Department of Electronics and Communication Engineering

EMPOWERING THROUGH INNOVATIONS
MLRS

MARRI LAXMAN REDDY
INSTITUTE OF TECHNOLOGY AND MANAGEMENT
(AN AUTONOMOUS INSTITUTION)

(Approved by AICTE, New Delhi & Affiliated to JNTUH, Hyderabad)

Accredited by NBA and NAAC with 'A' Grade & Recognized Under Section2(f) & 12(B)of the UGC act,1956

DIGITAL LOGIC DESIGN AND COMPUTER ORGANIZATION

- **UNIT-I: Basic Structure of Computers**

Presentation By

Dr. Bapi Debnath

Assoc. Prof.

Department of ECE

MARRI LAXMAN REDDY

INSTITUTE OF TECHNOLOGY AND MANAGEMENT

(AN AUTONOMOUS INSTITUTION)

(Approved by AICTE, New Delhi & Affiliated to JNTUH, Hyderabad)

Accredited by NBA and NAAC with 'A' Grade & Recognized Under Section2(f) & 12(B)of the UGC act,1956

Basic Operational Concepts in Computer Organization

A Computer has five functional independent units like Input Unit, Memory Unit, Arithmetic & Logic Unit, Output Unit, Control Unit.

Input Unit:-

Computers take coded information via input unit. The most famous input device is keyboard. Whenever we press any key it is automatically being translated to corresponding binary code & transmitted over a cable to memory or processor.

Memory Unit:-

It stores programs as well as data and there are two types- Primary and Secondary Memory

Primary Memory is quite fast which works at electronic speed. Programs should be stored in memory before getting executed. Random Access Memory are those memory in which location can be accessed in a shorter period of time after specifying the address. Primary memory is essential but expensive so we went for secondary memory which is quite cheaper. It is used when large amount of data & programs are needed to store, particularly the information that we don't access very frequently. Ex- Magnetic Disks, Tapes

Arithmetic & Logic Unit:-

All the arithmetic & Logical operations are performed by ALU and this operation are initiated once the operands are brought into the processor.

Output Unit: – It displays the processed result to outside world.

Functional Units

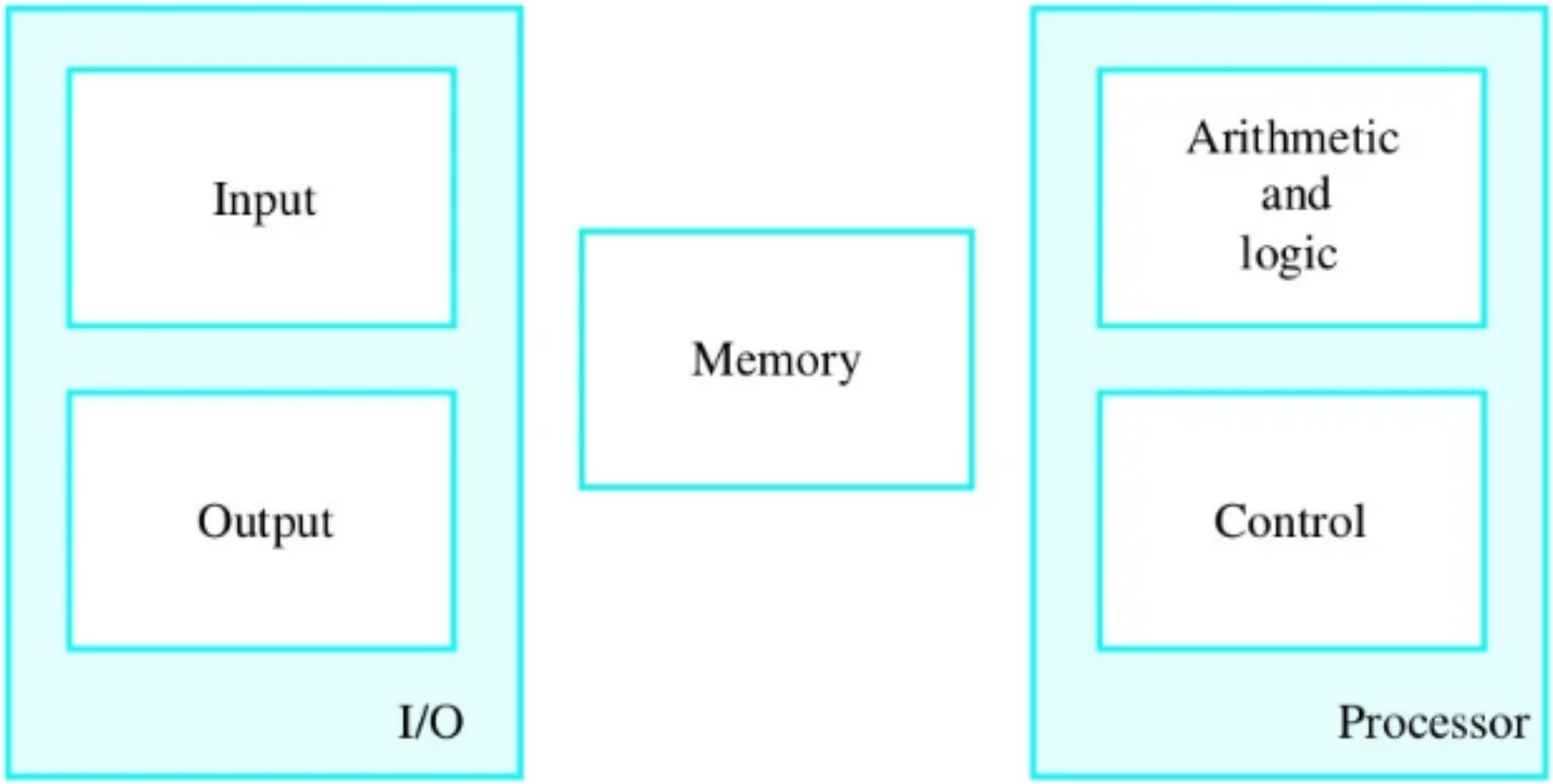


Figure : Basic functional units of a computer.

Basic Operational Concepts

- Instructions take a vital role for the proper working of the computer.
- An appropriate program consisting of a list of instructions is stored in the memory so that the tasks can be started.
- The memory brings the Individual instructions into the processor, which executes the specified operations.
- Data which is to be used as operands are moreover also stored in the memory.

Example:

A Typical Instruction:

Example: **Add R0, LOCA** // **R0 ← R0 + M [LOCA]**

- This instruction **adds** the **operand at memory location LOCA**, to **operand in register R0 in the processor** & **places the sum** into **register**.
  - The original content of LOCA is preserved and the original content of R0 is overwritten
- This **instruction** requires following steps,
  - 1st the instruction is fetched from the memory into the processor.
  - 2nd the operand at LOCA is fetched and added to the contents of R0
  - 3rd the resulting sum is stored in the register R0
- The preceding **add** instruction combines a ***Memory access operation*** with an ***ALU operations.***

➢ **An alternative way to implement the instruction:**

Separate ***memory access operation*** with an ***ALU operations.***

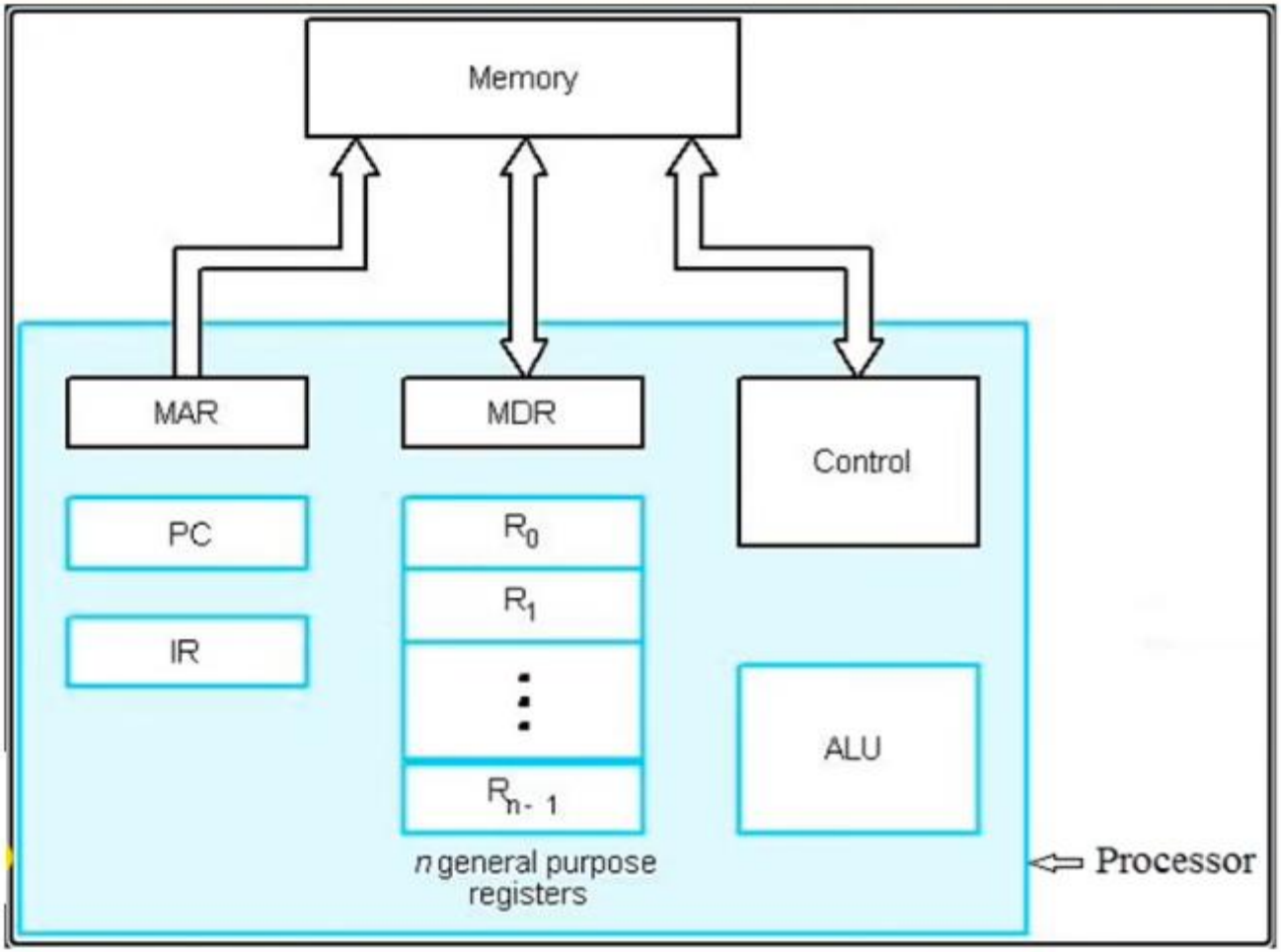
In some modern computers, the same task is performed using two separate instructions for performance reasons.

Load R1, LOCA // **R1 ← M [LOCA]**

Add R0, R1 // **R0 ← R0 + R1**

Bus structures in Computer Organization

Connection between the Processor and the Memory



Processor Contains:

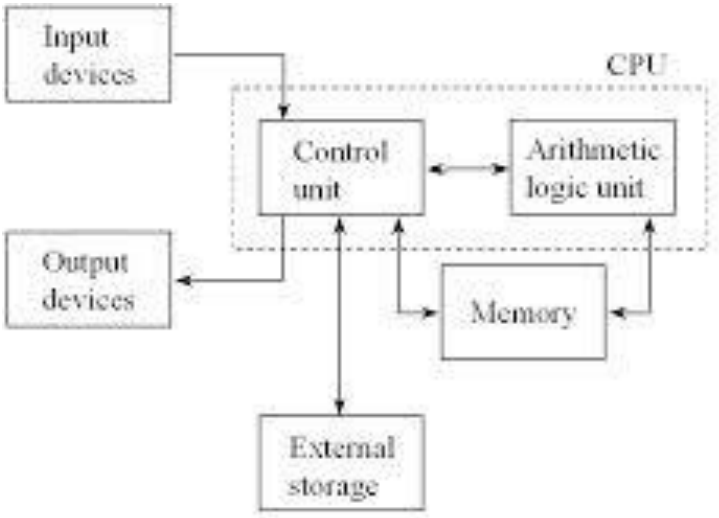
1. ALU (Arithmetic and Logic Unit)
2. Control Unit
3. Registers

Analyzing how processor and memory are connected :–

- Processors have various registers to perform various functions :-
- Program Counter: - It contains the memory address of next instruction to be fetched.
- Instruction Register:- It holds the instruction which is currently being executed.
- MDR: - It facilities communication with memory. It contains the data to be written into or read out of the addressed location.
- MAR :- It holds the address of the location that is to be accessed
- There are n general purpose registers that is R0 to Rn-1

Performance:-

- Performance means how quickly a program can be executed.



Computer organization

- In order to get the best performance it is required to design the compiler, machine instruction set & hardware in a coordinated manner.

Connection B / W Processor & Memory

Connection B/W Processor & Memory

- The above mentioned block diagram consists of the following components

1) Memory
2) MAR
3) MDR
4) PC
5) IR
6) General Purpose Registers
7) Control Unit
8) ALU

- The instruction that is currently being executed is held by the Instruction Register.
- IR output is available to the control circuits, which generates the timing signal that control the various processing elements involved in executing the instruction.
- The Memory address of the next instruction to be fetched and executed is contained by the Program Counter.
- It is a specialized register.
- It keeps the record of the programs that are executed.
- Role of these registers is to handle the data available in the instructions. They store the data temporarily.
- Two registers facilitate the communication with memory. These registers are:

9) MAR (Memory Address Register)
10) MDR (Memory Data Register)

Memory Address Register:

- The address of the location to be accessed is held by MAR. Memory

Data Register:

- It contains the data to be written into or to be read out of the addressed location.

Working Explanation

A **PC** is set to point to the first instruction of the program. The contents of the **PC** are transferred to the **MAR** and a Read control signal is sent to the memory. The addressed word is fetched from the location which is mentioned in the **MAR** and loaded into **MDR**. This post thus contains all the important basic operational concepts.

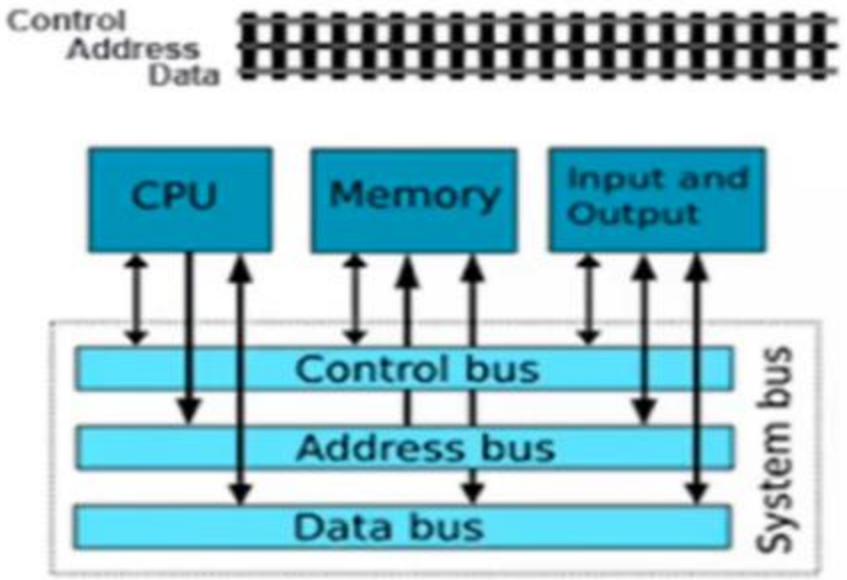
2. Bus Structures:

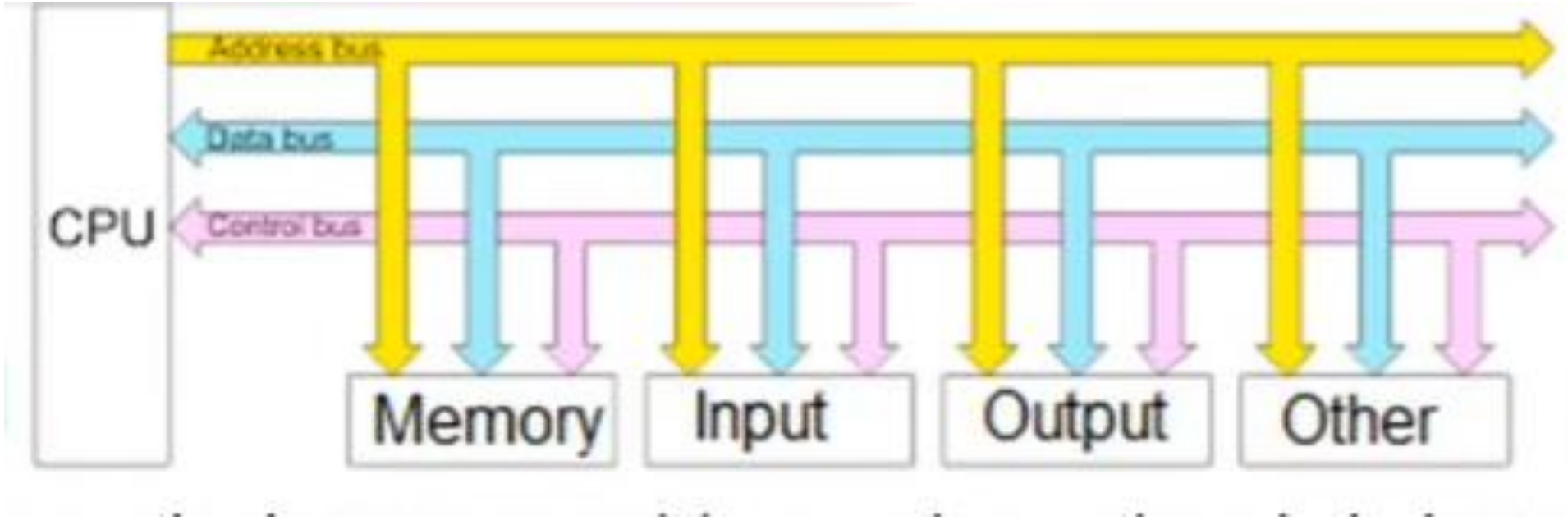
A Bus is a collection of wires that connects several devices.

Buses are used to send control signals and data between the processor and other components This is to achieve a reasonable speed of operation.

In computer system all the peripherals are connected to microprocessor through Bus. Types of Bus structure:

1. Address bus
2. Data bus
3. Control bus





Types of Buses in Computer Architecture

1.Address Bus:

1. Address bus carry the memory address while reading from writing into memory.
2. Address bus carry I/O post address or device address from I/O port.
3. In uni-directional address bus only the CPU could send address and other units could not address the microprocessor.
4. Now a days computers are having bi-directional address bus.

2. Data Bus:

1. Data bus carry the data.
2. Data bus is a bidirectional bus.
3. Data bus fetch the instructions from memory.
4. Data bus used to store the result of an instruction into memory.
5. Data bus carry commands to an I/O device controller or port.
6. Data bus carry data from a device controller or port.
7. Data bus issue data to a device controller or port.

3. Control Bus:

Different types of control signals are used in a bus:

1. Memory Read: This signal, is issued by the CPU or DMA controller when performing a read operation with the memory.
2. Memory Write: This signal is issued by the CPU or DMA controller when performing a write operation with the memory.
3. I/O Read: This signal is issued by the CPU when it is reading from an input port.
4. I/O Write: This signal is issued by the CPU when writing into an output port.
5. Ready: The ready is an input signal to the CPU generated in order to synchronize the show memory or I/O ports with the fast CPU.

A **system bus** is a single computer bus that connects the major components of a computer system, combining the functions of a data bus to carry information, an address bus to determine where it should be sent, and a control bus to determine its operation.

Software Performance

What is meant by software?

Software is a set of instructions, data or programs used to operate computers and execute specific tasks. It is the opposite of hardware, which describes the physical aspects of a computer. Software is a generic term used to refer to applications, scripts and programs that run on a device. It can be thought of as the variable part of a computer, while hardware is the invariable part.

The two main categories of software are **application software** and **system software.** An application is software that fulfills a specific need or performs tasks. System software is designed to run a computer's hardware and provides a platform for applications to run on top of.

Other types of software include programming software, which provides the programming tools software developers need; middleware, which sits between system software and applications; and driver software, which operates computer devices and peripherals.

Early software was written for specific computers and sold with the hardware it ran on. In the 1980s, software began to be sold on floppy disks, and later on CDs and DVDs. Today, most software is purchased and directly downloaded over the internet. Software can be found on vendor websites or application service provider websites.

Examples and types of software

Among the various categories of software, the most common types include the following:

- **Application software.** The most common type of software, application software is a computer software package that performs a specific function for a user, or in some cases, for another application. An application can be self-contained, or it can be a group of programs that run the application for the user. Examples of modern applications include office suites, graphics software, databases and database management programs, web browsers, word processors, software development tools, image editors and communication platforms.
- **System software.** These software programs are designed to run a computer's application programs and hardware. System software coordinates the activities and functions of the hardware and software. In addition, it controls the operations of the computer hardware and provides an environment or platform for all the other types of software to work in. The OS is the best example of system software; it manages all the other computer programs. Other examples of system software include the firmware, computer language translators and system utilities.
- **Driver software.** Also known as device drivers, this software is often considered a type of system software. Device drivers control the devices and peripherals connected to a computer, enabling them to perform their specific tasks. Every device that is connected to a computer needs at least one device driver to function. Examples include software that comes with any nonstandard hardware, including special game controllers, as well as the software that enables standard hardware, such as USB storage devices, keyboards, headphones and printers.

- **Middleware.** The term *middleware* describes software that mediates between application and system software or between two different kinds of application software. For example, middleware enables Microsoft Windows to talk to Excel and Word. It is also used to send a remote work request from an application in a computer that has one kind of OS, to an application in a computer with a different OS. It also enables newer applications to work with legacy ones.
- **Programming software.** Computer programmers use programming software to write code. Programming software and programming tools enable developers to develop, write, test and debug other software programs. Examples of programming software include assemblers, compilers, debuggers and interpreters.

Computer performance is the amount of work accomplished by a computer system. The word performance in computer performance means “How well is the computer doing the work it is supposed to do?”. It basically depends on response time, throughput and execution time of a computer system.

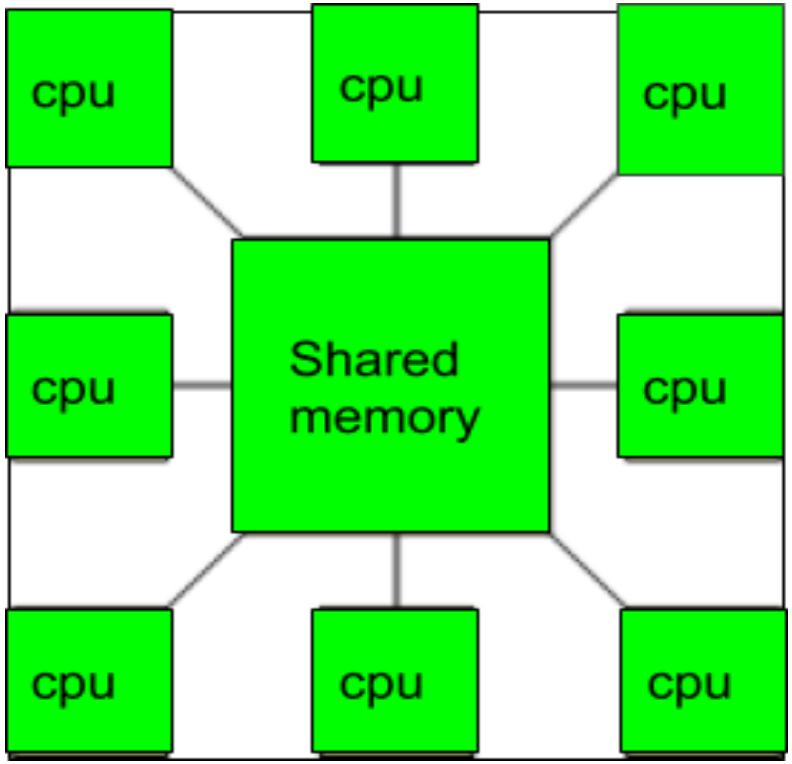
Response time is the time from start to completion of a task. This also includes:
Operating system overhead.
Waiting for I/O and other processes
Accessing disk and memory
Time spent executing on the CPU or execution time.

Multiprocessors and Multi computers

1. Multiprocessor:

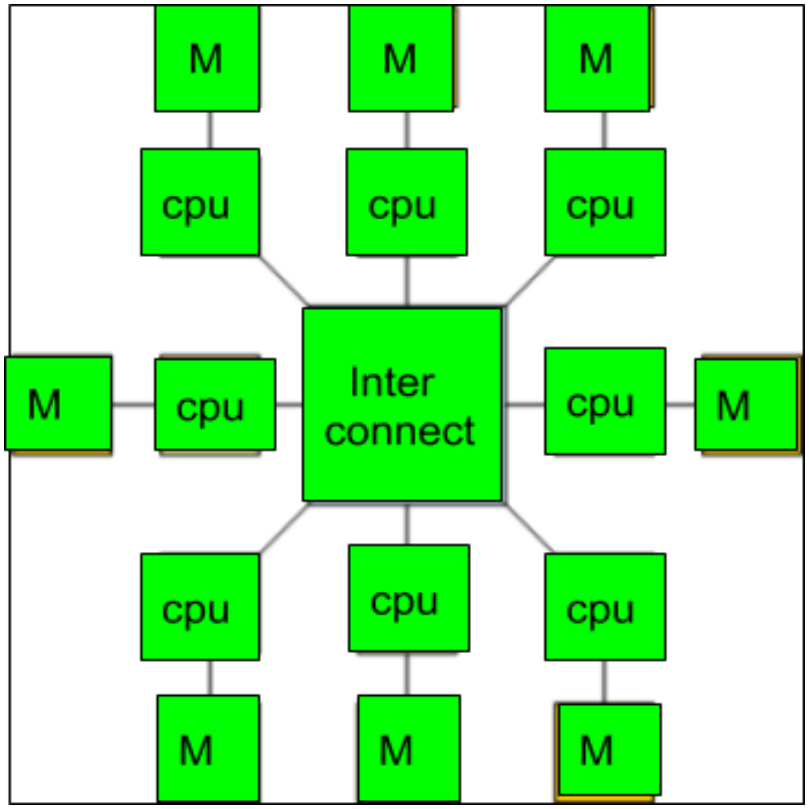
A Multiprocessor is a computer system with two or more central processing units (CPUs) share full access to a common RAM. The main objective of using a multiprocessor is to boost the system's execution speed, with other objectives being fault tolerance and application matching.

There are two types of multiprocessors, one is called shared memory multiprocessor and another is distributed memory multiprocessor. In shared memory multiprocessors, all the CPUs shares the common memory but in a distributed memory multiprocessor, every CPU has its own private memory.



Multicomputer:

A multicomputer system is a computer system with multiple processors that are connected together to solve a problem. Each processor has its own memory and it is accessible by that particular processor and those processors can communicate with each other via an interconnection network.



Historical Prospective of Computers:

The history of the computer dates back to several years. There are five prominent generations of computers. Each generation has witnessed several technological advances which change the functionality of the computers. This results in more compact, powerful, robust systems which are less expensive. The brief history of computers is discussed −

Historical perspective

Early History of Computer:

Since the evolution of humans, devices have been used for calculations for thousands of years. One of the earliest and most well-known devices was an abacus. Then in 1822, the father of computers, Charles Babbage began developing what would be the first mechanical computer. And then in 1833 he actually designed an Analytical Engine which was a general-purpose computer. It contained an ALU, some basic flow chart principles and the concept of integrated memory.

