Computer Organization –UNIT 1

Digital Components and Data Representation: Introduction, Numbering Systems, Decimal to Binary Conversion, Binary Coded Decimal Numbers, Weighted Codes, Self-Complementing Codes, Cyclic Codes, Error Detecting Codes, Error Correcting Codes, Hamming Code for Error Correction, Alphanumeric Codes, ASCII Code

Data Representation: Data types, Complements, Fixed Point Representation, Floating Point Representation.

Boolean Algebra: Theorems and properties, Boolean functions, canonical and standard forms, minimization of Boolean functions using algebraic identities; karnaugh representation and minimization using two and three variable Maps; Logical gates, universal gates and two-level realization using gates: AND-OR,OR-AND,NAND-NAND and NOR-NOR structures.

1.0 Digital logic circuits

Computer organization

Def:-It is concerned with the way the hardware components operate and the way they are connected together to form the computer system.

Computer design

Def:-It is concerned with the hardware design of the computer.

Computer architecture

Def:-It is concerned with the structure and behavior of the computer as seen by the user.

It includes the information, formats, the instruction set and techniques for addressing memory.

Types of computer architectures

Two basic types of computer architectures are **Von Neumann architecture** and **Harvard architecture.**

i)Von Neumann architecture

- It describes a general framework or structure that computer has been devised and implemented.
- Von Neumann architecture composed of the following components
  - The central arithmetic unit
  - Memory
  - A control unit
  - Man-machine interfaces
- In a computer with Von-Neumann architecture the CPU can be either reading an instruction or reading/writing data from/to the memory. Both cannot occur at the same time since the instructions and data use the same signal pathways and memory.

ii) Harvard architecture

It uses physically separate storage and signal pathways for their instructions and data. In a computer with Harvard architecture the CPU can read both an instruction and data from memory at the same time leading to double the memory width.

iii)Logic gates

The manipulation of binary information is done by logic circuits called gates.

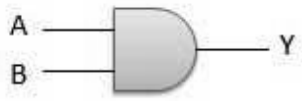
Logic gates are the basic building blocks of any digital system. It is an electronic circuit having one or more than one input and only one output. The relationship between the input and the output is based on a **certain logic**. Based on this, logic gates are named as AND gate, OR gate, NOT gate etc.

a)AND Gate

A circuit which performs an AND operation is shown in figure. It has n input (n >= 2) and one output.

Y = A AND B AND C ..... N
Y = A.B.C ..... N
Y = ABC ..... N

Logic diagram



Truth Table

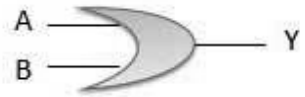
Inputs		Output
A	B	AB
0	0	0
0	1	0
1	0	0
1	1	1

b)OR Gate

A circuit which performs an OR operation is shown in figure. It has n input (n >= 2) and one output.

Y = A OR B OR C ..... N
Y = A + B + C ..... N

Logic diagram



Truth Table

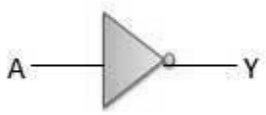
Inputs		Output
A	B	A + B
0	0	0
0	1	1
1	0	1
1	1	1

c) Inverter (or) NOT Gate

Inverter is also known as NOT gate. It has one input A and one output Y.

Y = NOT A
Y = $\overline{A}$

Logic diagram



Truth Table

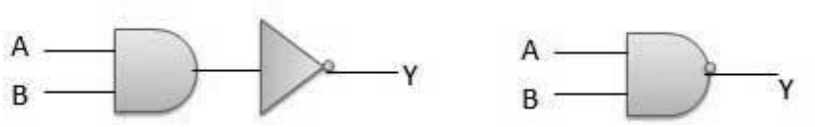
A	Y
0	1
1	0

d)NAND Gate

A NOT-AND operation is known as NAND operation. It has n input (n >= 2) and one output.

Y = A NOT AND B NOT AND C ..... N
Y = A NAND B NAND C ..... N

Logic diagram



Truth Table

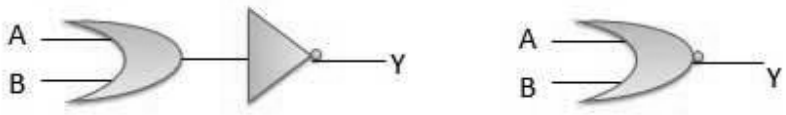
Inputs		Output
A	B	$\overline{AB}$
0	0	1
0	1	1
1	0	1
1	1	0

e)NOR Gate

A NOT-OR operation is known as NOR operation. It has n input (n >= 2) and one output.

Y = A NOT OR B NOT OR C ..... N
Y = A NOR B NOR C ..... N

Logic diagram



Truth Table

Inputs		Output
A	B	$\overline{A+B}$
0	0	1
0	1	0
1	0	0
1	1	0

f)XOR Gate

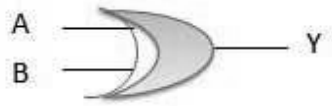
XOR or Ex-OR gate is a special type of gate. It can be used in the half adder, full adder and subtractor. The exclusive-OR gate is abbreviated as EX-OR gate or sometime as X-OR gate. It has n input (n >= 2) and one output.

Y = A XOR B XOR C ..... N

Y = $A \oplus B \oplus C$ ..... N

Y = $\overline{A}B + A\overline{B}$

Logic diagram



Truth Table

Inputs		Output
A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

g)XNOR Gate

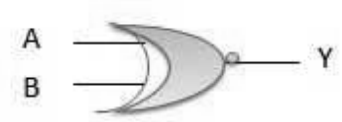
XNOR gate is a special type of gate. It can be used in the half adder, full adder and subtractor. The exclusive-NOR gate is abbreviated as EX-NOR gate or sometime as X-NOR gate. It has n input (n >= 2) and one output.

$Y = A \text{ XOR } B \text{ XOR } C \ldots\ldots N$

$Y = A \odot B \odot C \ldots\ldots N$

$Y = \overline{A}B + A\overline{B}$

Logic diagram



Truth Table

Inputs		Output
A	B	$A \odot B$
0	0	1
0	1	0
1	0	0
1	1	1

1.2 Digital Components

a)Integrated Circuits

Digital circuits are constructed with integrated circuits.

Def: An integrated circuit is a small silicon semiconductor crystal called a chip, containing the electronic components for the digital gates. The various gates are interconnected inside the chip to form the required circuit.

i)Construction of chip in IC

- The chip is mounted in ceramic or plastic container and connection are welded by thin gold wires to external **pins** to form the integrated circuit.
- The number of pins may range from 14 in a small IC package to 100 or more in a larger package.
- Each IC has a numeric designation printed on the surface of the package for identification.
- Each vendor publishes a data book or catalog that contains the exact description and all the necessary information about the ICs that it manufactures.
- As the technology of ICs has improved the number of gates that can be put in a single chip has increased considerably.

ii)Types of ICs

Small-scale integration(SSI)devices

- These contain several independent gates in a single package.
- The inputs and outputs of the gates are connected directly to the pins in the package.
- The number of gates is usually less than 10 and is limited by the number of pins available in the IC.

