



جامعة الفرات الأوسط التقنية

المعهد التقني / النجف

قسم تقنيات الأجهزة الطبية

Assist. Lect Sarah Alaa Tamkeen

sara.ala.inj@atu.edu.iq

2025-2026

First stage

30 weeks

120 Hours

المفردات النظرية	الاسبوع
A general idea of numerical systems (types and details)	1
Transfers between the numerical systems	2
Logic gates (types, working principle, truth tables, logical symbol)	3
How to connect the logic gates to form logic circuits.	4
Boolean algebra and the rule of de-Morgan	5
Simplification of logical equations using Boolean algebra and the laws of De Morgan's laws.	6
The design of the logical gates using NOR and NANDcircuits,	7
Ways of writing the equation from truth table (POS, SOP).	8
Karnaugh Map (for two variables, the three variables, the four variables)	9
Simplification of logical equations using Karnaugh Map	10
Calculations in the binary system (addition, subtraction, subtraction using complements).	11
Logic circuit applications(half adder, full adder, parallel adder circuits)	12
Binarysubtractorcircuits (half subtractor,full subtractorparallel subtractor) circuit using the adder circuit by method of 1s complements.	13
The circuit of digital comparator (one stage and two stages)	14

The circuit of decoder size of 2:4 ,3:8 and 4:10	15
The circuit of encoder size of 4:2, 8:3 and 10:4	16
Introduction to sequential logic circuits, a general idea of the Flip Flop, flip flop type (S-R).	17
The flip flop type J-K and master slave flip flop	18
The D- flip flop and T flip flop	19
The registers, design of registers, enter the information and output from registers	20
The shift register, shift to left, shift to right	21
The counter- asynchronous counter	22
The synchronous counter- the cycle counter	23
The multiplexer and its applications	24
The code convertor – the application of code convertor	25
Programmable logic array: Concepts of programmable logic array(PLA); Concepts of programmable array logic(PAL)	26
Buffers, Non inverting buffers, inverting buffers, Tri-state buffers, transmission gates	27
Introduction to Sequential logic latches and flip flops, Latches- Edgetriggered flip flop, Flip-flop operating characteristics, Flip-flop applications	28
Introduction To State Machine Design,	29
State diagram and State table	30

Digital Fundamentals

1.1 Number system

Number systems are the technique to represent numbers in the computer system architecture, every value that you are saving or getting into/from computer memory has a defined number system. Computer architecture supports following number systems.

- 1- **Binary number system**
- 2- **Octal number system**
- 3- **Decimal number system**
- 4- **Hexadecimal (hex) number system**

1.1.1 Binary number system

A Binary number system has only two digits that are **0 and 1**. Every number (value) represents with 0 and 1 in this number system. The base of binary number system is 2, because it has only two digits.

1.1.2 Octal number system

Octal number system has only eight (8) digits from **0 to 7**. Every number (value) represents with 0,1,2,3,4,5,6 and 7 in this number system. The base of octal number system is 8, because it has only 8 digits.

1.1.3 Decimal number system

Decimal number system has only ten (10) digits from **0 to 9**. Every number (value) represents with 0,1,2,3,4,5,6, 7,8 and 9 in this number system. The base of decimal number system is 10, because it has only 10 digits.

1.1.4 Hexadecimal (hex) number system

A Hexadecimal number system has sixteen (16) alphanumeric values from **0 to 9 and A to F**. Every number (value) represents with 0,1,2,3,4,5,6, 7,8,9,A,B,C,D,E and F in this number system.

The base of hexadecimal number system is 16, because it has 16 alphanumeric values. Here **A** is 10, **B** is 11, **C** is 12, **D** is 14, **E** is 15 and **F** is 16.

1.2 Conversions

1.2.1 Decimal to Binary

Example: Convert the following decimal numbers $(13)_{10}$ to binary

Division by 2	Quotient	Remainder
$13 \div 2$	6	1 (LSB)
$6 \div 2$	3	0
$3 \div 2$	1	1
$1 \div 2$	0	1 (MSB)



Example: Convert decimal Number $(29)_{10}$ to binary

Decimal Number $(29)_{10}$ = Binary Number $(11101)_2$.

Operation	Integer	Remainder (Result)	
$29 / 2$	14	1	(LSB) (MSB)
$14 / 2$	7	0	
$7 / 2$	3	1	
$3 / 2$	1	1	
$1 / 2$	0	1	

Note

LSB/ Stands for **L**east **S**ignificant **B**it

MSB/ Stands for **M**ost **S**ignificant **B**it

Example: Convert $(0.6875)_{10}$ to Binary

Operation	Integer	Fraction	Result (Integer)	
0.6875×2	1	0.3750	1	
0.3750×2	0	0.7500	0	
0.7500×2	1	0.5000	1	
0.5000×2	1	0.0000	1	

Therefore, the answer is $(0.6875)_{10} = (0.1011)_2$.

1.2.2 Decimal to Octal

Example: Convert $(153.513)_{10}$ to Octal

	Integer	Remainder			Integer	
$153 / 8 =$	19	1			$0.513 \times 8 = 4.104$	4
$19 / 8 =$	2	3			$0.104 \times 8 = 0.832$	0
$2 / 8 =$	0	2			$0.832 \times 8 = 6.656$	6
			$= (231)_8$		$= (406)_8$	

The answer is $(153.513)_{10} = (231.406)_8$

1.2.3 Decimal to Hexadecimal

Example: Decimal Number is : $(725)_{10}$

16	725	5	5	LSB
16	45	13	D	
	2	2	2	MSB

1.2.4 Binary to decimal

Multiply the digit with 2(with place value exponent). Eventually add all the multiplication becomes the Decimal number. $(11011001)_2 \longrightarrow (?)_{10}$

$$1 \cdot 2^7 + 1 \cdot 2^6 + 0 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 1 + 8 + 16 + 64 + 128 = 217$$

1.2.4 Binary to Octal

An easy way to convert from binary to octal is to group binary digits into sets of three, starting with the least significant (rightmost) digits.

Binary = $(011\ 100\ 101)_2$

Octal = $(3\ 4\ 5)_8$

Example: Convert $(101101.11)_2$ into Octal?

$101101 = 101\ 101\ .\ 110$
 ↑ ↑ ↑
 5 5 6

Binary	Octal
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

1.2.5 Binary to Hexadecimal

An equally easy way to convert from binary to hexadecimal is to group binary **digits into sets of four**, starting with the least significant (rightmost) digits.

Binary: $(11100101)_2 = (E5)_{16}$

Binary	Hexadecimal
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

1.2.6 Octal to decimal

The conversion can also be performed in the conventional mathematical way, by showing each digit place as an increasing power of 8.

$$(345)_8 = (3 * 8^2) + (4 * 8^1) + (5 * 8^0) = (3 * 64) + (4 * 8) + (5 * 1) = (229)_{10}$$

1.2.7 Hexadecimal to Binary

Converting from hexadecimal to binary is as easy as converting from binary to hexadecimal. Simply look up each hexadecimal digit to obtain the equivalent group of four binary digits.

$$(0.A38)_{16} = (0.101000111000)_2$$

Hexadecimal	Binary
A	1010
3	0011
8	1000

$$(E9A)_{16} = (111010011010)_2$$

Hexadecimal	Binary
E	1110
9	1001
A	1010

1.3 Binary Arithmetic(Operations)

It is a key for binary subtraction, multiplication, division. There are four rules of binary addition.

Addition operation

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 0 = 1$$

$$1 + 1 = 0; \text{ and carry } 1 \text{ to the next column.}$$

Subtraction operation

$$0 - 0 = 0$$

$$0 - 1 = 1; \text{ and borrow } 1 \text{ from the next column.}$$

$$1 - 0 = 1$$

$$1 - 1 = 0$$

Multiplication operation

$$0 \times 0 = 0$$

$$0 \times 1 = 0$$

$$1 \times 0 = 0$$

$$1 \times 1 = 1$$

1.3.1 Binary Addition

Example: $(0011010)_2 + (0001100)_2 = (00100110)_2$

$$\begin{array}{r} 11 \\ 0011010 = 26_{10} \\ 0001100 = 12_{10} \\ \hline 0100110 = 38_{10} \end{array}$$

1.3.2 Binary Multiplication

Example: Perform the following binary multiplications:

(a) $11_2 \times 11_2$

(b) $101_2 \times 111_2$

(a)

$$\begin{array}{r} 11 3 \\ \times 11 \times 3 \\ \hline 11 9 \\ + 11 \\ \hline 1001 \end{array}$$

Partial products {

(b)

$$\begin{array}{r} 111 7 \\ \times 101 \times 5 \\ \hline 111 35 \\ 000 \\ + 111 \\ \hline 100011 \end{array}$$

Partial products {

1.3.3 Binary Division

The numbers in a division are the *dividend*, the *divisor*, and the *quotient*. These are illustrated in the following standard. To illustrate, consider the following division examples:

Example:

$$\begin{array}{r} 110 \\ 100 \overline{) 11000} \\ \underline{100} \\ 0100 \\ \underline{100} \\ 0000 \end{array}$$

$$\frac{\text{dividend}}{\text{divisor}} = \text{quotient}$$

Example:

$$101010 / 000110 = 000111$$

$$\begin{array}{r} 111 = 7_{10} \\ 000110 \overline{) 101010} = 42_{10} \\ \underline{-110} = 6_{10} \\ 1001 \\ \underline{-110} \\ 110 \\ \underline{-110} \\ 0 \end{array}$$

Example: Perform the following binary divisions: (a) $(110)_2 \div (11)_2$ (b) $(110)_2 \div (10)_2$

$$\begin{array}{r} \text{(a)} \quad \begin{array}{r} 10 \\ 11 \overline{) 110} \\ \underline{11} \\ 000 \end{array} \quad \begin{array}{r} 2 \\ 3 \overline{) 6} \\ \underline{6} \\ 0 \end{array} \end{array}$$

$$\text{(b)} \quad \begin{array}{r} 11 \\ 10 \overline{) 110} \\ \underline{10} \\ 10 \\ \underline{10} \\ 00 \end{array} \quad \begin{array}{r} 3 \\ 2 \overline{) 6} \\ \underline{6} \\ 0 \end{array}$$

Various examples

Example1: Convert the following numbers to Decimal number?

