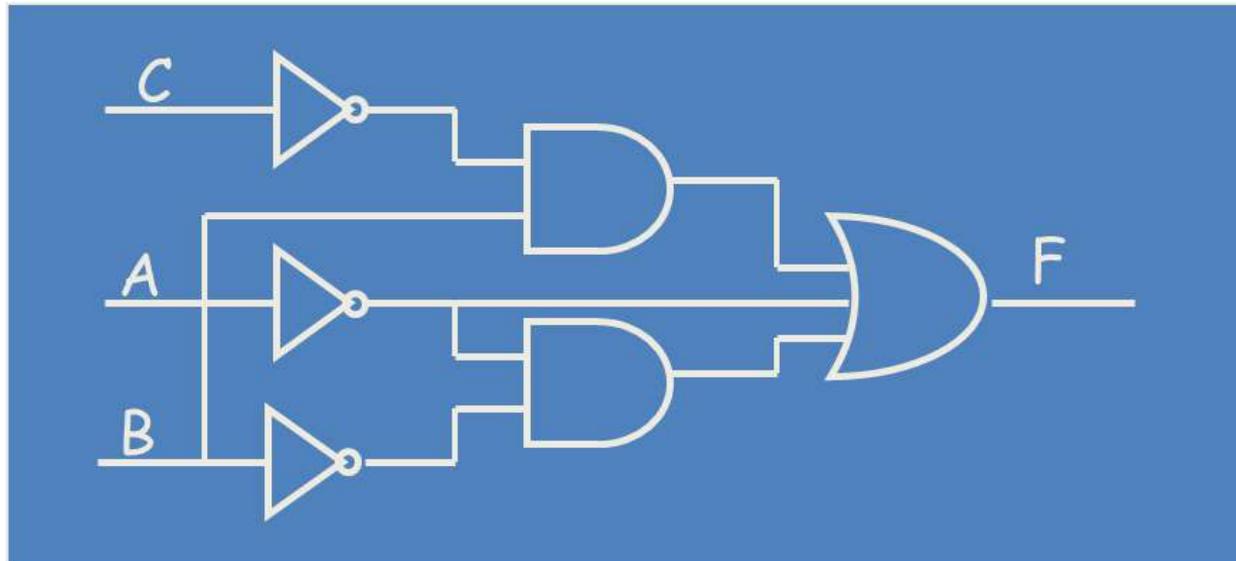
Combinational Logic Circuit from Logic Function

- In order to design a cost-effective and efficient circuit, we must minimize the circuit's size (area) and propagation delay (time required for an input signal change to be observed at the output line)

- Observe the truth table of F=A' + B•C' + A'•B' and G=A' + B•C'

- Truth tables for F and G are identical → same function

- Use G to implement the logic circuit (less components)

A	B	C	F	G
0	0	0	1	1
0	0	1	1	1
0	1	0	1	1
0	1	1	1	1
1	0	0	0	0
1	0	1	0	0
1	1	0	1	1
1	1	1	0	0

Combinational Logic Circuit from Logic Function
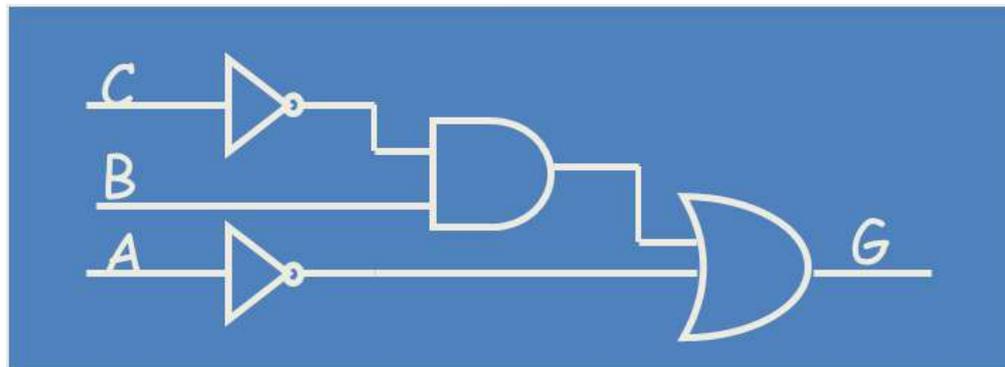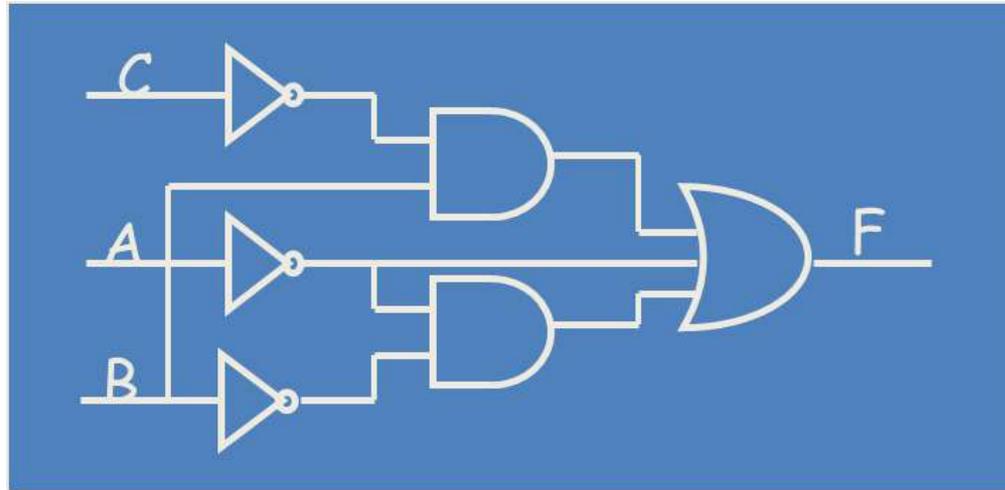
Boolean Algebra

Boolean Algebra : **George Boole(English mathematician), 1854**

- **It was Invented by George Boole in year 1854**
- **An algebraic structure defined by a set B = {0, 1}, together with two binary operators (+ and ·) and a unary operator (   )**

"An Investigation of the Laws of Thought, on Which Are Founded the Mathematical Theories of Logic and Probabilities"

Boolean Algebra

{(1,0), Var, (NOT, AND, OR)}

➢ **Boolean algebra is a useful tool for simplifying digital circuits.**
➢ **Terminology:**
  ✓ *Literal:* A variable or its complement
  ✓ *Product term:* literals connected by •
  ✓ *Sum term:* literals connected by +

Basic Functions

Boolean functions : NOT, AND, OR,

exclusive OR(XOR) : odd function   exclusive NOR(XNOR) : even function(equivalence)