Medium-scale integration (MSI) devices

- These have a complexity of approximately 10 to 200 gates in a single package.
- They usually perform specific elementary digital functions such as decoders, adders and registers.

Large-scale integration (LSI) devices

- These contain between 200 and a few thousand gates in a single package.
- They include digital systems such as processors, memory chips and programmable modules.

Very-large-scale integration(VLSI) devices

- These contain thousands of gates within a single package.
- These are small in size and having low-cost.

e.g

Large memory arrays and complex microcomputer chips.

iii)Logic families of ICs

- Digital integrated circuits are classified not only by their logic operation but also by their specific circuit technology to which they belong. The circuit technology is referred to as a digital logic family.

- ❖ Each logic family has its own basic electronic circuit upon which more complex digital circuits and functions are developed.
- ❖ There are four important logic families of integrated circuits are available.
  TTL-Transistor-transistor logic
  ECL- Emitter-coupled logic
  MOS- Metal-oxide semiconductor
  CMOS- Complementary metal-oxide semiconductor

1.2 Numbering system

a)radix

A number system of base or radix r is a system that uses distinct symbols for r digits.

b) Decimal number system

The decimal number system uses radix 10 system. It has 10 symbols 0,1,2,3,4,5,6,7,8 and 9.

e.g

The string of digits 724.5 is representing decimal number system/

c) Binary number system

The binary number system uses radix 2 system. It has two symbols 0 and 1.

e.g

The string of digits 101101 is representing binary number system.

1.3 Decimal to Binary conversion

Two simple operations divide and multiply give us a convenient way to convert a decimal number to its binary equivalent.

i) To convert the integral part we divide the number by 2 and write down the remainder which must be 0 or 1.The first remainder becomes the least significant binary digit. Now we divide the quotient of that division by 2 and write down new remainder in the second position. We repeat this process until the quotient becomes zero.

ii) To convert the fractional part we need to multiply the fraction part by two.

e.g

Convert decimal number 78.625 into binary number.

i)First convert integral part 78 into binary number.

.

Division	Remainder (R)

Division	Remainder (R)
78/ 2 = 39	0
39 / 2 = 19	1
19 / 2 = 9	1
9/ 2 = 4	1
4 / 2 = 2	0
32/ 2 = 1	0
1 / 2 = 0	1

Now, write remainder from bottom to up (in reverse order), this will be 1001110which is equivalent binary number of decimal integer 78.

ii)Second convert fractional part 0.625 into binary.

Multiplication	Resultant integer part (R)
0.625x 2= 1.25	1
0.25x 2= 0.50	0
0.50x 2= 1.0	1
0 x 2 = 0	0

Now, write these resultant integer part, this will be 0.1010 which is equivalent binary fractional number of decimal fractional 0.625.

78-1001110

0.625-0.1010

$(78.625)_{10}=(1001110.1010)_2$

1.4 Binary Coded Decimal Numbers

- Binary Coded Decimal, or BCD, is another process for converting decimal numbers into their binary equivalents.
- It is a form of binary encoding where each digit in a decimal number is represented in the form of bits.
- This encoding can be done in either 4-bit or 8-bit.usually 4-bit is preferred
- It is a fast and efficient system that converts the decimal numbers into binary numbers as compared to the existing binary system.

Decimal number	Binary-coded decimal number(BCD)
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	0001 0000
20	0010 0000
50	0101 0000
99	10011001
248	0010 0100 1000

1.5 Weighted codes

Weighted binary codes are those binary codes which obey the positional weight principle. Each position of the number represents a specific weight. Several systems of the codes are used to express the decimal digits 0 through 9. In these codes each decimal digit is represented by a group of four bits.

e.g

The 2421 is an example of weighted code. In a weighted code the bits are multiplied by the weights indicated and the sum of the weighted bits give the decimal digit.

Decimal number	Weighted code 2 4 2 1
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 10 1
6	0 1 1 0
7	0 11 1
8	1 1 1 0
9	1 1 1 1

Hexa decimal system

Radix-16

Different symbols- 0 to 9, A-10-1010 B-11-1011 C-12-1100 D-13-1101 E-14-1110

F-15-1111

1.6 Self-complementing codes

Self-complementing binary codes are those whose members complement on themselves. For a binary code to become a self-complementing code, the following two conditions must be satisfied:

- The complement of a binary number should be obtained from that number by replacing 1's with 0's and 0's with 1's .
- The sum of the binary number and its complement should be equal to decimal 9.

e.g a=4- 00001000 ~a=11110111
a+~a=11111111=-128+64+32+16+8+4+2+1=-128+127

The following example will illustrate this procedure.

The Excess-3 (Xs-3) Code

Let us now consider the excess-3 (Xs-3) binary coding system. An Xs-3 equivalent of a given binary number is obtained using the following steps:

- Find the decimal equivalent of the given binary number.
- Add +3 to the decimal equivalent obtained in 1.
- Convert the newly obtained decimal number back to binary number to get the desired Xs-3 equivalent.

Following the steps given above, we draw Table , which shows the binary and Xs-3 equivalents.

Table Xs-3 binary codes

Binary numbers	Decimal equivalent	Decimal +3	Xs-3 equivalent
0000	0	3	0011
0001	1	4	0100
0010	2	5	0101
0011	3	6	0110
0100	4	7	0111
0101	5	8	1000
0110	6	9	1001
0111	7	10	1010
1000	8	11	1011
1001	9	12	1100

1.7 Cyclic codes

Cyclic codes can be used to correct errors, like Hamming codes as a cyclic codes can be used for correcting single error.

e.g CRC

1.8 Error detecting codes

- An error detection code is a binary code that detects digital errors during transmission.
- The detected errors cannot be corrected but their presence is indicated.

Types of Error detection

a) Parity Checking
b) Cyclic Redundancy Check (CRC)

a)Parity Checking

- Parity bit means nothing but an additional bit added to the data at the transmitter before transmitting the data. Before adding the parity bit, number of 1's or zeros is calculated in the data.
- Based on this calculation of data an extra bit is added to the actual information / data. The addition of parity bit to the data will result in the change of data string size.
- This means if we have an 8 bit data, then after adding a parity bit to the data binary string it will become a 9 bit binary data string.
- Parity check is also called as "Vertical Redundancy Check (VRC)".
- There is two types of parity bits in error detection, they are

- Even parity
- Odd parity

i)Even Parity

If the data has even number of 1's, the parity bit is 0

Ex: data is 10000001 -> parity bit 0

If the data has odd number of 1's, the parity bit is 1.

Ex: data is 10010001 -> parity bit 1

ii)Odd Parity

If the data has odd number of 1's, the parity bit is 0.