1- $(1010.01)_2$

2- $(1A2)_{16}$

1- The weight of each digit : $2^3 \ 2^2 \ 2^1 \ 2^0 \ 2^{-1} \ 2^{-2}$

Binary number : 1 0 1 0 . 0 1

$$(1010.01)_2 = 8 + 0 + 2 + 0 + 0 + 0.25$$

$$= (10.25)_{10}$$

2- $(1A2)_{16} = 1 \cdot 16^2 + 10 \cdot 16^1 + 2 \cdot 16^0$

$$= 1 \cdot 256 + 10 \cdot 16 + 2 = 256 + 160 + 2 = (418)_{10}$$

Example2: Convert $(101101.01)_2$ to Decimal number?

Weight in power of 2: $2^5 \ 2^4 \ 2^3 \ 2^2 \ 2^1 \ 2^0 \ 2^{-1} \ 2^{-2}$

$$\begin{array}{cccccccc} \text{Weight} & 32 & 16 & 8 & 4 & 2 & 1 & 0.5 & 0.25 \\ \text{Binary} & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \end{array}$$

Binary number : 1 0 1 1 0 1 . 0 1

$$(101101.01)_2 = 32 + 8 + 4 + 1 + 0.25 = (45.25)_{10}$$

Example3: Convert $(101101.11)_2$ into Octal?

$$\begin{array}{ccc} 101101 = 101 & 101 & . & 110 \\ \uparrow & \uparrow & & \uparrow \\ 5 & 5 & & 6 \end{array}$$

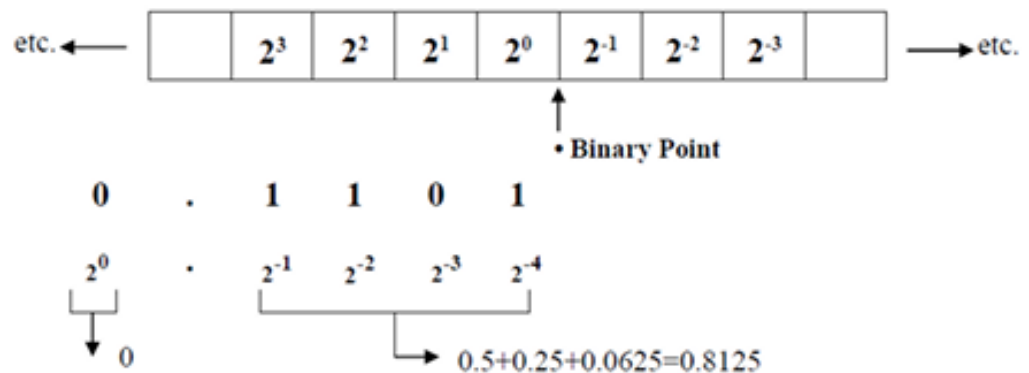
Thus $(101101)_2 = (55.6)_8$

Example8: Convert $(5B1.7)_{16}$ to Octal number?

$$(5B1.7)_{16} = (010\ 110\ 110\ 001.011\ 1)_2 = (2661.34)_8$$

$\uparrow \quad \uparrow \quad \uparrow \quad \uparrow \quad \uparrow \quad \uparrow$

Example9: Find the Decimal equivalent of the 0.1101_2 ?



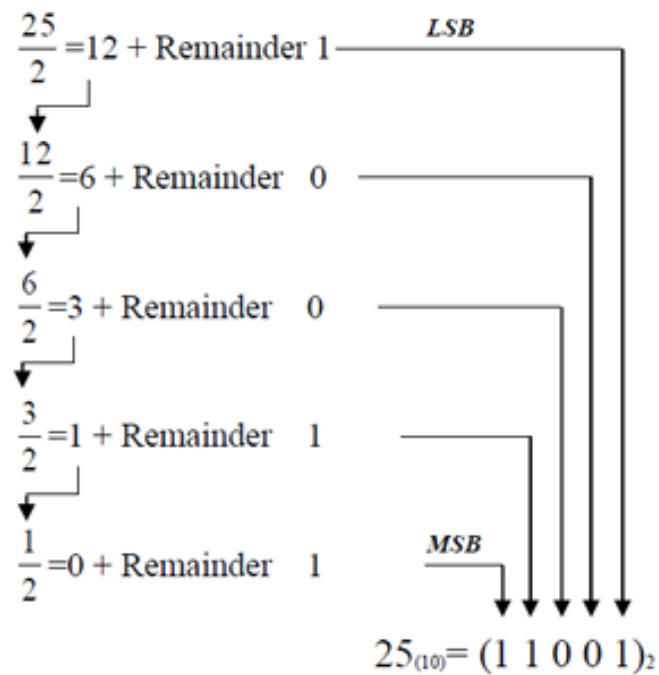
Hence: $0.1101_{(2)} = 0.8125_{(10)}$

Example10: Convert Binary $(110.001)_2$ to a Decimal number

$$\begin{array}{ccccccc}
 1 & 1 & 0 & . & 0 & 0 & 1 \\
 2^2 & 2^1 & 2^0 & . & 2^{-1} & 2^{-2} & 2^{-3}
 \end{array}
 \longrightarrow 6.125$$

$4 + 2 + 0.125 = (6.125)_{10}$

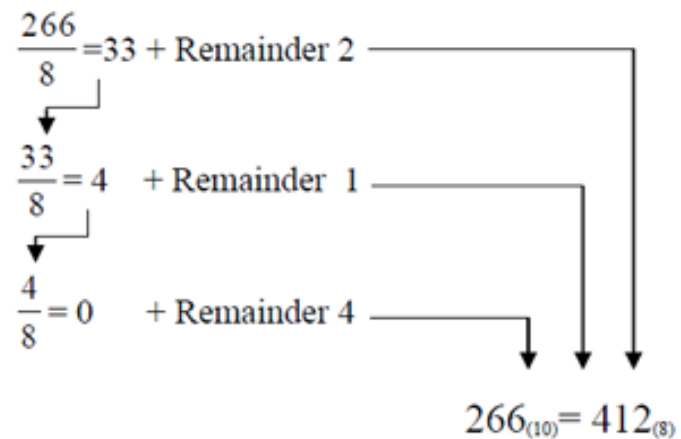
Example11: Convert Decimal to Binary number



Example12: Convert $24.6_{(8)}$ to its Decimal equivalent.

$$(24.6)_8 = 2 \times 8^1 + 4 \times 8^0 + 6 \times 8^{-1} = 2 \times 8 + 4 \times 1 + 6 \times 0.125 = (20.75)_{10}$$

Example13: Convert Decimal to Octal number



Example14: Convert the Decimal number [41] to Binary number

Integer	Remainder
$41 \div 2$	
$20 \div 2$	1
$10 \div 2$	0
$5 \div 2$	0
$2 \div 2$	1
1	0
	1

←

$(101001)_2 = \text{answer}$

So, $[41]_{10} = (101001)_2$

Example15: Convert the Decimal number [27.15] to Binary number

Integer	Remainder
$27 \div 2$	
$13 \div 2$	1
$6 \div 2$	1
$3 \div 2$	0
1	1
	1

Fraction	Coefficient
$0.15 \times 2 = 0.3$	0
$0.3 \times 2 = 0.6$	0
$0.6 \times 2 = 1.2$	1
$1.2 \times 2 = 0.4$	0
$0.4 \times 2 = 0.8$	0
$0.8 \times 2 = 1.6$	1
$1.6 \times 2 = 1.2$	1

SO $(27.15)_{10} = (11011.0010011)_2$

Example16: Convert the Decimal number [245.5] to Octal number system

Integer	Remainder
$245 \div 8$	
$30 \div 8$	5
3	6
	3

←
 $(365)_8 = \text{answer}$
 $[245]_{10} = (365)_8$

Fraction	Coefficient
$0.5 \times 8 = 4.0$	4
$4.0 \times 8 = 0.0$	0

→
 $(0.40)_8 = \text{answer}$

$[0.5]_{10} = (0.4)_8$

So, $[245.5]_{10} = (365.4)_8$

Example17: Convert the Decimal number [165.25] to Hexadecimal number system

Integer	Remainder
$165 \div 16$	
(10) A	5
	A

←
 $(A5)_{16} = \text{answer}$
 $[165]_{10} = (A5)_{16}$

Fraction	Coefficient
$0.25 \times 16 = 4.0$	4
$4.0 \times 16 = 0.0$	0

→
 $(0.40)_{16} = \text{answer}$

$[0.25]_{10} = (0.4)_{16}$

So, $[165.25]_{10} = (A5.4)_{16}$

Example18: Multiply; $(11.01 \times 1.01)_2$

$$\begin{array}{r}
 1101 \quad \times \\
 101 \\
 \hline
 1101 \quad + \\
 00000 \\
 110100 \\
 \hline
 (100.0001)_2
 \end{array}$$