Then more than a century later in the history of computers, we got our first electronic computer for general purpose. It was the ENIAC, which stands for Electronic Numerical Integrator and Computer. The inventors of this computer were John W. Mauchly and J.Presper Eckert.

And with times the technology developed and the computers got smaller and the processing got faster. We got our first laptop in 1981 and it was introduced by Adam Osborne and EPSON.

Historical perspective

First Generation (1940-1956)

The first generation computers had the following features and components –

Hardware

The hardware used in the first generation of computers was: **Vacuum Tubes** and **Punch Cards**.

Features

Following are the features of first generation computers –

- It supported machine language.
- It had slow performance
- It occupied large size due to the use of vacuum tubes.
- It had a poor storage capacity.
- It consumed a lot of electricity and generated a lot of heat.

Memory

The memory was of 4000 bits.

Data Input

The input was only provided through hard-wired programs in the computer, mostly through punched cards and paper tapes.

Examples

The examples of first-generation computers are −

- ENIAC
- UNIVACTBM 701

Second Generation (1956-1963)

Several advancements in the first-gen computers led to the development of second-generation computers. Following are various changes in features and components of second-generation computers −

Hardware

The hardware used in the second generation of computers were −

- Transistors
- Magnetic Tapes

Features

It had features like −

- Batch operating system
- Faster and smaller in size
- Reliable and energy efficient than the previous generation
- Less costly than the previous generation

Memory

The capacity of the memory was 32,000 bits. Data Input

The input was provided through punched cards.

Examples

The examples of second generation computers are −

- Honeywell 400
- CDC 1604
- IBM 7030

Third Generation (1964-1971)

Following are the various components and features of the third generation computers −

Hardware

The hardware used in the third generation of computers were −

- Integrated Circuits made from semi-conductor materials
- Large capacity disks and magnetic tapes

Features

The features of the third generation computers are −

- Supports time-sharing OS
- Faster, smaller, more reliable and cheaper than the previous generations
- Easy to access

Memory

The capacity of the memory was 128,000 bits. Data Input

The input was provided through keyboards and monitors.

Examples

The examples of third generation computers are −

- IBM 360/370
- CDC 6600
- PDP 8/11

Fourth Generation (1972-2010)

Fourth generation computers have the following components and features −

Hardware

The Hardware used in the fourth generation of computers were −

- ICs with Very Large Scale Integration (VLSI) technology
- Semiconductor memory
- Magnetic tapes and Floppy

Features

It supports features like −

- Multiprocessing & distributed OS
- Object-oriented high level programs supported
- Small & easy to use; hand-held computers have evolved
- No external cooling required & affordable
- This generation saw the development of networks and the internet
- It saw the development of new trends in GUIs and mouse

Memory

The capacity of the memory was 100 million bits.

Data Input

The input was provided through improved hand held devices, keyboard and mouse.

Examples

The examples of fourth generation computers are −

- Apple II
- VAX 9000
- CRAY 1 (super computers)

Fifth Generation (2010-Present)

These are the modern and advanced computers. Significant changes in the components and operations have made fifth generation computers handy and more reliable than the previous generations.

Hardware

The Hardware used in the fifth generation of computers are −

- Integrated Circuits with VLSI and Nano technology
- Large capacity hard disk with RAID support
- Powerful servers, Internet, Cluster computing

Features

It supports features like −

- Powerful, cheap, reliable and easy to use.
- Portable and faster due to use of parallel processors and Super Large Scale Integrated Circuits.
- Rapid software development is possible.

Memory

The capacity of the memory is unlimited. Data

Input

The input is provided through CDROM, Optical Disk and other touch and voice sensitive input devices.

Examples

The examples of fifth generation computers are −

- IBM
- Pentium
- PARAM

Department of Electronics and Communication Engineering

EMPOWERING THROUGH INNOVATIONS
MLRS

MARRI LAXMAN REDDY
INSTITUTE OF TECHNOLOGY AND MANAGEMENT
(AN AUTONOMOUS INSTITUTION)

(Approved by AICTE, New Delhi & Affiliated to JNTUH, Hyderabad)

Accredited by NBA and NAAC with 'A' Grade & Recognized Under Section2(f) & 12(B)of the UGC act,1956

DIGITAL LOGIC DESIGN AND COMPUTER ORGANIZATION

- **UNIT - I: Number Systems**

Presentation By

Dr. Bapi Debnath

Assoc. Prof.

Department of ECE

MARRI LAXMAN REDDY

INSTITUTE OF TECHNOLOGY AND MANAGEMENT

(AN AUTONOMOUS INSTITUTION)

(Approved by AICTE, New Delhi & Affiliated to JNTUH, Hyderabad)

Accredited by NBA and NAAC with 'A' Grade & Recognized Under Section2(f) & 12(B)of the UGC act,1956

Number Systems

- Number systems

Binary	**0,1**	**2**
Octal	**0,1,….7**	**8**
Decimal	**0,1,….9**	**10**
Hex-Dec	**0,1…9,A,B…..F**	**16**

- Conversions of No. Systems
  1. Decimal to other
  2. Other to Decimal
  3. Octal to Hex
  4. Hex to Octal

- Complements
  - Addition
  - Subtraction
  - 1's & 2's Comp.
  - 9's & 10's Comp.
- Binary Codes
  - Weighted
    - BCD
  - Non-Weighted
    - Ex-3,Gray
  - Alphanumeric
    - ASCII,EBCDIC
  - Error detecting & Correcting
    - Parity, Hamming

WHAT IS NUMBER SYSTEM…?

- The **number system** that we use in our day-to-day life is the decimal **number system**. Decimal **number system** has base 10 as it uses 10 digits from 0 to 9. In decimal **number system**, the successive positions to the left of the decimal point represent units, tens, hundreds, thousands and so on.

- Many number systems are in use in digital technology. The most common are :

Decimal	(Base 10)
Binary	(Base 2)
Octal	(Base 8)
Hexadecimal	(Base 16)

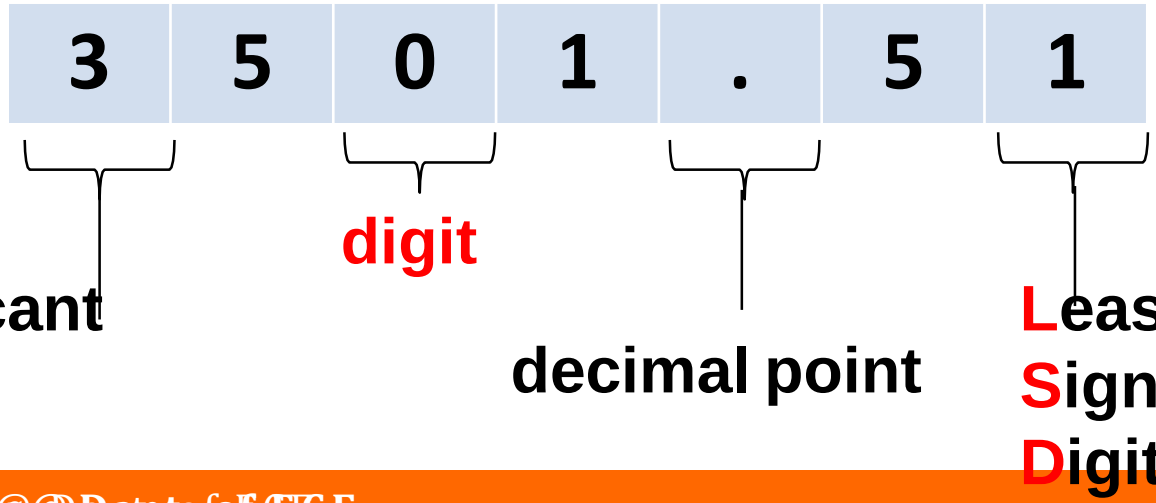
- The ***decimal system*** is the number system that we use everyday

- **Decimal system** uses symbols ***(digits)*** for ***the ten values*** *0, 1, 2, 3, 4, 5, 6, 7, 8, 9*
- **Binary System** uses digits for ***the two values*** *0,* and *1*
- **Octal System** uses digits for ***the eight values*** *0, 1, 2, 3, 4, 5, 6, 7*
- **Hexadecimal System** uses digits for ***the sixteen values*** *0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F*

to represent any number, no matter how large or how small.

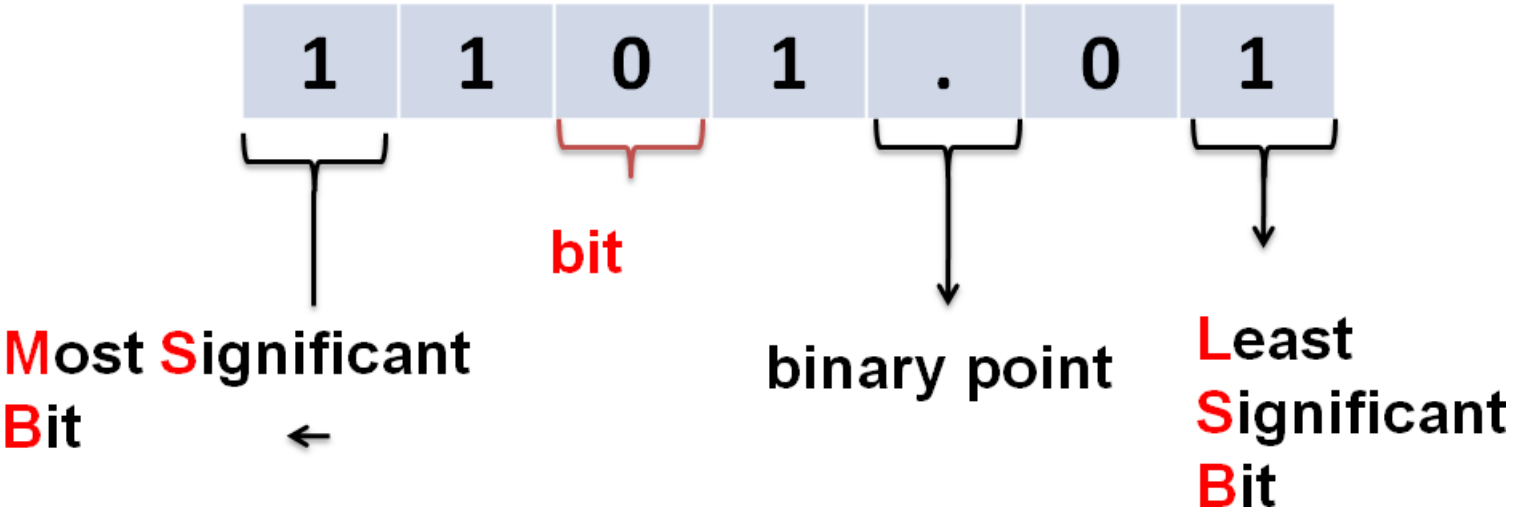
- **The decimal system** is composed of 10 numerals or symbols. These 10 symbols are 0,1,2,3,4,5,6,7,8,9; using these symbols as digits of a number, we can express any quantity.

  Example : 3501.51



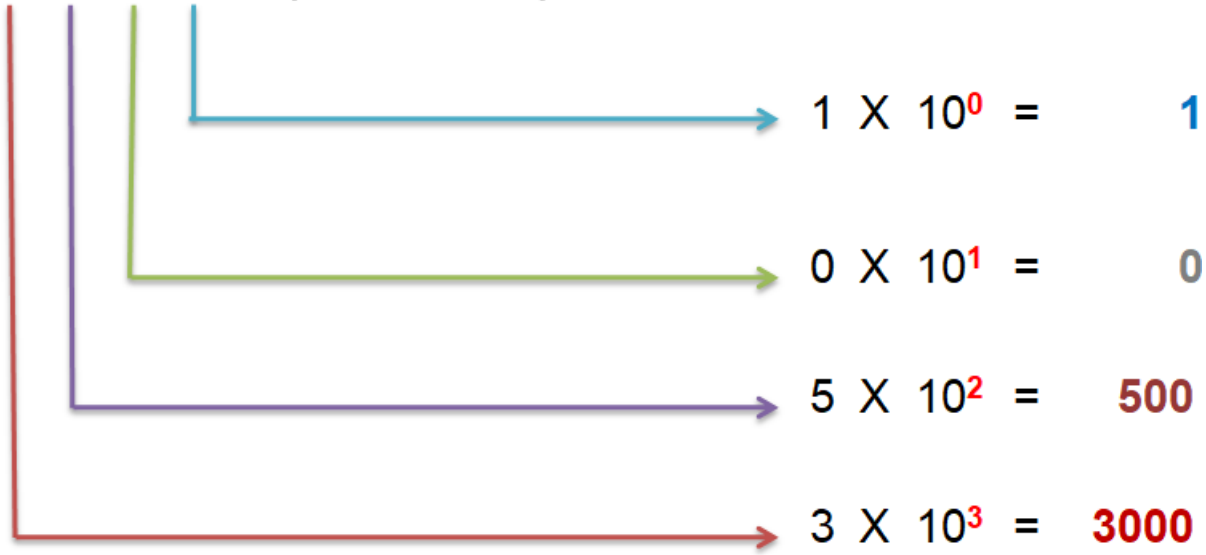
Binary System

- **The binary system** is composed of 2 numerals or symbols 0 and 1; using these symbols as digits of a number, we can express any quantity.
- **Example : 1101.01**



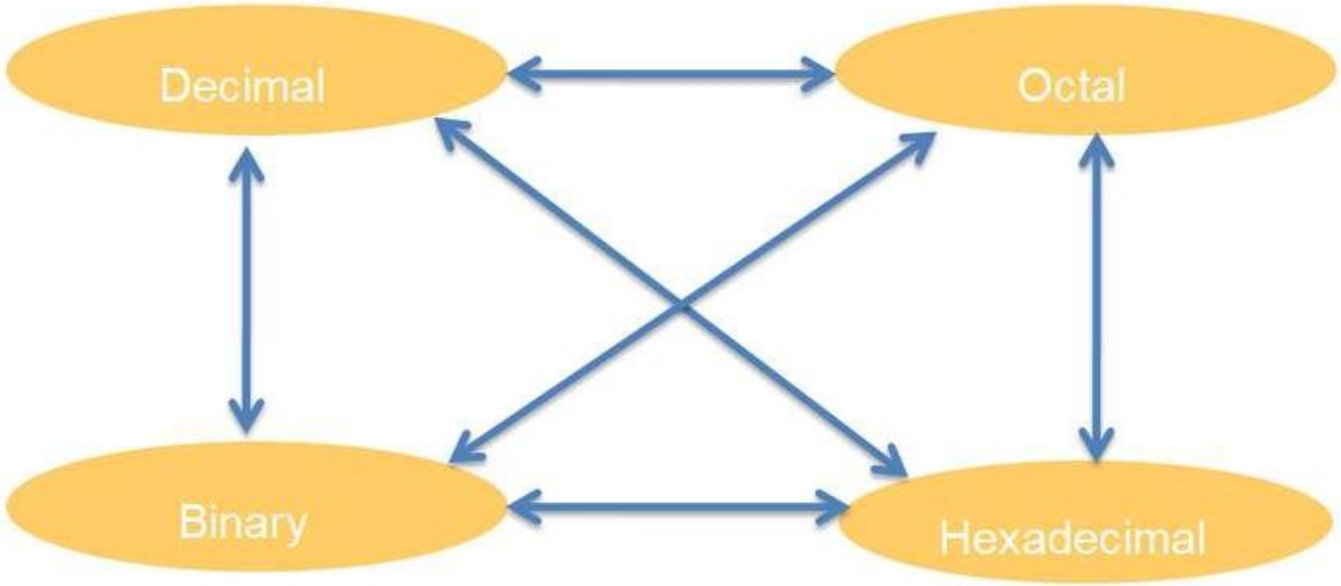
Decimal Number Quantity

- 3 5 0 1 (base-10)



3000 + 500 + 0 + 1 = 3501

CONVERSION AMONG BASES



Decimal to Other Base System

Steps

- **Step 1** – Divide the decimal number to be converted by the value of the new base.
- **Step 2** – Get the remainder from Step 1 as the rightmost digit (least significant digit) of new base number.
- **Step 3** – Divide the quotient of the previous divide by the new base.
- **Step 4** – Record the remainder from Step 3 as the next digit (to the left) of the new base number.

Repeat Steps 3 and 4, getting remainders from right to left, until the quotient becomes zero in Step 3.

The last remainder thus obtained will be the Most Significant Digit (MSD) of the new base number.

Other Base System to Decimal System

Steps

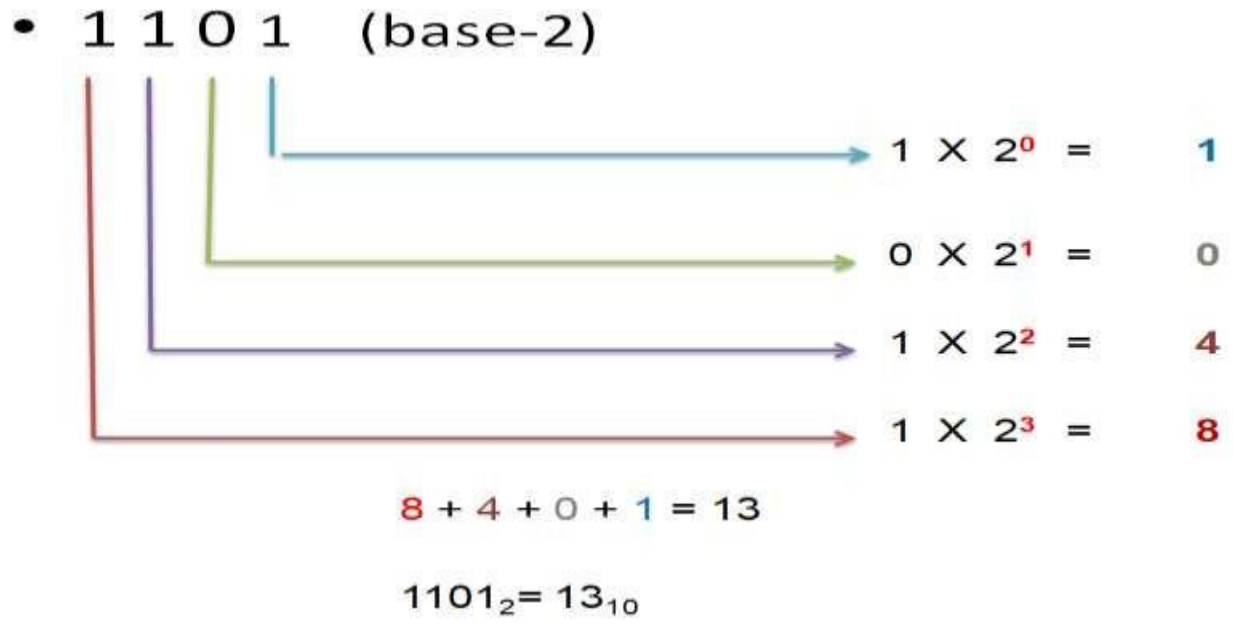
Step 1 – Determine the column (positional) value of each digit (this depends on the position of the digit and the base of the number system).

Step 2 – Multiply the obtained column values (in Step 1) by the digits in the corresponding columns.

Step 3 – Sum the products calculated in Step 2. The total is the equivalent value in decimal.

BINARY TO DECIMAL

- Technique
  - Multiply each bit by 2^n, where n is the "weight" of the bit
  - The weight is the position of the bit, starting from 0 on the right
  - Add the results



EXAMPLE

Bit "0"

101011_2 =>

$1 \times 2^0 = 1$

$1 \times 2^1 = 2$

$0 \times 2^2 = 0$

$1 \times 2^3 = 8$

$0 \times 2^4 = 0$

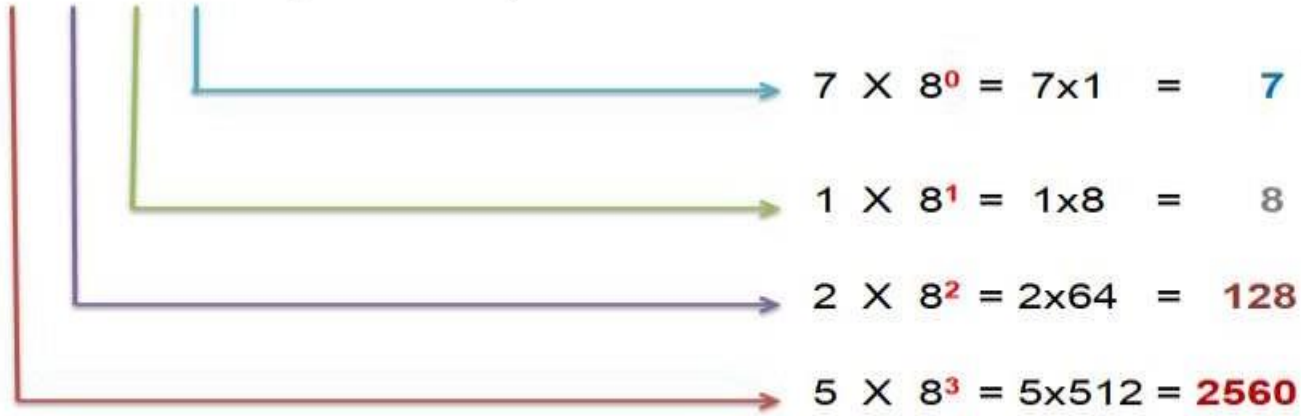
$1 \times 2^5 = 32$

43_{10}

OCTAL TO DECIMAL

- Technique
  - Multiply each bit by 8^n, where n is the "weight" of the bit
  - The weight is the position of the bit, starting from 0 on the right
  - Add the results

- 5 2 1 7 (base-8)



$2560 + 128 + 8 + 7 = 2703$

$5217_8 = 2703_{10}$

EXAMPLE

724_8 =>

$4 \times 8^0 = 4$

$2 \times 8^1 = 16$

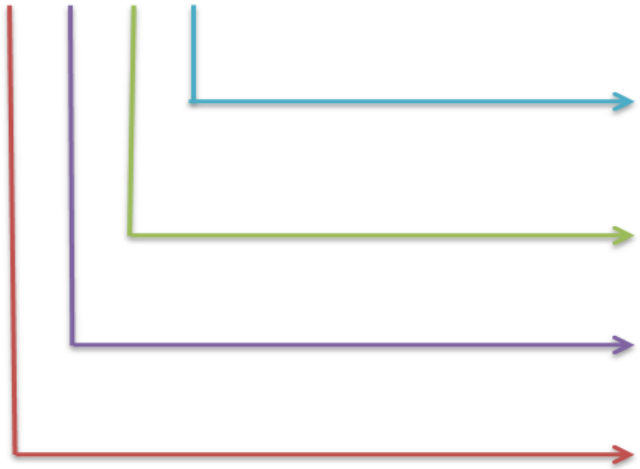
$7 \times 8^2 = 448$

468_{10}

HEXADECIMAL TO DECIMAL

- Technique
  - Multiply each bit by 16^n, where n is the "weight" of the bit
  - The weight is the position of the bit, starting from 0 on the right
  - Add the results

- 1 A C F (base-16) [A = 10, B = 11, C = 12, D = 13, E = 14, F = 15]



$15 \times 16^0 = 15 \times 1 = 15$

$12 \times 16^1 = 12 \times 16 = 192$

$10 \times 16^2 = 10 \times 256 = 2560$

$1 \times 16^3 = 1 \times 4096 = 4096$

15+192+2560+4096= 6863

$1ACF_{16} = 6863_{10}$

EXAMPLE

$ABC_{16} \Rightarrow$

$C \times 16^0 = 12 \times 1 = 12$

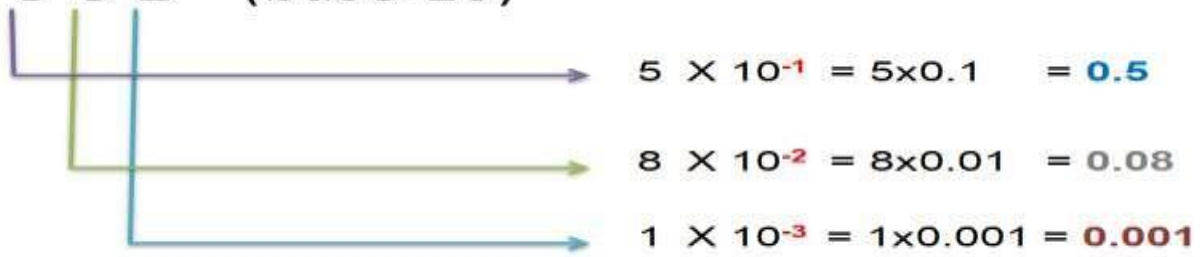
$B \times 16^1 = 11 \times 16 = 176$

$A \times 16^2 = 10 \times 256 = 2560$

2748_{10}

fractional number

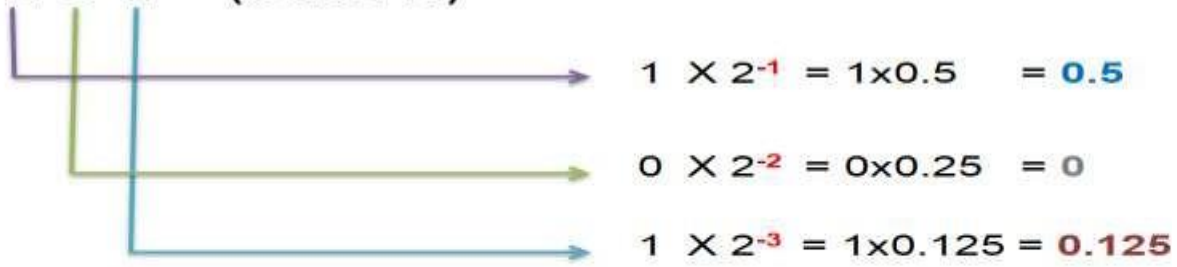
- . 5 8 1 (base-10)



0.5 + 0.08 + 0.001 = 0.581

Binary-to-Decimal Conversion

- . 1 0 1 (base-2)



0.5 + 0 + 0.125 = 0.625

$0.101_2 = 0.625_{10}$

Octal-to-Decimal Conversion

- . 2 5 (base-8)

$2 \times 8^{-1} = 2 \times 0.125 = 0.25$

$5 \times 8^{-2} =$ 5X 0.015625 = 0.07812

0.25+0.07812=0.328125

$0.25_8 = 0.328125_{10}$

Hexadecimal-to-Decimal Conversion

- . F 5 (base-16)

$15 \times 16^{-1} = 15 \times 0.0625 = 0.9375$

$5 \times 16^{-2} = 5 \times 0.00390625 = 0.01953125$

0.9375 + 0.01953125 = 0.95703125

$0.F5_{16} = 0.95703125_{10}$

Decimal to Other Base System

Steps

- **Step 1** – Divide the decimal number to be converted by the value of the new base.
- **Step 2** – Get the remainder from Step 1 as the rightmost digit (least significant digit) of new base number.
- **Step 3** – Divide the quotient of the previous divide by the new base.
- **Step 4** – Record the remainder from Step 3 as the next digit (to the left) of the new base number.