Ex: data is 10011101 -> parity bit 0

If the data has Even number of 1's, the parity bit is 1.

Ex: data is 10010101 -> parity bit 1

- ❖ The circuit which adds a parity bit to the data at transmitter is called "Parity generator". The parity bits are transmitted and they are checked at the receiver.
- ❖ If the parity bits sent at the transmitter and the parity bits received at receiver are not equal then an error is detected. The circuit which checks the parity at receiver is called "Parity checker".

Messages with even parity and odd parity

3 bit data			Message with even parity		Message with odd parity	
A	B	C	Message	Parity	Message	Parity
0	0	0	000	0	000	1
0	0	1	001	1	001	0
0	1	0	010	1	010	0
0	1	1	011	0	011	1
1	0	0	100	1	100	0
1	0	1	101	0	101	1
1	1	0	110	0	110	1
1	1	1	111	1	111	0

b) Cyclic Redundancy Check (CRC)

CRC or Cyclic Redundancy Check is a method of detecting accidental changes/errors in the communication channel.

CRC uses **Generator Polynomial** which is available on both sender and receiver side. An example generator polynomial is of the form like $x^3 + x + 1$. This generator polynomial represents key 1011

n : Number of bits in data to be sent

from sender side.

k : Number of bits in the key obtained

from generator polynomial.

Sender Side (Generation of Encoded Data from Data and Generator Polynomial (or Key)):

1. The binary data is first augmented by adding k-1 zeros in the end of the data
2. Use ***modulo-2 binary division*** to divide binary data by the key and store remainder of division.
3. Append the remainder at the end of the data to form the encoded data and send the same

Receiver Side (Check if there are errors introduced in transmission)

Perform modulo-2 division again and if the remainder is 0, then there are no errors.

Modulo 2 Division:

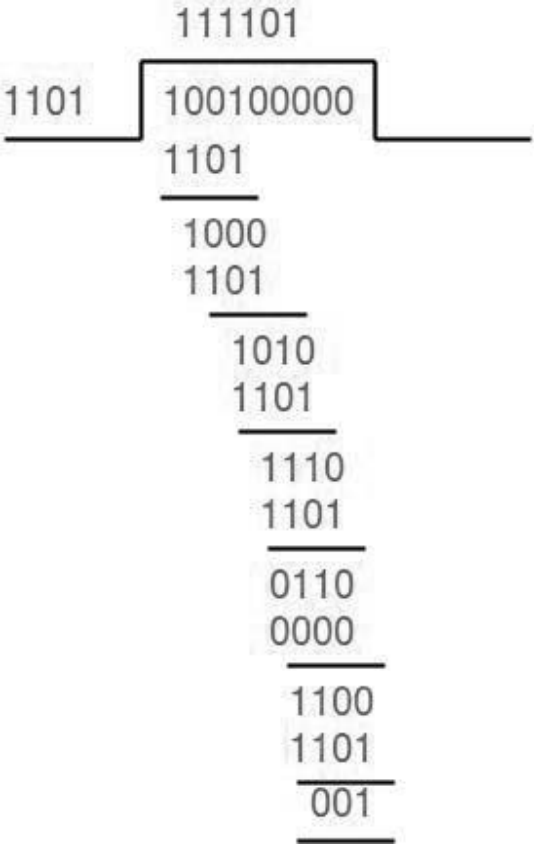
The process of modulo-2 binary division is the same as the familiar division process we use for decimal numbers. Just that instead of subtraction, we use XOR here.

- In each step, a copy of the divisor (or data) is XORed with the k bits of the dividend (or key).
- The result of the XOR operation (remainder) is (n-1) bits, which is used for the next step after 1 extra bit is pulled down to make it n bits long.
- When there are no bits left to pull down, we have a result. The (n-1)-bit remainder which is appended at the sender side.

Example 1 (No error in transmission):
Data word to be sent - 100100

Key - 1101 [Or generator polynomial $x^3 + x^2 + 1$]

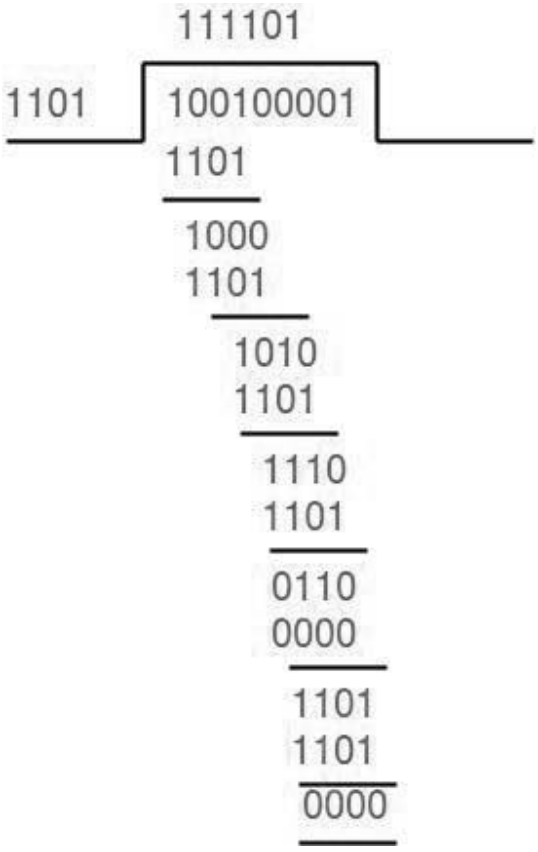
Sender Side:



Therefore, the remainder is 001 and hence the encoded data sent is 100100001.

Receiver Side

Code word received at the receiver side 100100001



Therefore, the remainder is all zeros. Hence, the

data received has no error.

. Another example is $x^2 + 1$ that represents key 101.

1.9 Error Correcting Codes

- The codes which are used for both error detecting and error correction are called as “Error Correction Codes”. The error correction techniques are of two types. They are
  - Single bit error correction
  - Burst error correction
- The process or method of correcting single bit errors is called “single bit error correction”.
- The method of detecting and correcting burst errors in the data sequence is called “Burst error correction”.
- Hamming code or Hamming Distance Code is the best error correcting code we use in most of the communication network and digital systems.

1.10 Hamming code for error correction

- Hamming code is used to detect and correct the error in the transmitted data. So, it is an error detection and correction code.
- It was originally invented by *Richard W. Hamming* in the year 1950.
- Hamming codes detect 1-bit and 2-bit errors.
- While transmitting the message, it is encoded with the redundant bits. The redundant bits are the extra bits that are placed at certain locations of the data bits to detect the error. At the receiver end, the code is decoded to detect errors and the original message is received.
- So before transmitting, the sender has to encode the message with the redundant bits.
- It involves three steps.
  a) Selecting the number of redundant bits
  b) Choosing the location of redundant bits
  c) Assigning the values to redundant bits

a) Selecting the number of redundant bits

- The hamming code uses the number of redundant bits depending on the number of information bits in the message.
- Let n be the number of information or data bits, then the number of redundant bits P is determined from the following formula,
  - $2^P >= n+P+1$

e.g

If 4-bit information is to be transmitted then n=4.the number of redundant bits is determined by the trial and error method

Let P=2, we get,

$2^2 >= 4+2+1$

The above equation implies 4 is not greater than or equal to 7.So let's choose another value of P=3.