Boolean functions for (a) AND, (b) OR, (c) XOR, and (d) NOT

x	y	$x \wedge y$		x	y	$x \vee y$		x	y	$x \oplus y$		x	x'
0	0	0		0	0	0		0	0	0		0	1
0	1	0		0	1	1		0	1	1		1	0
1	0	0		1	0	1		1	0	1			
1	1	1		1	1	1		1	1	0			
		(a)				(b)				(c)			(d)

Basic functions

- **AND   Z=X · Y or Z=XY**

  Z=1 if and only if X=1 and Y=1, otherwise Z=0

- **OR   Z=X + Y**

  Z=1 if X=1 or if Y=1, or both X=1and Y=1. Z=0 if and only if X=0 and Y=0

- **NOT   Z=X′ or**

  Z=1 if X=0, Z=0 if X=1

Boolean functions for (a) NAND, (b) NOR, and (c) XNOR

x	y	NAND
0	0	1
0	1	1
1	0	1
1	1	0

(a)

x	y	NOR
0	0	1
0	1	0
1	0	0
1	1	0

(b)

x	y	XNOR
0	0	1
0	1	0
1	0	0
1	1	1

(c)

All possible binary boolean functions

x	y	0	∧	xy′	x	x′y	y	⊕	∨	NOR	XNOR	y′	x + y′	x′	x′+ y	NAND	1
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Basic Identities of Boolean Algebra

Basic Identities of Boolean Algebra

1.	$X + 0 = X$	2.	$X \cdot 1 = X$
3.	$X + 1 = 1$	4.	$X \cdot 0 = 0$
5.	$X + X = X$	6.	$X \cdot X = X$
7.	$X + \overline{X} = 1$	8.	$X \cdot \overline{X} = 0$
9.	$\overline{\overline{X}} = X$		

The relationship between a single variable X, its complement X′, and the binary constants 0 and 1

10.	$X + Y = Y + X$	11.	$XY = YX$	Commutative
12.	$X + (Y + Z) = (X + Y) + Z$	13.	$X(YZ) = (XY)Z$	Associative
14.	$X(Y + Z) = XY + XZ$	15.	$X + YZ = (X + Y)(X + Z)$	Distributive
16.	$\overline{X + Y} = \overline{X} \cdot \overline{Y}$	17.	$\overline{X \cdot Y} = \overline{X} + \overline{Y}$	DeMorgan's

Duality: **The dual of an expression is obtained by exchanging (• and +), and (1 and 0) in it, provided that the precedence of operations is not changed.**

Laws of Boolean Algebra

- Commutative Law: the order of literals does not matter

A + B = B + A   A B = B A
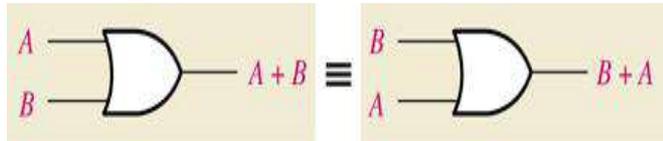
- Associative Law: the grouping of literals does not matter

A + (B + C) = (A + B) + C (=A+B+C)   A(BC) = (AB)C (=ABC)
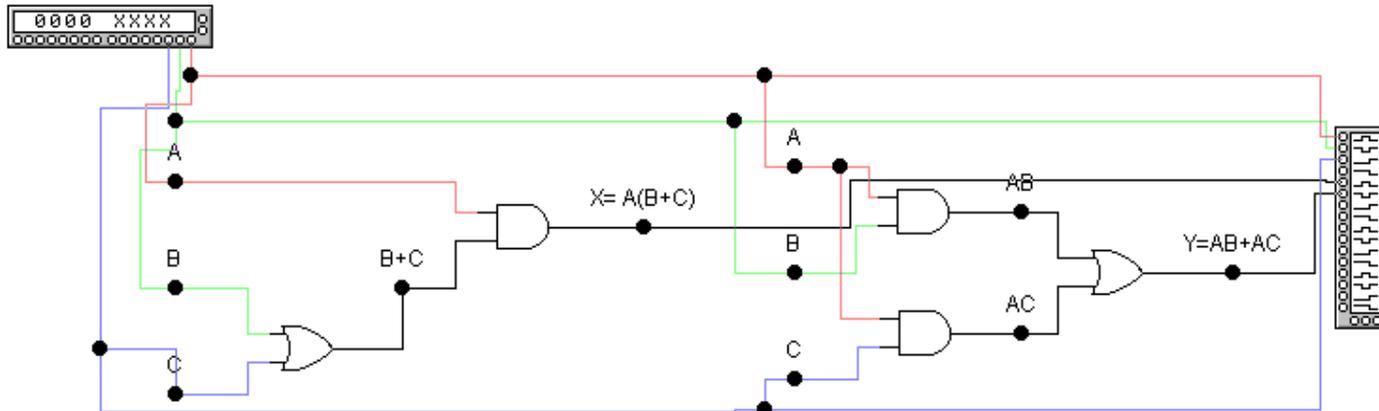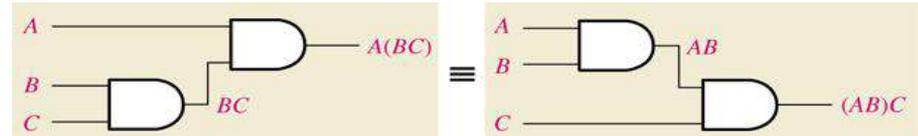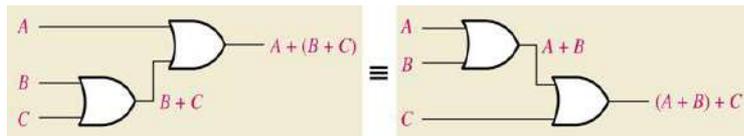
Rules of Boolean Algebra

✓ A+0=A   In math if you add 0 you have changed nothing in Boolean Algebra ORing with 0 changes nothing

✓ A•0=0   In math if 0 is multiplied with   anything you get 0. If you AND anything with 0 you get 0

✓ A•1 =A   ANDing anything with 1 will yield the anything

✓ A+A = A   ORing with itself will give the same

✓ A+A'=1   Either A or A' must be 1 so A + A' =1

✓ A•A = A   ANDing with itself will give the same result

✓ A•A' =0 In digital Logic 1' =0 and 0' =1, so AA'=0 since one of the inputs must be 0.