Example19: Divide; $(1101.01 \div 10)_2$

$$\begin{array}{r}
 110.101 \\
 10 \overline{) 110101} \\
 \underline{10} - \\
 10 \\
 \underline{10} - \\
 010 \\
 \underline{10} - \\
 010 \\
 \underline{10} - \\
 00
 \end{array}$$

So, $1101.01 \div 10 = (110.101)_2$

Example20: Convert $(153.513)_{10}$ to Octal

	Integer	Remainder			Integer	
$153 / 8 =$	19	1	$\uparrow$ $= (231)_8$	$ $	$0.513 \times 8 = 4.104$	4
$19 / 8 =$	2	3			$0.104 \times 8 = 0.832$	0
$2 / 8 =$	0	2			$0.832 \times 8 = 6.656$	6
					$\downarrow$ $= (406)_8$	

The answer is $(153.513)_{10} = (231.406)_8$

Binary coded System

2.1 Introduction

In general, coding is the process of assigning a group of binary digits to represent multivalued items of information. Binary coded decimal (BCD) is a system of writing numerals that assigns a four-digit binary code to each digit 0 through 9 in a decimal (base- 10) numeral. The four-bit BCD code for any particular single base-10 digit is its representation in binary notation, as follows:

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

Numbers larger than 9, having two or more digits in the decimal system, are expressed digit by digit, For example, the BCD interpretation of the base-10 number 1895 is

1 8 9 5 ie., 1895 = 0001 1000 1001 0101
0001 1000 1001 0101

The binary equivalents of 1, 8, 9, and 5, always in a four-digit format, go from left to right. Binary codes are broadly classified into Numeric codes, Alphanumeric codes and Error detecting codes. Numeric codes are further classified into weighted codes and nonweighted codes. The most obvious way of encoding digits is "natural BCD" (NBCD), where each decimal digit is represented by its corresponding four-bit binary value. This is also called "8421" encoding. Standard binary coded decimal code is commonly known as a weighted 8421 BCD code, with 8, 4, 2 and 1 representing the weights of the different bits starting from the most significant bit (MSB) and proceeding towards the least significant bit (LSB). The weights of the individual positions of the bits of a BCD code are: $2^3 = 8$, $2^2 = 4$, $2^1 = 2$, $2^0 = 1$. The main advantage of the Binary Coded Decimal system is that it is a fast and efficient system to convert the decimal numbers into binary numbers as compared to the pure binary system. However, the disadvantage is that BCD code is inefficient as the states between 1010(decimal 10), and 1111

(decimal 15) are not used. In non-weighted code, there is no positional weight i.e. each position within the binary number is not assigned a prefixed value. No specific weights are assigned to bit position in non-weighted code. The non-weighted codes are classified to

- a) The Excess-3 code b) The Gray code

2.2 Gray code

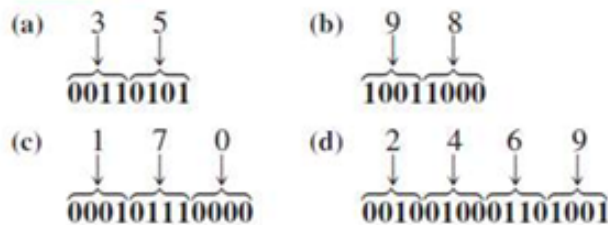
The Gray code was designed by *Frank Gray* at Bell Labs in 1953. It belongs to a class of codes called the minimum change code. The successive coded characters never differ in more than one-bit. The Gray code is a non-weighted code. Because of this, the gray code is not suitable for arithmetic operations but finds applications in input/output devices, some analog-to-digital converters and designation of rows and columns in Karnaugh map.... etc.

2.2.1 Decimal to Gray code conversion

Example: Convert each of the following decimal numbers to BCD

- (a) 35 (b) 98 (c) 170 (d) 2469

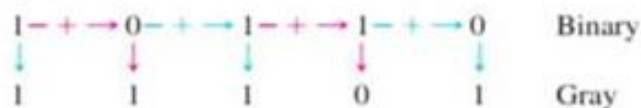
Solution



2.2.2 Binary to Gray code conversion

- 1- The most significant bit (MSB) in the Gray code is same as the corresponding bit in the binary number
- 2- Going from left to right, add each adjacent pair of binary bit to get the next Gray code bit and discard the carry.

Example: conversion of the binary number 10110 to Gray code

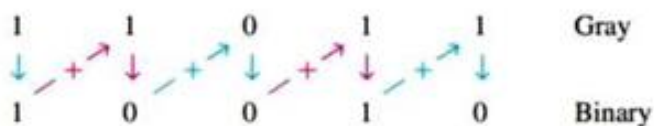


The Gray code is (11101).

2.2.3 Gray code to Binary conversion

- 1- The most significant bit (MSB) in the Binary code is same as the corresponding bit in the Gray code
- 2- Going from left to right, add each binary bit generated to the gray digit in the next adjacent position and discard the carry.

Example: conversion of the Gray code 11011 to binary



The binary number is (10010).

Example: Convert the following numbers to BCD code: $(136)_{10}$, $(491)_{10}$, $(101101)_2$?

$$(136)_{10} = (\overbrace{0001}^1 \overbrace{0011}^3 \overbrace{0110}^6)_{BCD}$$

$$(491)_{10} = (0100 \ 1001 \ 0001)_{BCD}$$

To convert the binary, octal or hexadecimal number to BCD code, must convert the number to decimal system firstly, and then convert the decimal number to BCD code.

$$(101101)_2 = (45)_{10} = (0100 \ 0101)_{BCD}$$

Example: convert the BCD code (1001 0101 0000 0100) to decimal number?

BCD code: 1001 0101 0000 0100

Decimal digit: 9 5 0 4

The decimal number = $(9504)_{10}$

Example: convert the number $(193)_{10}$ to

(a) Binary number (b) BCD code

(a) $(193)_{10} = (11000001)_2$

(b) $(193)_{10} = (0001\ 1001\ 0011)$ in BCD code.

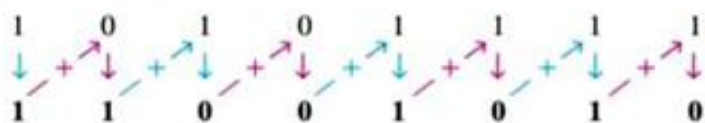
Example: (a) Convert the binary number 11000110 to Gray code.

(b) Convert the Gray code 10101111 to binary.

(a) Binary to Gray code:



(b) Gray code to binary:



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

Decimal	Binary	Gray
0	000	000
1	001	001
2	010	011
3	011	010
4	100	110
5	101	111
6	110	101
7	111	100

2.3 Addition in BCD

To add two BCD codes, use the following steps:

1. Add the two BCD numbers like addition in binary system rules.
2. If the fourth digit, MSB, is less or equal to 9, it is accepted in BCD code.
3. If the fourth digit, MSB, is greater than 9, or a carry out of the fourth digit, it is an unaccepted in BCD code. add 6 (0110) to the sum of addition operation in order to skip the six invalid codes and return the code to the weight 8,4,2,1. If a carry results when 6 is added, add the carry to the next group.

Example: Add the following BCD codes

- a) (0101) + (0001)
- b) (0011 0101) + (0010 0010)
- c) (1001) + (1000)
- d) (0110 0101) + (0010 0101)
- e) $(1000\ 0111)_2 + (0101\ 0011)_2$

a)
$$\begin{array}{r} 0101 \\ 0001 + \\ \hline 0110 \end{array}$$
 Valid in BCD code.
$$\begin{array}{r} \leftarrow 5 \\ \leftarrow 1 \\ \hline 6 \end{array}$$

b)
$$\begin{array}{r} 0011 \ 0101 \\ 0010 \ 0010 + \\ \hline 0101 \ 0111 \end{array}$$
 Valid in BCD code
$$\begin{array}{r} \leftarrow 35 \\ \leftarrow 22 + \\ \hline 57 \end{array}$$

c) (1001) + (1000)

$$\begin{array}{r} 1001 \\ 1000 + \\ \hline 1 \ 0001 \end{array}$$
 Invalid because carry
 Add 6

$$\begin{array}{r} 0110 + \\ \hline 0001 \ 0111 \\ 1 \quad 7 \end{array}$$

$$\begin{array}{r} \leftarrow 9 \\ \leftarrow 8 + \\ \hline 17 \end{array}$$

d) (0110 1001) + (0010 0101)

$$\begin{array}{r} 0110 \ 0101 \\ 0010 \ 0101 \\ \hline 1000 \ 1010 \end{array}$$
 Right group invalid (>9)
$$\begin{array}{r} \leftarrow 65 \\ \leftarrow 25 + \\ \hline 90 \end{array}$$

 Add 6 to right group

$$\begin{array}{r} 0110 \\ \hline 1001 \ 0000 \end{array}$$
 Add carry to left group

$$\begin{array}{r} 9 \quad 0 \end{array}$$

e) (1000 0111) + (0101 0011)

$$\begin{array}{r} 1000 \ 0111 \\ 0101 \ 0011 + \\ \hline 1101 \ 1010 \end{array}$$
 Invalid both group (>9)
$$\begin{array}{r} 87 \\ 53 + \\ \hline 140 \end{array}$$

 Add 6 to both groups

$$\begin{array}{r} 0110 \ 0110 \\ \hline 0001 \ 0100 \ 0000 \end{array}$$

2.4 Complements

Complements are used in digital computers for simplifying the subtraction operation and for logical manipulation. There are two types of complements, one is r 's complement and another is $(r - 1)$'s complement, where r is base of the system. For binary system the base r is 2, therefore 2's complement and 1's complement is possible.