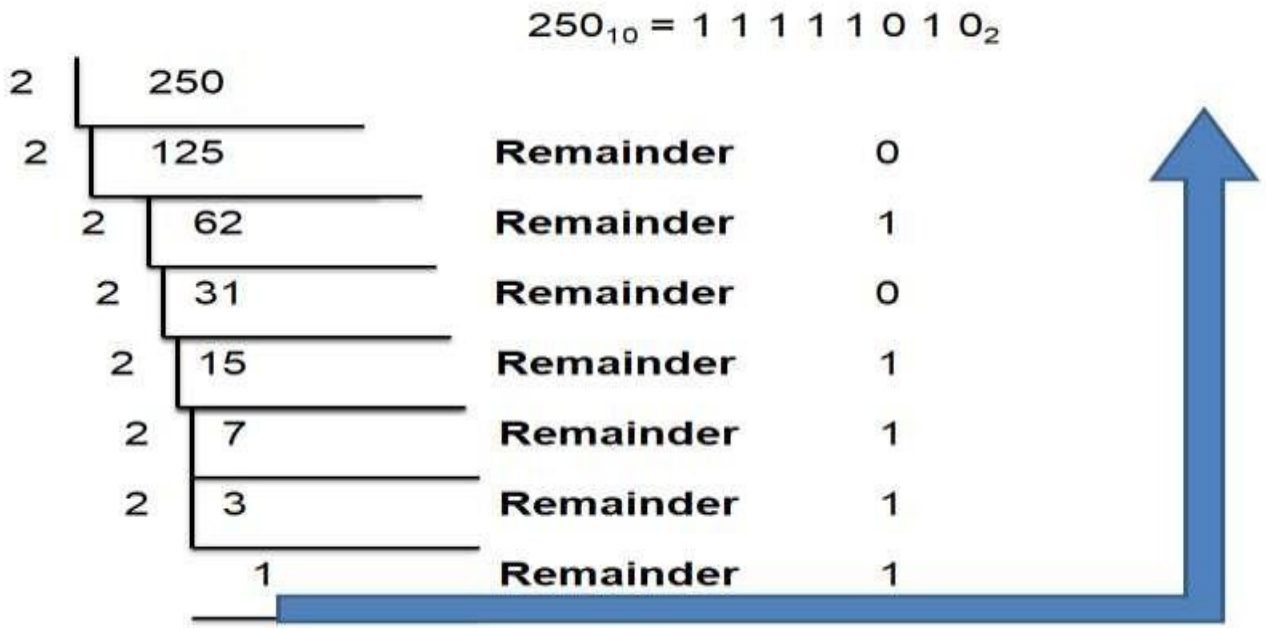
Repeat Steps 3 and 4, getting remainders from right to left, until the quotient becomes zero in Step 3.

The last remainder thus obtained will be the Most Significant Digit (MSD) of the new base number.

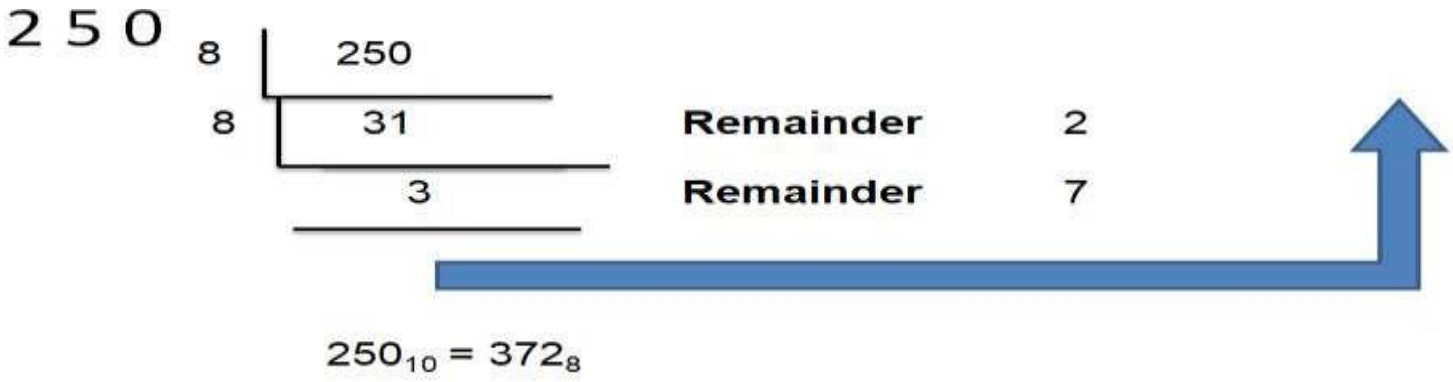
DECIMAL TO BINARY

- Technique
  - Divide by two, keep track of the remainder
  - First remainder is bit 0 (LSB, least- significant bit)
  - Second remainder is bit 1
  - Etc.

2 5 0

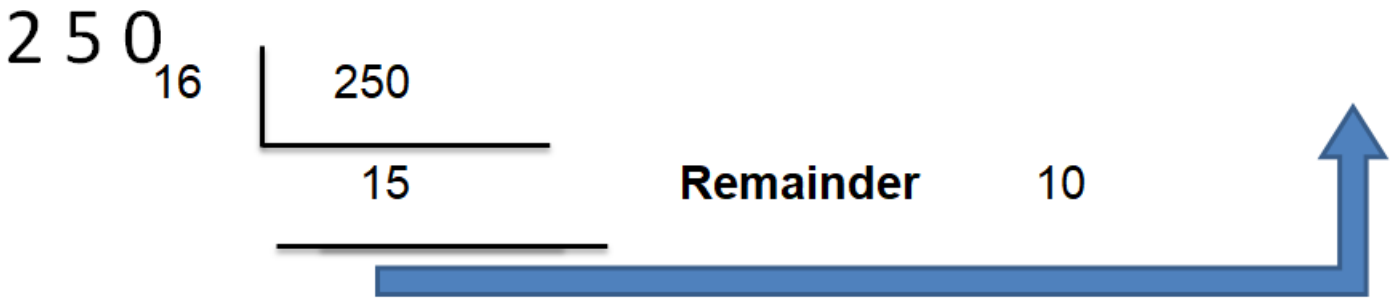


Decimal-to-Octal Conversion



$250_{10} = 372_8$

Decimal-to-Hexadecimal Conversion



$250_{10} = 15\ 10_{16}$?

$= FA_{16}$

Decimal-to-Binary Conversion (fractional number)

0 . 4375

0.4375 x 2 = 0.8750
0.8750 x 2 = 1.75
0.75 x 2 = 1.5
0.5 x 2 = 1.0

$0.4375_{10} = 0.0111_2$

Decimal-to-Octal Conversion

0 . 4375

0.4375 x 8 = 3.5
0.5 x 8 = 4.0

$0.4375_{10} = 0.34_8$

Decimal-to-Hexadecimal Conversion

0 . 4375

0.4375 x 16 = 7.0

$0.4375_{10} = 0.7_{16}$

Decimal-to-Binary Conversion (Estimation)

$11001_2 \rightarrow 2^{-1} + 2^{-2} + 2^{-5}$
$\rightarrow$ 0.5 + 0.25 +0.03125
$\rightarrow$ **0.78125**

• 0 . 7 8 2

0.782 x 2 = 1.564
0.564 x 2 = 1.128
0.128 x 2 = 0.256
0.256 x 2 = 0.512
0.512 x 2 = 1.024
0.024 x 2 = 0.048
0.048 x 2 = 0.096
0.192 x 2 = 0.384
0.384 x 2 = 0.768
0.768 x 2 = 1.536

1100100001_2
$\rightarrow 2^{-1} + 2^{-2} + 2^{-5} + 2^{-10}$
$\rightarrow$ 0.5 + 0.25 +0.03125 + 0.0009765625
$\rightarrow$ **0.7822265625**

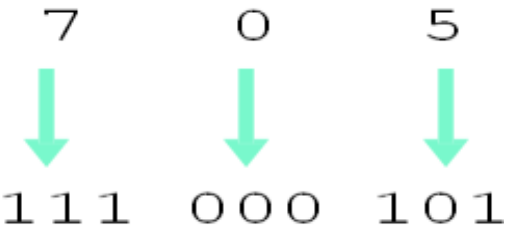
HEXADECIMAL TO BINARY

$10AF_{16} = ?_2$

$10AF_{16} = 0001000010101111_2$

OCTAL TO BINARY

$705_8 = ?_2$

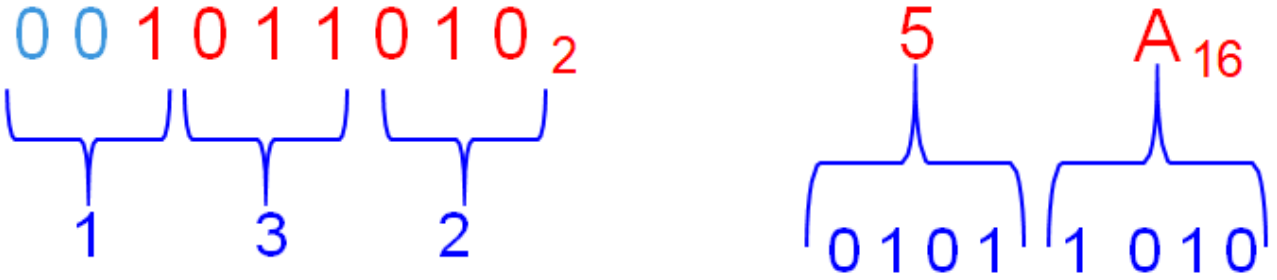


$705_8 = 111000101_2$

OCTAL TO HEXADECIMAL

- 132_8 $(?)_{16}$

Octal Binary Hex



Department of Electronics and Communication Engineering

EMPOWERING THROUGH INNOVATIONS
MLRS

MARRI LAXMAN REDDY
INSTITUTE OF TECHNOLOGY AND MANAGEMENT
(AN AUTONOMOUS INSTITUTION)
(Approved by AICTE, New Delhi & Affiliated to JNTUH, Hyderabad)
Accredited by NBA and NAAC with 'A' Grade & Recognized Under Section2(f) & 12(B)of the UGC act,1956

DIGITAL LOGIC DESIGN AND COMPUTER ORGANIZATION

- **UNIT-I: Complements, Signed binary numbers**

Presentation By

Dr. Bapi Debnath

Assoc. Prof.

Department of ECE

MARRI LAXMAN REDDY

INSTITUTE OF TECHNOLOGY AND MANAGEMENT

(AN AUTONOMOUS INSTITUTION)

(Approved by AICTE, New Delhi & Affiliated to JNTUH, Hyderabad)

Accredited by NBA and NAAC with 'A' Grade & Recognized Under Section2(f) & 12(B)of the UGC act,1956

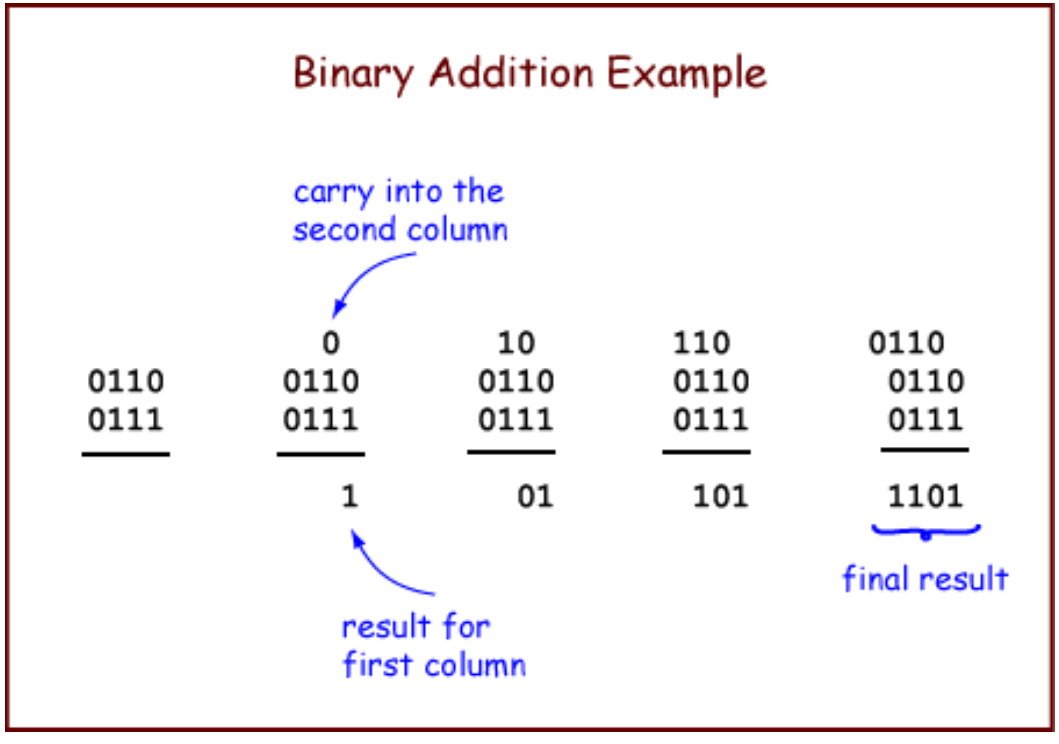
Table for Number System Conversion

Decimal (base=10)	Binary (base=2) 8 4 2 1	Octal (base=8)	Hex-Dec (base=16)
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7

Decima l (base=10)	Binary (base=2) 8 4 2 1	Octal (base=8)	Hex-Dec (base=16)
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

BINARY ADDITION

A + B	SUM	CARRY
0 + 0	0	0
0 + 1	1	0
1 + 0	1	0
1 + 1	0	1



ADDITION

Base-2 Addition:

$(1010)_2+(0011)_2$

```
  1
  1 0 1 0
  0 0 1 1
  ---------
  1 1 0 1
```

Base-10 Addition:

$(10)_{10}+(3)_{10}$

```
  10
   3
  ---
  13
```

Base-8 Addition:

$(723)_8+(237)_8$

```
  1 0 1
    7 2 3        Carry
    2 3 7
  -------
Carry
  1 1 6 2   Ans:(1162)8
```

8 | 10 → 1→2 (12)

8 | 9 → 1→1 (11)

Base-16 Addition:

$(3F8)_{16}+(5B3)_{16}$

```
  1
  3 F 8     F+B=15+11=26
  5 B 3
  -------
  9 A B          (9AB)16
```

16 | 26 → 1→A (1A)

BINARY SUBSTRACTION

A - B	Subtract	Borrow
0 – 0	0	0
0 – 1	0	1
1 – 0	1	0
1 – 1	0	0

BINARY	DECIMAL
Borrow 0 10	
1 ~~1~~ 0 0	12
– 0 0 1 0	– 2
1 0 1 0	10

Example Subtract 1001010_2 - 10100_2

	0	1	10	0	10	
	~~1~~	~~0~~	0	~~1~~	0	1	0
(-)			1	0	1	0	0
		1	1	0	1	1	0

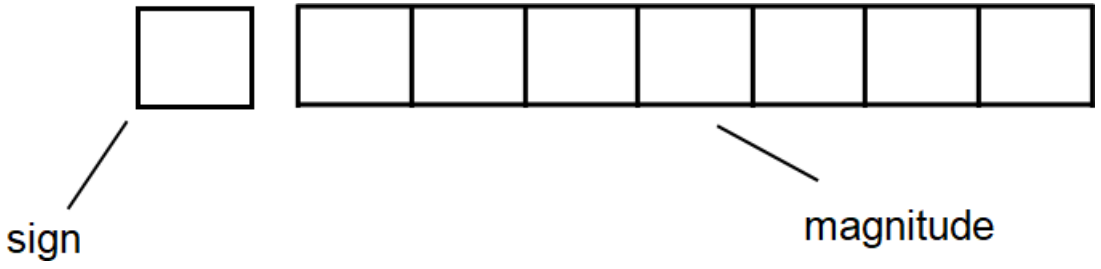
2-bit	3-bit	4-bit	5-bit binary	6-bit
2 1	4 2 1	8 4 2 1	16 8 4 2 1	32 16 8 4 2 1

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

64 32 16 8 4 2 1

Sign-and-Magnitude

- Example: an 8-bit number can have 1-bit sign and 7-bits magnitude.



Complements (General)

- Complement numbers can help perform subtraction. With complements, subtraction can be performed by addition.

Hence, A – B can be performed by A + (-B) where (-B) is represented as the complement of B.

- In general for Base-*r* number, there are:
  - *(i) r*-1's Complement
  - *(ii) Radix* (or *r*'s) Complement
- For Base-2 number, we have seen:
  - (i) 1's Complement
  - (ii) 2's Complement

For Base-10 number
- (i) 9's
- (ii) 10's

For Base-8 number
- (i) 7's
- (ii) 8's

For Base-16 number
- (i) 15's
- (ii) 16's

Complements of Binary Numbers

1's complement :

The 1's **complement** of a binary number is defined as the value obtained by inverting all the bits in the binary representation of the number .

(basically known as swapping 0 with **1** and vice versa)

1s Complement

♣ Given a number x which can be expressed as an n-bit binary number, its negative value can be obtained in 1s- complement representation using:

$-x = r^n - x - 1$ **here, r=2 n=No. of bits**

Example:

❑With an 8-bit number 00001100, its negative value, expressed in 1s complement, is obtained as follows:

$-(00001100)_2 = -(12)_{10}$

$= (2^8 - 12 - 1)_{10}$

$= (243)_{10}$

$= \mathbf{(11110011)_2}$

00001100

11110011

1s Complement

❑**Essential technique: invert all the bits.**

Examples: **1s complement of $(00000001)_{1s}$ = $(11111110)_{1s}$**

1s complement of $(01111111)_{1s}$ = $(10000000)_{1s}$

♣ **Largest Positive Number: 0 1111111 $+(127)_{10}$**

♣ **Largest Negative Number: 1 0000000 $-(127)_{10}$**

❑**Range: $-(127)_{10}$ to $+(127)_{10}$**

The most significant bit still represents the sign:

0 = +ve

1 = -ve.

- Examples :

(assuming 8-bit binary numbers):

$-(14)_{10} = -(00001110)_2 = (11110001)_{1's}$

$-(80)_{10} = -(\ ?\ )_2 = (\ ?\ )_{1's}$

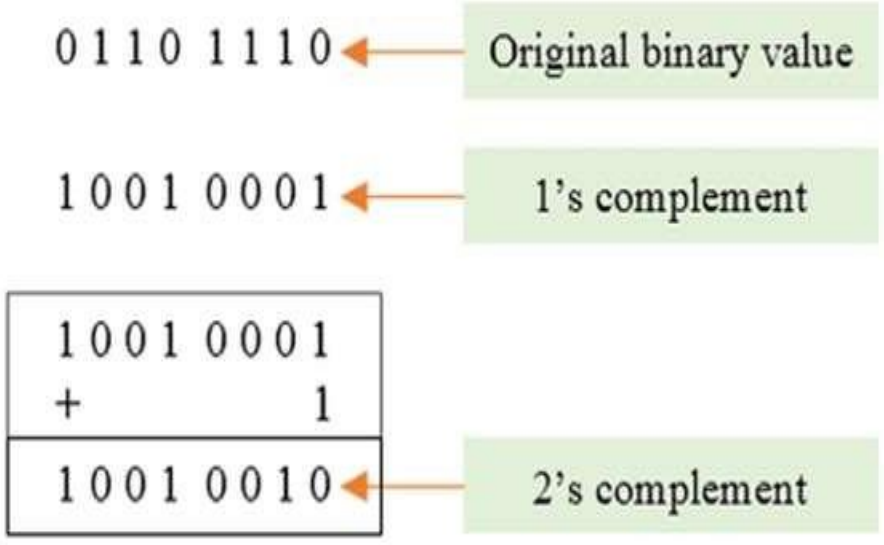
$(01010000)_2$ **128 64 32 16 8 4 2 1**

(10101111) 1's

2's Complements of Numbers

2's complement:

1. Take a given number First find one's complement.
2. Then add 1 to LSB side results of 1's complement.
3. Result is 2's complement of the given number.



2s Complement

♣ Given a number x which can be expressed as an n-bit binary number, its negative number can be obtained in 2s-complement representation using:

$-x = r^n - x$ **here, r=2 n=No. of bits**

Example: With an 8-bit number 00001100, its negative value in 2s complement is thus:

$-(00001100)_2 = -(12)_{10}$

$= (2^8 - 12)_{10}$

$= (244)_{10}$

$= \mathbf{(11110100)_{2's}}$

00001100

11110011

1

$(11110100)_{2's}$

2s Complement

- ✓ Largest Positive Number: 0 1111111 $+(127)_{10}$
- ✓ Largest Negative Number: 1 0000000 $-(128)_{10}$
- ✓ Zero: 0 0000000
- ✓ Range: $-(128)_{10}$ to $+(127)_{10}$
- ✓ The most significant bit still represents the sign: 0 = +ve;

  1 = -ve.

Complements of Octal (7's & 8's)

7's complement :

The 7's **complement** of a octal number is defined as the value obtained by subtracting of each bit in the octal with 7.

Examples:

1.7's complement of "7254" is "0523"

```
  7 7 7 7
- 7 2 5 4
---------
  0 5 2 3
```

2.7's complement of "2745" is "5032"

Complements of Octal (7's & 8's)

8's complement:

1. Take a given number First find 7's complement.
2. Then add 1 to LSB side results of 7's complement.
3. Result is 8's complement of the given number.

Examples:

1. 8's complement of "7254" is "0524"

```
  7 7 7 7
  7 2 5 4
  -------
  0 5 2 3 +1=0524
```

Complements of Hex Decimal (15's & 16's)

15's complement :

The 15's **complement** of a hex decimal number is defined as the value obtained by subtracting of each bit in the hex decimal with 15.

Examples:

1.15's complement of "ABCD" is "5432"

```
   15 15 15 15
   A  B  C  D
  ------------
   5  4  3  2
```

2.15's complement of "1B74" is "D48B"

Complements of Hex Decimal (15's & 16's)

16's complement:

1. Take a given number First find 15's complement.
2. Then add 1 to LSB side results of 15's complement.
3. Result is 16's complement of the given number.

Examples:

1. 16's complement of "ABCD" is "5433"

	15	15	15	15	
	A	B	C	D	
	5	4	3	2	+ 1 = 5433

2. 15's complement of "1B74" is "D48C"

Subtraction by Complements

Subtractions are 2 types

1)Large-Small value (L-S) 2)Small-Large value (S-L)

For type-1:A-B (L-S)

By 1's Complement:

Step 1: Take B & do 1's Comp.
Step 2: Then add A to step:1 result.
If carry is generated remove it & add that carry to step:2
Step 3: Result

By 2's Complement:

Step 1: Take B & do 2's Comp. (1's + 1 = 2's)
Step 2: Then add A to step:1 result.
If carry is generated removed/ignored it.
Step 3: Result

1. Perform the Subtraction by using 1's & 2's Complement methods for $(1010)_2$-$(0111)_2$?

Sol: by 1's Complement

A-B=1010-0111

Step1: Take **B** i.e., 0111

1's Comp 1000

Step2: add **A** with Step1 result

```
  1010
  1000
 10010
```

Carry is generated & remove carry

Add it 0010

```
     1
  0011
```

Result: $(0011)_2 = (3)_{10}$

Sol: by 2's Complement

A-B=1010-0111

Step1: Take **B** i.e., 0111 do 2's Comp.

1's Comp 1000

```
     1
  1001
```

Step2: add **A** with Step1 result

```
  1010
  1001
 10011
```

Carry is generated remove/ignore it

Result: $(0011)_2 = (3)_{10}$

2. Perform the Subtraction by using 1's & 2's Complement methods for $(111001)_2$-$(101011)_2$?

Sol: by 1's Complement

A-B=111001-101011

Step1: Take **B** i.e., 101011

1's Comp 010100

Step2: add **A** with Step1 result

```
  111001
  010100
 1001101
```

Carry is generated & remove carry

Add it 001101

```
       1
  001110
```

Result: $(001110)_2 = (14)_{10}$

Sol: by 2's Complement

A-B=111001-101011

Step1: Take **B** i.e., 101011 do **2's**

1's Comp 010100

```
       1
  010101
```

Step2: add **A** with Step1 result

```
  111001
  010101
 1001110
```

Carry is generated & remove carry

Result: $(001110)_2 = (14)_{10}$

Subtraction by Complements

Type-2 Small-Large value (S-L) of A-B

By 1's Complement:

Step 1: Take B & do 1's Comp.

Step 2: Then add A to step:1 result.
If carry is generated remove it & add that carry to step:2

Step 3: Do the 1's Comp. for result of Step:2

Step 4: Final Result with "–" sign

By 2's Complement:

Step 1: Take B & do 2's Comp.

Step 2: Then add A to step:1 result.
If carry is generated remove it

Step 3: Do the 2's Comp. for result of Step:2

Step 4: Final Result "–" sign

1. Perform the Subtraction by using 1's & 2's Complement methods for $(0111)_2$-$(1010)_2$?

Sol: by 1's Complement

A-B=0111-1010

Step1: Take **B** i.e.,1010 do **1's**

1's Comp 0101

Step2: add **A** with Step1 result

```
  0111
  0101
 ----
  1100
```

Carry is not generated so **1100**

Step3: Take step2 result 1100 do **1's**

& result is **–ve** 0011

Result: $-(0011)_2 = -(3)_{10}$

Sol: by 2's Complement

A-B=0111-1010

Step1: Take **B** i.e.,1010 do **2's**

1's Comp 0101 +1 =0110

Step2: add **A** with Step1 result

```
  0111
  0110
 ----
  1101
```

Carry is not generated so **1101**

Step3: Take step2 result 1101 do **2's**

& result is **–ve**

1's Comp 0010 + 1 = 0011

Result: $-(0011)_2 = -(3)_{10}$

2. Perform the Subtraction by using 1's & 2's Complement methods for $(101011)_2$-$(111001)_2$?

Sol: by 1's Complement

A-B=101011-111001

Step1: Take **B** i.e.,111001 do **1's**

1's Comp 000110

Step2: add **A** with Step1 result

101011
000110
110001

Carry is not generated so **110001**

Step3: Take step2 result 110001 do **1's** & result is – ve 001110

Result: $-(001110)_2 = -(14)_{10}$

Sol: by 1's Complement

A-B=101011-111001

Step1: Take **B** i.e.,111001 do **2's**

1's Comp 000110 +1 = 000111

Step2: add **A** with Step1 result

101011
000111
110010

Carry is not generated so **110010**

Step3: Take step2 result 110010 do **2's** & result is – ve **001101+1=** 001110

Result: $-(001110)_2 = -(14)_{10}$

Department of Electronics and Communication Engineering

MARRI LAXMAN REDDY
INSTITUTE OF TECHNOLOGY AND MANAGEMENT

(AN AUTONOMOUS INSTITUTION)

(Approved by AICTE, New Delhi & Affiliated to JNTUH, Hyderabad)

Accredited by NBA and NAAC with 'A' Grade & Recognized Under Section2(f) & 12(B)of the UGC act,1956

EMPOWERING THROUGH INNOVATIONS

MLRS

DIGITAL LOGIC DESIGN AND COMPUTER ORGANIZATION

• UNIT-I: BINARY CODES

Presentation By

Dr. Bapi Debnath

Assoc. Prof.

Department of ECE

MARRI LAXMAN REDDY

INSTITUTE OF TECHNOLOGY AND MANAGEMENT

(AN AUTONOMOUS INSTITUTION)

(Approved by AICTE, New Delhi & Affiliated to JNTUH, Hyderabad)

Accredited by NBA and NAAC with 'A' Grade & Recognized Under Section2(f) & 12(B)of the UGC act,1956

Codes

- Code is a symbolic representation of discrete information, which may be present in the form of numbers , letters or physical quantities.
- Required to conveniently input data into digital systems and interpret results.