✓ A = (A')' If you not something twice you are back to the beginning
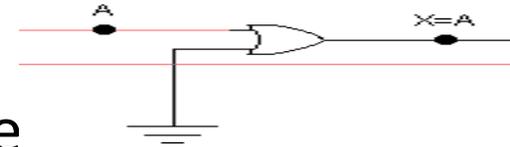
✓ A + A'B = A + B

If A is 1 the output is 1   If A is 0 the output is B

✓ A + AB = A

✓ (A + B)(A + C) = A + BC

DeMorgan's Theorem

- (A • B)' = A' + B' and (A + B)' = A' • B'

- DeMorgan's theorem will help to simplify digital circuits using NORs and NANDs his theorem states
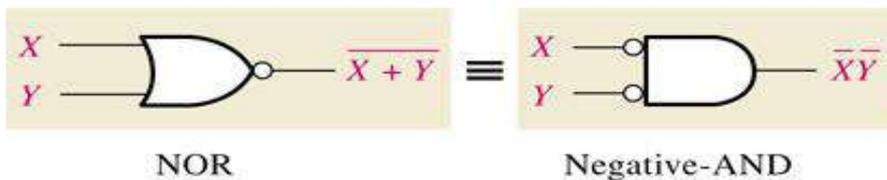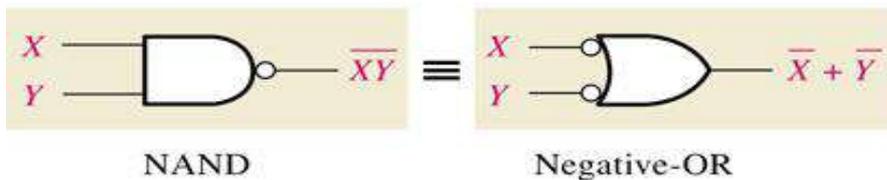
Inputs		Output	
X	Y	$\overline{XY}$	$\overline{X} + \overline{Y}$
0	0	1	1
0	1	1	1
1	0	1	1
1	1	0	0

Inputs		Output	
X	Y	$\overline{X + Y}$	$\overline{X}\overline{Y}$
0	0	1	1
0	1	0	0
1	0	0	0
1	1	0	0

Boolean Analysis of Logic Circuits

- Constructing a Truth Table for a Logic Circuit
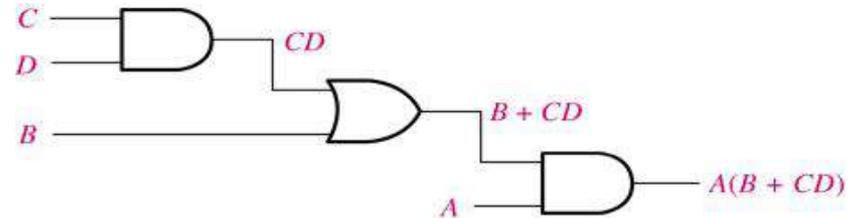
    - Convert the expression into the min-terms containing all the input literals
    - Get the numbers from the min-terms
    - Putting '1's in the rows corresponding to the min-terms and '0's in the remains

Ex) A(B+CD)=AB(C+C') (D+D') +A(B+B')CD =ABC(D+D') +ABC'(D+D') +ABCD+AB'CD =ABCD+ABCD'+ABC'D+ABC'D' +ABCD+AB'CD =ABCD+ABCD'+ABC'D+ABC'D' +AB'CD =m11+m12+m13+m14+m15=Σ(11,12,13,14,15)

A(B+CD) = m11+m12+m13+m14+m15 =Σ(11,12,13,14,15)

Input				Output
A	B	C	D	A(B+CD)
0	0	0	0	0
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	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Standard Forms of Boolean Expressions

Any logic expression may be expressed in two standard forms:

- The Sum-of-Products(SOP) Form   Ex) AB+ABC, ABC+CDE+B'CD'
- The Product-of-Sums(POS) Form Ex) (A+B)(A+B+C), (A+B+C)(C+D+E)(B'+C+D')

❏Principle of Duality : SOP ⇔ POS (change AND ⇔ OR, 0 ⇔1)

- Domain of a Boolean Expression : The set of variables contained in the expression
  Ex) A'B+AB'C : the domain is {A, B, C}

✓**Standard SOP Form (Canonical SOP Form)**

 – For all the missing variables, apply (x+x')=1 to the AND terms of the expression

 – List all the min-terms in forms of the complete set of variables in ascending order

Ex : Convert the following expression into standard SOP form: AB'C+A'B'+ABC'D

Sol) domain={A,B,C,D}, AB'C(D'+D)+A'B'(C'+C)(D'+D)+ABC'D
=AB'CD'+AB'CD+A'B'C'D'+A'B'C'D+A'B'CD'+A'B'CD+ABC'D
=1010+1011+0000+0001+0010+0011+1101 =0+1+2+3+10+11+13 =
Σ(0,1,2,3,10,11,13)

Standard POS Form (Canonical POS Form)

- **For all the missing variables, apply $(x'x)=0$ to the OR terms of the expression**
- **List all the max-terms in forms of the complete set of variables in ascending order**

Ex : Convert the following expression into standard POS form:
(A+B'+C)(B'+C+D')(A+B'+C'+D)

Sol) domain={A,B,C,D},   (A+B'+C)(B'+C+D')(A+B'+C'+D)

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

=(A+B'+C+D')(A+B'+C+D)(A'+B'+C+D')(A+B'+C+D')(A+B'+C'+D)=(0100))(0101)(0110)(1101)= Π(4,5,6,13)

A+BC = (A+B) (A+C)
Distributed Law

Converting Standard SOP to Standard POS

Step 1. Evaluate each product term in the SOP expression. Determine the binary numbers that represent the product terms

Step 2. Determine all of the binary numbers not included in the evaluation in Step 1

Step 3. Write in equivalent sum term for each binary number Step 2 and expression in POS form

Ex : Convert the following SOP to POS

Sol) SOP= A'B'C'+A'BC'+A'BC+AB'C+ABC=0+2+3+5+7 =$\Sigma(0,2,3,5,7)$

POS=(1)(4)(6) = $\Pi(1, 4, 6)$ (=(A+B+C')(A'+B+C)(A'+B'+C))

❑   **SOP and POS Observations**

• Canonical Forms (Sum-of-minterms, Product-of-Maxterms), or other standard forms (SOP, POS) differ in complexity

• Boolean algebra can be used to manipulate equations into simpler forms

• Simpler equations lead to simpler implementations

Definitions

- *Literal:* A variable or its complement

- *Product term:* literals connected by •

- *Sum term:* literals connected by +

- *Minterm:* a product term in which all the variables appear exactly once, either complemented or uncomplemented

- *Maxterm:* a sum term in which all the variables appear exactly once, either complemented or uncomplemented

Minterms

- Represents exactly one combination in the truth table.

- Denoted by m_j, where j is the decimal equivalent of the minterm's corresponding binary combination (b_j).

- A variable in m_j is complemented if its value in b_j is 0, otherwise is uncomplemented.

- Example: Assume 3 variables (A,B,C), and j=3. Then, b_j = 011 and its corresponding minterm is denoted by m_j = A'BC

Maxterm

- Represents exactly one combination in the truth table.

- Denoted by M_j, where j is the decimal equivalent of the maxterm's corresponding binary combination (b_j).

- A variable in M_j is complemented if its value in b_j is 1, otherwise is uncomplemented.