$2^3 >= 4+2+1$

Now the equation satisfies the condition. So the number of redundant bits P=3.

In this way, the number of redundant bits is selected for the number of information bits to be transmitted.

b) Choosing the location of redundant bits

For the above example, the number of data bits n=4, and the number of redundant bits P=3. So the message consists of 7 bits in total that are to be coded. Let the rightmost bit be designated as bit 1, the next successive bit as bit 2 and so on.

The seven bits are bit 7, bit 6, bit 5, bit 4, bit 3, bit 2, bit 1.

In this, the redundant bits are placed at the positions that are numbered corresponding to the power of 2, i.e., 1, 2, 4, 8,… Thus the locations of data bit and redundant bit are *D4, D3, D2, P3, D1, P2, P1*.

c)Assigning the values to redundant bits

Now it is time to assign bit value to the redundant bits in the formed hamming code group. The assigned bits are called a parity bit.

Each parity bit will check certain other bits in the total code group. It is one with the bit location table, as shown below.

Bit Location	7	6	5	4	3	2	1
Bit designation	D4	D3	D2	**P3**	D1	**P2**	**P1**
Binary representation	111	110	101	100	011	010	001
Information / Data bits	D4	D3	D2		D1		
Parity bits				**P3**		**P2**	**P1**

Parity bit P1 covers all data bits in positions whose binary representation has 1 in the least significant position (001, 011, 101, 111, etc.). Thus P1 checks the bit in locations 1, 3, 5, 7, 9, 11, etc..

Parity bit P2 covers all data bits in positions whose binary representation has 1 in the second least significant position(010, 011, 110, 111, etc.). Thus P2 checks the bit in locations 2, 3, 6, 7, etc.

Parity bit P4 covers all data bits in positions whose binary representation has 1 in the third least significant position(100, 101, 110, 111, etc.). Thus P2 checks the bit in locations 4, 5, 6, 7, etc.

Example problem 1

Encode a binary word 11001 into the even parity hamming code.

Given, number of data bits, n =5.

***To find the number of redundant bits*,**

.

- $2^P >= n+P+1$

The above condition is true at P=4.

The equation is satisfied and so 4 redundant bits are selected.

So, total code bit = n+P = 9

The redundant bits are placed at bit positions 1, 2, 4 and 8.

Construct the bit location table.

Bit Location	9	8	7	6	5	4	3	2	1
Bit designation	D5	**P4**	D4	D3	D2	**P3**	D1	**P2**	**P1**
Binary representation	1001	1000	0111	0110	0101	0100	0011	0010	0001
Information bits	1		1	0	0		1		
Parity bits		1				1		0	1

To determine the parity bits

For P1: Bit locations 3, 5, 7 and 9 have three 1s. To have even parity, P1 must be 1.

For P2: Bit locations 3, 6, 7 have two 1s. To have even parity, P2 must be 0.

For P3: Bit locations 5, 6, 7 have one 1s. To have even parity, P3 must be 1.

For P4: Bit locations 8, 9 have one 1s. To have even parity, P2 must be 1.

Thus the encoded 9-bit hamming code is 111001101.

Advantages of Hamming code

- Hamming code method is effective on networks where the data streams are given for the single-bit errors.
- Hamming code not only provides the detection of a bit error but also helps you to indent bit containing error so that it can be corrected.
- The ease of use of hamming codes makes it best them suitable for use in computer memory and single-error correction.

Disadvantages of Hamming code

- Single-bit error detection and correction code. However, if multiple bits are founded error, then the outcome may result in another bit which should be correct to be changed. This can cause the data to be further errored.
- Hamming code algorithm can solve only single bits issues.

Application of Hamming code

- Satellites
- Computer Memory
- Modems
- PlasmaCAM
- Open connectors
- Shielding wire
- Embedded Processor

1.11 Alphanumeric codes

- A binary digit or bit can represent only two symbols as it has only two states '0' or '1'. But this is not enough for communication between two computers because there we need many more symbols for communication. These symbols are required to represent 26 alphabets with capital and small letters, numbers from 0 to 9, punctuation marks and other symbols.
- The alphanumeric codes are the codes that represent numbers and alphabetic characters. Mostly such codes also represent other characters such as symbol and various instructions necessary for conveying information.
- An alphanumeric code should at least represent 10 digits and 26 letters of alphabet i.e. total 36 items.
- The following three alphanumeric codes are very commonly used for the data representation.
  a)American Standard Code for Information Interchange (ASCII).
  b) Extended Binary Coded Decimal Interchange Code (EBCDIC).
  c) Five bit Baudot Code.
- ASCII code is a 7-bit code whereas EBCDIC is an 8-bit code. ASCII code is more commonly used worldwide while EBCDIC is used primarily in large IBM computers.

1.12 ASCII code

- The standard alphanumeric binary code is the ASCII which uses seven bits to code 128 characters.

ASCII - Binary Character Table

Letter	ASCII Code	Binary	Letter	ASCII Code	Binary
a	097	01100001	A	065	01000001
b	098	01100010	B	066	01000010
c	099	01100011	C	067	01000011
d	100	01100100	D	068	01000100
e	101	01100101	E	069	01000101
f	102	01100110	F	070	01000110
g	103	01100111	G	071	01000111
h	104	01101000	H	072	01001000
i	105	01101001	I	073	01001001
j	106	01101010	J	074	01001010
k	107	01101011	K	075	01001011
l	108	01101100	L	076	01001100
m	109	01101101	M	077	01001101
n	110	01101110	N	078	01001110
o	111	01101111	O	079	01001111

p	112	01110000	P	080	01010000
q	113	01110001	Q	081	01010001
r	114	01110010	R	082	01010010
s	115	01110011	S	083	01010011
t	116	01110100	T	084	01010100
u	117	01110101	U	085	01010101
v	118	01110110	V	086	01010110
w	119	01110111	W	087	01010111
x	120	01111000	X	088	01011000
y	121	01111001	Y	089	01011001
z	122	01111010	Z	090	01011010

1.13 Data types

The data types found in the registers of digital computers may be classified as being one of the following categories.
a) numbers used in arithmetic computations
b) letters of the alphabet used in data processing
c) other discrete symbols used for specific purposes
All types of data except binary numbers are represented in computers registers in binary-coded form.