Types of Codes

- Weighted binary codes.
- Non-weighted codes.
- Error detecting codes.
- Alphanumeric codes.

Weighted Binary Codes

- Weighted binary codes obey their positional weighting principles.
- Each positions of a number represents a specific weight.
- In a weighted binary code, the bits are multiplied by the weights indicated.The sum of these weighted bits gives the equivalent decimal digit.

Examples:

8421

2421

5421

5211

Weighted Code

Decimal no	8421	5421	2421
0	0000	0000	0000
1	0001	0001	0001
2	0010	0010	0010
3	0011	0011	0011
4	0100	0100	0100
5	0101	1000	1011
6	0110	1001	1100
7	0111	1010	1101
8	1000	1011	1110
9	1001	1100	1111

- Many weighted codes are possible
- Straight binary coding is a method of representing a decimal number by its binary equivalent.
- Weights must be chosen in a way that their sum is not greater than 15 and not less than 9
- One of the weights must be 1

BCD Code

- BCD is weighted code
- Binary coded decimal (bcd) uses the binary number system to specify the decimal numbers 0 to 9.
- It has four bits.
- The weights are assigned according to the positions occupied by these digits

Example

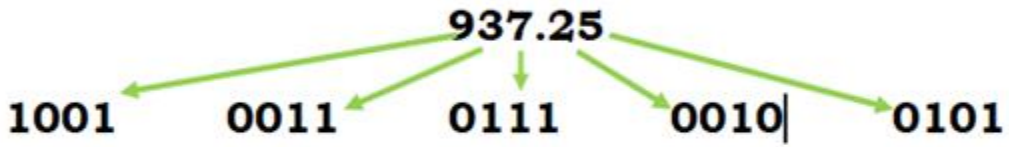
Bcd code for the decimal number 874

Solution: $(874)_{10} = (1000\ 0111\ 0100)_{bcd}$

BCD code (8421 code)

- *Simplest form*: each decimal digit is replaced by its binary equivalent.

Example1: 937.25 is represented by



(937.25)= $(1001\,0011\,0111.0010\,0101)_{BCD}$

- This representation is referred to as *"Binary-Coded-Decimal"*: **BCD** or mor explicitly as **8-4-2-1(8421 code).**

> **Note:**
> **The result is quite different than that obtained by converting the number as a whole into binary.**

Example 2:

= $100001010100_{(BCD)}$

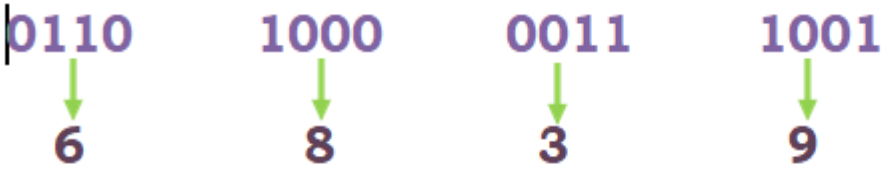
- BCD is *inefficient,* e.g. to represent **999** and **999999** bits needed:
  - 10 and 20 in binary numbers
  - 12 and 24 for BCD code.

BCD code (8421 code)

Example 3: convert 0110100000111001(BCD) to its decimal equivalent.

Solution:

Divide the BCD number into four-bit groups and convert each to decimal:



0110100000111001(BCD) = 6839_{10}

- BCD is used in interfacing between a digit device and a human being, e.g. digital voltmeter (DVM).

BCD code (8421 code)

Example 4: Convert the following decimal and binary numbers to BCD.

a) 5648_{10}
b) 10001101_2

Solution:

a) 5648_{10} =0101 0110 0100 1000
b) $10001101_2 = 141_{10}$=0001 0100 0001

Example 5: convert the BCD number 011111000001 to its decimal equivalent.

$0111\ 1100\ 0001_{BCD}$ = error

Doesn't exist in the BCD Code

Non – Weighted Codes

- Non –weighted codes are not positionally weighted, this means that each position within a binary number is not a fixed value.

Examples

- Excess –3 codes
- Gray codes

Excess-3 Code

- It represents a decimal number, in binary form , as a number greater than 3.
- It is obtained by adding 3 to a decimal number.

Example

$[643]_{10}$ into excess-3 code

Decimal number	6	4	3
Add 3 to each bit	+3	+3	+3
Excess-3 code	**9**	**7**	**6**

Binary	Excess-3	Gray code
0000	0011	0000
0001	0100	0001
0010	0101	0011
0011	0110	0010
0100	0111	0110
0101	1000	0111
0110	1001	0101
0111	1010	0100
1000	1011	1100
1001	1100	1101
1010	1101	1111
1011	1110	1110
1100	1111	1010
1101	0000	1011
1110	0001	1001
1111	0010	1000

EXCESS 3 CODES

Decimal digit → BCD CODE 8 4 2 1 CODE → (ADD 3) → EXCESS 3 CODE

Decimal	8 4 2 1 CODE BCD CODE		XS 3
0	0000		0011
1	0001		0100
2	0010	+0011	0101
3	0011	+0011	0110
4	0100	+0011	0111
5	0101	+0011	1000
6	0110	+0011	1001
7	0111	+0011	1010
8	1000	+0011	1011
9	1001	+0011	1100

```
0000
+0011
------
0011

0001
+0011
------
0100
```

FIND EXCESS 3 CODE

Exmaple-1

```
        35
      /    \
    11      111
  0011     0101
+0011    + 0011
------   ------
  0110     1000
  ____     ____
   6        8
```

Exmaple-2

```
          111
 0001  0001  0001
+0011  0011  0011
-----------------
 0100  0100  0100
```

Gray Codes

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

Decimal numbers	Binary code	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

GRAY CODE

Decimal	binary	Gray code
0	0 0 0 0	0 0 0 0
1	0 0 0 1	0 0 0 1
2	0 0 1 0	0 0 1 1
3	0 0 1 1	0 0 1 0
4	0 1 0 0	0 1 1 0
5	0 1 0 1	0 1 1 1
6	0 1 1 0	0 1 0 1
7	0 1 1 1	0 1 0 0
8	1 0 0 0	1 1 0 0
9	1 0 0 1	1 1 0 1
11		
12		
13		

Two values differ in only one bit

Binary no. is convert to gray code
To reduce switching circuit

Binary to Gray code Conversion

- The first bit(MSB) of the gray code is the same as the first bit of the binary number;
- The second bit of the gray code equals the exclusive OR of the first and second bits of the binary number.
- The third gray code bit equals the exclusive-OR of the second and third bits of the binary number and so on.

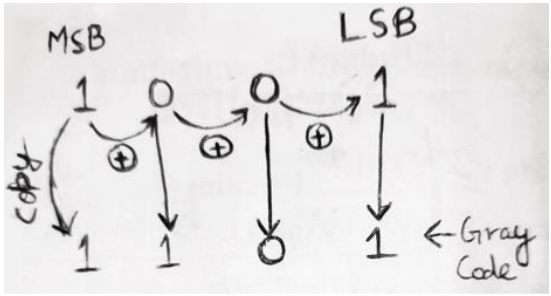
EX-OR (X-OR) Gate Truth Table

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

Example

Convert [10110]2 to graycode

Ans : 11101



BINARY TO GRAY CODE CONVERSION

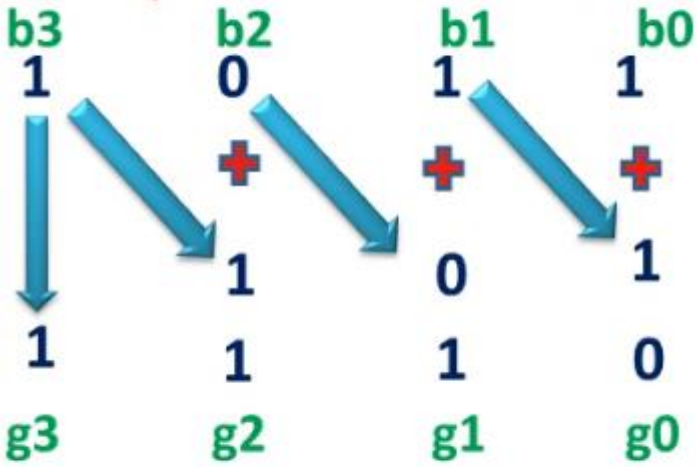
$g3 = b3$

$g2 = b2 \oplus b3$

$g1 = b1 \oplus b2$

$g0 = b0 \oplus b1$

Example = convert 1011 binary to gray



Gray code

$A \oplus B$

A	B	$Y = AB' + A'B$
0	0	0
0	1	1
1	0	1
1	1	0

MSB write as it is

Add MSB in next bit neglect carry if produce

We can represent it as X-OR (odd 1's detector)

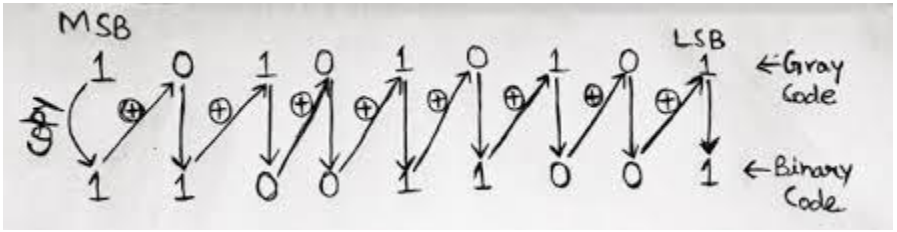
Conversion from Gray to Binary

- The first binary bit(MSB) is the same as that of the first gray code bit.
- The second gray bit is XOR ed with the first binary, the third gray bit is XOR ed with the second binary and so on.

Example

Convert the gray code 110101 to binary form

Ans : $[100110]_2$



GRAY TO BINARY CODE CONVERSION

$b3 = g3$

$b2 = b3 \oplus g2$

$b1 = b2 \oplus g1$

$g0 = b1 \oplus g0$

Example = convert 1101 gray code to binary

g3	g2	g1	g0	
1	1	0	1	gray code
1	0	0	1	binary code
b3	b2	b1	b0	

$A \oplus B$

A	B	$Y = AB' + A'B$
0	0	0
0	1	1
1	0	1
1	1	0

MSB write as it is

Add MSB in next bit of gray code, neglect carry if produce

We can represent it as X-OR (odd 1's detector)

Other Codes

Reflective Codes

- A code is said to be reflective when the code for 9 is the complement of the code for 0, 8 for 1, 7 for 2, 6 for 3, and 5 for 4.

2421, 5211 and excess-3 codes	**reflective codes.**
8421 code	**not reflective code.**

Sequential Codes

- A code is said to be a sequential when each succeeding code is one binary number greater than its preceding code.

8421 and excess-3 codes	**sequential codes**
2421 and 5421 codes	**not sequential codes.**

Error Detecting Codes

- **Odd parity code:**

Total number of 1s in the code group must be an odd(including parity bit).

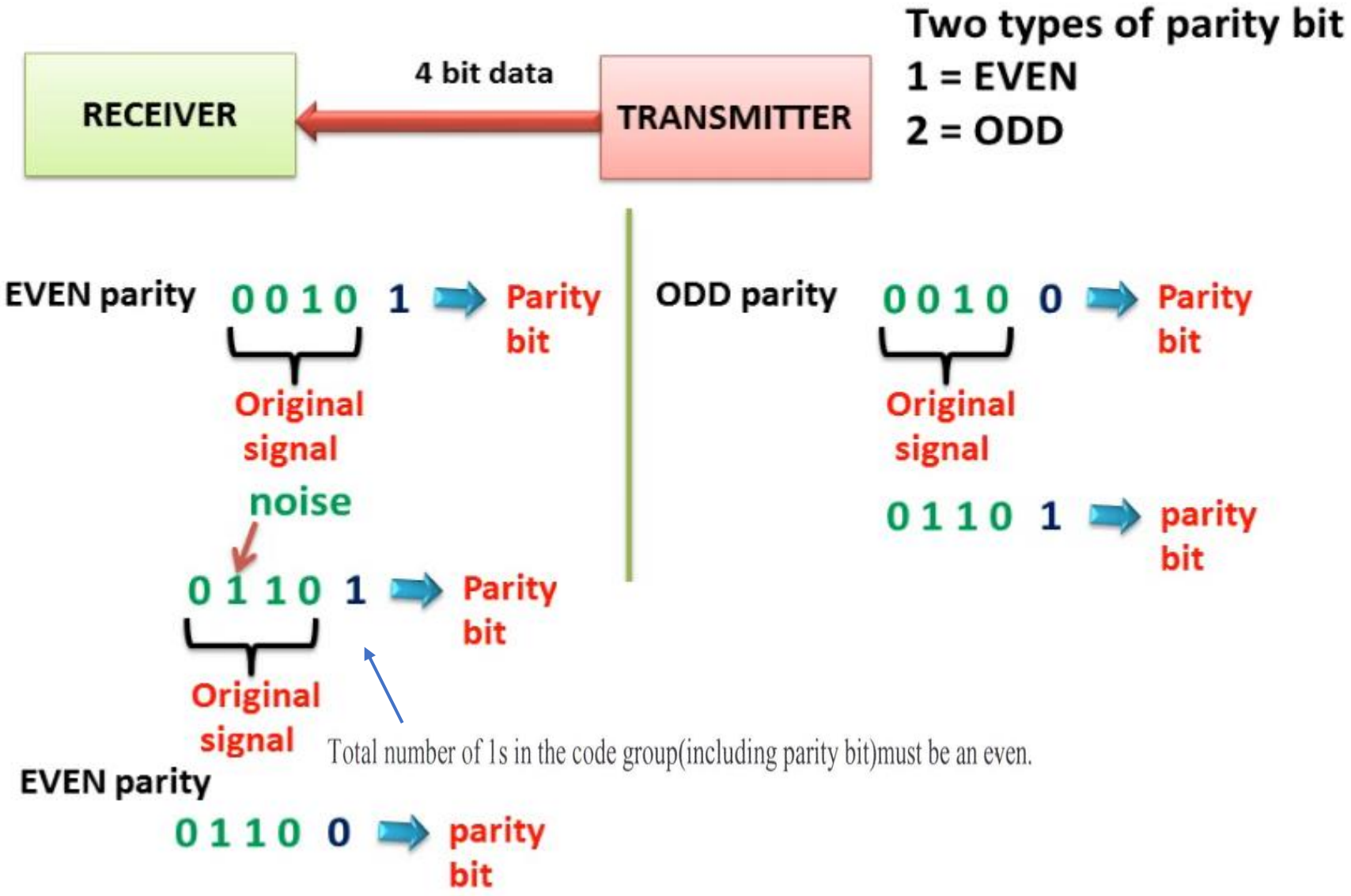
- **Even parity code:**

Total number of 1s in the code group(including parity bit)must be an even.

WHAT IS PARITY

PARITY is a bit in form of 0 and 1 and use for detect the error in signal
It only detect single bit error not more then single bit .
PARITY bit is a single bit or extra bit send with the original data and it Tell total no. of 1's present in signal .
Two types of parity bit
1 = EVEN
2 = ODD

WHAT IS PARITY



Other Codes

- Hamming code
- Alphanumeric codes
- ASCII code
- EBCDIC code
- Hollerith code

Department of Electronics and Communication Engineering

MARRI LAXMAN REDDY
INSTITUTE OF TECHNOLOGY AND MANAGEMENT

(AN AUTONOMOUS INSTITUTION)

(Approved by AICTE, New Delhi & Affiliated to JNTUH, Hyderabad)

Accredited by NBA and NAAC with 'A' Grade & Recognized Under Section2(f) & 12(B)of the UGC act,1956

EMPOWERING THROUGH INNOVATIONS

MLRS

DIGITAL LOGIC DESIGN AND COMPUTER ORGANIZATION

- **UNIT-I: Binary storage and registers**

Presentation By

Dr. Bapi Debnath

Assoc. Prof.

Department of ECE

MARRI LAXMAN REDDY

INSTITUTE OF TECHNOLOGY AND MANAGEMENT

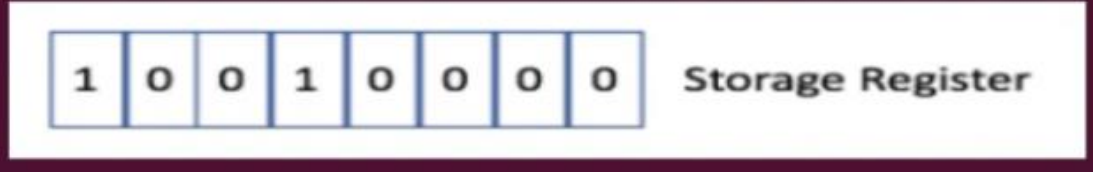
(AN AUTONOMOUS INSTITUTION)

(Approved by AICTE, New Delhi & Affiliated to JNTUH, Hyderabad)

Accredited by NBA and NAAC with 'A' Grade & Recognized Under Section2(f) & 12(B)of the UGC act,1956

Registers

- ★ A register consists of a collection of (flip flops) binary storage cells, each implemented by a flip-flop.
- ★ A flip flop is used to store single bit digital data. For storing a large number of bits, the storage capacity is increased by grouping more than one flip flops.
- ★ The number of cells determines the length of the register.
- ★ Thus, a 'n' bit register contains n flip flops and can store n bits (2^n binary combinations, or distinct states.)



Operations by registers

- The register is used to perform different types of operations.
- For performing the operations, the CPU use these registers.
- The result returned by the system will store in the registers.
- There are the following operations which are performed by the registers:

Fetch:

It is used

- To take the instructions given by the users.
- To fetch the instruction stored into the main memory.

Operations by registers

Decode:

The decode operation is used to interpret the instructions.

In decode, the operation performed on the instructions is identified by the CPU.

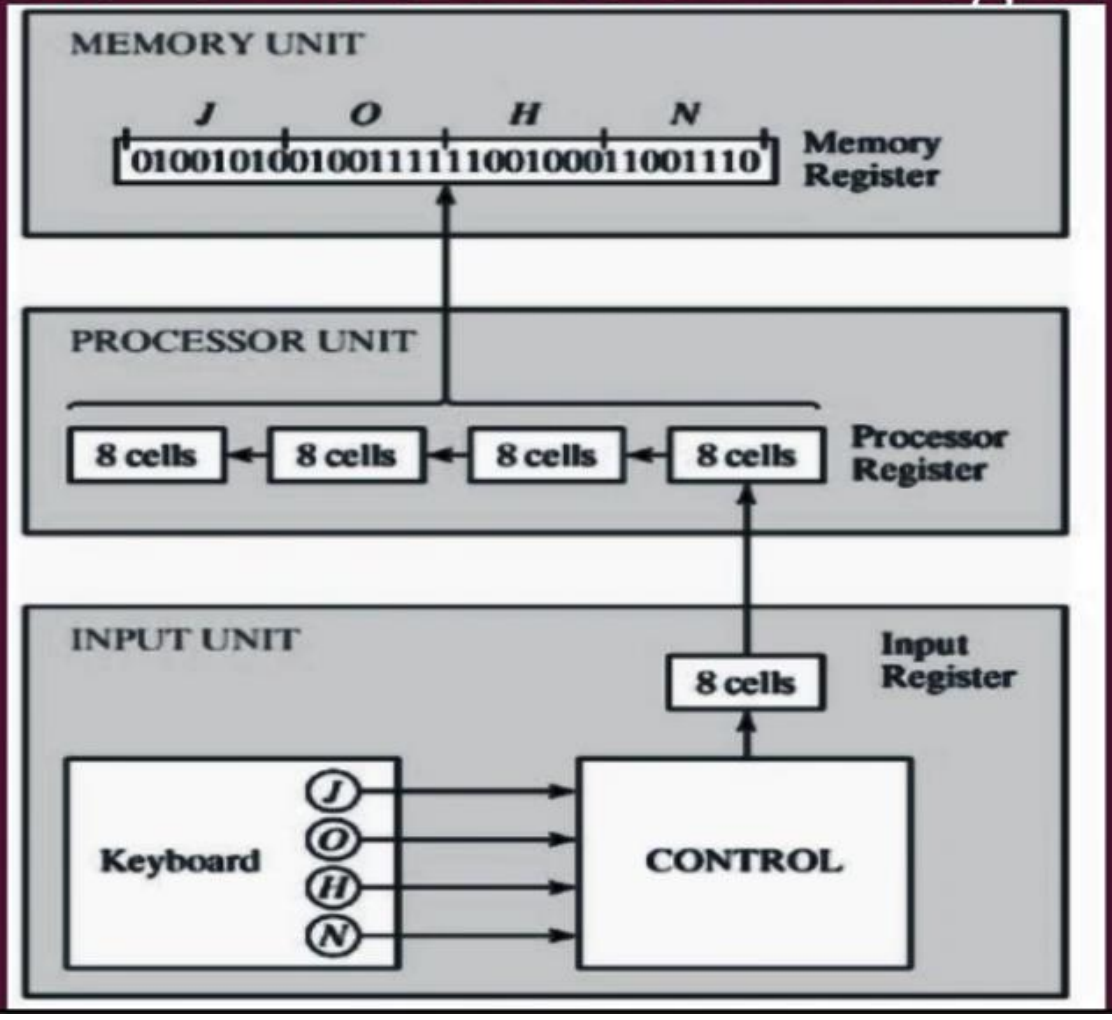
In simple words, the decode operation is used to decode the instructions.

Execute:

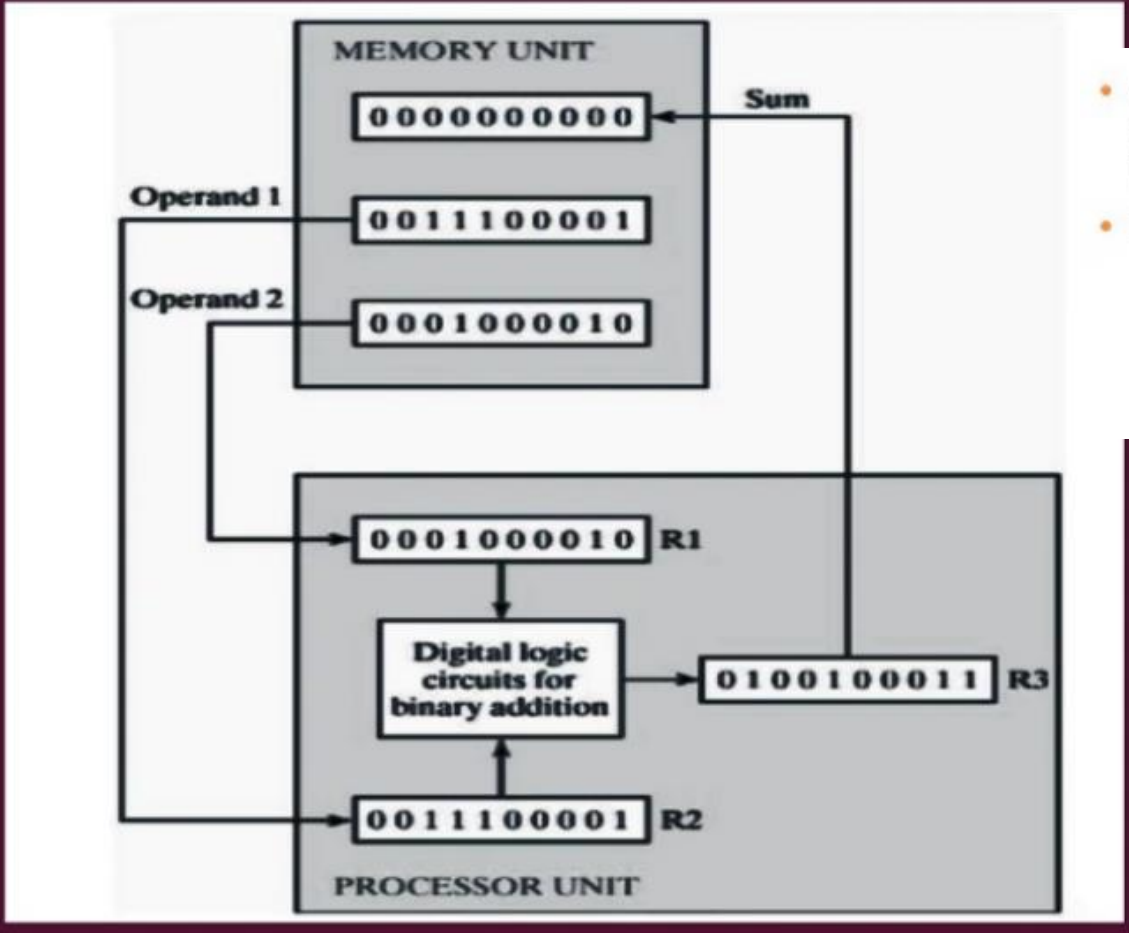
The execution operation is used to store the result produced by the CPU into the memory.

After storing this result, it is displayed on the user screen.

Transfer of information in Register



Example of Binary information processing



- Circuit elements to manipulate individual bits of information
- Load-store machine

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

Department of Electronics and Communication Engineering

MARRI LAXMAN REDDY
INSTITUTE OF TECHNOLOGY AND MANAGEMENT

(AN AUTONOMOUS INSTITUTION)

(Approved by AICTE, New Delhi & Affiliated to JNTUH, Hyderabad)

Accredited by NBA and NAAC with 'A' Grade & Recognized Under Section2(f) & 12(B)of the UGC act,1956

EMPOWERING THROUGH INNOVATIONS

MLRS

DIGITAL LOGIC DESIGN AND COMPUTER ORGANIZATION

- **UNIT-I: Binary logic, Boolean algebra, basic theorems**

Presentation By

Dr. Bapi Debnath

Assoc. Prof.

Department of ECE

MARRI LAXMAN REDDY

INSTITUTE OF TECHNOLOGY AND MANAGEMENT

(AN AUTONOMOUS INSTITUTION)

(Approved by AICTE, New Delhi & Affiliated to JNTUH, Hyderabad)

Accredited by NBA and NAAC with 'A' Grade & Recognized Under Section2(f) & 12(B)of the UGC act,1956

Logic Gate

A gate is an digital circuit which operates on one or more signals and produce single output.

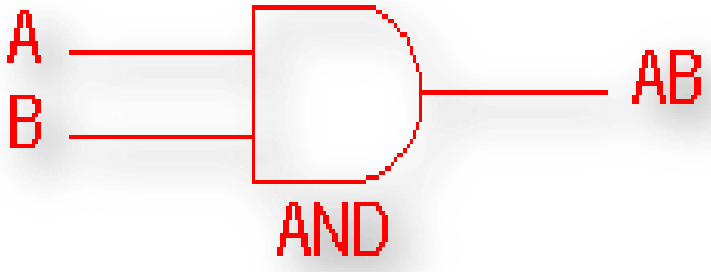
Gates are digital circuits because the input and output signals are denoted by either 1(high voltage) or 0(low voltage).

There are three basic gates and are:

1. AND gate
2. OR gate
3. NOT gate

AND gate

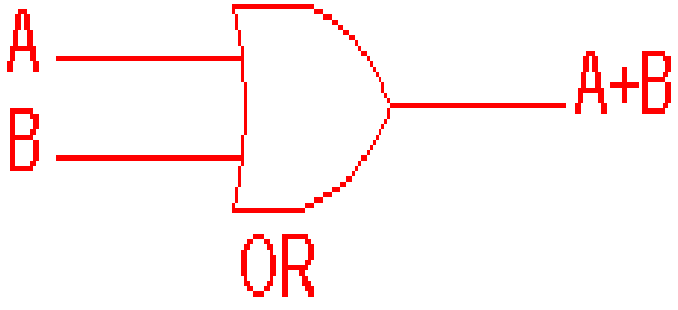
- The AND gate is an electronic circuit that gives a high output (1) only if all its inputs are high.
- AND gate takes two or more input signals and produce only one output signal.