- Example: Assume 3 variables (A,B,C), and j=3. Then, b_j = 011 and its corresponding maxterm is denoted by M_j = A+B'+C'

Truth Table notation for Minterms and Maxterms

- Minterms and Maxterms are easy to denote using a truth table.

- Example:
  Assume 3 variables x,y,z
  (order is fixed)

x	y	z		Minterm	Maxterm
0	0	0		$x'y'z' = m_0$	$x+y+z = M_0$
0	0	1		$x'y'z = m_1$	$x+y+z' = M_1$
0	1	0		$x'yz' = m_2$	$x+y'+z = M_2$
0	1	1		$x'yz = m_3$	$x+y'+z' = M_3$
1	0	0		$xy'z' = m_4$	$x'+y+z = M_4$
1	0	1		$xy'z = m_5$	$x'+y+z' = M_5$
1	1	0		$xyz' = m_6$	$x'+y'+z = M_6$
1	1	1		$xyz = m_7$	$x'+y'+z' = M_7$

Canonical Forms (Unique)

- **Each term of Boolean expression contain all input variables in either true form or complemented form**

- Any Boolean function F( ) can be expressed as a *unique* **sum** of **min**terms and a unique **product** of **max**terms (under a fixed variable ordering).

- In other words, every function F() has two canonical forms:
  - ✓ Canonical Sum-Of-Products (sum of minterms)
  - ✓ Canonical Product-Of-Sums (product of maxterms)

- **Canonical Sum-Of-Products:**
  The minterms included are those m_j such that F( ) = 1 in row *j* of the truth table for F( ).

- Canonical Product-Of-Sums:
  The maxterms included are those Mj such that F( ) = 0 in row j of the truth table for F( ).

Example

- Truth table for $F_1(A,B,C)$ at right

- The canonical sum-of-products form for F_1 is

$$F_1(A,B,C) = m_1 + m_2 + m_4 + m_6$$
$$= A'B'C + A'BC' + AB'C' + ABC'$$

- The canonical product-of-sums form for F_1 is

$$F_1(A,B,C) = M_0 \bullet M_3 \bullet M_5 \bullet M_7$$
$$= (A+B+C)\bullet(A+B'+C')\bullet$$
$$(A'+B+C')\bullet(A'+B'+C').$$

- **Observe that: $m_j = M_j'$**

A	B	C		F_1
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		1
1	1	1		0

Shorthand: ∑ and ∏

- $F_1(A,B,C) = \sum m(1,2,4,6)$, where $\sum$ indicates that this is a sum-of-products form, and $m(1,2,4,6)$ indicates that the minterms to be included are m_1, m_2, m_4, and m_6.

- $F_1(A,B,C) = \prod M(0,3,5,7)$, where $\prod$ indicates that this is a product-of-sums form, and $M(0,3,5,7)$ indicates that the maxterms to be included are M_0, M_3, M_5, and M_7.

- Since $m_j = M_j{}'$ for any *j*,
  $\sum m(1,2,4,6) = \prod M(0,3,5,7) = F_1(A,B,C)$

Conversion Between Canonical Forms

- Replace ∑ with ∏ (or *vice versa*) and replace those *j*'s that appeared in the original form with those that do not.

- Example:

$$f_1(a,b,c) = a'b'c + a'bc' + ab'c' + abc'$$
$$= m_1 + m_2 + m_4 + m_6$$
$$= \sum(1,2,4,6)$$
$$= \prod(0,3,5,7)$$
$$= (a+b+c) \cdot (a+b'+c') \cdot (a'+b+c') \cdot (a'+b'+c')$$

Standard Forms (NOT Unique)

- **If there is exists at least one term that does not contain all variables.**

- Standard forms are "*like*" canonical forms, except that not all variables need appear in the individual product (SOP) or sum (POS) terms.

- Example

  $f_1(a,b,c) = a'b'c + bc' + ac'$

  is a *standard* sum-of-products form

  $f_1(a,b,c) = (a+b+c) \cdot (b'+c') \cdot (a'+c')$

  is a *standard* product-of-sums form.

Conversion of SOP from standard to canonical form

- Expand *non-canonical* terms by inserting equivalent of 1 in each missing variable x: (x + x') = 1

- Remove duplicate minterms

- f_1(a,b,c) = a'b'c + bc' + ac'

$$= a'b'c + (a+a')bc' + a(b+b')c'$$
$$= a'b'c + abc' + a'bc' + abc' + ab'c'$$
$$= a'b'c + abc' + a'bc + ab'c'$$

Conversion of POS from standard to canonical form

- Expand noncanonical terms by adding 0 in terms of missing variables (*e.g.*, xx' = 0) and using the distributive law

- Remove duplicate maxterms

- f_1(a,b,c) = (a+b+c)•(b'+c')•(a'+c')

  = (a+b+c)•(aa'+b'+c')•(a'+bb'+c')

  = (a+b+c)•(a+b'+c')•(a'+b'+c')•

  (a'+b+c')•(a'+b'+c')

  = (a+b+c)•(a+b'+c')•(a'+b'+c')•(a'+b+c')

Karnaugh Maps

- Karnaugh maps (K-maps) are *graphical* representations of boolean functions.

- One *map cell* corresponds to a row in the truth table.

- Also, one map cell corresponds to a minterm or a maxterm in the boolean expression

- Multiple-cell areas of the map correspond to standard terms.

Two-Variable Map

x_1 \ x_2	0	1
0	0 m_0	1 m_1
1	2 m_2	3 m_3

OR

x_2 \ x_1	0	1
0	0 m_0	2 m_2
1	1 m_1	3 m_3

NOTE: ordering of variables is IMPORTANT for $f(x_1,x_2)$, x_1 is the row, x_2 is the column.

Cell **0** represents $x_1'x_2'$; Cell **1** represents $x_1'x_2$; etc. If a minterm is present in the function, then a 1 is placed in the corresponding cell.

Two-Variable Map

- Any two adjacent cells in the map differ by ONLY one variable, which appears complemented in one cell and uncomplemented in the other.

Exp: m_0 (=$x_1'x_2'$) is adjacent to m_1 (=$x_1'x_2$) and m_2 (=x_1x_2') but NOT m_3 (=x_1x_2)

2-Variable Map -- Example

- $f(x_1,x_2) = x_1'x_2' + x_1'x_2 + x_1x_2'$
  $= m_0 + m_1 + m_2$
  $= x_1' + x_2'$

- 1s placed in K-map for specified minterms m_0, m_1, m_2

- Grouping (ORing) of 1s allows simplification

- What (simpler) function is represented by each dashed rectangle?