2.4.1 1's complement

1's complement of a binary number is formed by simply changing each 1 in the number to 0 and each 0 in the number to 1.

1 0 1 1 0 1	Binary No.	0 1 1 0 1 0	Binary No.
↓ ↓ ↓ ↓ ↓ ↓		↓ ↓ ↓ ↓ ↓ ↓	
0 1 0 0 1 0	1's complement	1 0 0 1 0 1	1's complement

2.4.3 2's Complement:

The 2's complement of a binary number is formed by taking 1's complement of the number and adding 1 to the least significant bit (LSB) position

$$\text{2's complement} = \text{1's complement} + 1$$

<u>Binary Number</u>	<u>1's complement</u>	<u>2's complement</u>
1010	0101	0110
0101	1010	1011
1001	0110	0111
0001	1110	1111

2.5 Signed Magnitude of Binary Numbers

Digital circuitry requires both positive and negative numbers. A signed binary number consists of a sign, either positive or negative and magnitude. In a signed magnitude representation of binary numbers, the most significant digit is zero for the representation of positive binary number and one for the representation of negative binary numbers. This msd (0 or 1) represents whether the number is positive or negative and the magnitude is the value of the numbers. **There are three methods by which a signed number can be represented i.e. signed magnitude, 1's-complement and 2's-complement.** Digital computers store negative binary numbers in the form of 2's complement. Of the three methods mentioned above, for the representation of signed binary numbers, 2's-complement method is most widely used and sign magnitude is rarely used. A byte as signed binary number can be represented as follows for decimal values $(45)_{10}$ and $(-67)_{10}$:

$$(45)_{10} = (00101101)_2$$



+sign magnitude

$$(-67)_{10} = (11000011)_2$$



-sign magnitude

Example: Express the decimal number -39 as an 8-bit number in the sign-magnitude, 1's complement, and 2's complement forms.

First, write the 8-bit number for +39.

00100111

In the *sign-magnitude form*, -39 is produced by changing the sign bit to a 1 and leaving the magnitude bits as they are. The number is

10100111

In the *1's complement form*, -39 is produced by taking the 1's complement of +39 (00100111).

11011000

In the *2's complement form*, -39 is produced by taking the 2's complement of +39 (00100111) as follows:

11011000	1's complement
+ 1	
11011001	2's complement

Example: Determine the decimal values of the signed binary numbers expressed in 1's complement: (a) 00010111 (b) 11101000

- (a) The bits and their powers-of-two weights for the positive number are as follows:

-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
0	0	0	1	0	1	1	1

Summing the weights where there are 1s,

$$16 + 4 + 2 + 1 = +23$$

- (b) The bits and their powers-of-two weights for the negative number are as follows. Notice that the negative sign bit has a weight of -2^7 or -128 .

-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	1	1	0	1	0	0	0

Summing the weights where there are 1s,

$$-128 + 64 + 32 + 8 = -24$$

Adding 1 to the result, the final decimal number is

$$-24 + 1 = -23$$

Example : Determine the decimal values of the signed binary numbers expressed in 2's complement: (a) 01010110 (b) 10101010

- (a) The bits and their powers-of-two weights for the positive number are as follows:

-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
0	1	0	1	0	1	1	0

Summing the weights where there are 1s,

$$64 + 16 + 4 + 2 = +86$$

- (b) The bits and their powers-of-two weights for the negative number are as follows. Notice that the negative sign bit has a weight of $-2^7 = -128$.

-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	0	1	0	1	0	1	0

Summing the weights where there are 1s,

$$-128 + 32 + 8 + 2 = -86$$

Homework #1

Solve the following questions carefully

Q1/ Add, subtract, and multiply in binary:

(a) 1111 and 1010 (b) 110110 and 11101 (c) 100100 and 10110

Q2/ Convert the following binary numbers to their decimal equivalents:

(a) 00001110 (b) 11100000 (c) 10000011 (d) 10011010.

Q3/ Convert the following decimal numbers to their binary equivalents:

(a) 32 (b) 200 (c) 170 (d) 258

Q4/ Convert the following whole decimal numbers to their hexadecimal equivalents:

(a) 8 (b) 10 (c) 14 (d) 16

Q5/ Convert the following binary numbers to their hexadecimal equivalents:

(a) 11110010 (b) 11011001 (c) 111110.000011 (d) 10001.11111

Q6/ Divide in binary

(a) $10001101 \div 110$ (b) $110000011 \div 1011$ (c) $1110100 \div 1010$

Q7/ Convert the following binary numbers to decimal numbers.

(a) $(101.1011)_2$ (b) $(110110)_2$ (c) $(11010.101)_2$

Q8/ Perform the following arithmetic operations: -

a. $010111 + 001100$ b. $1011 - 0110$ c. $1010 * 1001$ d. $1100 \div 0100$

Q9/ Solve the following MCQ questions carefully

1- The decimal equivalent of 1000 is

(a) 2 (b) 4 (c) 6 (d) 8

2- The binary number 11011101 is equal to the decimal number

- (a) 121 (b) 221 (c) 441 (d) 256

3- The decimal number 21 is equivalent to the binary number

- (a) 10101 (b) 10001 (c) 10000 (d) 11111

4- The decimal number 250 is equivalent to the binary number

- (a) 11111010 (b) 11110110 (c) 11111000 (d) 11111011

5- The sum of 1111 + 1111 in binary equals

- (a) 0000 (b) 2222 (c) 11110 (d) 11111

6- The difference of 1000 - 100 equals

- (a) 100 (b) 101 (c) 110 (d) 111

7- The 1's complement of 11110000 is

- (a) 11111111 (b) 11111110 (c) 00001111 (d) 10000001

8- The 2's complement of 11001100 is

- (a) 00110011 (b) 00110100 (c) 00110101 (d) 00110110

9- The decimal number 1122 is expressed in the 2's complement form as

- (a) 01111010 (b) 11111010 (c) 01000101 (d) 10000101

10- The binary number 101100111001010100001 can be written in octal as

- (a) (5471230)₈ (b) (5471241)₈ (c) (2634521)₈ (d) (23162501)₈

11- The binary number 10001101010001101111 can be written in hexadecimal as

- (a) (AD467)₁₆ (b) (8C46F)₁₆ (c) (8D46F)₁₆ (d) (AE46F)₁₆

12- The binary number for F7A916 is

- (a) 1111011110101001 (b) 1110111110101001 (c) 1111111010110001 (d) 1111011010101001
-

13- The BCD number for decimal 473 is

(a) 111011010 (b) 110001110011 (c) 010001110011 (d) 010011110011

Q10/ Determine the 1's complement of each binary number:

(a) 00011010 (b) 11110111 (c) 10001101

Q11/ Determine the 2's complement of each binary number:

(a) 00010110 (b) 11111100 (c) 10010001

Q12/ Convert the following decimal numbers to BCD:

(a) 6 (b) 15 (c) 273 (d) 849

Q13/ What decimal numbers are represented by each BCD code?

(a) 10001001 (b) 001001111000 (c) 000101010111

Q14/

(a) Convert binary 101101 to Gray code.

(b) Convert Gray code 100111 to binary.

Binary Logic Gates

The basic building blocks of a computer are called **logical gates**. Gates are basic circuits that have at least one (and usually more) **input** and exactly one output. Input and output values are the logical values **true** and **false**. In computer architecture, it is common to use 0 for false and 1 for true. Gates have no memory. The value of the output depends only on the current value of the inputs. A useful way of describing the relationship between the inputs of gates and their output is the truth table. In a truth table, the value of each output is tabulated for every possible combination of the input values. usually consider three basic kinds of gates, **AND**-gates, **OR**-gates, and **NOT**-gates (or **Inverters**).

3.1 Basic Gates

3.1.1 AND Gate

The AND operation is represented by a dot(.) or by the absence of an operator. **E.g.** $A.B=F$ $AB=F$ are all read as A AND B=F. The logical operation AND is interpreted to mean that $F=1$ if $A=1$ and $B=1$ otherwise $F=0$

A	B	$F = A.B$
0	0	0
0	1	0
1	0	0

1	Truth Table

Symbol



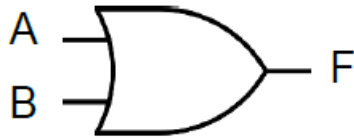
3.1.2 OR Gate

The OR operation is represented by a (+) sign for example, $A+B=F$ is interpreted as $A \text{ OR } B=F$ meaning that $F=1$ if $A=1$ or $B=1$ or if both $A=1$ and $B=1$. If both A and B are 0, then $F=0$.