1.14 Complements

- Complements are used in digital computers for simplifying the subtraction operation and for logical operations.
- There are two types of complements for each base r system: the r's complement and (r-1)'s complement.

  e.g

  When the value of the base r is substituted in the name, the two types are referred to as the 2's complement and 1's complement for binary numbers and the 10's and 9's complement for decimal numbers.

a)(r-1)'s complement

Given a number N in base r having n digits the (r-1)'s complement of N is defined as$(r^n-1)-N$.

b)(r's complement)

The r's complement of an n-digit number N in base r is defined as r^n-N for N!=0 and 0 for N=0;

c)9's complement

The 9's complement is used to find the subtraction of the decimal numbers. The 9's complement of a number is calculated by subtracting each digit of the number by 9.

example

suppose we have a number 1423, and we want to find the 9's complement of the number. For this, we subtract each digit of the number 1423 by 9. So, the 9's complement of the number 1423 is 9999-1423= 8576.

Subtraction using 9's complement

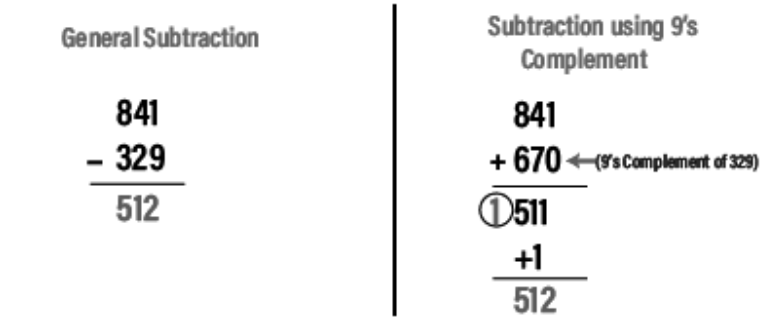
With the help of the 9's complement, the process of subtraction is done in a much easier way. Generally, we subtract the subtrahend from the minuend, but in a case when we perform subtraction using 9's complement, there is no need to do the same.

For subtracting two numbers using 9's complement, we first have to find the 9's complement of the subtrahend and then we will add this complement value with the minuend. There are two possible cases when we subtract the numbers using 9's complement.

Case 1: When the subtrahend is smaller than the minuend.

For subtracting the smaller number from the larger number using 9's complement, we will find the 9's complement of the subtrahend, and then we will add this complement value with the minuend. By adding both these values, the result will come in the formation of carry. At last, we will add this carry to the result obtained previously.

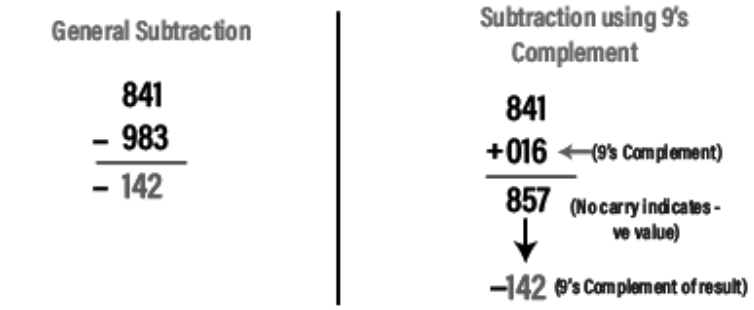
When subtrahend is smaller than the minuend



Case 2: When the subtrahend is greater than the minuend.

In this case, when we add the complement value and the minuend, the result will not come in the formation of carry. This indicates that the number is negative, and for finding the final result, we need to find the 9's complement of the result.

When subtrahend is greater than the minuend



d)10's Complement

The 10's complement is also used to find the subtraction of the decimal numbers. The 10's complement of a number is calculated by subtracting each digit by 9 and then adding 1 to the result. Simply, by adding 1 to its 9's complement we can get its 10's complement value.

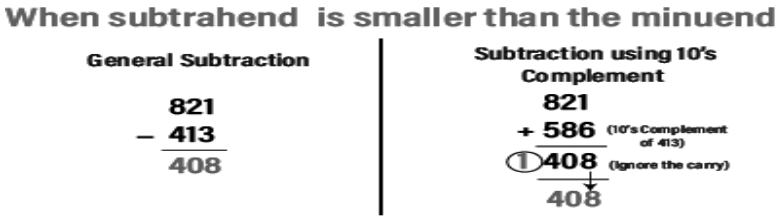
For example, suppose we have a number 1423, and we want to find the 10's complement of the number. For this, we find the 9's complement of the number 1423 that is 9999-1423= 8576, and now we will add 1 to the result. So the 10's complement of the number 1423 is 8576+1=8577.

Subtraction using 10's complement

For subtracting two numbers using 10's complement, we first have to find the 10's complement of the subtrahend, and then we will add this complement value with the minuend. There are two possible cases when we subtract the numbers using 10's complement.

Case 1: When the subtrahend is smaller than the minuend.

For subtracting the smaller number from the larger number using 10's complement, we will find the 10's complement of the subtrahend and then we will add this complement value with the minuend. By adding both these values, the result will come in the formation of carry. We ignore this carry and the remaining digits will be the answer.



Case 2: When the subtrahend is greater than the minuend.

In this case, when we add the complement value and the minuend, the result will not come in the formation of carry. This indicates that the number is negative and for finding the final result, we need to find the 10's complement of the result obtained by adding complement value of subtrahend and minuend.

325 **-641** **-316**	**325** **359 (358+1)** **684(no carry)** **-316(Applying 10's complement on 684)**

e) 1's complement

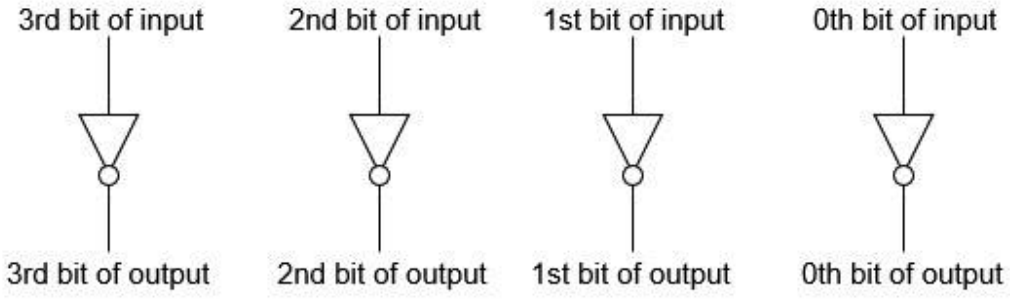
To get 1's complement of a binary number, simply invert the given number.

example,

1's complement of binary number 110010 is 001101.

1's Complement of a Binary Number:

There is a simple algorithm to convert a binary number into 1's complement. To get 1's complement of a binary number, simply invert the given number. You can simply implement logic circuit using only NOT gate for each bit of Binary number input. Implementation of logic circuit of 4-bit 1's complement is given as following below.



f) 2's complement

To get 2's complement of binary number is **1's complement** of given number **plus 1** to the least significant bit (LSB).