  - $x_1' = m_0 + m_1$
  - $x_2' = m_0 + m_2$

- Note m_0 covered twice
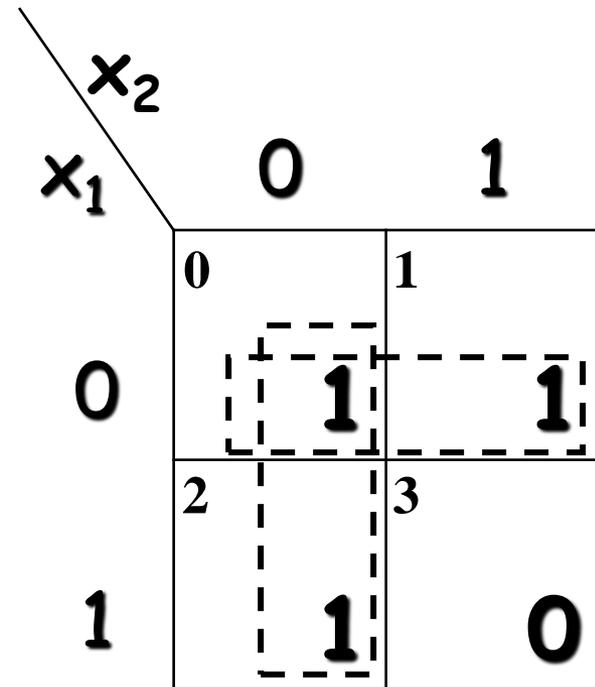
Minimization as SOP using K-map

- Enter 1s in the K-map for each product term in the function

- Group *adjacent* K-map cells containing 1s to obtain a product with fewer variables. Group size must be in power of 2 (2, 4, 8, ...)

- Handle "boundary wrap" for K-maps of 3 or more variables.

- Realize that answer may not be unique

Three-Variable Map
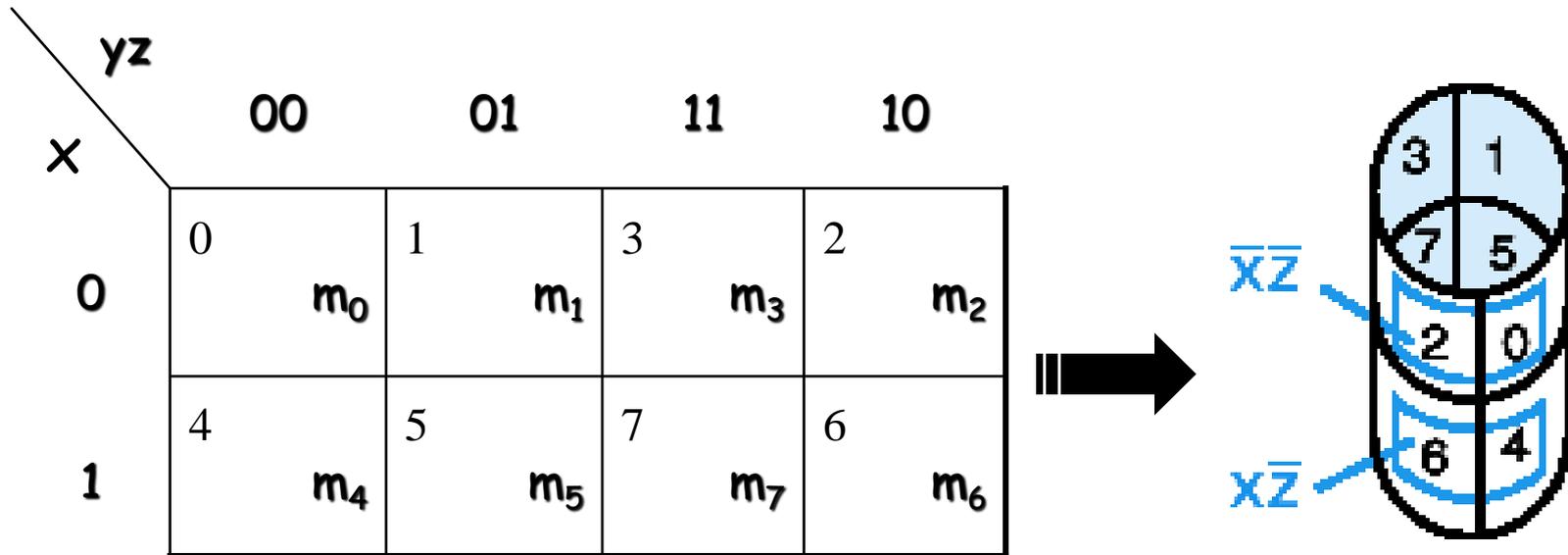
-Note: variable ordering is (x,y,z); yz specifies column, x specifies row.
-Each cell is adjacent to **_three_** other cells (left or right or top or bottom or edge wrap)

Three-Variable Map (cont.)

minterm

**The types of structures that are either minterms or are generated by repeated application of the minimization theorem on a three variable map are shown at right.
Groups of 1, 2, 4, 8 are possible.**

group of 2 terms

group of 4 terms
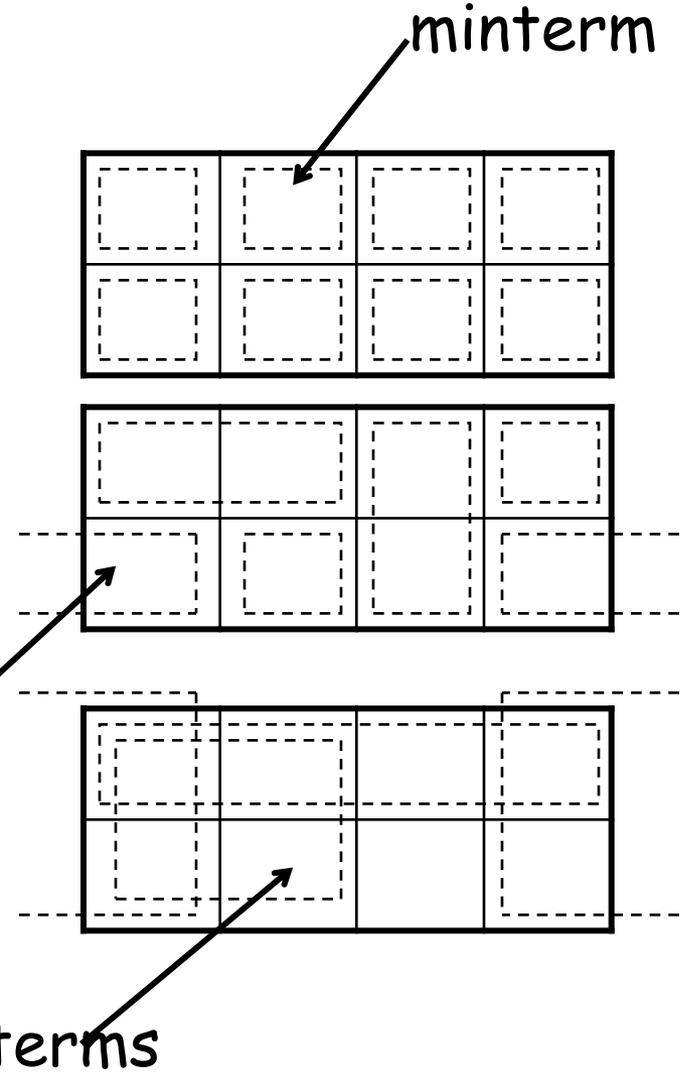
Simplification

- Enter minterms of the Boolean function into the map, then group terms

- Example: f(a,b,c) = a'c + abc + bc'