Input A	Input B	Output AB
0	0	0
0	1	0
1	0	0
1	1	1

OR gate

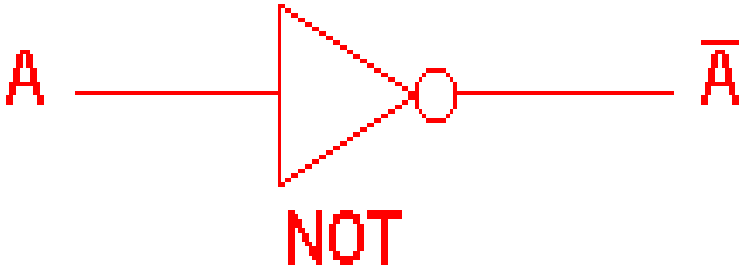
- The OR gate is an electronic circuit that gives a high output (1) if one or more of its inputs are high.
- OR gate also takes two or more input signals and produce only one output signal.



Input A	Input B	Output A+B
0	0	0
0	1	1
1	0	1
1	1	1

NOT gate

- The NOT gate is an electronic circuit that gives a high output (1) if its input is low .
- NOT gate takes only one input signal and produce only one output signal.
- The output of NOT gate is complement of its input.
- It is also called inverter.



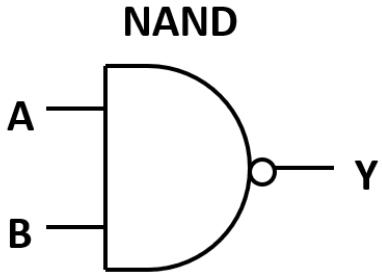
Input A	Output $\overline{A}$
0	1
1	0

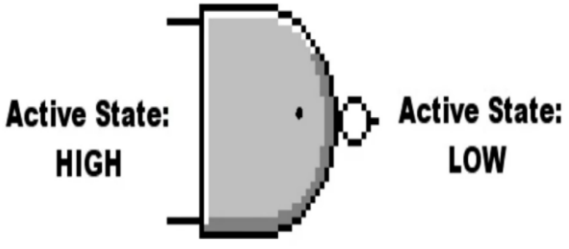
NAND Gate

- The NAND gate implements the NAND function, which means NOT-AND.
  - The inputs are AND & then NOT to get a single output.
  - The output of NAND gate is HIGH if any or all of the inputs are LOW.
  - When all inputs are HIGH, the output is LOW. Below table depicts the truth table for a two-input NAND gate.

Table Truth table for a two-input NAND gate

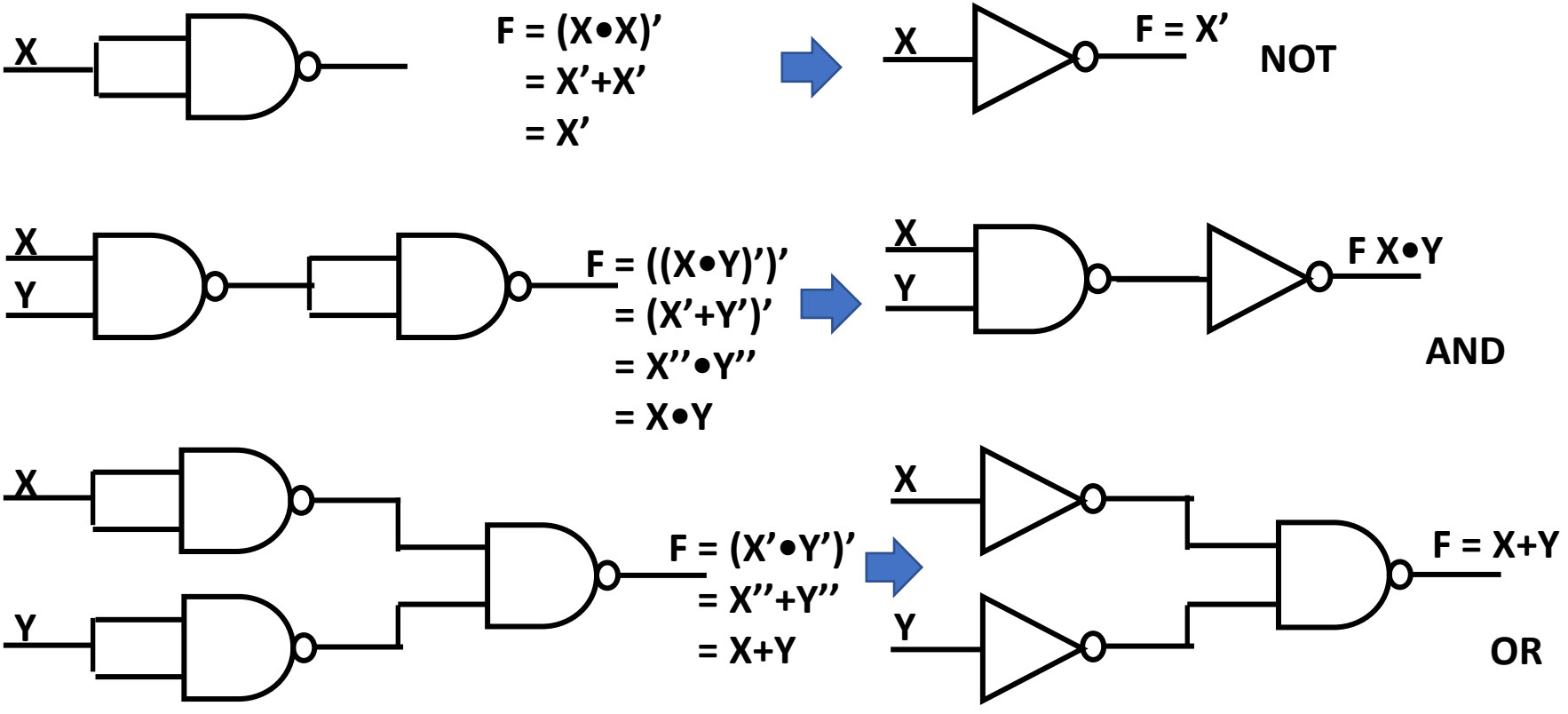
Inputs		Output
A	*B*	$Y = \overline{AB}$
0	0	1
0	1	1
1	0	1
1	1	0





NAND Gate

Known as a "universal" gate because ANY digital circuit can be implemented with NAND gates alone.



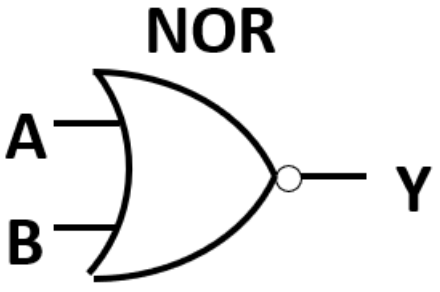
NOR Gate

The NOR gate is an OR gate with inverted output.

- Whereas the OR gate allows the output to be HIGH (logic 1) if any one or more of its inputs are HIGH, the NOR gate inverts this and forces the output to logic 0 when any input is HIGH, i.e., the output of a NOR gate is LOW if any of the inputs are HIGH.
- The output is HIGH when all inputs are LOW.
- The NOR function uses the plus sign (+) operator with the output represented by a expression with an over bar to indicate the OR inversion.
- The truth table of a two-input NOR gate is given in Table.

Table Truth table for a two-input NOR gate

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



Also known as a “universal” gate

NOR Gate

Also known as a “universal” gate

- Similarly, Fig. depicts how NOR gates are used to implement AND, OR, and NOT operations.

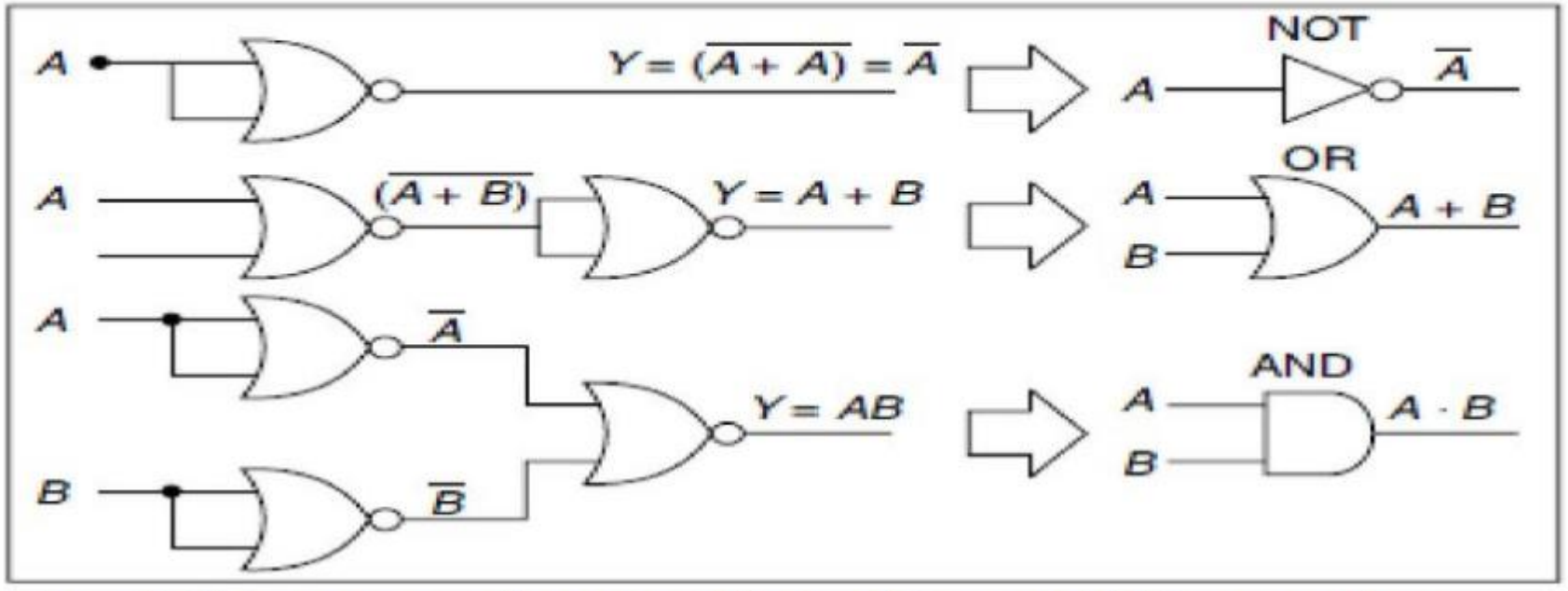


Fig. NOR gates performing NOT, OR, and AND operations

XOR Gate

- An encircled plus ⊕ sign is used to show the XOR operation.

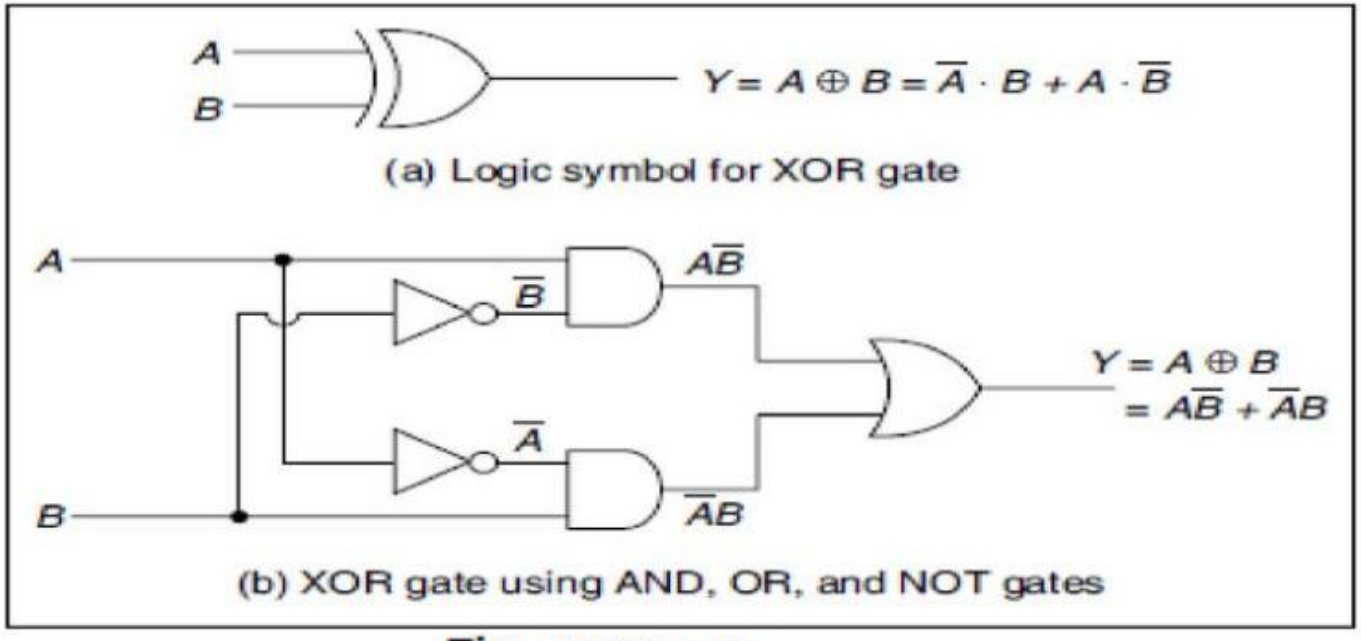


Fig. XOR gate

XNOR Gate

- The Exclusive-NOR gate is a XOR gate followed by a NOT gate. XNOR gate is a two-input and one-output logic gate circuit.
- In the gate, the output is HIGH if both inputs are either LOW or HIGH. The logic symbol for a XNOR is shown in Fig.

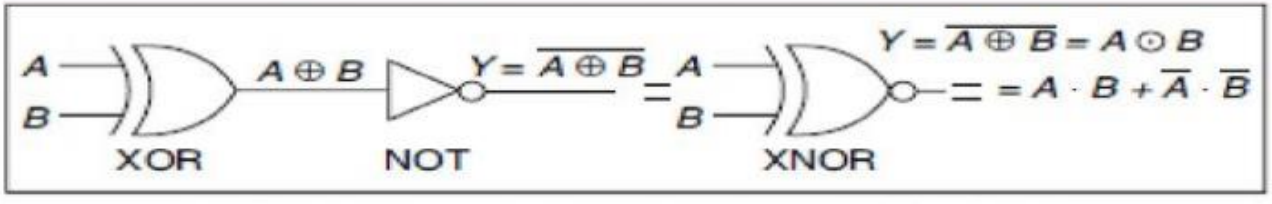
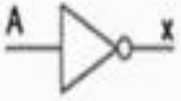


Fig. Logic symbol for a two-input XNOR gate

Table Truth table for a two-input XNOR gate

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

Name	NOT	AND	NAND	OR	NOR	XOR	XNOR
Alg. Expr.	$\overline{A}$	AB	$\overline{AB}$	$A+B$	$\overline{A+B}$	$A \oplus B$	$\overline{A \oplus B}$
Symbol							
Truth Table	A | X 0 | 1 1 | 0	B A | X 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1	B A | X 0 0 | 1 0 1 | 1 1 0 | 1 1 1 | 0	B A | X 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1	B A | X 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 0	B A | X 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 0	B A | X 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 1

Boolean Algebra

Theorems of Boolean Algebra

Theorem		
1. (*a*) $A + A = A$	(*b*) $A.A = A$	Tautology Law
2. (*a*) $A + 1 = 1$	(*b*) $A.0 = 0$	Union Law
3. $(A')' = A$		Involution Law
4. (*a*) $A + (B + C) = (A + B) + C$	(*b*) $A.(B.C) = (A.B).C$	Associative Law
5. (*a*) $(A + B)' = A'B'$	(*b*) $(A.B)' = A' + B'$	De Morgan's Law
6. (*a*) $A + AB = A$	(*b*) $A(A + B) = A$	Absorption Law
7. (*a*) $A + A'B = A + B$	(*b*) $A(A' + B) = AB$	
8. (*a*) $AB + AB' = A$	(*b*) $(A + B)(A + B') = A$	Logical adjancy
9. (*a*) $AB + A'C + BC = AB + A'C$	(*b*) $(A + B)(A' + C)(B + C) = (A + B)$	Consensus Law

Boolean Algebra

The proofs of the theorem

- Theorem 1(*a*): **A + A = A**

$$
\begin{aligned}
A + A &= A \\
A + A &= (A + A).1 && \text{by postulate } 2(b) \\
&= (A + A)\,(A + A') && 5(a) \\
&= A + AA' && 4(b) \\
&= A + 0 && 5(b) \\
&= A && 2(a)
\end{aligned}
$$

Theorem 1(b): **A . A = A**

$$
\begin{aligned}
A.A &= A. \\
A.A &= A.A + 0 && \text{by postulate } 2(a) \\
&= A.A + A.A' && 5(b) \\
&= A(A + A') && 4(a) \\
&= A.1 && 5(a) \\
&= A && 2(b)
\end{aligned}
$$

Boolean Algebra

Theorem 2 (A + 1 = 1 and A.0 = 0 ⟶ Union Law)

A + A	= 1		
A + 1	= 1.(A + 1)	by postulate	2(*b*)
	= (A + A′) (A + 1)		5(*a*)
	= A + A′.1		4(*b*)
	= A + A′		2(*b*)
	= 1		5(*a*)
A.0	= 0	by duality.	

Theorem 3. (A′)′ = A ⟶ Involution Law

Boolean Algebra

Theorem 4. (*a*) A + (B + C) = (A + B) + C
(*b*) A.(B.C) = (A.B).C → Associative Law

A	*B*	*C*	*(B + C)*	*A + (B + C)*	*(A + B)*	*(A + B) + C*
0	0	0	0	0	0	0
0	0	1	1	1	0	1
0	1	0	1	1	1	1
0	1	1	1	1	1	1
1	0	0	0	1	1	1
1	0	1	1	1	1	1
1	1	0	1	1	1	1
1	1	1	1	1	1	1

- Theorem 4(*b*)—can be proved in similar fashion.
- Theorem 5(*a*) and 5(*b*)—can also be proved by perfect induction method.

Boolean Algebra

Theorem 6. (*a*) A + AB = A
(*b*) A(A + B) = A → Absorption Law

$$A + AB = A$$

$$A + AB = A\,(1 + B)$$

$$= A\,(1)$$

$$= A$$

$$A.(A + B) = A \quad \text{By duality.}$$

Boolean Algebra

Theorem 7 (*a*) $A + A'B = A + B$
(*b*) $A(A' + B) = AB$

$A + A'B = A + B$

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

- $A.(A' + B) = A.B$ By duality

Boolean Algebra

Theorem 8 (*a*) $AB + AB' = A$
(*b*) $(A + B)(A + B') = A$ → Logical adjancy

- 8(*a*)

$AB + AB' = A$

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

$= A$

- 8(*b*)

$(A + B) . (A + B') = A$ By duality.

Boolean Algebra

Theorem 9. (*a*) AB + A′C + BC = AB + A′C
(*b*) (A + B) (A′ + C) (B + C) = (A + B) → Consensus Law

OR

Redundancy theorem

- 9(*a*)

AB + A′C + BC = AB + A′C

AB + A′C + BC = AB + A′C + BC(A + A′)

= AB + A′C + ABC + A′BC

= AB (1 + C) + A′C (1 + B)

= AB + A′C

- 9(*b*)
- (A + B) (A′ + C) (B + C) = (A + B) (A′ + C) By duality.

Minimization using theorems

Example : Simplify & Describe logic circuit for

$$Y = A` B` C` + A` B C` + A B` C` + A B C`$$

$$Y = (A` B` + A` B + A B` + A B) C`$$
$$= [A`( B` + B) + A (B` + B ) ] C`$$
$$= [A' (1) + A (1)] C'$$
$$= (A` + A ) C`$$
$$Y = C`$$

Example

.. $(X+Y)(\overline{X}+Y+Z)(\overline{X}+Y+\overline{Z})$

Solution:

$(X+Y)(\overline{X}+Y+Z)(\overline{X}+Y+\overline{Z})$

$= (X+Y)((\overline{X}+Y)+Z)((\overline{X}+Y)+\overline{Z})$ (Associative)

$= (X+Y)((\overline{X}+Y)+(Z\cdot\overline{Z}))$ (Distributive)

$= (X+Y)(\overline{X}+Y)+(0)$ (Complement)

$= (X+Y)(\overline{X}+Y)$ (Null)

$= (X\cdot\overline{X})+Y$ (Distributive)

$= (0)+Y$ (Complement)

$= Y$ (Null)

There are two instances in this problem that require the use of the distributive property of '+' to simplify the problem. Another way to solve this problem is to use the principle of duality:

$(X+Y)(\overline{X}+Y+Z)(\overline{X}+Y+Z)$

The dual of the expression above is:

$XY+\overline{X}YZ+\overline{X}Y\overline{Z}$

After simplifying this expression and obtaining an answer, the dual of the answer has to be taken so that the simplified form of the original expression is obained.

$XY+\overline{X}YZ+\overline{X}Y\overline{Z}$ simplifies to Y

So,

Dual $(X+Y)(\overline{X}+Y+Z)(\overline{X}+Y+\overline{Z})$ = Dual (Y)

The dual of Y is still Y, hence,

$(X+Y)(\overline{X}+Y+Z)(\overline{X}+Y+\overline{Z})=Y$

Department of Electronics and Communication Engineering

MARRI LAXMAN REDDY
INSTITUTE OF TECHNOLOGY AND MANAGEMENT

(AN AUTONOMOUS INSTITUTION)

(Approved by AICTE, New Delhi & Affiliated to JNTUH, Hyderabad)

Accredited by NBA and NAAC with 'A' Grade & Recognized Under Section2(f) & 12(B)of the UGC act,1956

EMPOWERING THROUGH INNOVATIONS

MLRS

DIGITAL LOGIC DESIGN AND COMPUTER ORGANIZATION

- **UNIT-I: Canonical and Standard forms**

Presentation By

Dr. Bapi Debnath

Assoc. Prof.

Department of ECE

MARRI LAXMAN REDDY

INSTITUTE OF TECHNOLOGY AND MANAGEMENT

(AN AUTONOMOUS INSTITUTION)

(Approved by AICTE, New Delhi & Affiliated to JNTUH, Hyderabad)

Accredited by NBA and NAAC with 'A' Grade & Recognized Under Section2(f) & 12(B)of the UGC act,1956

DEFINITIONS

- ***Literal:*** A variable or its complement
- ***Product term:*** Literals connected by •
- ***Sum term:*** Literals connected by +
- ***Minterm:*** A product term in which all the variables appear exactly once, either complemented or uncomplemented
- ***Maxterm:*** A sum term in which all the variables appear exactly once, either complemented or uncomplemented

MINTERM

- Represents exactly one combination in the truth table.
- Denoted by m_j, where j is the decimal equivalent of the minterm's corresponding binary combination (b_j).
- A variable in m_j is complemented if its value in b_j is 0, otherwise is uncomplemented.
- **Example:** Assume 3 variables (A, B, C), and $j = 3$. Then, b_j = 011 and its corresponding minterm is denoted by

$$m_j = A'BC$$

MAXTERM

- Represents exactly one combination in the truth table.
- Denoted by M_j, where j is the decimal equivalent of the maxterm's corresponding binary combination (b_j).
- A variable in M_j is complemented if its value in b_j is 1, otherwise is uncomplemented.
- **Example:** Assume 3 variables (A, B, C), and $j = 3$. Then, $b_j = 011$ and its corresponding maxterm is denoted by

$$M_j = A + B' + C'$$

TRUTH TABLE NOTATION FOR MINTERMS AND MAXTERMS

- Minterms and Maxterms are easy to denote using a truth table.
- **Example:**
  Assume 3 variables x, y, z (order is fixed)

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

CANONICAL FORMS (UNIQUE)

- Any Boolean function F( ) can be expressed as a *unique* **sum** of **min**terms and a unique **product** of **max**terms (under a fixed variable ordering).
- In other words, every function F() has two canonical forms:
  - Canonical Sum-Of-Products (sum of minterms)
  - Canonical Product-Of-Sums (product of maxterms)
- **Canonical Sum-Of-Products:**
  The minterms included are those m_j such that F( ) = 1 in row j of the truth table for F( ). It is also called **Disjunctive Normal Form (DNF).**
- **Canonical Product-Of-Sums:**
  The maxterms included are those M_j such that F( ) = 0 in row j of the truth table for F( ). It is also called **Conjunctive Normal Form (CNF).**

EXAMPLE

- Truth table for $f_1(a, b, c)$ at right
- The canonical sum-of-products form (DNF) for f_1 is

  $f_1(a, b, c) = m_1 + m_2 + m_4 + m_6$

  $= a'b'c + a'bc' + ab'c' + abc'$
- The canonical product-of-sums form (CNF) for f_1 is

  $f_1(a, b, c) = M_0 \bullet M_3 \bullet M_5 \bullet M_7$

  $= (a+b+c) \bullet (a+b'+c') \bullet$

  $(a'+b+c') \bullet (a'+b'+c').$
- Observe that: $m_j = M_j'$

a	b	c		f_1
0	0	0		0
0	0	1		1
0	1	0		1
0	1	1		0
1	0	0		1
1	0	1		0
1	1	0		1
1	1	1		0

SHORTHAND: ∑ AND ∏

- $f_1(a, b, c) = \sum m(1, 2, 4, 6)$, where $\sum$ indicates that this is a sum-of-products form, and $m(1, 2, 4, 6)$ indicates that the minterms to be included are m_1, m_2, m_4, and m_6.
- $f_1(a, b, c) = \prod M(0, 3, 5, 7)$, where $\prod$ indicates that this is a product-of-sums form, and $M(0, 3, 5, 7)$ indicates that the maxterms to be included are M_0, M_3, M_5, and M_7.
- Since $m_j = M_j'$ for any j,
  $\sum m(1, 2, 4, 6) = \prod M(0, 3, 5, 7) = f_1(a, b, c)$

CONVERSION BETWEEN CANONICAL FORMS

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

- **Example:**

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

$$= m_1 + m_2 + m_4 + m_6$$

$$= \sum(1, 2, 4, 6)$$

$$= \prod(0, 3, 5, 7)$$

$$= (a+b+c)\bullet(a+b'+c')\bullet$$

$$(a'+b+c')\bullet(a'+b'+c')$$

a	b	c	Minterm	Maxterm
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		$a+b'+c' = M_3$
1	0	0	$ab'c' = m_4$	
1	0	1		$a'+b+c' = M_5$
1	1	0	$abc' = m_6$	
1	1	1		$a'+b'+c' = M_7$

CONVERSION BETWEEN CANONICAL FORMS

- **Example:** Convert the Boolean function $f(x, y) = x.y' + x'.y + x'.y'$ to its conjunctive normal form.

Sol:

Here $f(x, y) = x.y' + x'.y + x'.y'$

$= m_2 + m_1 + m_0$

$= \sum(0, 1, 2)$

$= \prod(3)$

$= (x' + y')$

which is required CNF.

x	y	Minterm	Maxterm
0	0	$x'y' = m_0$	
0	1	$x'y = m_1$	
1	0	$xy' = m_2$	
1	1		$x'+y' = M_3$

CONVERSION BETWEEN CANONICAL FORMS

- **Example:** Convert the Boolean function

  $f(x, y, z) = (x' + y + z').(x' + y + z).(x + y' + z)$

  in disjunctive normal form.