Symbol

Truth Table

A	B	$F = A + B$
0	0	0
0	1	1

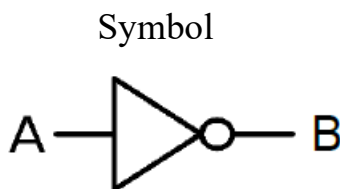


1	0	1
1	1	1

3.1.3 NOT Gate or INVERTER

The inverter is a little different from AND and OR gates in that it always has exactly one input as well as one output. Whatever logical state is applied to the input, the opposite state will appear at the output. This operation is represented by a bar or a prime. For example, $A' = A = B$ is interpreted as NOT $A = B$.

Truth Table



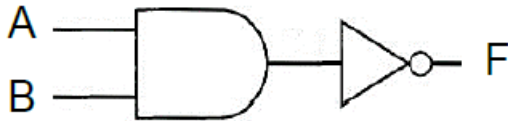
A	B
0	1
1	0

3.2 Combined gates

Sometimes, it is practical to combine functions of the basic gates into more complex gates, for instance, to save space in circuit diagrams. This section, shows some such combined gates together with their truth tables.

3.2.1 NAND Gate

The NAND-gate is an AND-gate with an inverter on the output. This operation is represented by $F = \overline{A \cdot B}$. So instead of drawing several gates like this:



draw a single NAND-gate with

little ring on the output like this:



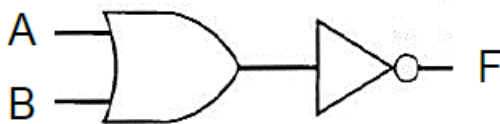
A	B	$F = \overline{A \cdot B}$
0	0	1
0	1	1
1	0	1
1	1	0

a

3.2.2 NOR Gate

The NOR-gate is an OR-gate with an inverter on the output. This operation is represented by $F = \overline{A + B}$. So instead of drawing several gates like this:

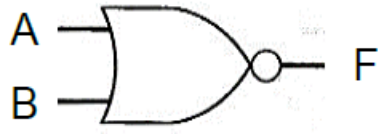
Truth Table



draw a single NOR gate with a little

ring on the output like this:

A	B	$F = \overline{A + B}$
0	0	1
0	1	0
1	0	0
1	1	0



3.2.3 Exclusive OR Gate(Ex-OR/ XOR)

The Exclusive-OR-Gate is similar to an OR-gate. It can have an arbitrary number of inputs, and its output value is 1 if exactly one input is 1 (and thus the others 0).

Otherwise, the output is 0. This operation is represented by
Truth Table

$$F = A \cdot \bar{B} + \bar{A} \cdot B = A \oplus B$$



A	B	$F = A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

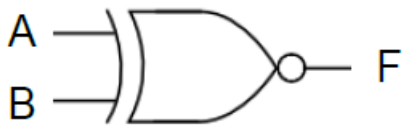
3.2.4 Exclusive NOR Gate(Ex-NOR/ XNOR)

The Exclusive-NOR-Gate is similar to an NOR-Gate. It can have an arbitrary number of inputs, and its output value is 1 if the two inputs are of the same values (1 and 1 or 0 and 0). Otherwise, the output is 0.

$$F = A.B + \bar{A}.\bar{B}$$

This operation is represented by

$$A \odot B$$

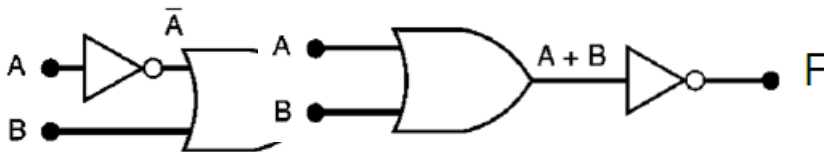


A	B	$F = \overline{A \oplus B}$
0	0	1
0	1	0
1	0	0
1	1	1

Examples: Draw the following functions

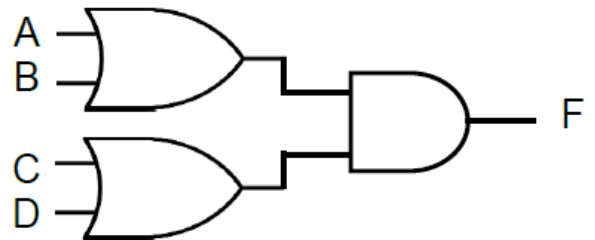
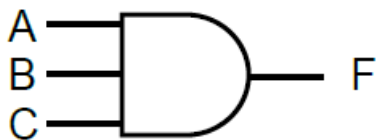
1- $F = \bar{A} + B$

2- $F = \overline{A + B}$



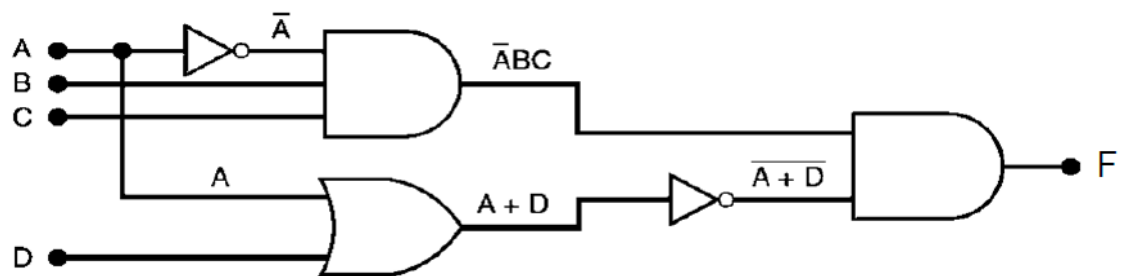
3- $F = A.B.C$

4- $F = (A+B).(C+D)$

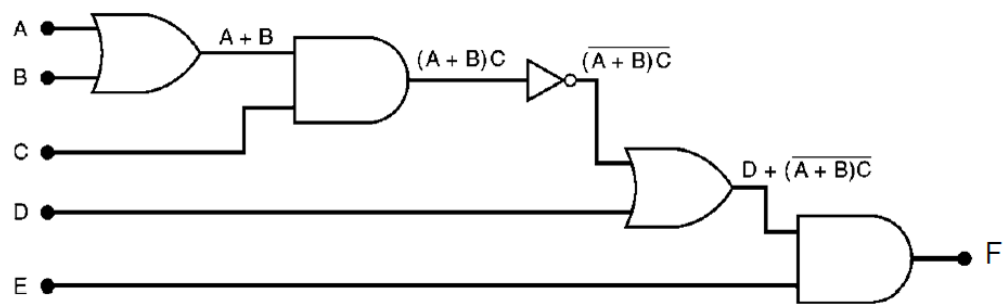


-

5- $F = ABC(A + D)$

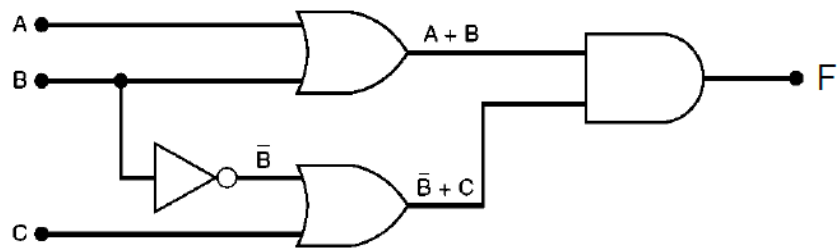


$$6- F = [D + (A + B) C] \cdot E$$

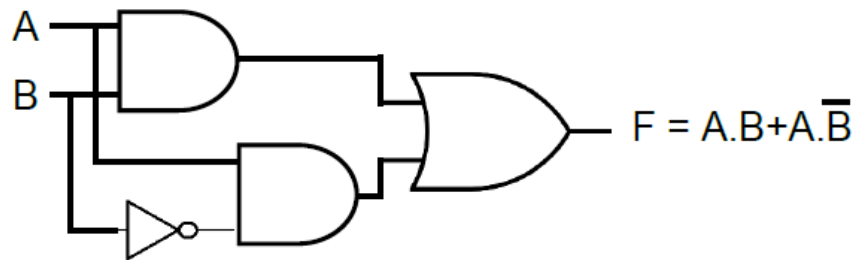


—

$$7- F = (A + B)(B + C)$$



Example: Find the truth table of F.