- Result: f(a,b,c) = a'c+ b

abc

a

More Examples

yz
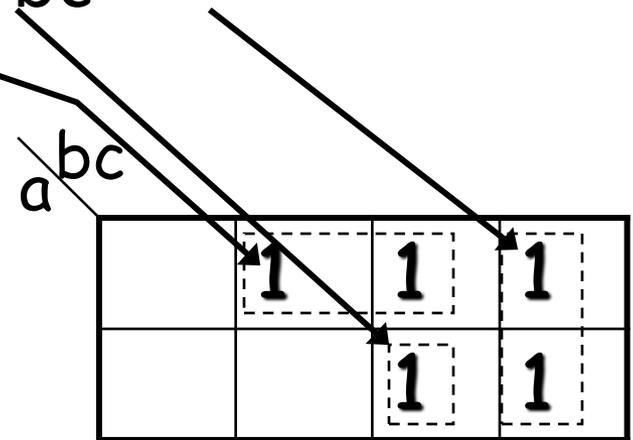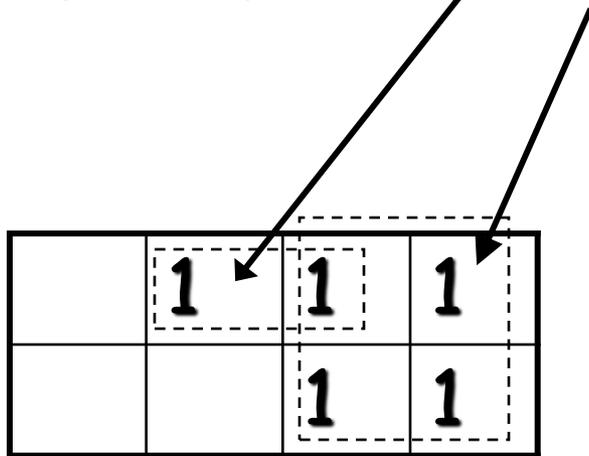
x \ yz	00	01	11	10
0			1	1
1		1	1	

- $f_1(x, y, z) = \sum m(2,3,5,7)$

  ■ $f_1(x, y, z) = x'y + xz$

- $f_2(x, y, z) = \sum m(0,1,2,3,6)$

  ■ $f_2(x, y, z) = x'+yz'$

1	1	1	1
			1

Four-Variable Maps

YZ

WX	00	01	11	10
00	m_0	m_1	m_3	m_2
01	m_4	m_5	m_7	m_6
11	m_{12}	m_{13}	m_{15}	m_{14}
10	m_8	m_9	m_{11}	m_{10}
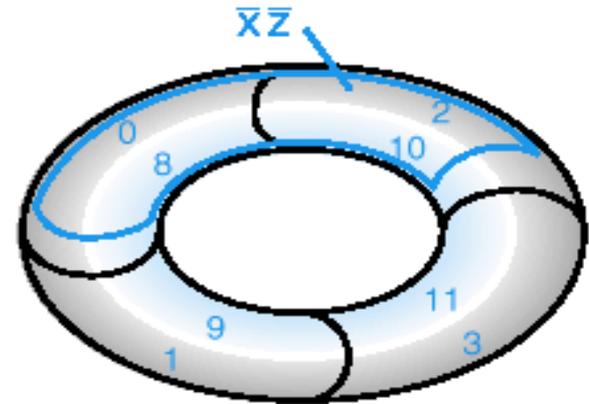
- Top cells are adjacent to bottom cells. Left-edge cells are adjacent to right-edge cells.

- Note variable ordering (WXYZ).

Four-variable Map Simplification

- One square represents a minterm of 4 literals.

- A rectangle of 2 adjacent squares represents a product term of 3 literals.

- A rectangle of 4 squares represents a product term of 2 literals.

- A rectangle of 8 squares represents a product term of 1 literal.

- A rectangle of 16 squares produces a function that is equal to logic 1.

Example

- Simplify the following Boolean function (A,B,C,D) = ∑m(0,1,2,4,5,7,8,9,10,12,13).

- First put the function g( ) into the map, and then group as many 1s as possible.

cd

ab

1	1		1
1	1	1	
1	1		
1	1		1

1	1		1
1	1	1	
1	1		
1	1		1

g(A,B,C,D) = c'+b'd'+a'bd

Don't Care Conditions

- There may be a combination of input values which
  - will **never** occur
  - if they do occur, the output is of no concern.

- The function value for such combinations is called a *don't care*.

- They are denoted with **x** or **–**. Each **x** may be arbitrarily assigned the value 0 or 1 in an implementation.

- Don't cares can be used to *further* simplify a function

Minimization using Don't Cares

- Treat don't cares as if they are 1s to generate PIs.

- Delete PI's that cover only don't care minterms.

- Treat the covering of remaining don't care minterms as optional in the selection process (*i.e.* they may be, but need not be, covered).

Example

cd

ab 00 01 11 10

	00	01	11	10
00	0	1	0	1
01	1	1	0	1
11	0	0	×	×
10	1	1	×	×

- Simplify the function f(a,b,c,d) whose K-map is shown at the right.

- f = a'c'd+ab'+cd'+a'bc'

  or

- f = a'c'd+ab'+cd'+a'bd'

0	1	0	1
1	1	0	1
0	0	×	×
1	1	×	×

0	1	0	1
1	1	0	1
0	0	×	×
1	1	×	×

Another Example

ab\cd

×	1	0	0
1	×	0	×
1	×	×	1
0	×	×	0

- Simplify the function g(a,b,c,d) whose K-map is shown at right.