Sol: Here $f(x, y, z) = (x' + y + z').(x' + y + z).(x + y' + z)$

$= M_5 + M_4 + M_2$

$= \prod(2, 4, 5)$

$= \sum(0, 1, 3, 6, 7)$

$= x'y'z' + x'y'z + x'yz + xyz' + xyz$

which is required DNF.

STANDARD FORMS (NOT UNIQUE)

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

- **Example:**

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

  is a *standard* sum-of-products form.

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

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

CONVERSION OF SOP FROM STANDARD TO CANONICAL FORM

- Expand *non-canonical* terms by inserting equivalent of 1 in each missing variable x:

$$(x + x') = 1$$

- Remove duplicate minterms.

- **Example:**

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

$$= a'b'c + (a+a')\,bc' + a(b+b')c'$$

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

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

CONVERSION OF POS FROM STANDARD TO CANONICAL FORM

- Expand noncanonical terms by adding 0 in terms of missing variables (*e.g.*, $xx' = 0$) and using the distributive law.
- Remove duplicate maxterms.
- **Example:** $f_1(a,b,c) = (a+b+c)\bullet(b'+c')\bullet(a'+c')$

$$= (a+b+c)\bullet(aa'+b'+c')\bullet(a'+bb'+c')$$

$$= (a+b+c)\bullet(a+b'+c')\bullet(a'+b'+c')\bullet$$

$$(a'+b+c')\bullet(a'+b'+c')$$

$$= (a+b+c)\bullet(a+b'+c')\bullet(a'+b'+c')\bullet(a'+b+c')$$

- **Example:** Express the Boolean function $f(x, y, z) = x \,.\, (y' \,.\, z)'$ in its disjunctive normal form.

- **Sol:** Here $f(x, y, z) = x \,.\, (y' \,.\, z)'$

$= x \,.\, [(y')' + z']$ [By De-Morgan's law]

$= x \,.\, (y + z')$ [Involution law]

$= x \,.\, y + x \,.\, z'$ [Distributive law]

$= x.\, y.\, (z + z') + x.\, z'.(y + y')$ [(y+y'=1), (z+z'=1)]

$= xyz + xyz' + xz'y + xz'y'$ [Distributive law]

$= xyz + xyz' + xyz' + xy'z'$ [Commutative law]

$= xyz + xyz' + xy'z'$

Disjunctive Normal Form

- **Example:** Express the Boolean function $f(x, y, z) = (x\,y' + x\,z)' + z'$ in its conjunctive normal form.

- **Sol:** Here $f(x, y, z) = (x\,y' + x\,z)' + z'$

$= (x\,y')'.\,(x\,z)' + z'$ [By De-Morgan's law]

$= [x' + (y')']\,.\,(x' + z') + z'$ [By De-Morgan's law]

$= (x' + y)\,.\,(x' + z') + z'$ [Involution law]

$= (x' + y + z')\,.\,(x' + z' + z')$ [Distributive law]

$= (x' + y + z')\,.\,(x' + z' + y.y')$ [Idempotent law] $(yy' = 0)$

$= (x' + y + z').(x' + z' + y).(x' + z' + y')$ [Distributive law]

$= (x' + y + z').(x' + y + z').(x' + y' + z')$

Conjunctive Normal Form

- **Example:** Prove that the complete disjunctive normal form and complete conjunctive normal form in three variables in a Boolean algebra is equal to the unit and zero element of the Boolean algebra.

- **Sol:** Complete disjunctive normal form in x, y and z

$= x'y'z' + x'y' z + x'yz' + x'yz + xy'z' + xy'z + xyz' + xyz$

$= x'(y'z' + y'z + yz' + yz) + x\,(y'z' + y'z + yz' + yz)$ [Distributive law]

$= (x' + x)\,.\,(y'z' + y'z + yz' + yz)$ [Distributive law]

$= 1\,.\,(y'z' + y'z + yz' + yz)$ [Complement law]

$= y'(z' + z)\,.\,y\,(z' + z)$ [Distributive law]

$= (y' + y)\,.\,(z' + z)$ [Distributive law]

$= 1\,.\,1$ [Complement law]

$= 1$ [Idempotent law]

Unit element of the Boolean algebra.

- Again, Complete conjunctive normal form in x, y and z

$= (x + y + z).(x + y + z').(x + y' + z).(x + y' + z').$

$(x' + y + z).(x' + y + z').(x' + y' + z).(x' + y' + z')$

$= (x + y + z.\ z').(x + y' + z.\ z').(x' + y + z.\ z').(x' + y' + z.\ z')$ [Distributive law]

$= (x + y + 0).(x + y' + 0).(x' + y + 0).(x' + y' + 0)$ [Complement law]

$= (x + y) . (x + y') . (x' + y) . (x' + y')$ [Identity law]

$= (x + y.\ y') . (x' + y.\ y')$ [Distributive law]

$= (x + 0) . (x' + 0)$ [Complement law]

$= x . x' = 0$ [Identity and Complement law]

Zero element of the Boolean algebra.

Department of Electronics and Communication Engineering

MARRI LAXMAN REDDY
INSTITUTE OF TECHNOLOGY AND MANAGEMENT

(AN AUTONOMOUS INSTITUTION)

(Approved by AICTE, New Delhi & Affiliated to JNTUH, Hyderabad)

Accredited by NBA and NAAC with 'A' Grade & Recognized Under Section2(f) & 12(B)of the UGC act,1956

EMPOWERING THROUGH INNOVATIONS

MLRS

DIGITAL LOGIC DESIGN AND COMPUTER ORGANIZATION

- **UNIT-I: K Maps**

Presentation By

Dr. Bapi Debnath

Assoc. Prof.

Department of ECE

MARRI LAXMAN REDDY

INSTITUTE OF TECHNOLOGY AND MANAGEMENT

(AN AUTONOMOUS INSTITUTION)

(Approved by AICTE, New Delhi & Affiliated to JNTUH, Hyderabad)

Accredited by NBA and NAAC with 'A' Grade & Recognized Under Section2(f) & 12(B)of the UGC act,1956

Minimization Using Karnaugh Map (K-map) Method

- In 1953 Maurice Karnaugh developed K-map in his paper titled 'The map method for synthesis of combinational logic circuits.
- The map method provides simple straight forward procedure for minimizing Boolean functions that may be regarded as pictorial form of truth table.
- K-map orders and displays the minterms in a geometrical pattern such that the application of the logic adjacency theorem becomes obvious.
- The K-map is a diagram made up of squares. Each square represents one minterm.
- Since any function can be expressed as a sum of minterms, it follows that a Boolean function can be recognized from a map by the area enclosed by those squares Whose minterms are included in the operation.
- By various patterns, we can derive alternative algebraic expression for the same operation, from which we can select the simplest one. (One that has minimum member of literals).

- A Karnaugh map is a two-dimensional truth-table. Note that the squares are numbered so that the binary representations for the numbers of two adjacent squares **differ in exactly one position**.

Rules for Grouping together adjacent cells containing 1's

- Groups must contain 1, 2, 4, 8, 16 (2^n) cells.
- Groups must contain only 1 (and X if don't care is allowed).
- Groups may be horizontal or vertical, but not diagonal.
- Groups should be as large as possible.
- Each cell containing a 1 must be in at least one group.
- Groups may overlap.
- Groups may wrap around the table. The leftmost cell in a row may be grouped with the rightmost cell and the top cell in a column may be grouped with the bottom cell.
- There should be as few groups as possible.

Description of K Maps and Terminology

- A Kmap is a matrix consisting of rows and columns that represent the output values of a Boolean function.
- The output values placed in each cell are derived from the minterms of a Boolean function.
- A *minterm* is a product term that contains all of the function's variables exactly once, either complemented or not complemented.

Description of K Maps and Terminology

- For example, the minterms for a function having the inputs x and y are:
- Consider the Boolean function,
- Its minterms are:

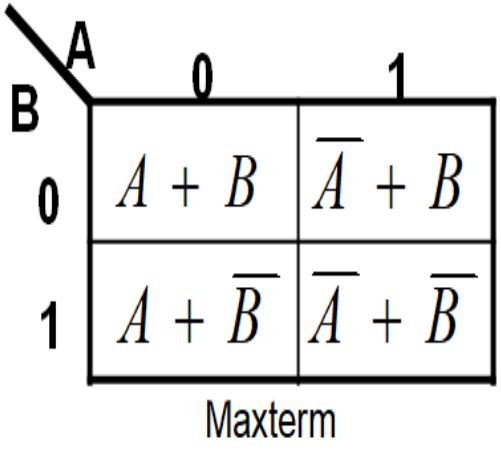
Minterm	X	Y
$\bar{X}\bar{Y}$	0	0
$\bar{X}Y$	0	1
$X\bar{Y}$	1	0
XY	1	1

Karnaugh maps

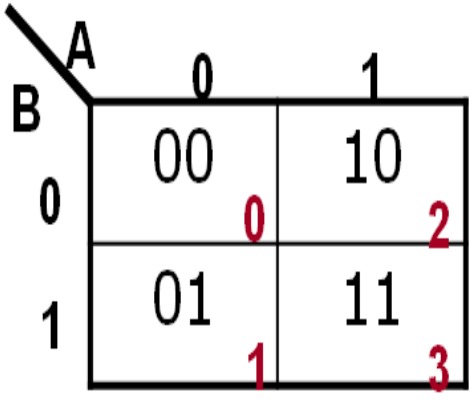
- Karnaugh maps, or K-maps, are often used to simplify logic problems with 2, 3 or 4 variables.

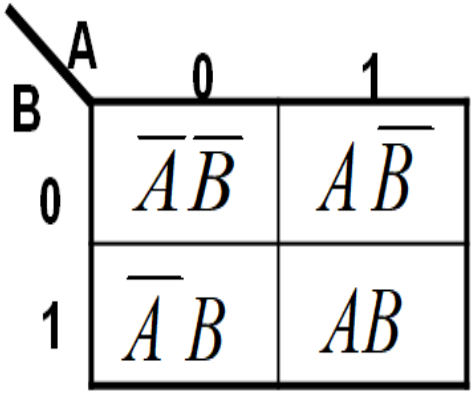
Cell = 2^n ,where n is a number of variables

For the case of 2 variables, we form a map consisting of 2^2=4 cells as shown in Figure



Maxterm



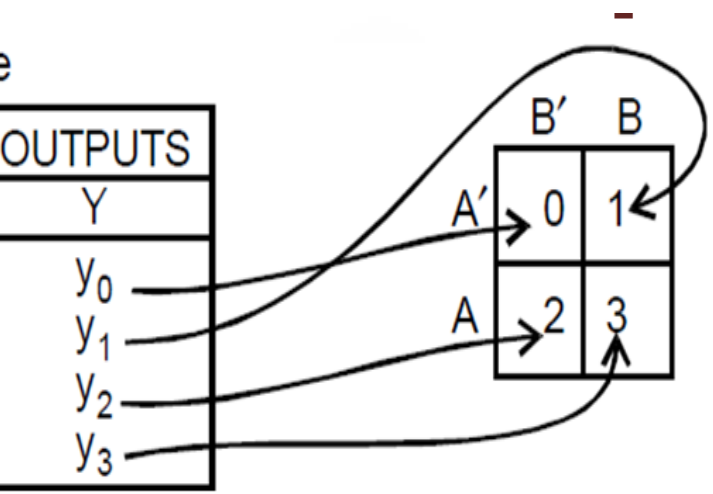


Minterm

Two-Variable K-Map

Truth Table

Min-term	INPUTS A	INPUTS B	OUTPUTS Y
m_0	0	0	y_0
m_1	0	1	y_1
m_2	1	0	y_2
m_3	1	1	y_3



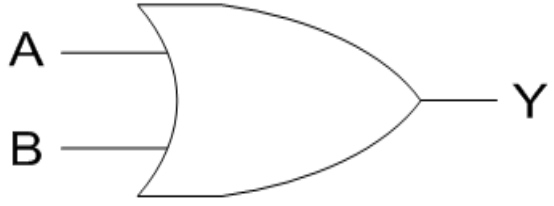
or

A \ B	0	1
0	y_0	y_1
1	y_2	y_3

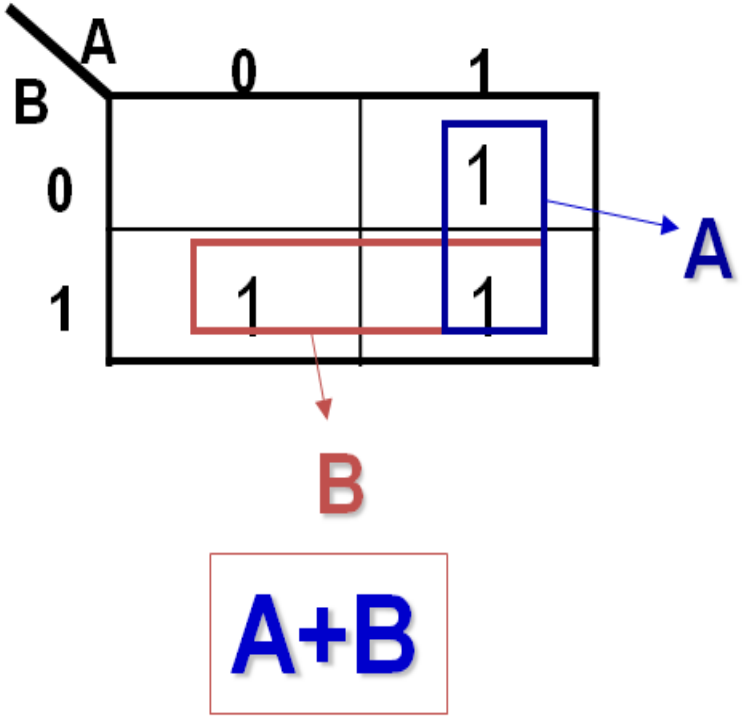
- The four squares (0, 1, 2, 3) represent the four possibly combinations of A and B in a two variable truth table.
- Square 0 in the K-map stands for A'B', (00)
- Square 1 for A'B, (01)
- Square 2 for AB', (10)
- Square 3 for AB, (11)

Example

2-variable Karnaugh maps are trivial but can be used to introduce the methods you need to learn. The map for a 2-input OR gate looks like this:



A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1



Now let us map a Boolean function Y = A + B.

Method I

- (i) Draw the truth table of given function.
- (ii)Draw a two variable K-map and fill those squares with a 1 for which the value of minterm in the function is equal to 1

Min-term	*A*	*B*	*Y*
m_0	0	0	0
m_1	0	1	1
m_2	1	0	1
m_3	1	1	1

Method II

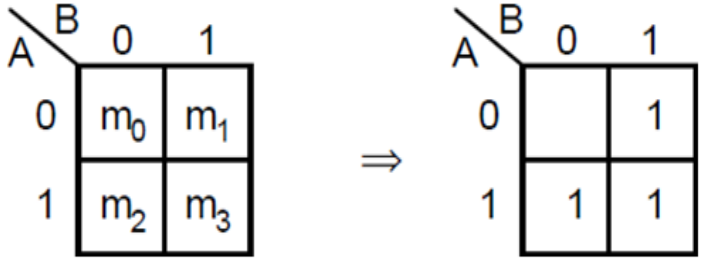
(i)Find all the minterms of the function Y = A + B.

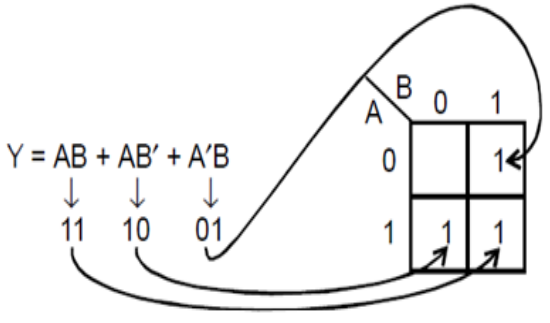
Y = A + B = A(B + B′) + B(A + A′)

= AB + AB′ + AB + A′B

= AB + AB′ + A′B

(ii)Draw a two variable K map using this sum of minterms expression.





3- Variable Karnaugh maps

- 3 variables Karnaugh map

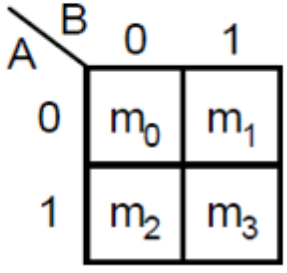
Cell = 2^3=8

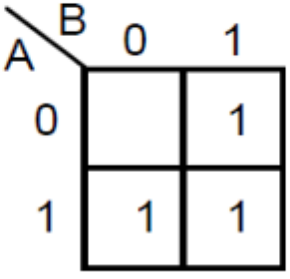
C \ AB	00	01	11	10
0	$\overline{A}\,\overline{B}\,\overline{C}$ 0	$\overline{A}\,B\,\overline{C}$ 2	$AB\,\overline{C}$ 6	$A\,\overline{B}\,\overline{C}$ 4
1	$\overline{A}\,\overline{B}\,C$ 1	$\overline{A}\,BC$ 3	ABC 7	$A\,\overline{B}\,C$ 5

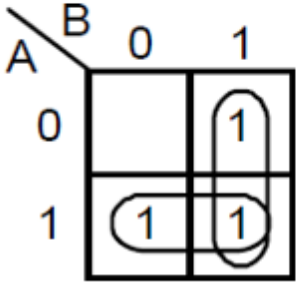
1. A group of 2 squares is called a ***pair***.
2. A group of 4 squares is called a ***quad***.
3. A group of 8 squares is called ***octet***.
4. *Using Logic Adjacency Theorem,*
5. *A* group of two squares eliminates one variable,
6. A group of four squares eliminates two variable and
7. A group of eight squares eliminates three variables.

Simplify the Boolean function for F = AB + AB′ +A′B. Using two variable K-map.

- *This function can also be written as* F(A, B) = Σ(1, 2, 3)



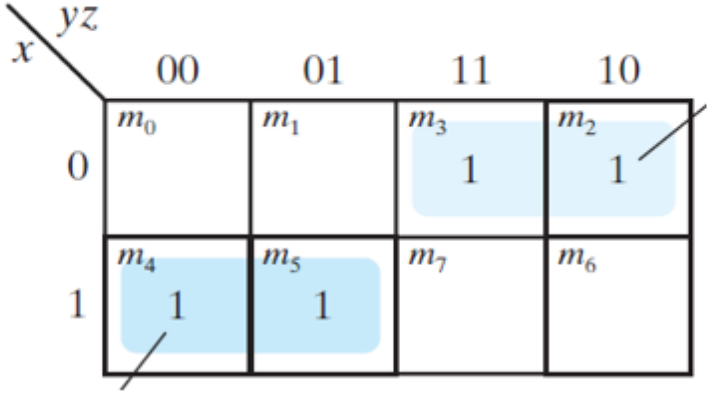




F = A + B

Simplify the function $F(x, y, z) = \Sigma(2, 3, 4, 5)$

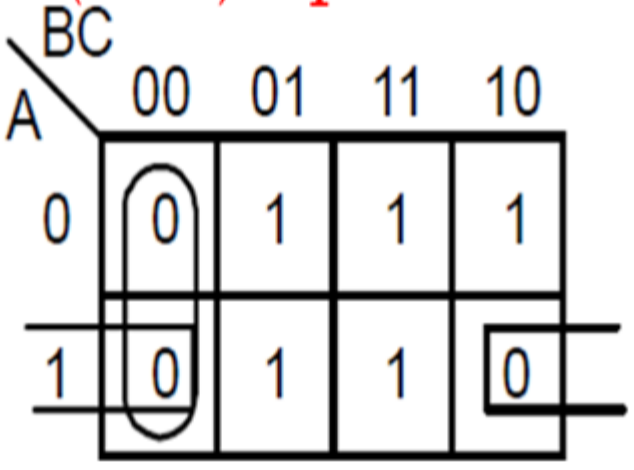
$F(x, y, z) = (2, 3, 4, 5) = x'y + xy'$



Given the following Boolean function: F = A′C + A′B + AB′C + BC

Find the simplified products of sum (POS)expression.

- Minterms 0 and 4 forms a pair, giving value = B′C′.
- Similarly minterms 4 and 6 forms a second pair giving value = AC′.
- Therefore we get **F′ = AC′ + B′C′.**



Applying De Morgan's theorem automatically con expression, giving the value of F

$$(F')' = [AC' + B'C']'$$

$$F = [(AC')' \cdot (B'C')']$$

$$\boxed{F = (A + C) \cdot (B + C)}$$

Four Variable K-Map

Four-Variable K-Map

1. A group of 2 squares is called a ***pair***.
2. A group of 4 squares is called a ***quad***.
3. A group of 8 squares is called ***octet***.
4. A group of 16 squares is called ***Hexa***.
5. *Using Logic Adjacency Theorem,*
6. *A* group of two squares eliminates one variable,
7. A group of four squares eliminates two variable and
8. A group of eight squares eliminates three variables.
9. A group of eight squares eliminates four variables.

- 4 variables Karnaugh map

Cell = 2^4= 16

AB \ CD	C'D' 00	C'D 01	CD 11	CD' 10
A'B' 00	0	1	3	2
A'B 01	4	5	7	6
AB 11	12	13	15	14
AB' 10	8	9	11	10

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

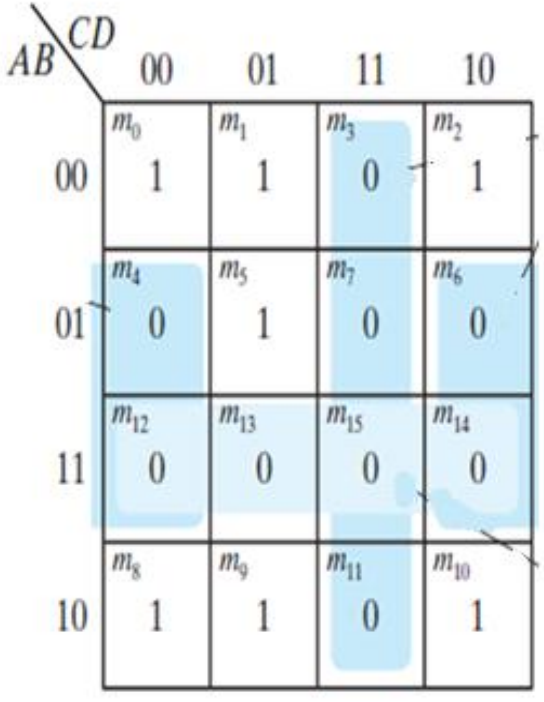
AB \ CD	00	01	11	10
00	0 A' B' C' D'	1 A' B' C' D	3 A' B' C D	2 A' B' C D'
01	4 A' B C' D'	5 A' B C' D	7 A' B C D	6 A' B C D'
11	12 A B C' D'	13 A B C' D	15 A B C D	14 A B C D'
10	8 A B' C' D'	9 A B' C' D	11 A B' C D	10 A B' C D'

Simplify the following Boolean function into

(a) sum-of-products form and

(b) product-of-sums form:

$F(A, B, C, D) = \Sigma(0, 1, 2, 5, 8, 9, 10)$

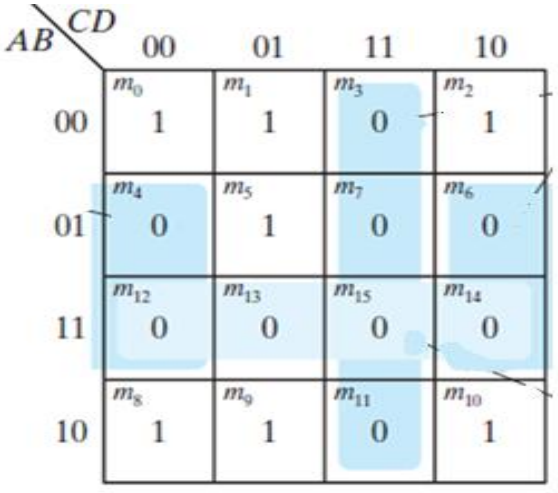


- $F(A, B, C, D) = \Sigma(0, 1, 2, 5, 8, 9, 10)$
  $= B'D' + B'C' + A'C'D$
  $= (A' + B')(C' + D')(B' + D)$

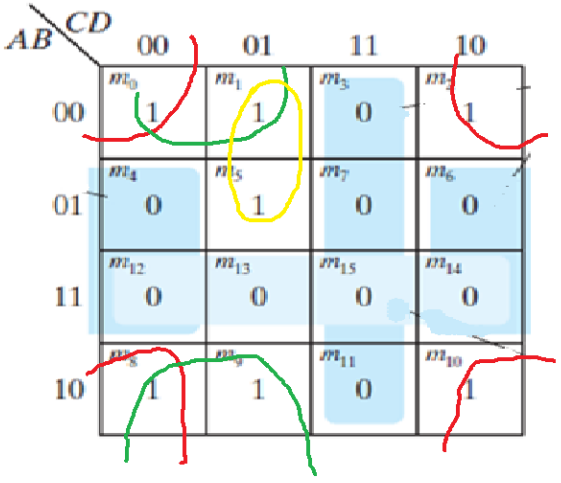
Simplify the following Boolean function into

(a) sum-of-products form and
(b) product-of-sums form:

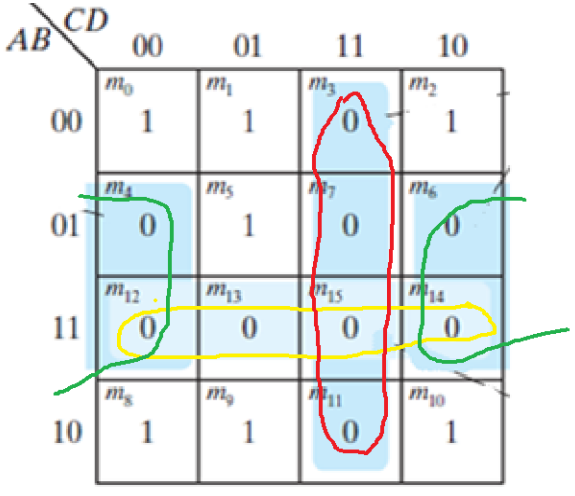
$F(A, B, C, D) = \Sigma(0, 1, 2, 5, 8, 9, 10)$