A	B	$\bar{B}$	AB	$A\bar{B}$	F
0	0	1	0	0	0
0	1	0	0	0	0
1	0	1	0	1	1
1	1	0	1	0	1

Logic simplification

4.1 Boolean algebra

Boolean algebra or switching algebra is a system of mathematical logic to performs different mathematical operations in a binary system. Boolean algebra which was formulated by George Boole, an English mathematician (1815-1864) described propositions whose outcome would be either true or false. There are only three basic binary operations, AND, OR, and NOT by which all simple as well as complex binary mathematical operations are to be done. There are many rules in Boolean algebra by which those mathematical operations are done. In Boolean algebra, the variables are represented by English Capital Letters like A, B, C, etc. and the value of each variable can be either 1 or 0, nothing else. In Boolean algebra, an expression given can also be converted into a logic diagram using different logic gates like AND gate, OR gate and NOT gate, NOR gates, NAND gates, XOR gates, XNOR gates, etc. Some basic logical Boolean operations:

Input		Output							
A	B	AND	NAND	OR	NOR	X-OR	X-NOR	NOT ($\bar{A}$)	NOT ($\bar{B}$)
0	0	0	1	0	1	0	1	1	1
0	1	0	1	1	0	1	0	1	0
1	0	0	1	1	0	1	0	0	1
1	1	1	0	1	0	0	1	0	0

4.2 Some Basic Laws for Boolean Algebra

4.2.1 Boolean Postulates

$$1. A + 0 = A$$

$$2. A + 1 = 1$$

$$3. A \cdot 0 = 0$$

$$4. A \cdot 1 = A$$

$$5. A + A = A$$

$$6. A + \bar{A} = 1$$

$$7. A \cdot A = A$$

$$8. A \cdot \bar{A} = 0$$

$$9. \bar{\bar{A}} = A$$

$$10. A + AB = A$$

$$11. A + \bar{A}B = A + B$$

$$12. (A + B)(A + C) = A + BC$$

4.2.2

Boolean

Laws

1. Commutative Law: According to Commutative Law, the order of OR operations and AND operations conducted on the variables makes no difference.

$$(a) A + B = B + A$$

$$(b) AB = BA$$

2. Associate Law: This law is for several variables, where the OR operation of the variables result is the same though the grouping of the variables is different. This law is quite the same in the case of AND operators.

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

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

3. Distributive Law: This law is composed of two operators, AND and OR.

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

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

4. Identity Law

$$(a) A + A = A$$

$$(b) AA = A$$

6. De Morgan's

5. Redundance Law

Theorem

$$\text{a) } A + AB = A$$

$$\text{b) } A(A + B) = A$$

$$\text{c) } AB + A\bar{B} = A$$

$$\text{d) } A(\bar{A} + B) = AB$$

$$\text{e) } A + \bar{A}B = A + B$$

$$\text{f) } (A + B)(A + \bar{B}) = A$$

$$\text{a) } \overline{(A + B)} = \bar{A}\bar{B}$$

$$\text{b) } \overline{(AB)} = \bar{A} + \bar{B}$$

The laws of Boolean algebra are also true for more than two variables. Let us show one use of this law to prove the expression $A + (B \cdot C) = (A + B) \cdot (A + C)$

Proof:

$$A + BC = A \cdot 1 + B \cdot C \text{ (since, } A \cdot 1 = A)$$

$$= A \cdot 1 + B + B \cdot C \text{ (since, } B + 1 = 1)$$

$$= A \cdot 1 + C + AB + BC \text{ (since, } A \cdot A = A \cdot 1 = A)$$

$$= A \cdot 1 + AB + BC$$

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

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

Proof of De Morgan's Theorem by truth table

$$\overline{A + B} = \bar{A} \cdot \bar{B}$$

$$\overline{A \cdot B} = \bar{A} + \bar{B}$$

Inputs		Outputs					
A	B	$\bar{A}$	$\bar{B}$	$\overline{A + B}$	$\bar{A} \cdot \bar{B}$	$\overline{A \cdot B}$	$\bar{A} + \bar{B}$
0	0	1	1	1	1	1	1
0	1	1	0	0	0	1	1
1	0	0	1	0	0	1	1
1	1	0	0	0	0	0	0

Column for $\overline{A + B}$ and $\bar{A} \cdot \bar{B}$ are same

Column for $\overline{A \cdot B}$ and $\bar{A} + \bar{B}$ are same

Examples

1. Simplify, $\overline{(A + \bar{B})(C + \bar{D})}$

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

$$= \bar{A} \cdot \bar{\bar{B}} + \bar{C} \cdot \bar{\bar{D}}$$

$$= \bar{A}B + \bar{C}D$$

2. Simplify, $\overline{\overline{AB} + \bar{A} + AB}$

$$\overline{\overline{AB} + \bar{A} + AB} = \overline{\overline{AB}} \cdot \bar{\bar{A}} \cdot \overline{AB}$$

$$= AB \cdot A \cdot \overline{AB}$$

$$= 0$$

3. Simplify, $AB + A\bar{B}C + B\bar{C}$

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

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

$$= AB + AC + B\bar{C} \text{ Since } B + \bar{B} = 1$$

$$= AB(C + \bar{C}) + AC + B\bar{C}$$

$$= ABC + AB\bar{C} + AC + B\bar{C}$$

$$= AC(1 + B) + B\bar{C}(A + 1)$$

$$= AC + B\bar{C}$$

4. Simplify, $C + \overline{BC}$:

<u>Expression</u>	<u>Rule(s) Used</u>
$C + \overline{BC}$	Original Expression
$= C + (\overline{B} + \overline{C})$	DeMorgan's Law.
$= (C + \overline{C}) + \overline{B}$	Commutative, Associative Laws.
$= 1 + \overline{B}$	Complement Law.
$= 1$	Identity Law.

5. Simplify $\overline{A}\overline{B}\overline{C} + \overline{A}B\overline{C} + A\overline{B}\overline{C} + ABC\overline{C}$

$$\begin{aligned}
 \overline{A}\overline{B}\overline{C} + \overline{A}B\overline{C} + A\overline{B}\overline{C} + ABC\overline{C} &= \overline{A}\overline{C}(\overline{B} + B) + A\overline{C}(\overline{B} + B) \\
 &= \overline{A}\overline{C} + A\overline{C} \text{ Since } \overline{B} + B = 1 \\
 &= \overline{C}(\overline{A} + A) \\
 &= \overline{C}
 \end{aligned}$$

4.2.3 Representation of Boolean function in the truth table

Boolean algebra deals with binary variables and logic operations. A Boolean function is described by an algebraic expression called Boolean expression which function variables, the constants 0 and 1, and the logic operation symbols. of binary consists

Examples1:

Truth table for $F = A + BC$

Inputs			Output	
A	B	C	BC	$F = A + BC$
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	1	1
1	0	0	0	1
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Truth table for $F = \bar{A}B + \bar{B}C$

A	B	C	$\bar{A}$	$\bar{B}$	$\bar{A}B$	$\bar{B}C$	$\bar{A}B + \bar{B}C$
0	0	0	1	1	0	0	0
0	0	1	1	1	0	1	1
0	1	0	1	0	1	0	1
0	1	1	1	0	1	0	1
1	0	0	0	1	0	0	0
1	0	1	0	1	0	1	1
1	1	0	0	0	0	0	0
1	1	1	0	0	0	0	0

Example 2: Simplify the following Boolean functions.

1) $F = AB + AB' = A.(B+B')$

$= A.1 = A$

2) $F = AB + AB'C + AB'C'$

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

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

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

$= A(B+1)$

$= A.1 = A$

A	B	C	B'	C'	AB	AB'C	AB'C'	F
0	0	0	1	1	0	0	0	0
0	0	1	1	0	0	0	0	0
0	1	0	0	1	0	0	0	0
0	1	1	0	0	0	0	0	0
1	0	0	1	1	0	0	1	1
1	0	1	1	0	0	1	0	1
1	1	0	0	1	1	0	0	1
1	1	1	0	0	1	0	0	1

$\therefore F = A$

$$\begin{aligned}
3) \quad F &= (A+B).(A+C) \\
&= AA+AC+AB+BC \\
&= A+AC+AB+BC \\
&= A(1+C)+AB+BC \\
&= A+AB+BC \\
&= A(1+B)+BC \\
&= A+BC
\end{aligned}$$

$$\begin{aligned}
4) \quad F &= (A+B)(A+B')(A'+B)(A'+B') \\
&= (A+BB')(A'+BB') \\
&= (A+0)(A'+0) \\
&= A.A' \\
&= 0
\end{aligned}$$

$$5) F = AC' + ABC + ACD' + CD$$

$$= A(C' + BC) + C(AD' + D)$$

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

$$= AC' + AB + AC + CD$$

$$= A(C' + C) + AB + CD$$

$$= A + AB + CD$$

$$= A(1 + B) + CD$$

$$= A + CD$$

$$6) F = AB + A'BC' + A'BC$$

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

$$= AB + A'B$$

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

$$= B$$

A	B	C	A'	C'	AB	A'BC'	A'BC	F
0	0	0	1	1	0	0	0	0
0	0	1	1	0	0	0	0	0
0	1	0	1	1	0	1	0	1
0	1	1	1	0	0	0	1	1
1	0	0	0	1	0	0	0	0
1	0	1	0	0	0	0	0	0
1	1	0	0	1	1	0	0	1
1	1	1	0	0	1	0	0	1

$$\therefore F = B$$

4.3 Standard Forms of Boolean Expressions

Logical functions are generally expressed in terms of different combinations of logical variables with their true forms as well as the complement forms. Binary logic values obtained by the logical functions and logic variables are in binary form. An arbitrary logic function can be expressed in the following forms:

1) Sum of Products (SOP)

2) Product of Sums (POS)

Sum of Products (SOP): In Boolean algebra, the logical sum of two or more logical product terms is referred to as a sum of product expressions. It is an OR operation on AND-operated variables.

For example: $Y = AB + BC + AC$ or $Y = A'B + BC + AC'$ are the sum of product expressions.

Product of Sums (POS): Similarly, the logical product of two or more logical sum terms is called a product of sums expression. It is an AND operation on OR-operated variables.