For example

2's complement of binary number 10010 is (01101) + 1 = 01110.

1.15 Fixed-point representation

- Positive numbers including zero can be represented as unsigned numbers.
- To represent negative integers we need a notation for negative values.
- The MSB bit contains the sign bit of the given number.
- The **sign bit** equal to **0** for **positive** and **1** for **negative.**

a)Integer representation

When an integer binary number is positive the sign is represented by 0 and the magnitude by a positive binary number.

When the number is negative the sign is represented by 1 but rest of the number may be represented in one of three possible ways.

i)Signed-magnitude representation

ii) Signed 1's complement representation

iii) Signed 2's complement representation

i)Signed-magnitude representation

- The signed-magnitude representation of -14 is obtained from +14 by **complementing only the sign bit.**
- **e.g**

  14- 00001110

  -14-10001110

ii) Signed 1's complement representation

- The signed 1's complement representation of -14 is obtained by complementing all bits of +14 including sign bit.
- **e.g**

  14- 00001110

  -14-11110001

iii) Signed 2's complement representation

- The signed 2's complement representation of -14 is obtained by adding 1 to the One's complement.
- **e.g**

	14-00001110
1's complement	-14-11110001
	+____ _1_
2's complement	11110010
	_____

b)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 small magnitude from the larger and give the result the sign of the larger magnitude.

+6 00000110 **+13** 00001101 _____ **+19** 00010011 _____	***-6*** *11111010* **+13** 00001101 _____ **+7** 00000111 _____
+6 00000110 **-13** 11110011 _____ **-7** 11111001 _____	**-6** 11111010 **-13** 11110011 _____ **-19** 11101101 _____-

In each of the four cases the operation performed is always addition including the sign bits. Any carry out of the sign bit position is discarded and negative results are automatically in 2's complement form.

c) Arithmetic Subtraction

- Subtraction of two signed binary numbers when negative numbers are in 2's complement form is very simple and can be stated as follows: Take the 2's complement of the subtrahend and add it to the minuend.
- But changing a positive number to a negative number is easily done by taking its 2's complement. The reverse is also true because the complement of a negative number in complement form produces the equivalent positive number.
- Consider the subtraction of(-6)-(-13)=+7.In binary with eights is written as (11111010)-(11110011).The subtraction is changed to addition by taking the 2's complement of the subtrahend (-13) to give (+13).In binary this is 11111010+00001101=100000111. Removing the end carry we obtain the correct answer 00000111(+7).

d)Overflow

- When the two numbers of n digits each are added and the sum occupies n+ 1 digit then an overflow occurred.
- An overflow cannot occur after an addition if one number is positive and other is negative, since adding a positive number to a negative number produces a result that is smaller than the larger of the two original numbers.
- An overflow may occur if the two numbers added are either positive or negative.

e.g

+70 **0** 1000110

+80 **0** 1010000

_____

+150 **1** 0010110

_____

The **carry** of 70 & 80 is **0**, the result 150 **carry** is **1.**Two carries are different.

Overflow detection

An overflow condition can be detected by observing the carry into the sign bit position and the carry out of the sign bit position. If these two carries are not equal an overflow condition is produced.

e) Decimal Fixed-Point Representation

- The representation of decimal numbers in registers is a function of the binary code used to represent a decimal digit.
- A 4-bit decimal code requires four flip-flops for each decimal digit. The representation of 4385 in BCD requires 16 flip-flops, four flip-flops for each digit. The number will be represented in a register with 16 flip-flops as follows:
  - 0100 0011 1000 0101

1.16 Floating-Point representation

The floating-point representation of a number has two parts. The first part represents a signed, fixed point number called the mantissa. The second part designates the position of the decimal point is called the exponent.

The fixed point mantissa may be fraction or an integer.

e.g

The decimal number +6132.789 is represented in floating-point with a fraction and exponent as follows.

Fraction	**Exponent**
0.6132789	+04

The value of the exponent indicates that the actual position of the decimal point is four positions to the right of the indicated decimal point in the fraction.

1.17 Basic theorems and properties of boolean algebra

Postulate 2	(a) x+0=x	(b) x.1=x
Postulate 5	(a) $x+x^1=1$	(b) $x.x^1=0$
Theorem 1	(a)x+x=x	(b) x.x=x
Theorem 2	(a) x+1=1	(b)x.0=0
Theorem 3,involution	(a) $(x^1)^1=x$	
Postulate 3,commutative	(a)x+y=y+x	(b) xy=yx
Theorem 4,associative	(a)x+(y+z)=(x+y)+z	(b)x(yz)=(xy)z
Postulate 4,distributive	(a)x(y+z)=xy+xz	(b)x+yz=(x+y)(x+z)
Theorem 5,DeMorgan	(a)$(x+y)^1=x^1y^1$	(b) $(xy)^1=x^1+y^1$
Theorem 6,absorption	(a)x+xy=x	(b)x(x+y)=x

Table: Postulates and theorems of Boolean algebra

Theorem 1(a): x+x=x

x+x=(x+x).1 by postulate: 2(b)

=(x+x).(x+ x^1) by postulate:5(a)

=x+xx^1 by postulate:4(b)

=x+0 by postulate:5(b)

=x by postulate:2(a)

Theorem 1(b): x.x=x

x.x=xx+0 by postulate:2(a)

=xx+xx^1 by postulate:5(b)

= x(x+ x^1) by postulate:4(a)

=x.1 by postulate:5(a)

=x by postulate:2(b)

Theorem 2(a) : x+1=x

x+1=1.(x+1) by postulate:2(b)

=(x+x^1)(x+1) by postulate:5(a)

=x+x^1.1 by postulate:4(b)

=x+x^1 by postulate:2(b)

=1 by postulate:5(a)

Theorem 2(b): x.0=0 by duality

Theorem 6(a): x+xy=x

x+xy=x.1+xy by postulate 2(b)

=x(1+y) by postulate:4(a)

=x(y+1) by postulate:3(a)

=x.1 by theorem 2(a)

=x by postulate:2(b)

1.18 Boolean Functions

A Boolean function is an expression formed with binary variables,the two binary operators OR and AND the unary operator NOT.

For a given value of the variables the function can be either 0 0r 1.Consider for example the Boolean function

$F_1=xyz^1$

The function F_1 is equal to 1 if x=1 and y=1 and z^1=1; otherwise F_1=0.The above is an example of a Boolean function represented as an algebraic expression.

Consider now the function

$F_2=x+y^1z$

F2=1 if x=1 or if y=0 while z=1 otherwise F_2=0.