- $F(A, B, C, D) = \Sigma(0, 1, 2, 5, 8, 9, 10)$
  $= B'D' + B'C' + A'C'D$
  $= (A' + B')(C' + D')(B' + D)$

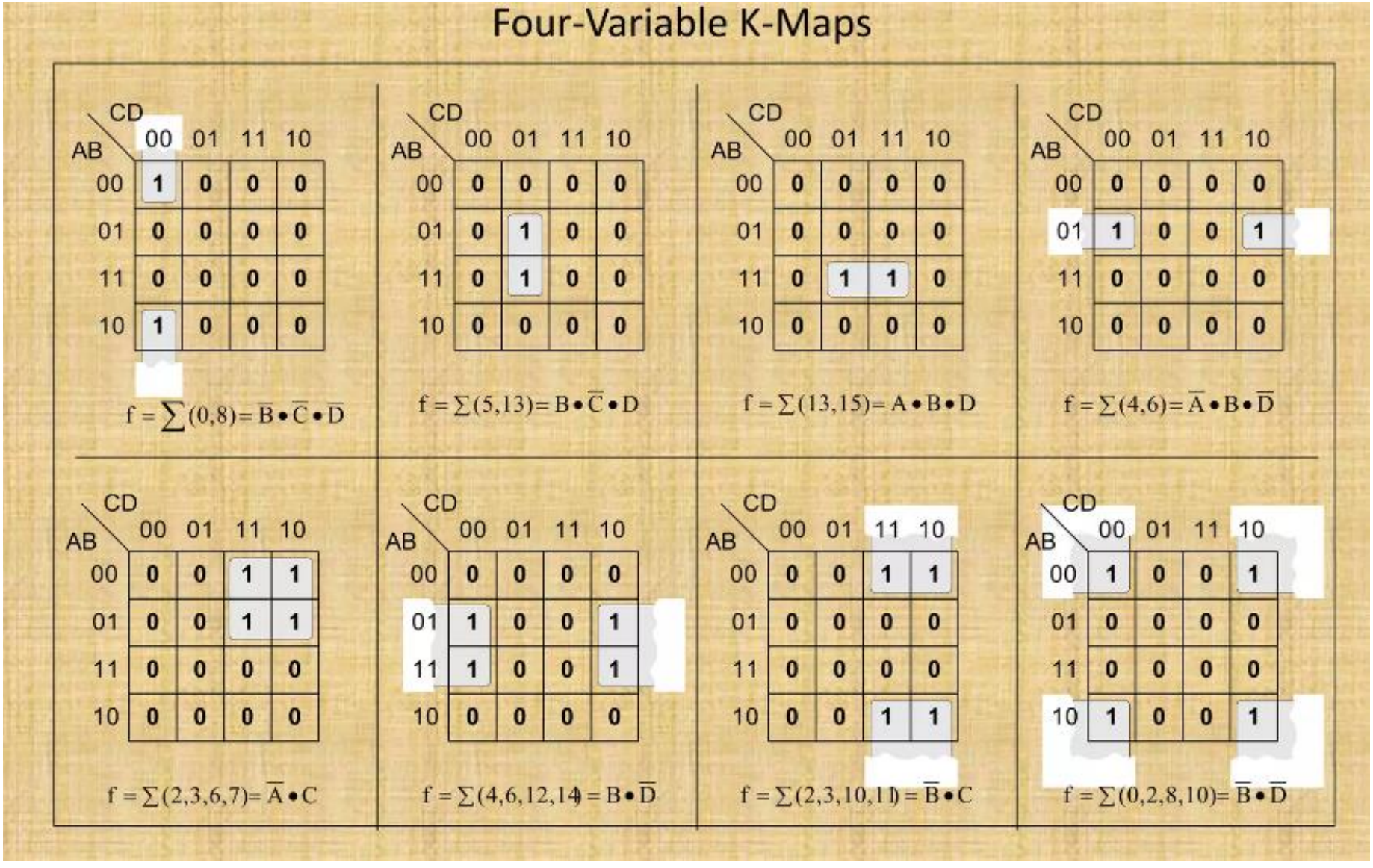


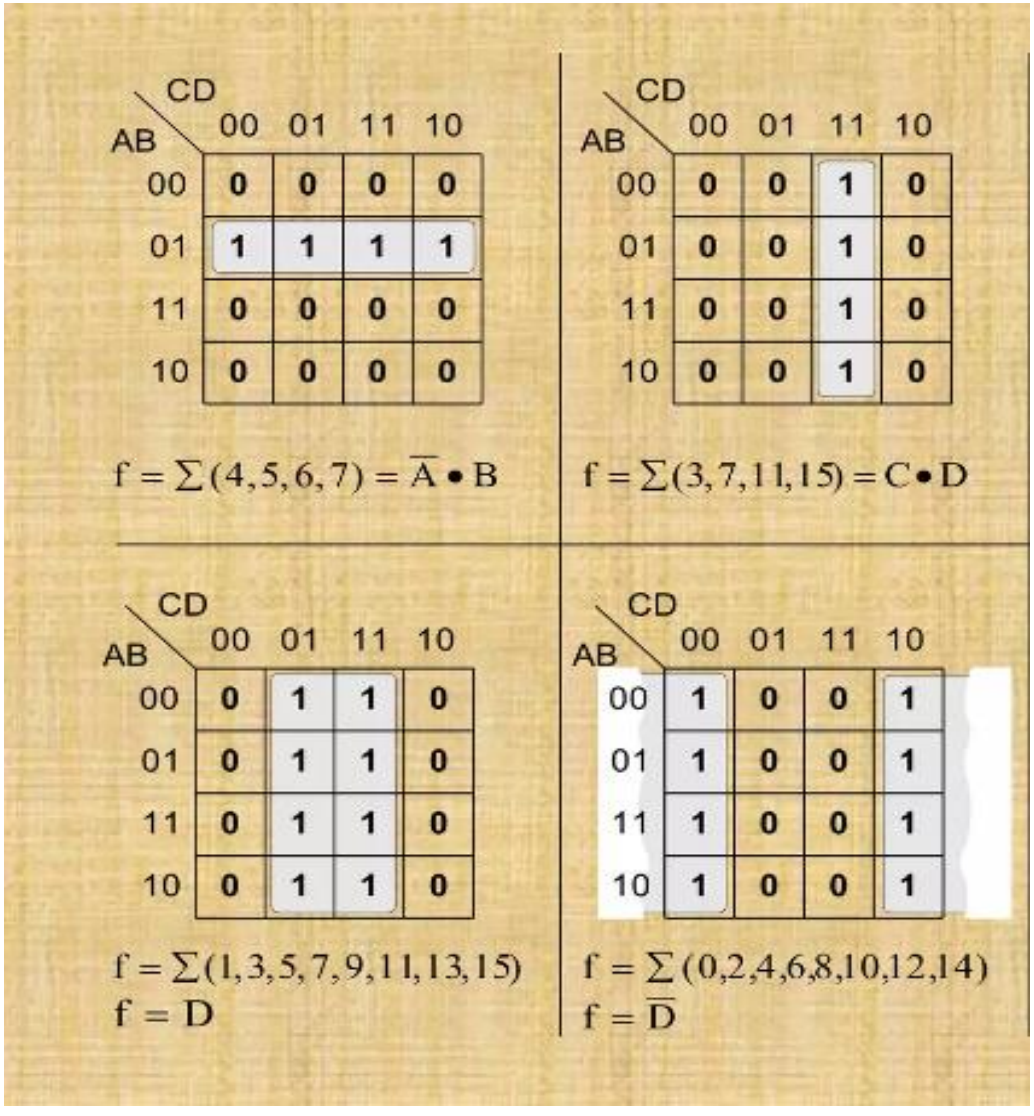
$= B'D' + B'C' + A'C'D$



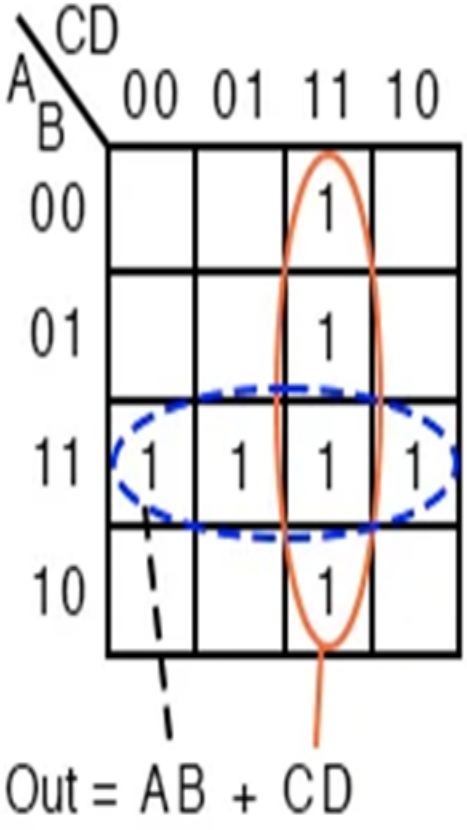
$= (A' + B')(C' + D')(B' + D)$

Four-Variable K-Maps



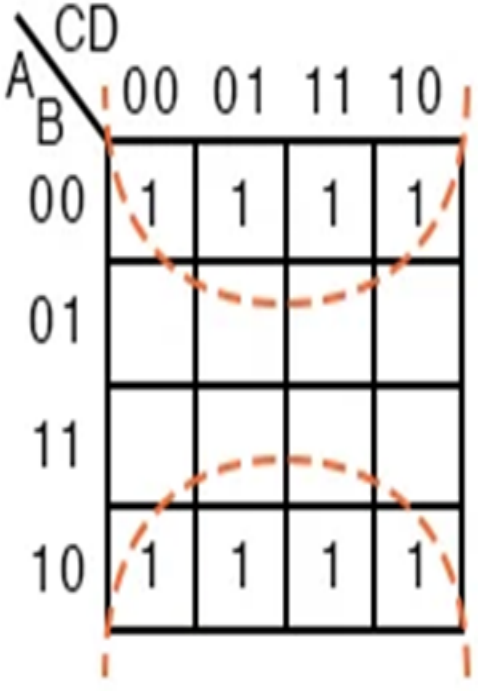

CD
AB
00 01 11 10
00 0 0 0 0
01 1 1 1 1
11 0 0 0 0
10 0 0 0 0
$f = \sum(4,5,6,7) = \overline{A} \bullet B$
CD
AB
00 01 11 10
00 0 0 1 0
01 0 0 1 0
11 0 0 1 0
10 0 0 1 0
$f = \sum(3,7,11,15) = C \bullet D$
CD
AB
00 01 11 10
00 0 1 1 0
01 0 1 1 0
11 0 1 1 0
10 0 1 1 0
$f = \sum(1,3,5,7,9,11,13,15)$
$f = D$
CD
AB
00 01 11 10
00 1 0 0 1
01 1 0 0 1
11 1 0 0 1
10 1 0 0 1
$f = \sum(0,2,4,6,8,10,12,14)$
$f = \overline{D}$

$$Out = \bar{A}\bar{B}CD + \bar{A}B\bar{C}D + ABCD + A\bar{B}CD + AB\bar{C}\bar{D} + AB\bar{C}D + ABC\bar{D}$$



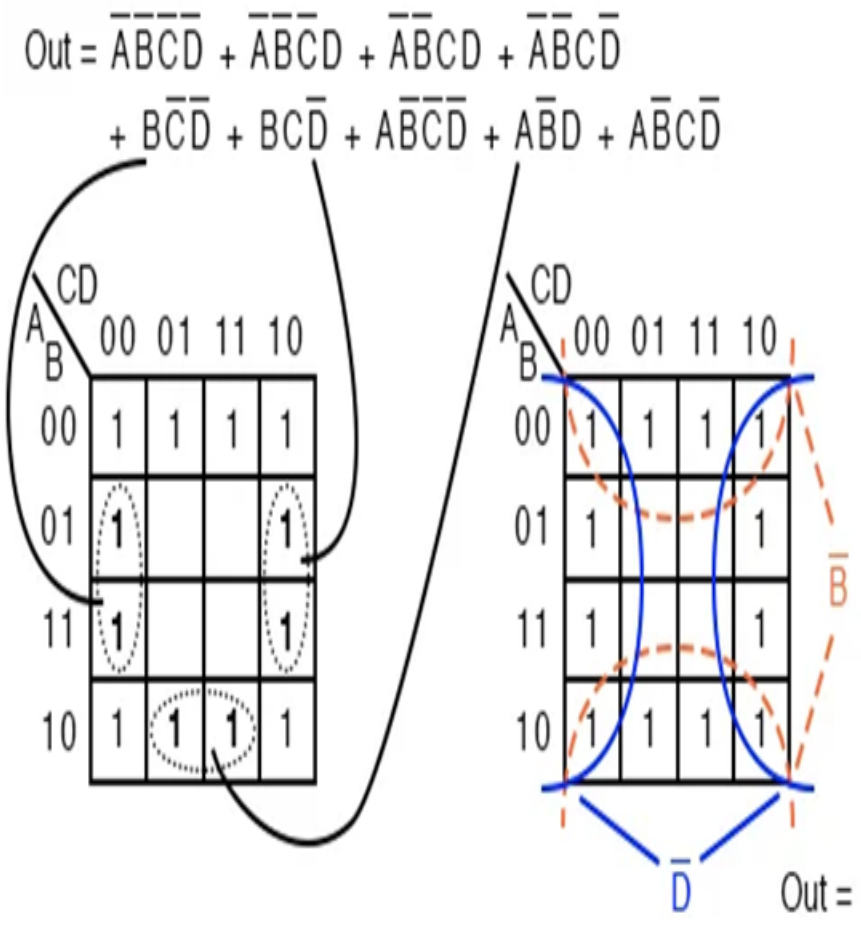
- The dashed horizontal group corresponds to the simplified product term **AB**.
- The vertical group corresponds to Boolean CD.
- Since there are two groups, there will be two product terms in the Sum-Of-Products result of **Out=AB+CD**.

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



$Out = \bar{B}$

- This group of eight has one Boolean variable in common: **B=0**.
- Therefore, the one group of eight is covered by one p-term: **B'**.
- The original eight-term Boolean expression simplifies to **Out=B'**



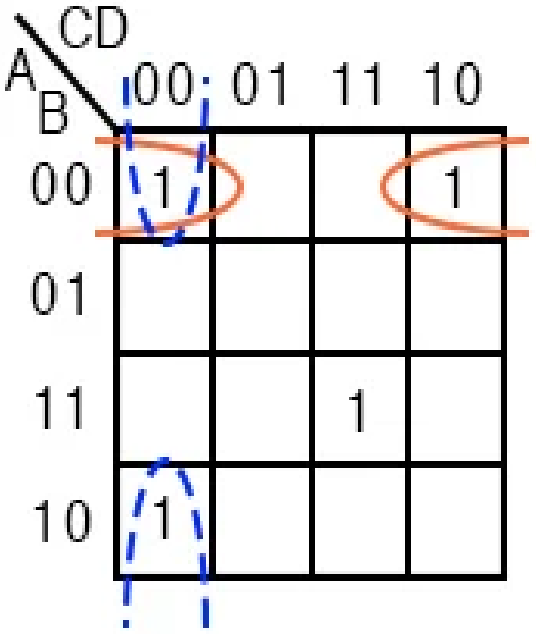
1. The six product terms of four Boolean variables map in the usual manner above as single cells. The three Boolean variable terms (three each) map as cell pairs, which is shown above.

2. Note that we are mapping p-terms into the K-map, not pulling them out at this point.

3. For the simplification, we form two groups of eight. Cells in the corners are shared with both groups. This is fine. In fact, this leads to a better solution than forming a group of eight and a group of four without sharing any cells. Final Solution is **Out=B'+D'**

Four-Variable K-Map

$$\text{Out} = \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} + \overline{A}\,\overline{B}\,C\,\overline{D} + A\,\overline{B}\,\overline{C}\,\overline{D} + A\,B\,C\,D$$

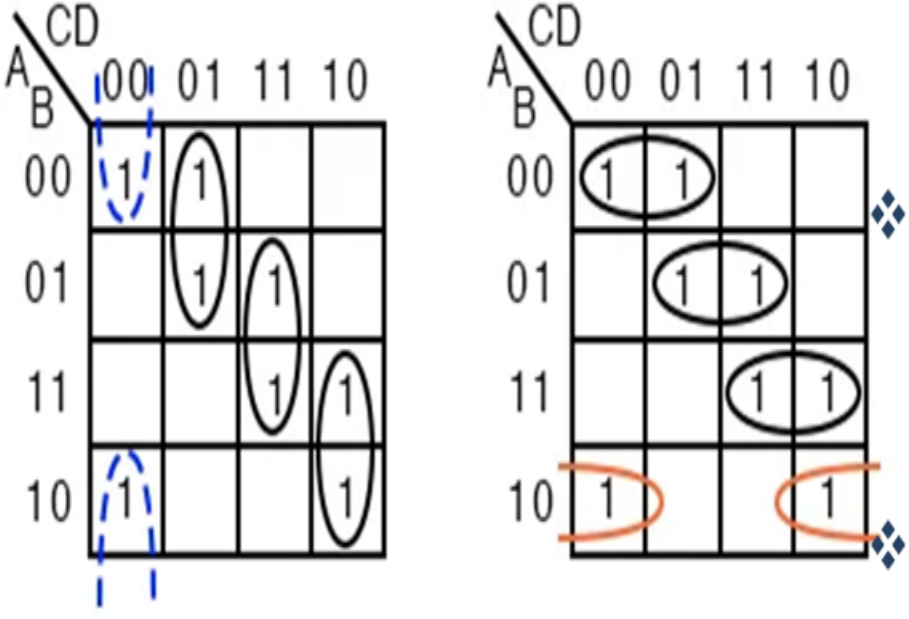


$$\text{Out} = \overline{B}\,\overline{C}\,\overline{D} + \overline{A}\,\overline{B}\,\overline{D} + A\,B\,C\,D$$

1. Above, three of the cells form into groups of two cells.
2. A fourth cell cannot be combined with anything, which often happens in "real world" problems.
3. In this case, the Boolean p-term **ABCD** is unchanged in the simplification process. Result:

Out= B'C'D'+A'B'D'+ABCD

$$Out = \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}\bar{C}D + \bar{A}\bar{B}C\bar{D} + \bar{A}BCD + ABCD + ABC\bar{D} + A\bar{B}\bar{C}\bar{D} + A\bar{B}C\bar{D}$$

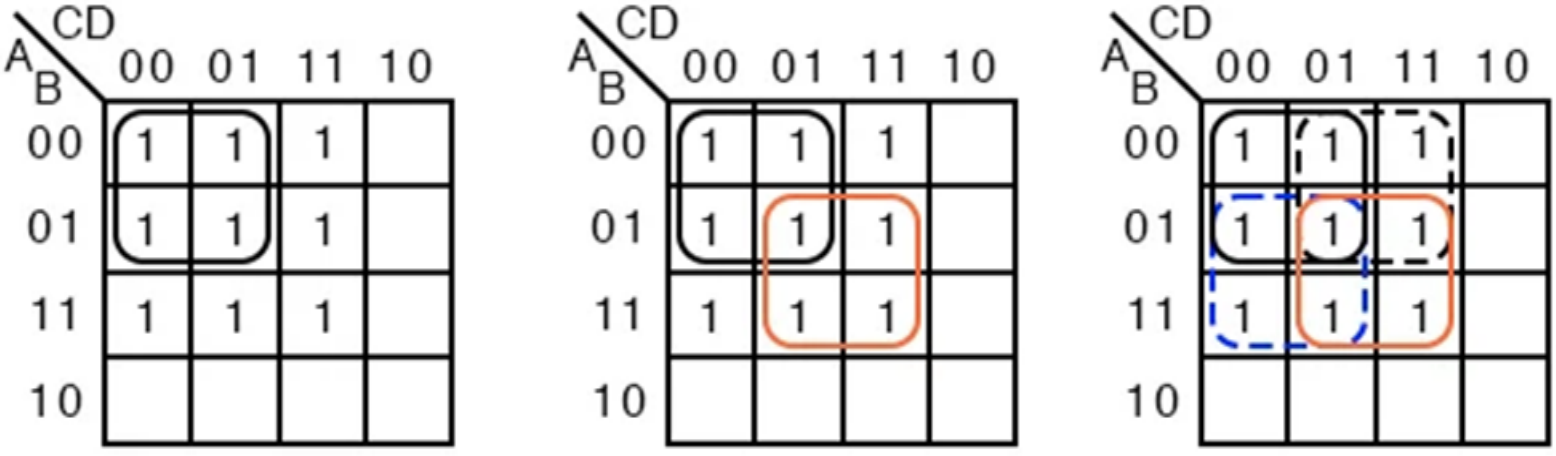


$$Out = \bar{B}\bar{C}\bar{D} + \bar{A}\bar{C}D + BCD + AC\bar{D}$$

$$Out = \bar{A}\bar{B}\bar{C} + \bar{A}BD + ABC + A\bar{B}\bar{D}$$

- Both results above have four product terms of three Boolean variable each. Both are equally valid *minimal cost* solutions.
- The difference in the final solution is due to how the cells are grouped as shown above.
- A minimal cost solution is a valid logic design with the minimum number of gates with the minimum number of inputs

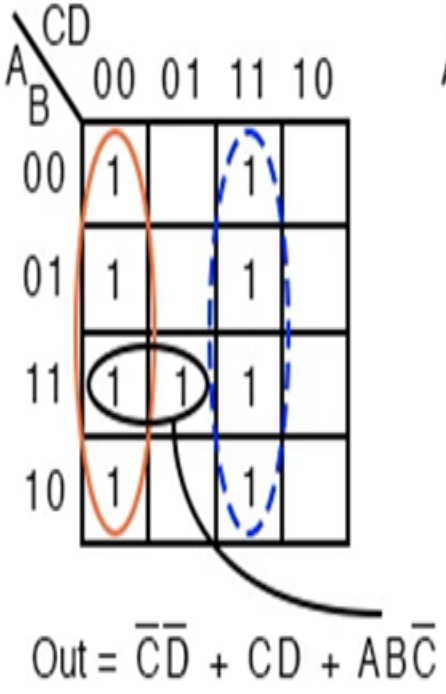
$$Out = \overline{A}\overline{B}\overline{C}\overline{D} + \overline{A}\overline{B}\overline{C}D + \overline{A}\overline{B}CD + \overline{A}B\overline{C}\overline{D} + \overline{A}B\overline{C}D + \overline{A}BCD + AB\overline{C}\overline{D} + AB\overline{C}D + ABCD$$



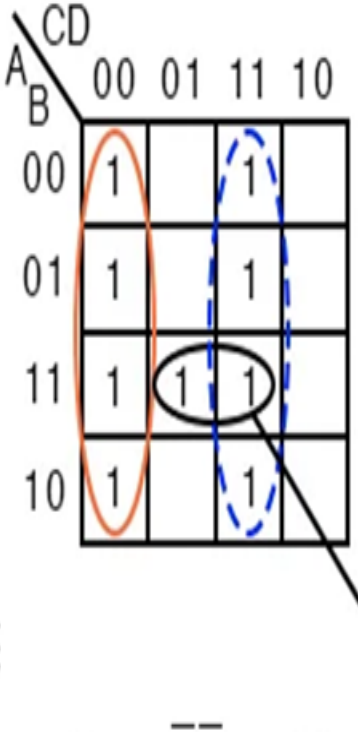
$$Out = \overline{A}\overline{C} + \overline{A}D + B\overline{C} + BD$$

- There are still two cells remaining. the minimal cost method to pick up those is to group them with neighboring cells as groups of four as at above right.
- On a cautionary note, do not attempt to form groups of three. Groupings must be powers of 2, that is, 1, 2, 4, 8 ...

$$Out = \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}CD + \bar{A}B\bar{C}\bar{D} + \bar{A}BCD + AB\bar{C}\bar{D} + AB\bar{C}D + ABCD + A\bar{B}\bar{C}\bar{D} + A\bar{B}CD$$

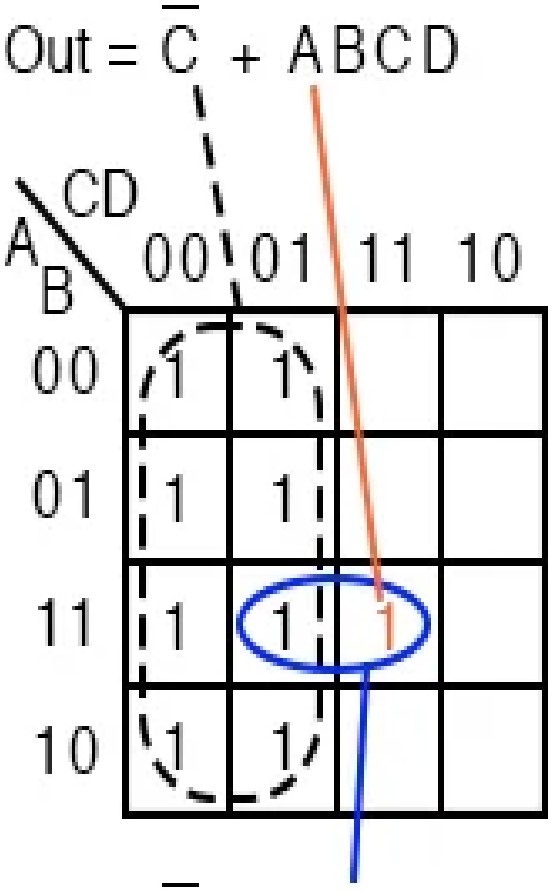


$Out = \bar{C}\bar{D} + CD + AB\bar{C}$



$Out = \bar{C}\bar{D} + CD + ABD$

- The two solutions depend on whether the single remaining cell is grouped with the first or the second group of four as a group of two cells.
- That cell either comes out as either **ABC'** or **ABD**, your choice.
- Either way, this cell is covered by either Boolean product term. Final results are shown above.



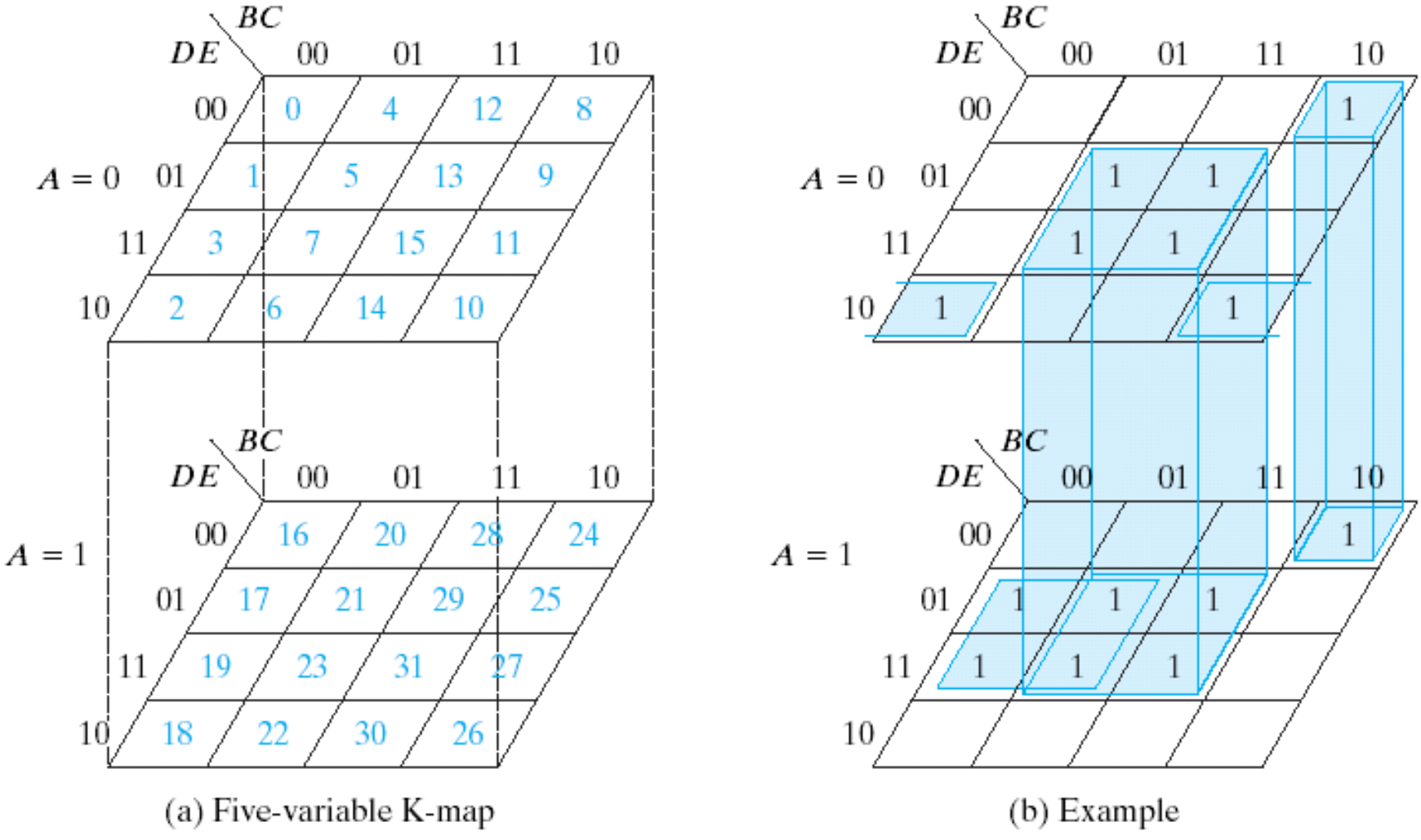
Simplification by Boolean Algebra

Out = $\overline{C}$ + ABCD

Applying rule **A + $\overline{A}$B = A + B** to the $\overline{C}$ + ABCD term

Out = $\overline{C}$ + ABD

5 Variable K-Map



(a) Five-variable K-map

(b) Example

Five-variable K-map and example.