- g = a'c'+ ab
  or

- g = a'c'+b'd

×	1	0	0
1	×	0	×
1	×	×	1
0	×	×	0

×	1	0	0
1	×	0	×
1	×	×	1
0	×	×	0

Karnaugh Map

- ## Simplification methods
  - ### Boolean algebra(algebraic method)
  - ### Karnaugh map(map method))
  - ### Quine-McCluskey(tabular method)
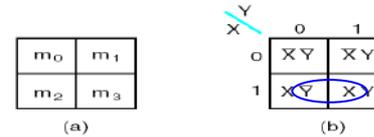
Two-Variable Map

$$XY+XY'=X(Y+Y')=X$$

- Three- and Four-input Kanaugh maps

Gray code
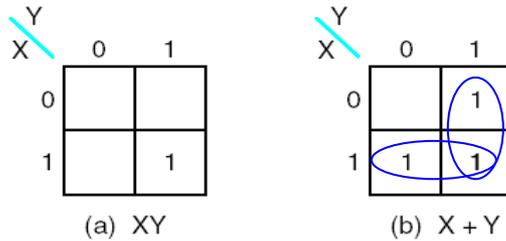
Representation of Functions in the Map

(a) XY   (b) X + Y

(a)

(b)
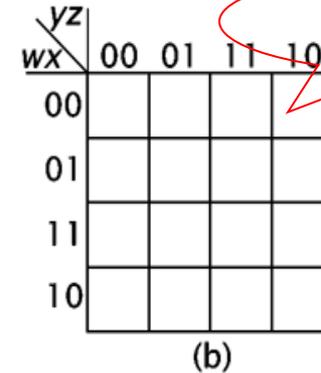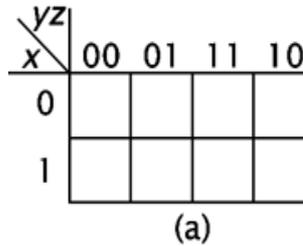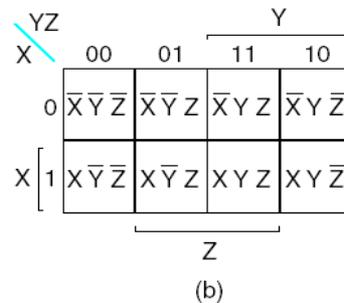
Three-Variable Map

(a)   (b)
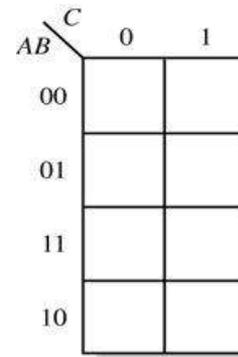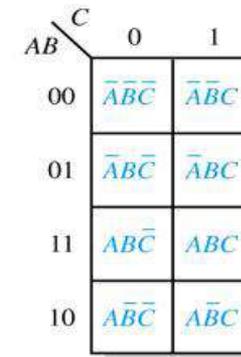
(a)   (b)

Karnaugh Map (K- Map) Steps

1. Sketch a Karnaugh map grid for the given problem.in power of 2^N Squares
2. Fill in the 1's and 0's from the truth table of sop or pos Boolean function
3. Circle groups of 1's.

   ◆ Circle the largest groups of 2, 4, 8, etc. first.
   ◆ Minimize the number of circles but make sure that every 1 is in a circle.

4. Write an equation using these circles.

Example) $F(X,Y,Z)=\Sigma m(2,3,4,5) = X'Y+XY'$
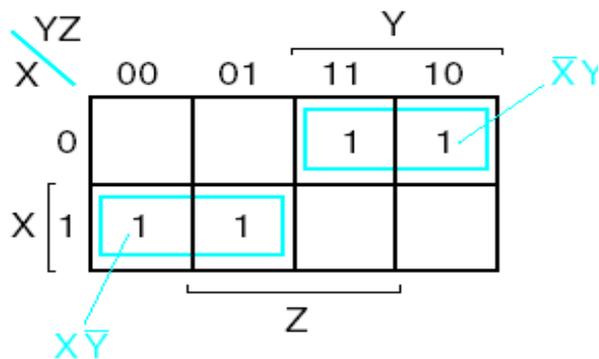
Example) $F(X,Y,Z)=\Sigma m(0,2,4,6) = X'Z'+XZ' = Z'(X'+X)=Z'$
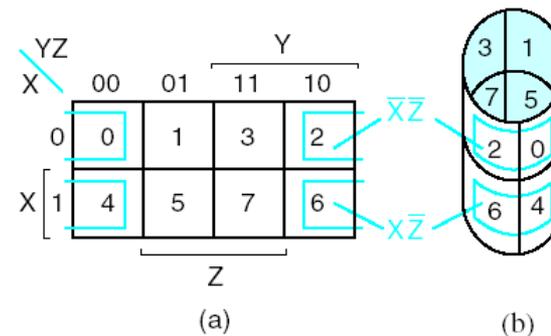
Three-Variable Map: Flat and on a Cylinder to Show Adjacent Squares

Four-Variable K-Map : 16 minterms : m_0 ~ m_{15}

Rectangle group

- 2-squares(minterms) : 3-literals product term
- 4-squares : 2-literals product term
- 8-squares : 1-literals product term
- 16-squares : logic 1

(a)   (b)

Fig. 2-17   Four-Variable Map

F(W, X,Y,Z)=Σm(0,2,7,8,9,10,11) = WX' + X'Z' + **W'XYZ**

(a)

(b)

Ex 4-28) Minimize the following expression

AB'C+A'BC+A'B'C+A'B'C'+AB'C'

Sol) B'+A'C
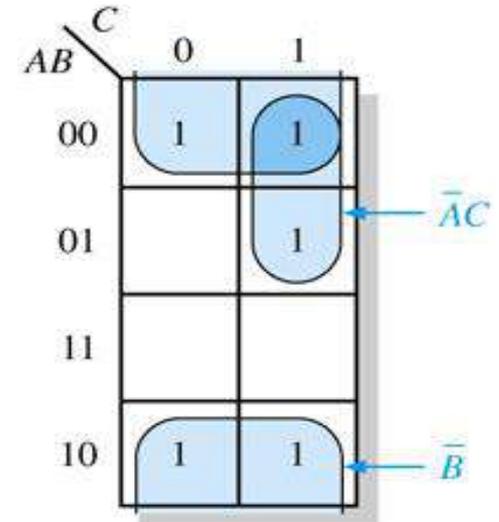
Ex. Minimize the following expression

B'C'D'+A'BC'D'+ABC'D'+A'B'CD+AB'CD+A'B'C'

Sol) D'+B'C

❑ Don't Care Conditions

• it really does not matter since they

will never occur(its output is either '0' or '1')

• The don't care terms can be

used to advantage on the

Karnaugh map

Inputs	Output
A B C D	Y
0 0 0 0	0
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	1
1 0 0 0	1
1 0 0 1	1
1 0 1 0	X
1 0 1 1	X
1 1 0 0	X
1 1 0 1	X
1 1 1 0	X
1 1 1 1	X