For example: $Y = (A + B + C)(A + B' + C)(A + B + C')$ or $Y = (A + B + C)(A' + B' + C')$ are product of sums expressions.

❖ Standard form

The standard form of the Boolean function is when it is expressed in the sum of the products or products of the sums fashion. The examples stated above, like $Y = AB + BC + AC$ or

$Y = (A + B + C)(A + B' + C)(A + B + C')$ are the standard forms. However, Boolean functions are also sometimes expressed in nonstandard forms like $F = (AB + CD)(A'B' + C'D')$, which is neither a sum of products form nor a product of sums form. However, the same expression can be converted to a standard form with the help of various Boolean properties, such as:

$$F = (AB + CD)(A'B' + C'D') = A'B'CD + ABC'D'$$

❖ Canonical Sum of Product Expression (Minterm)

A product term containing all n variables of the function in either true or complemented form is called the minterm. Each minterm is obtained by an AND operation of the variables in their true form or complemented form. For a two-variable function, four different combinations are possible, such as $A'B'$, $A'B$, AB' , and AB . These product terms are called fundamental products standard products or minterms. In the minterm, a variable will possess the value 1 if it is in true or uncomplemented form, whereas, it contains the value 0 if it is in complemented form. For three variables function, eight minterms are possible as listed in the following table:

a Boolean function is expressed as logical sum of minterms the rows of a table, for the value of function is 1,

A	B	C	Minterms	designation
0	0	0	$A'B'C'$	m_0
0	0	1	$A'B'C$	m_1
0	1	0	$A'BC'$	m_2
0	1	1	$A'BC$	m_3
1	0	0	$AB'C'$	m_4
1	0	1	$AB'C$	m_5
1	1	0	ABC'	m_6
1	1	1	ABC	m_7

When the all the from truth which the

it is referred to as the canonical sum of product expression. The same can be expressed in a compact form by listing the corresponding decimal-equivalent codes of the minterms containing a function value of 1. For example, if the canonical sum of the product form of a three-variable logic function F has the minterms $A'BC$,

$AB'C$, and ABC' , this can be expressed as the sum of the decimal codes corresponding to these minterms as below:

$F(A, B, C) = \Sigma(3, 5, 6) = m_3 + m_5 + m_6 = A'BC + AB'C + ABC'$ where $\Sigma(3, 5, 6)$ represents the summation of minterms corresponding to decimal codes 3, 5, and 6.

1) Check each term in the given logic function. Retain if it is a minterm, and continue to examine the next term in the same manner.

2) Examine the variables that are missing in each product which is not a minterm. If the missing variable in the minterm is X , multiply that minterm with $(X + X')$.

3) Multiply all the products and discard the redundant terms.

Example3: Obtain the canonical sum of product form of the following function

$$F(A, B) = A + B$$

Solution: The given function contains two variables A and B . The variable B is missing from the first term of the expression and the variable A is missing from the second term of the expression. Therefore, the first term is to be multiplied by $(B + B')$ and the second term is to be multiplied by $(A + A')$ as demonstrated below:

$$F(A, B) = A + B$$

$$= A.1 + B.1$$

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

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

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

A	B	Minterms	designation
0	0	$A'B'$	m_0
0	1	$A'B$	m_1
1	0	AB'	m_2
1	1	AB	m_3

$$F(A, B) = AB + AB' + A'B = m_3 + m_2 + m_1 = \Sigma(1,2,3)$$

❖ Canonical Product of Sum Expression (Maxterm)

A sum term containing all n variables of the function in either true or complemented form is called the maxterm. Each maxterm is obtained by an OR operation of the variables in their true form or complemented form. Four different combinations are possible for a two-variable function, such as $A + B$, $A + B'$, $A' + B$, and $A' + B'$.

These sum terms are called standard sums or maxterms. Note that, in the maxterm,

a variable will possess the value 0, if it is in true or uncomplemented form, whereas, it contains the value 1 if it is in complemented form. Like minterms, for a three-variable function, eight maxterms are also possible as listed in the following table:

A	B	C	Maxterms	designation
0	0	0	$A+B+C$	M_0
0	0	1	$A+B+C'$	M_1
0	1	0	$A+B'+C$	M_2
0	1	1	$A+B'+C'$	M_3
1	0	0	$A'+B+C$	M_4
1	0	1	$A'+B+C'$	M_5
1	1	0	$A'+B'+C$	M_6
1	1	1	$A'+B'+C'$	M_7

When a Boolean function is expressed as the logical product of all the maxterms from the rows of a truth table, for which the value of the function is 0, it is referred to as the canonical product of sum expression. The same can be expressed in a compact form by listing the corresponding decimal equivalent codes of the maxterms containing a function value of 0. For example, if the canonical product of sums form of a three-variable logic function F has the maxterms $A + B + C$, $A + B' + C$, and $A' + B + C'$, this can be expressed as the product of the decimal codes corresponding to these maxterms as below:

$$F(A,B,C) = \pi(0,2,5) = M_0 M_2 M_5 = (A + B + C)(A + B' + C)(A' + B + C')$$

where $\pi(0,2,5)$ represents the product of maxterms corresponding to decimal codes 0, 2, and 5.

The canonical product of the sum form of a logic function can be obtained by using the following procedure.

- 1) Check each term in the given logic function. Retain it if it is a maxterm, and continue to examine the next term in the same manner.

2) Examine the variables that are missing in each sum term that is not a maxterm. If the missing variable in the maxterm is X, add that maxterm with (X.X').

3) Expand the expression using the properties and postulates as described earlier and discard the redundant terms.

: Represent (SOP) and (POS) expressions by using the following truth **4Example** table:

SOP:-

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

$$F = \Sigma (2,4,5,6) \quad \text{POS}$$

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

$$F = \pi (0,1,3,7)$$

A	B	C	F	SOP	POS
0	0	0	0		A+B+C
0	0	1	0		A+B+C'
0	1	0	1	A'BC'	
0	1	1	0		A+B'+C'
1	0	0	1	AB'C'	
1	0	1	1	AB'C	
1	1	0	1	ABC'	
1	1	1	0		A'+B'+C'

: Simplify the expression for $F(A,B) = \Sigma (0,2,3)$ **5Example**

$$\text{Solution: } F = m_0 + m_2 + m_3$$

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

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

$$= B' + AB$$

$$= B' + A$$

Summary

- 1-** Boolean Algebra: a mathematical tool used in the analysis and design of digital circuits
- 2-** OR, AND, NOT: basic Boolean operations
- 3-** OR: HIGH output when any input is HIGH
- 4-** AND: HIGH output only when all inputs are HIGH
- 5-** NOT: output is the opposite logic level as the input
- 6-** NOR: OR with its output connected to an INVERTER
- 7-** NAND: AND with its output connected to an INVERTER
- 8-** Boolean theorems and rules: to simplify the expression of a logic circuit and can lead to a simpler way of implementing the circuit
- 9-** NAND, NOR: can be used to implement any of the basic Boolean operations

Karnaugh maps

5.1 Introduction

The Boolean theorems and De-Morgan's theorems are useful in manipulating the logic expression. We can realize the logical expression using gates. The number of logic gates required for the realization of a logical expression should be reduced to the minimum possible value. One of the methods used to minimize the logical expression is the K-map method. A Karnaugh map provides a pictorial method of grouping together expressions with common factors and therefore eliminating unwanted variables. The Karnaugh map can also be described as a special arrangement of a truth table. The K-map is a graphical device used to simplify a logical equation or to convert a truth table to its corresponding logic circuit in a simple, logical method. It is also known as the Veitch diagram. A K-map is a diagram made up of squares and may be considered to be the graphic representation of the minterm canonical form. Each minterm is represented by a cell, and the cells are assembled in an orderly arrangement such that adjacent cells represent minterms that differ by one variable. The number of cells in a K-map depends upon the number of variables in the Boolean expression. The variables map contains four cells, the three variables map contains eight cells, and the n variables map contains 2^n cells. Each row and column of the map is assigned by 0's and 1's

5.2 Two-Variable K-Map

The two-variable map is shown below. There are four minterms for two variables; hence, the map consists of four squares, one for each minterm.

		B	
		0	1
A	0	00 ⁰	01 ¹
	1	10 ²	11 ³

Two Variables K-map
With cell number

Example 1: Simplify the following Boolean functions using the (K- Map) Method.

1) $F(x,y) = x'y + x'y'$

Solution:

x \ y	0	1
0	1	1
1		

$$F(x,y) = x'$$

2) $F(x,y) = xy + x'y$

Solution:

x \ y	0	1
0		1
1		1

$$F(x,y) = y$$

3) $F(x,y) = x'y' + xy' + xy$

Solution:

x \ y	0	1
0	1	
1	1	1

$F(x,y) = x + y'$

4) $F(x,y) = xy + x'y + xy' + x'y'$

Solution:

x \ y	0	1
0	1	1
1	1	1

$F(x,y) = 1$

5.3 Three-Variable K-Map

A three-variable K-map is shown below. There are eight minterms for three binary variables; therefore, the map consists of eight squares. Note that the minterms are arranged, not in a binary sequence, but in a sequence similar to the Gray code.

The characteristic of this sequence is that **only one bit changes in value from one adjacent column to the next**. As more adjacent squares are combined, we obtain a

One square represents one minterm, giving a product term with fewer literals.
 Two adjacent squares represent a term with two literals.
 Four adjacent squares represent a term with one literal. Eight adjacent squares encompass the entire map and produce a function that is always equal to 1.