How many k-map is needed?

- If you have 5 variables, you'll need 2 k-map…
- Let's say the variables are A, B, C, D and E.

E = 0

A B \ C D	0 0	0 1	1 1	1 0
0 0	0	1	3	2
0 1	4	5	7	6
1 1	12	13	15	14
1 0	8	9	11	10

How many k-map is needed?

E = 1

AB \ CD	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11	12	13	15	14
10	8	9	11	10

Try this out…

- Simplify the Boolean function F(A,B,C,D,E) = ∑(0,1,4,5,16,17,21,25,29)

  Soln: F(A,B,C,D,E) = A'B'D'+AD'E+B'C'D'

1st step – convert the minterm into Boolean equation

- F(A,B,C,D,E) = $\sum$(0,1,4,5,16,17,21,25,29)
- How to convert … ??

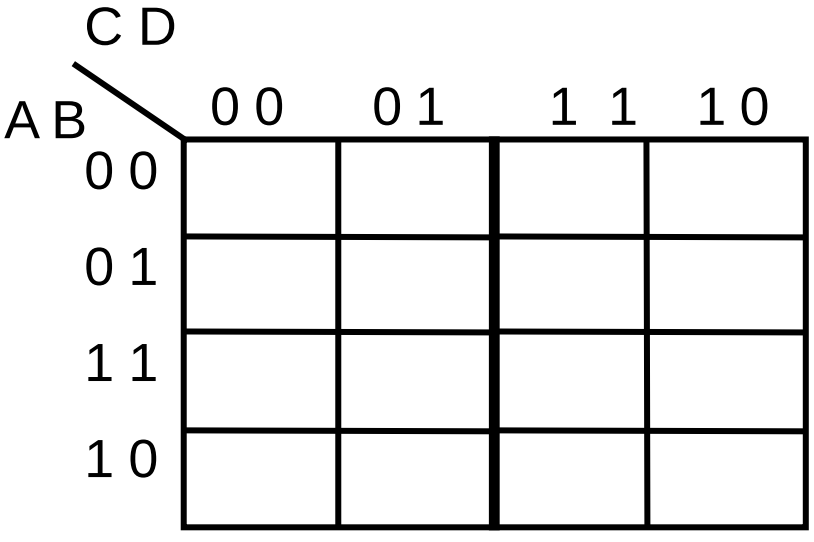
- 0 = 00000 ➔ $\overline{A}\,\overline{B}\,\overline{C}\,\overline{D}\,\overline{E}$
- 1 = 00001 ➔ $\overline{A}\,\overline{B}\,\overline{C}\,\overline{D}E$

1st step – convert the minterm into Boolean equation

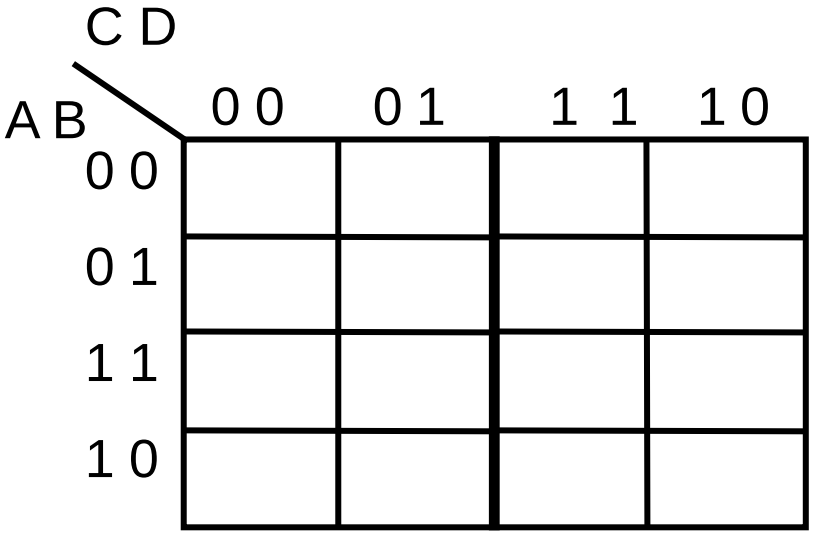
- F(A,B,C,D,E) = $\sum$(0,1,4,5,16,17,21,25,29)

$$F = \bar{A}\bar{B}\bar{C}\bar{D}\bar{E} + \bar{A}\bar{B}\bar{C}\bar{D}E + \bar{A}\bar{B}C\bar{D}\bar{E} +$$
$$\bar{A}\bar{B}CD\bar{E} + A\bar{B}\bar{C}\bar{D}\bar{E} + A\bar{B}\bar{C}D\bar{E} +$$
$$A\bar{B}C\bar{D}\bar{E} + AB\bar{C}\bar{D}E + ABC\bar{D}\bar{E}$$

2nd – prepare 2 k-map

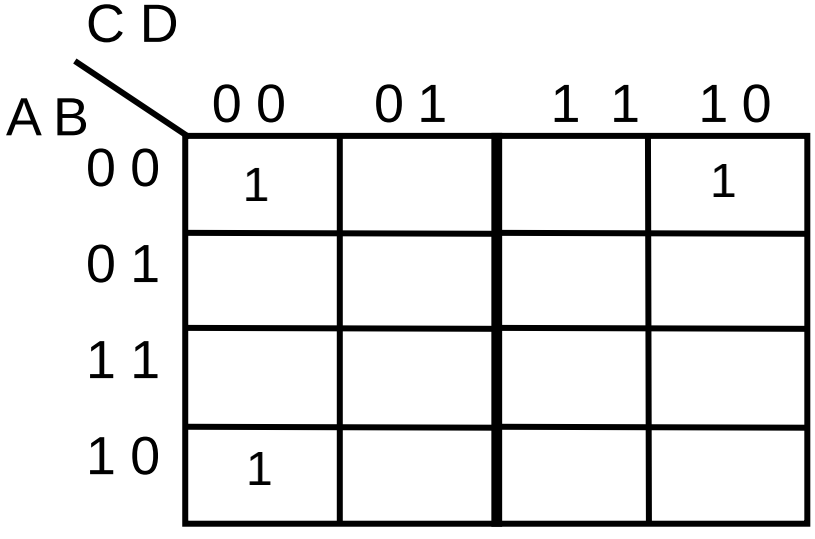


E = 0

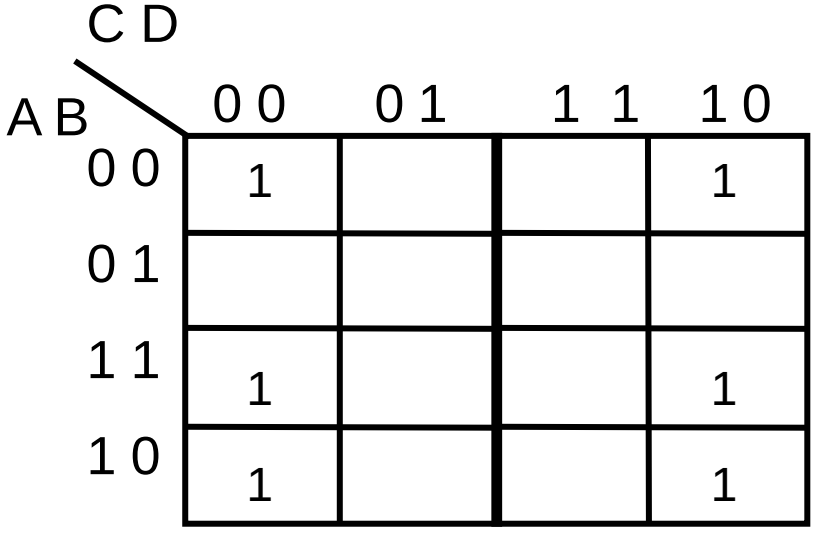


E = 1

3rd- plug in the Boolean term/minterm into k-map

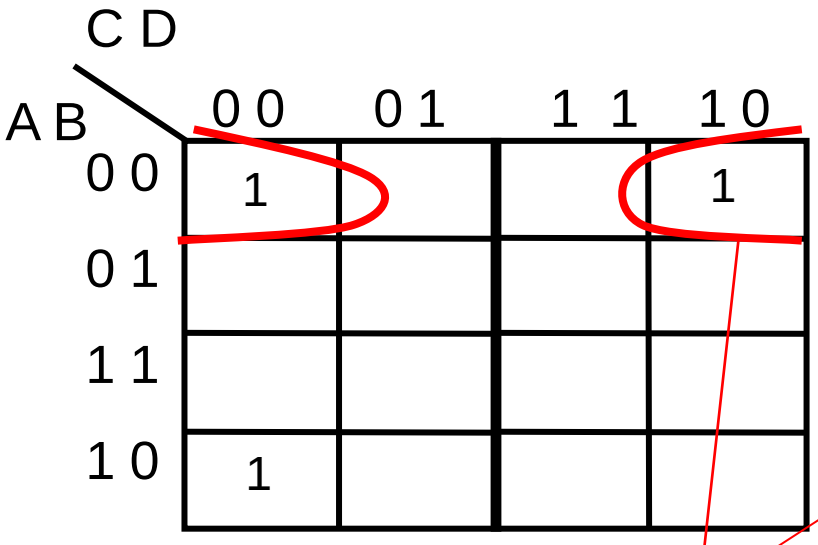


E = 0

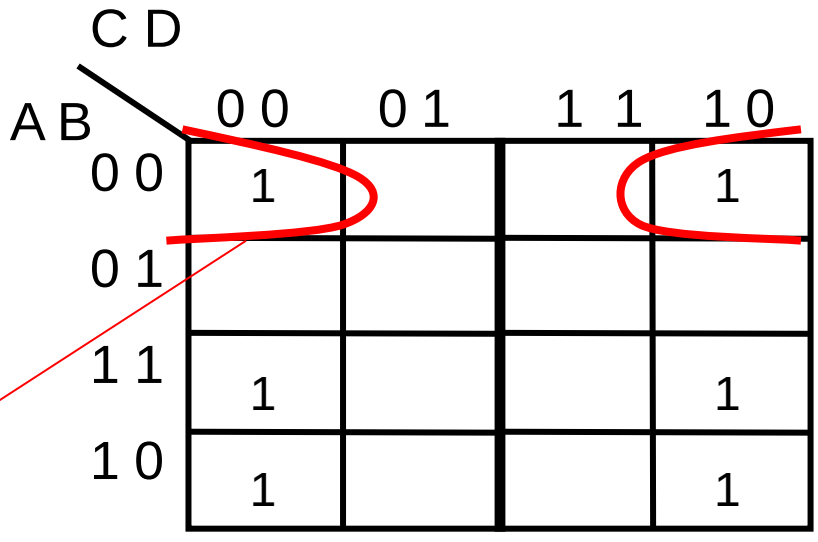


E = 1

4th- look for similar grouping that can be done in both k-map



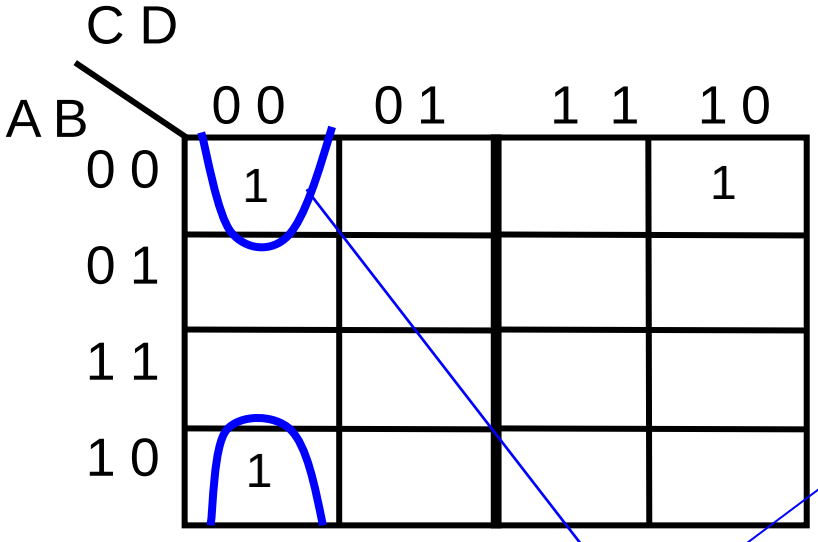
E = 0



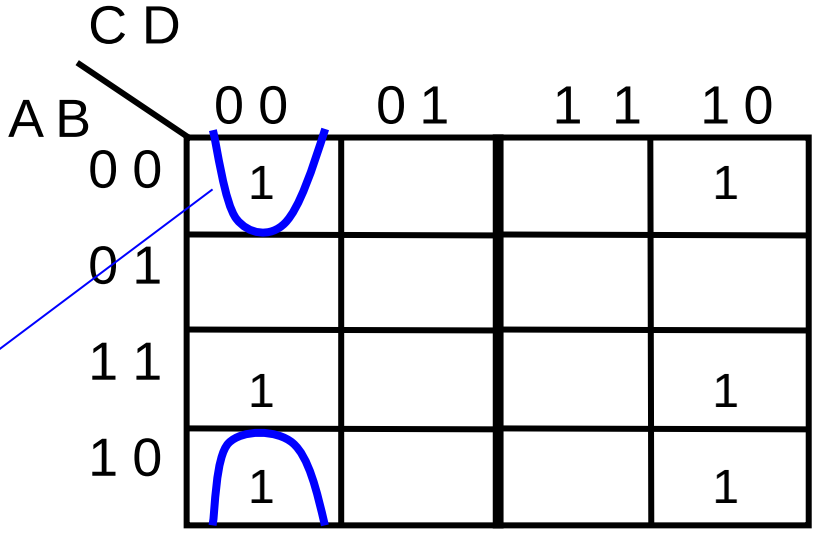
E = 1

That will give us $\overline{A}\overline{B}\overline{D}$

4th- look for similar grouping that can be done in both k-map



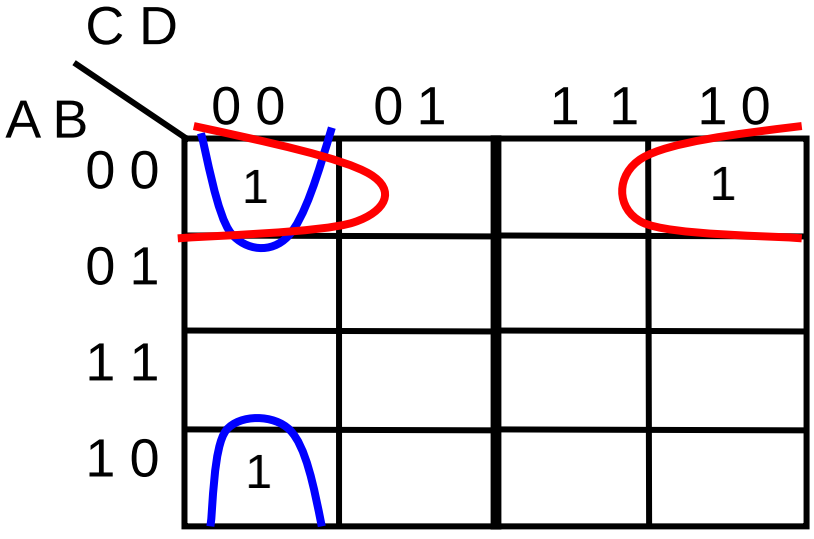
E = 0



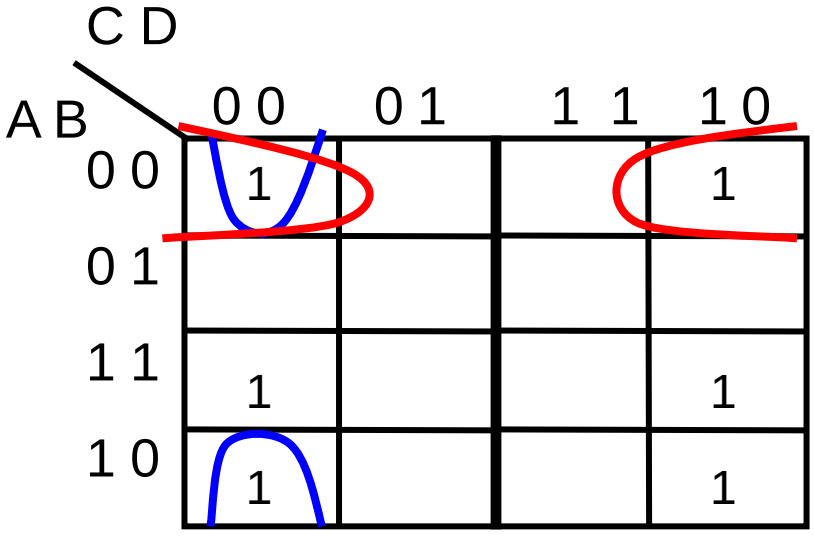
E = 1

That will give us $\overline{B}\overline{C}\overline{D}$

The combination will look like this…



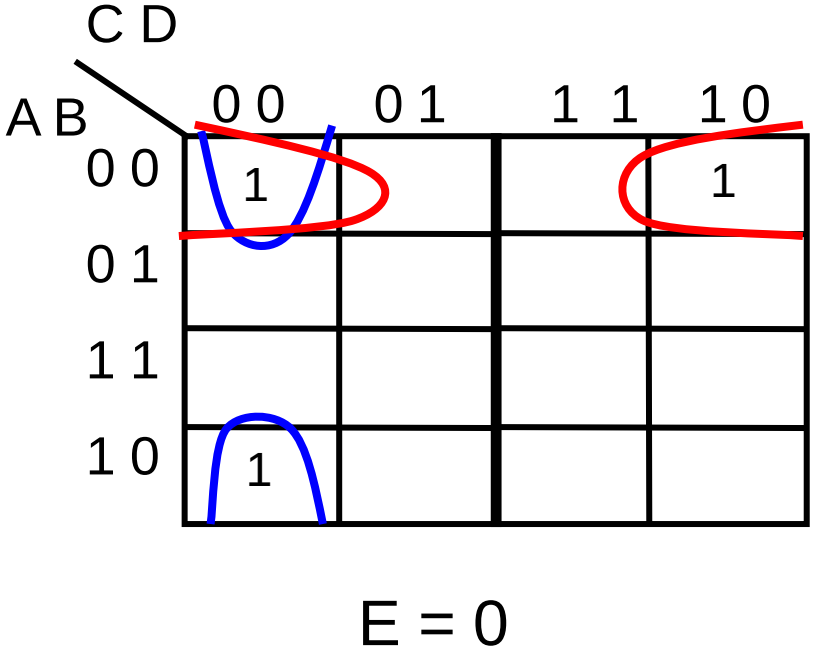
E = 0



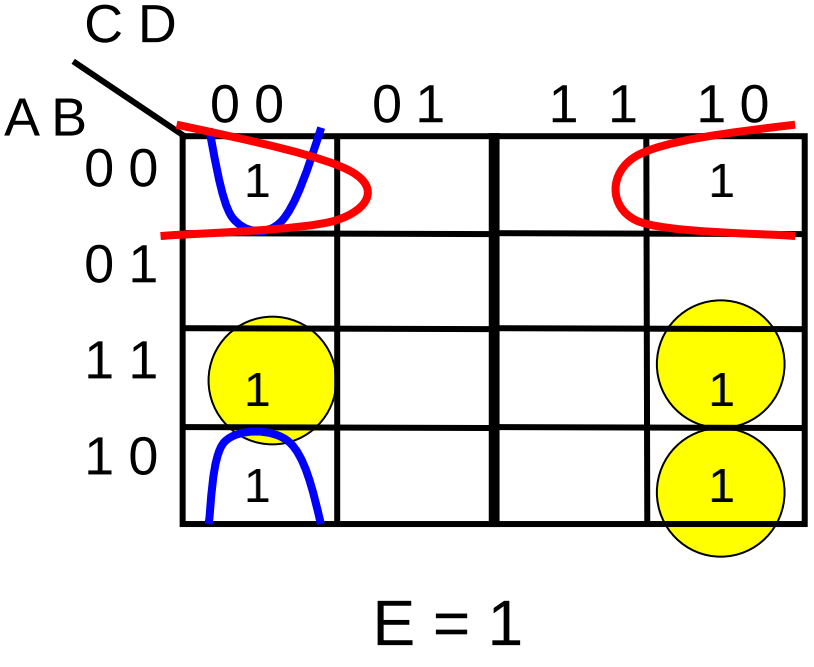
E = 1

That will give us $\overline{A}\overline{B}D + \overline{B}\overline{C}D$

5th- look for the remaining 1's that has not been included yet

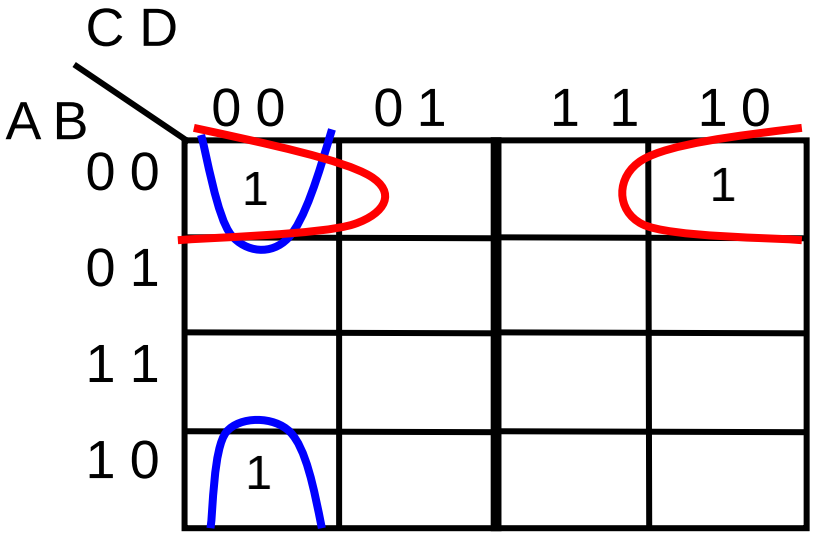


E = 0

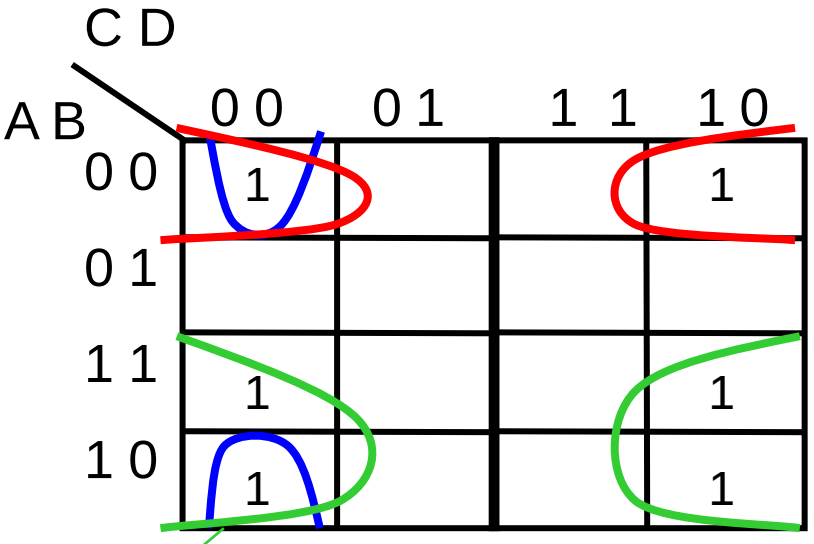


E = 1

6th- group the remaining 1's



E = 0



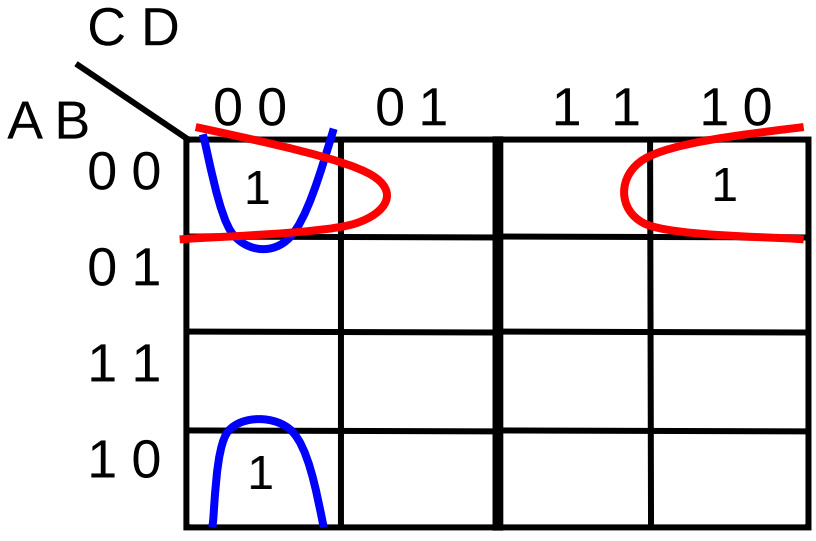
$A\bar{D}E$

E = 1

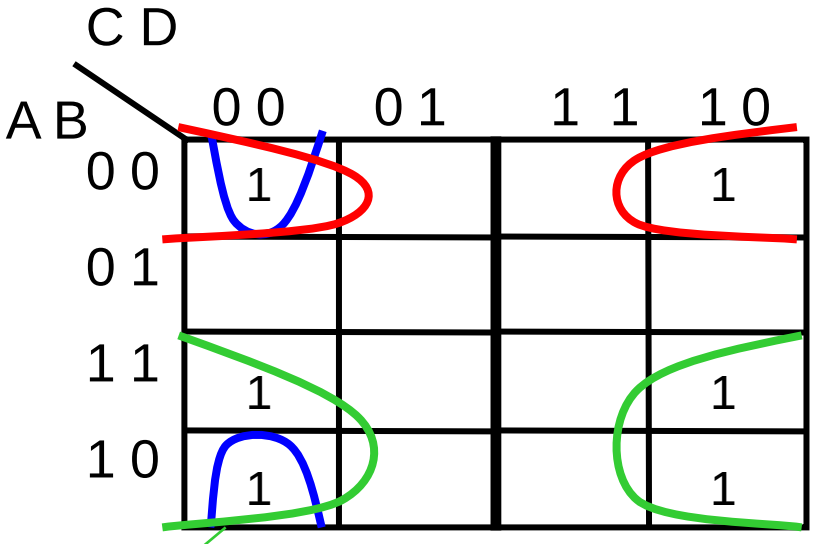
That new grouping will give us

Note that this time, E need to be included in the term, since the grouping is in the E = 1 k-map only.

6th- group the remaining 1's



E = 0



E = 1

Full expression will be,

$$F = \overline{A}\,\overline{B}\,\overline{D} + \overline{B}\,\overline{C}\,\overline{D} + A\,\overline{D}\,E$$