Consider the function

$F_3=x^1y^1z+x^1yz+xy^1$

Consider the function

$F4=xy^1+x^1z$

x	y	z	F_1	F_2	F_3	F_4
0	0	0	0	0	0	0
0	0	1	0	1	1	1
0	1	0	0	0	0	0
0	1	1	0	0	1	1
1	0	0	0	1	1	1
1	0	1	0	1	1	1
1	1	0	1	1	0	0
1	1	1	0	1	0	0

Table:Truth tables for F_1 F_2 F_3 F_4

1.19 Canonical and standard forms

- A binary variable may appear either in its normal form(x) or in its complement form(x^1**).**
- There are two ways in which we can put the Boolean function. These ways are the minterm canonical form and maxterm canonical form.

a)Literal

- A Literal signifies the Boolean variables including their complements. Such as B is a boolean variable and its complements are ~B or B', which are the literals.

b) Minterm

- The product of all literals, either with complement or without complement, is known as **minterm**.
- Now consider two binary variables x and y combined with an AND operation. Since each variable may appear in either form there are four possible combinations x^1y^1, x^1y,x $y^1$and xy.

Minterm from values

Using variable values, we can write the minterms as:

- If the variable value is 1, we will take the variable without its complement.
- If the variable value is 0, take its complement.

Example

- Let's assume that we have three Boolean variables A, B, and C having values A=1,B=0,c=0
- Now, we will take the complement of the variables B and C because these values are 0 and will take A without complement. So, the minterm will be:
  Minterm=A.B'C'

c)Maxterm

- The sum of all literals, either with complement or without complement, is known as **maxterm**.
- Now consider two binary variables x and y combined with an OR operation. Since each variable may appear in either form there are four possible combinations x^1+y^1, x^1+y,$x+y^1$and x+y.

Maxterm from values

Using the given variable values, we can write the maxterm as:

- If the variable value is 1, then we will take the variable without a complement.
- If the variable value is 0, take the complement of the variable.

Example

- Let's assume that we have three Boolean variables A, B, and C having values A=1,B=0,c=0
- Now, we will take the complement of the variables B and C because these values are 0 and will take A without complement. So, the maxterm will be:
  Maxterm=A+B'+C'

d) Standard forms

There are two types of standard forms

i)The sum of products

ii) The product of sums

i)The sum of products

The sum of products is a boolean expression containing AND terms called product terms,of one or more literals each.The sum denotes the ORing of these terms.

e.g

$F_1=y^1+xy+x^1yz^1$

The expression has three product terms of one, two and three literals each respectively. Their sum is in effect an OR operation.

ii)The product of sums

A product of sums is a Boolean expression containing OR terms called sum terms. Each term may have any number of literals. The product denotes the ANDing of these terms.

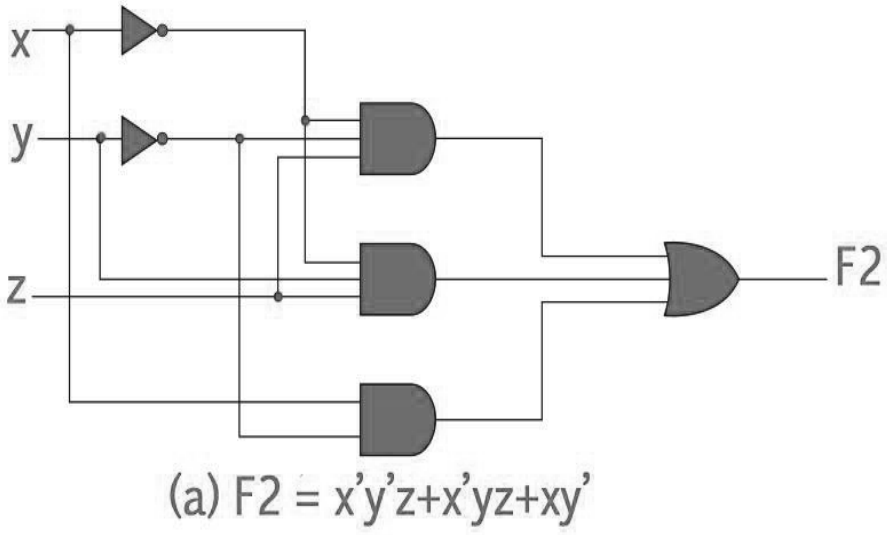
$F2=x(y^1+z)(x^1+y+z^1+w)$

The above expression has three sum terms of one, two, and four literals each. The product is an AND operation.

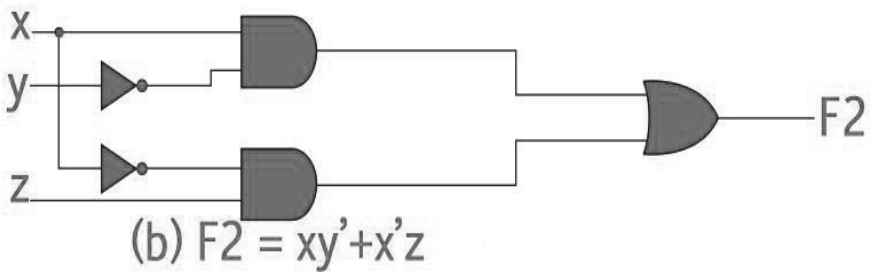
1.20 Minimization of Boolean functions using algebraic identities

The process of simplifying the algebraic expression of a boolean function is called **minimization**. Minimization is important since it reduces the cost and complexity of the associated circuit.

For example, the function $F=x^1y^1z+x^1yz+xy^1$ can be minimized to $F=x^1z+xy^1$. The circuits associated with above expressions is –



(a) F2 = x'y'z+x'yz+xy'



(b) F2 = xy'+x'z

It is clear from the above image that the minimized version of the expression takes a less number of logic gates and also reduces the complexity of the circuit substantially. Minimization is hence important to find the most economic equivalent representation of a boolean function.

Minimization can be done using Algebraic Manipulation or K-Map method. Each method has it's own merits and demerits.

a)Minimization using Algebraic Manipulation –

This method is the simplest of all methods used for minimization. It is suitable for medium sized expressions involving 4 or 5 variables. Algebraic manipulation is a

manual method, hence it is prone to human error.
Common Laws used in algebraic manipulation :

i. $A+A^1=1$
ii. $A+ A^1B=A+B$
iii. $A+AB=A$

e.g

$CD+AB^1C+ABC^1+BCD$
$=CD+BCD+AB^1C+ABC^1$ (A+AB=A)
$=CD++AB^1C+ABC^1$

1.21 Karnaugh map representation

- In many digital circuits and practical problems we need to find expression with minimum variables.
- We can minimize Boolean expressions of 3, 4 variables very easily using K-map without using any Boolean algebra theorems.
- K-map can take two forms Sum of Product (SOP) and Product of Sum (POS) according to the need of problem.
- K-map is table like representation but it gives more information than TRUTH TABLE. We fill grid of K-map with 0's and 1's then solve it by making groups.