BC A		00		01		11		10	
		0	1	3	2	4	5	7	6
0	000	001	011	010					
1	100	101	111	110					

Three Variables K-map
with cell number

the following

Example 2: Simplify

Boolean functions using the (K- Map) Method.

$$1) F(A,B,C) = \sum (1,3,5,7)$$

Solution: $F = C$

BC A		00	01	11	10
		m_0	m_1	m_3	m_2
0			1	1	
1			1	1	

$$2) F(A,B,C) = \sum (0,1,2,4,7)$$

Solution:

BC A		00	01	11	10
		m_0	m_1	m_3	m_2
0		1	1		1
1		1		1	

$$F = B'C' + A'B' + ABC + A'C'$$

3) $F(A,B,C) = \sum (1,3,6,7)$

Solution:

A \ BC	BC			
	00	01	11	10
0	m_0	m_1	m_3	m_2
1	m_4	m_5	m_7	m_6

5) $F = A'B'C + A'BC' + A'BC + AB'C + ABC$

$F = AB + A'C$

Solution:

A \ BC	BC			
	00	01	11	10
0	m_0	m_1	m_3	m_2
1	m_4	m_5	m_7	m_6

$F = C + A'B$

4) $F(A,B,C) = \sum (0,2,4,5,6)$

Solution:

A \ BC	BC			
	00	01	11	10
0	m_0	m_1	m_3	m_2
1	m_4	m_5	m_7	m_6

$+ AB'$

6) $F(A,B,C) = \sum (3,4,6,7)$

Solution:

A \ BC	BC			
	00	01	11	10
0	m_0	m_1	m_3	m_2
1	m_4	m_5	m_7	m_6

$F = AC' + BC$

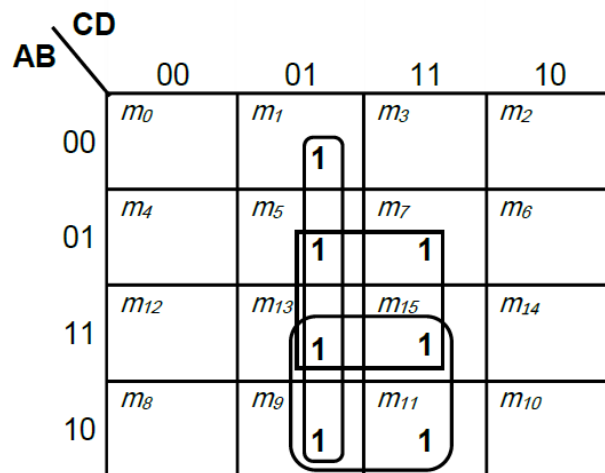
5.4 Four-Variable K-Map

The number of squares in 4 variable K-map is sixteen since the number of variables is four. The following figure shows 4 variable K-Map. The combination

of adjacent squares that is useful during the simplification process is easily
 One square represents one determined from inspection of the four-variable map:
 Two adjacent squares represent a term minterm, giving a term with four literals.
 Eight Four adjacent squares represent a term with two literals. with three literals.
 Sixteen adjacent squares adjacent squares represent a term with one literal.
 function that is always equal to 1. produce a

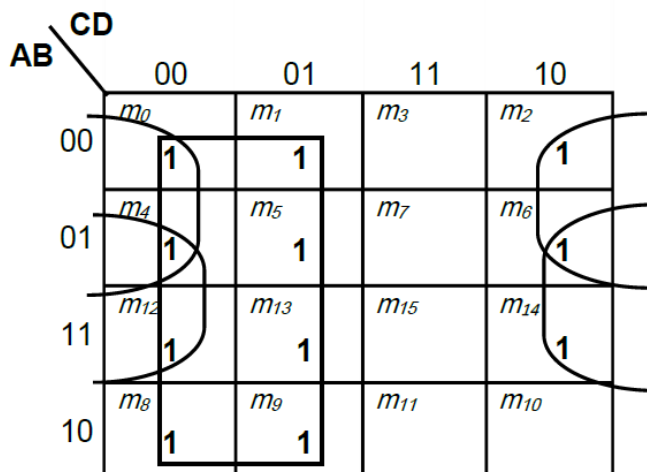
Example 3: Simplify the following Boolean functions using the (K- Map)
 Method.

$$F(A,B,C,D) = \Sigma (1,5,7,9,11,13,15)$$



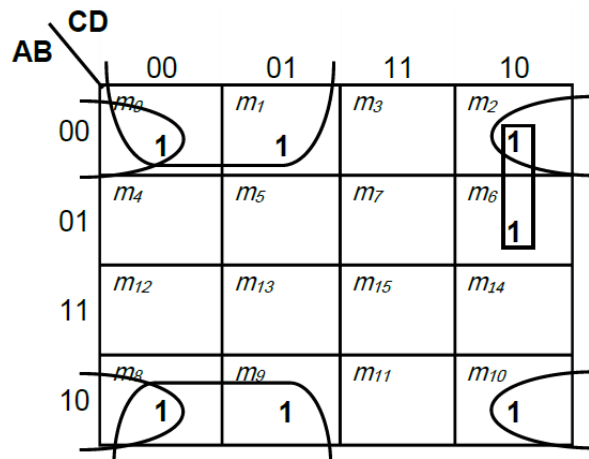
$$F = AD + BD + C'D$$

Example4: $F(A, B, C, D) = \Sigma (0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14)$



$$F = C' + A'D' + BD'$$

Example5: $F = A'B'C'D' + A'B'C'D + A'B'CD' + A'BCD' + AB'C'D' + AB'C'D + AB'CD'$



$$F = B'D' + B'C' + A'CD'$$

Example6: Minimize the following expressions using K-map

(B) $Y(A, B, C) = \sum m$

(A) $Y(A, B, C) = \sum m(1, 3, 5, 7)$

(0, 1, 4, 5)

(C) $Y(A, B, C) = \sum m(0, 2, 4, 6)$

(A) $Y(A, B, C) = \sum m(1, 3, 5, 7)$

A \ BC				
	00	01	11	10
0	0 ⁰	1 ¹	1 ³	0 ²
1	0 ⁴	1 ⁵	1 ⁷	0 ⁶

$$Y(A, B, C) = C$$

(B) $Y(A, B, C) = \sum m(0, 1, 4, 5)$

A \ BC				
	00	01	11	10
0	1 ⁰	1 ¹	0 ³	0 ²
1	1 ⁴	1 ⁵	0 ⁷	0 ⁶

$$Y(A, B, C) = \bar{B}$$

A \ BC				
	00	01	11	10
0	1 ⁰	0 ¹	0 ³	1 ²
1	1 ⁴	0 ⁵	0 ⁷	1 ⁶

$$Y(A, B, C) = \bar{C}$$

(C) $Y(A, B, C) = \sum m(0, 2, 4, 6)$

Example7: Simplify the expression using K-map $Y(A, B, C) = \sum(0, 1, 2, 3, 4, 5, 6, 7)$

A \ BC					
		00	01	11	10
0		0 1	1 1	3 1	2 1
1		4 1	5 1	7 1	6 1

$Y(A, B, C) = 1$

Example8: Minimize the following expressions using K-map

(A) $Y(A, B, C) = \Pi M(1, 3, 5, 7)$

(B) $Y(A, B, C) = \Pi M(0, 1, 4, 5)$

(C) $Y(A, B, C) = \Pi M(0, 2, 4, 6)$

(A) $Y(A, B, C) = \Pi M(1, 3, 5, 7)$

(B) $Y(A, B, C) = \Pi M(0, 1, 4, 5)$

A \ BC					
		00	01	11	10
0		0 1	1 0	3 0	2 1
1		4 1	5 0	7 0	6 1

$Y(A, B, C) = \bar{C}$

A \ BC					
		00	01	11	10
0		0 0	1 0	3 1	2 1
1		4 0	5 0	7 1	6 1

$Y(A, B, C) = B$

(C) $Y(A, B, C) = \Pi M(0, 1, 4, 5)$

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

$$Y(A, B, C) = C$$

5.5 Don't-care conditions

don't-care conditions can be used on a map to provide further simplification of the Boolean expression. To distinguish the don't-care condition from 1's and 0's, an X or d is used. In choosing adjacent squares to simplify the function in a map, the don't-care minterms may be assumed to be either 0 or 1.

Example 9: Simplify the following Boolean functions using the (K- Map) Method.

1) $F(A, B, C) = \Sigma(2, 3, 4, 5)$ which has the don't-care conditions $d(A, B, C) = \Sigma(6, 7)$

BC		00		01		11		10	
A	0	m_0	m_1	m_3	m_2	1	1		
	1	m_4	m_5	m_7	m_6	1	1	X	X

$$F = A + B$$

2) $F(A, B, C, D) = \Sigma(1, 3, 7, 11, 15)$ which has the don't-care conditions

$$d(A, B, C, D) = \Sigma(0, 2, 5)$$

AB \ CD		CD			
		00	01	11	10
AB	00	m_0 X	m_1 1	m_3 1	m_2 X
	01	m_4	m_5 X	m_7 1	m_6
	11	m_{12}	m_{13}	m_{15} 1	m_{14}
	10	m_8	m_9	m_{11} 1	m_{10}

$$F = A'B' + CD$$