(a) Truth table
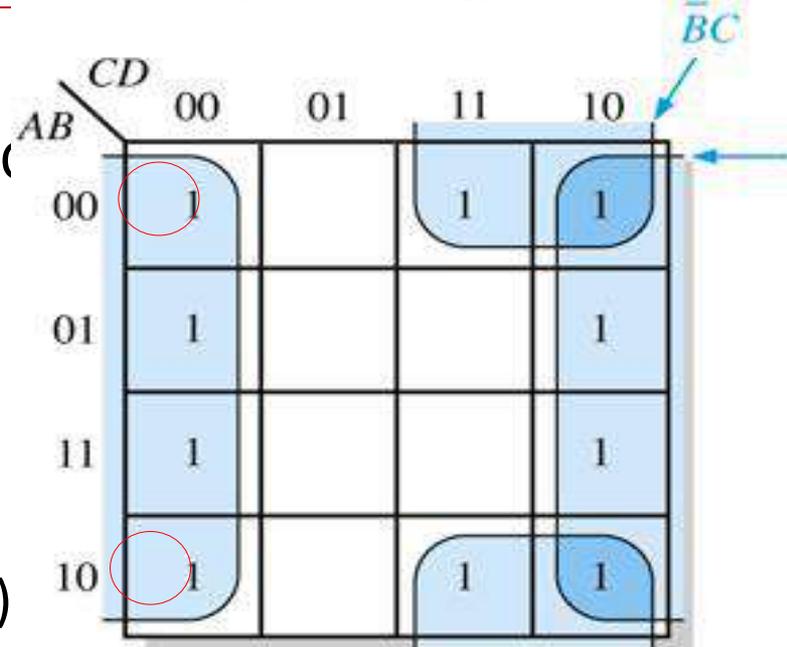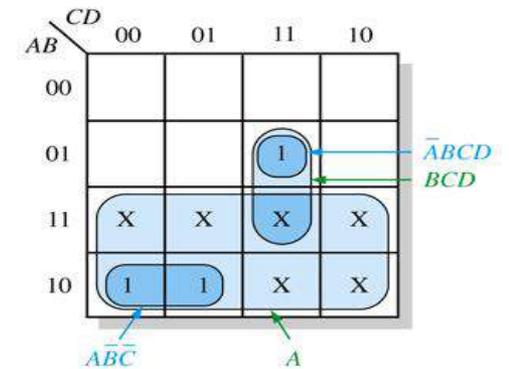
(b) Without "don't cares" $Y = A B\bar{C} + \bar{A}BCD$
With "don't cares" $Y = A + BCD$

Ex K- Map for POS (B+C+D)(A+B+C'+D)(A'+B+C+D')(A+B'+C+D)(A'+B'+C+D)

Sol) (B+C+D)=(A'A+B+C+D)=(A'+B+C+D)(A+B+C+D)

(1+0+0+0)(0+0+0+0)(0+0+1+0)

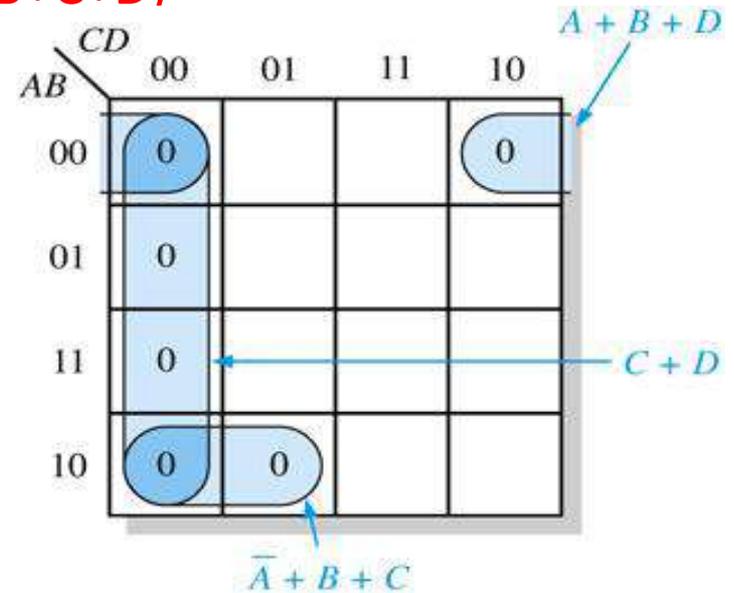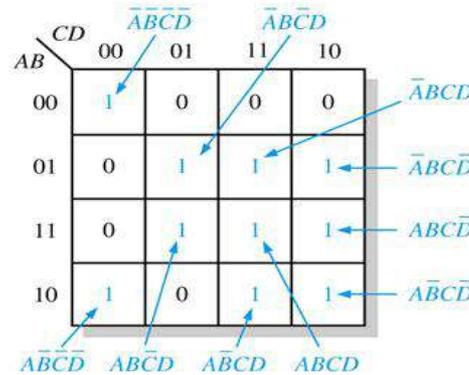(1+0+0+1)(0+1+0+0)(1+1+0+0)

  F=(C+D)(A'+B+C)(A+B+D)

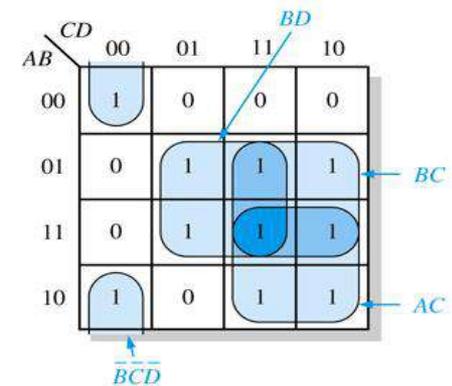❑ Converting Between POS and SOP Using the K−map

Ex 4-33) (A'+B'+C+D)(A+B'+C+D)

(A+B+C+D')(A+B+C'+D') (A'+B+C+D')

(
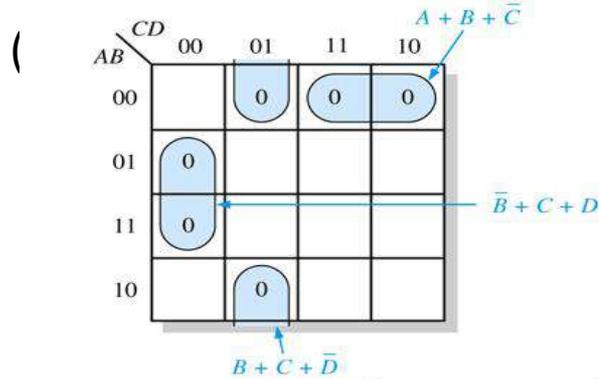
(a) Minimum POS: $(A + B + \bar{C})(\bar{B} + C + D)(B + C + \bar{D})$

(b) Standard SOP:
$\bar{A}\bar{B}\bar{C}D + \bar{A}B\bar{C}D + \bar{A}BCD + \bar{A}BC\bar{D} + ABC\bar{D} + A\bar{B}C\bar{D} + A\bar{B}\bar{C}\bar{D} + AB\bar{C}D + A\bar{B}CD + ABCD$

(c) Minimum SOP: $AC + BC + BD + \bar{B}\bar{C}\bar{D}$

Five Variable K-Map : {A,B,C,D,E}

Six Variable K-Map : {A,B,C,D,E,F}