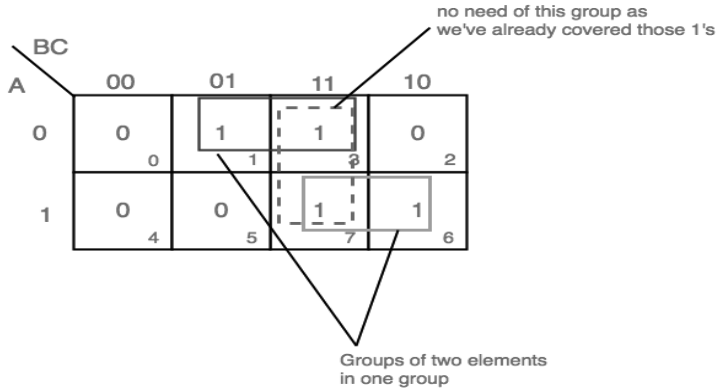
Steps to solve expression using K-map-

- Select K-map according to the number of variables.
- Identify minterms or maxterms as given in problem.
- For SOP put 1's in blocks of K-map respective to the minterms.
- For POS put 0's in blocks of K-map respective to the maxterms.
- Make rectangular groups containing total terms in power of two like 2,4,8 ..(except 1) and try to cover as many elements as we can in one group.
- From the groups made in step 5 find the product terms and sum them up for SOP form.

SOP FORM

1. **K-map of 3 variables-**

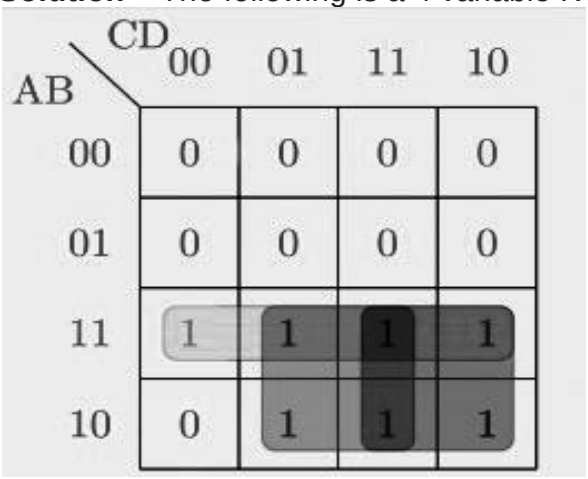
Z= ∑A,B,C(1,3,6,7)



From **red** group we get product term— A'C
From **green** group we get product term—AB
Summing these product terms we get- **Final expression (A'C+AB)**

Minimization using K-Map –
The Algebraic manipulation method is tedious and cumbersome. The K-Map method is faster and can be used to solve boolean functions of upto 5 variables.

- **Example 2 –** Consider the same expression from example-1 and minimize it using K-Map.
  - **Solution –** The following is a 4 variable K-Map of the given expression.



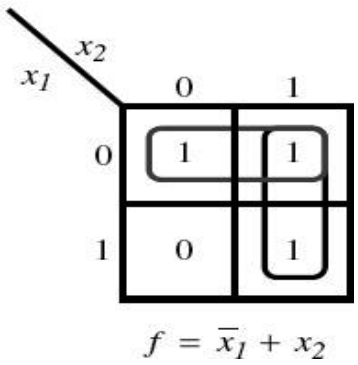
The above figure highlights the prime implicants in green, red and blue.
The green one spans the whole third row, which gives us – AB
The red one spans 4 squares, which gives us – AD
The blue one spans 4 squares, which gives us – AC
So, the minimized boolean expression is- AB+AC+AD

1.22 Two variable MAPS

m0	**m1**
m2	**m3**

Fig(a) Two-variable map

x_1	x_2	f
0	0	1
0	1	1
1	0	0
1	1	1



Fig(b):Representation of function in the map

There are four minterms for two variables; hence the map consists of four squares, one for each minterm.This is represented in fig(a).

The fig(b) shows representation of the function in map.

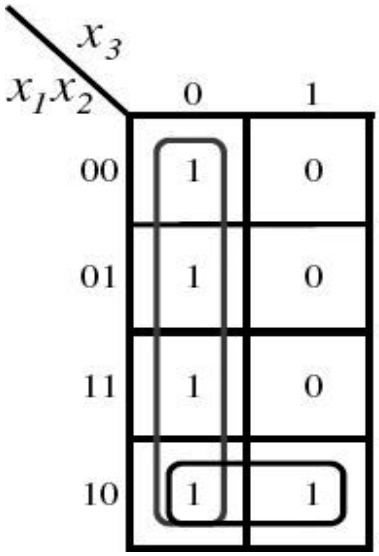
The 0's and 1's marked for each row and each column designate the values of variables x_1 and x_2 respectively.

x_1 appears primed in row 0 and unprimed in row 1. x_2 appears primed in column 0 and unprimed in column1.

1.23 Three variable MAPS

m0	m1	m2	m3
m4	m5	m6	m7

Fig(a)-Three-variable map



$$f = \overline{x}_3 + x_1\overline{x}_2$$

Fig(b)- Representation of function in the map

A three variable map is shown in Fig(a).There are eight minterms for three variables. Therefore a map consists of eight squares.
The fig(b) shows representation of the function in map.

1.24 Universal gates

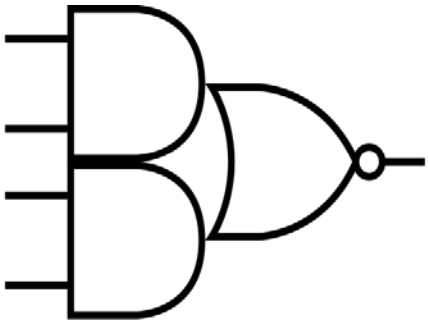
- A **universal gate** is a logic gate which can implement any Boolean function without the need to use any other type of logic gate. The NOR gate and NAND gate are universal gates. This means that we can create any logical Boolean expression using only NOR gates or only NAND gates.
- In practice, this is advantageous since NOR and NAND gates are economical and easier to fabricate than other logic gates.
- An AND gate is typically implemented as a NAND gate followed by an inverter. Similarly, an OR gate is typically realised as a NOR gate followed by an inverter.

1.25 Two-level realization using gates

a)AND-OR-INVERT implementation

The two forms NAND-AND and AND-NOR are equivalent form and can be treated together. Both perform the AND-OR-INVERT function.
AOI gates are two-level compound logic functions constructed from the combination of one or more AND gates followed by a NOR gate. Construction of AOI cells is

particularly efficient using CMOS technology where the total number of transistor gates can be compared to the same construction using NAND logic or NOR logic.



.

b) OR-AND- INVERT implementation

- The OR-NAND and NOR-OR forms perform the OR-AND-INVERT function.
- The OR-NAND form resembles the OR-AND form except for the inversion done by the circle in the NAND gate.
- It implements the function : $F=[(A+B)(C+D)E]^1$