2026

Ammar Télidji University UATL

Faculty of Sciences

Department of Computer Science

Machine Structure 1

Course and Exercises

MOHAMED EL AMINE AMEUR

| me.ameur@lagh-univ.dz

Foreword

Dear Students,

This course manual has been developed to support you throughout the **"Machine Structure 1"** module. It serves as a structured pedagogical resource designed to facilitate the progressive acquisition of both fundamental and advanced concepts in computer architecture a core discipline within your academic training in computer science.

The organization of this document follows a methodical and incremental approach. Theoretical concepts are systematically reinforced through detailed explanations, illustrative examples, and practical exercises, thereby establishing a clear connection between abstract principles and their concrete applications.

A thorough understanding of machine structure constitutes a fundamental pillar of computer science. It underpins system performance, optimization strategies, and technological innovation. I therefore encourage you to approach this module with intellectual curiosity, rigor, and perseverance.

This material is intended to serve as a reference throughout your studies. It should be complemented by your personal notes and further exploration of the topics addressed. I remain at your disposal for any clarification or further guidance you may require.

I wish you every success in this module and in your academic journey.

Sincerely,

1. Dedication

With heartfelt appreciation, I dedicate this pedagogical work to my parents, whose unwavering support and encouragement have been the foundation of my academic and professional journey.

I also dedicate it in loving memory of my father, whose profound wisdom and guidance continue to resonate within me. His steadfast belief in the pursuit of knowledge was the guiding force that led me toward higher studies and inspired my commitment to sharing that knowledge with others.

This work is in honor of you, Messaoud Ameur. Your legacy continues to inspire me.

2. Acknowledgements

First and foremost, I express my deepest gratitude to Allah for granting me the strength and perseverance to complete this work. All praise is due to Allah, the Lord of the Worlds.

I would like to extend my sincere appreciation to Pr. Nasreddine Lagraa, Pr. Noureddine Chaib, and Dr. Fatima Zohra Bousbaa for their kindness in serving on the review committee. I am truly grateful for the time, expertise, and effort they dedicated to evaluating this pedagogical material.

My thanks also go to my colleagues within the Department of Computer Science at Laghouat University. Their support, along with the encouragement of the friends I have made throughout my academic journey, has been invaluable.

Finally, I wish to express my profound gratitude to my family for their unwavering love and support.

3. Table of Contents

4. List of Tables and Figures:

Figures:

First Part Course Summary

5. Chapter 1: General Introduction

Basic Definitions, Binary Representation, and Units of Measurement

1. Introduction

Computer science is a fundamental discipline that studies the representation, processing, and transmission of information. Modern society relies heavily on computers for communication, data storage, automation, and problem solving. This chapter introduces the basic concepts required to understand how computers handle information.

2. Basic Definitions

- **Information** is a set of data that has meaning and can be understood by humans.
- **Computer science** is the set of sciences and technologies concerned with the automatic processing of information.
- A **computer** is a machine capable of performing the automatic processing of information.

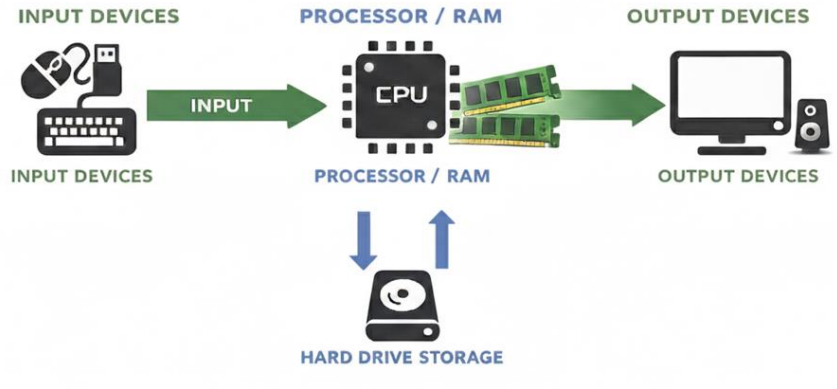


Figure 5-1 The Computer as an Information Processing Machine

A computer system consists of hardware and software components working together.

1-Hardware

Hardware refers to the physical components of a computer system, such as the CPU, memory, storage devices, and input/output devices.

2-Software

Software includes the programs and instructions that control hardware operations.

3. Units of Information

In computer science, information is quantified according to the number of binary elements used to represent it. These elements correspond to the physical binary states employed by digital systems and form the basis for measuring data size and storage capacity.

a) **Bit (Binary Digit):** The bit is the smallest unit of information. It can take one of two possible values: 0 or 1. we can compare a Bit to a simple electric light bulb (ampoule). Just like a computer circuit, a light bulb can only be in one of two distinct states.

1. OFF (0)  : The switch is open, no current flows, and the bulb is dark. This represents the binary digit 0.
2. ON (1)  : The switch is closed, current flows (High Voltage), and the bulb is lit. This represents the binary digit 1.

b) **Byte (Octet):** A byte is a group of 8 bits ( ). It is a fundamental unit used to represent characters, small numbers, and basic data elements.

One byte is capable of representing 2^8 (or 256) different values.

 value 0

...

 value255

c) **Word:** A word is a unit of data composed of two bytes, which is equivalent to 16 bits. Historically, the term "16-bit computer" referred to machines that processed data in 16-bit word increments.

4. The Universality of Binary Representation

The information processed by a computer comes in various types, such as **text**, **numbers**, **images**, and **sound**. Despite this diversity, these data types are always represented and manipulated by the computer in binary form.

a) How Different Data Types Become Binary:

- **Text:** Information encoding establishes a correspondence between external characters (like the letter 'A') and an internal sequence of bits. For example, the number 36 is an external representation, while its internal representation is the bit sequence `100100`.
- **Numbers:** Computers primarily use binary to represent numbers using the digits 0 and 1.
- **Images & Sound:** These complex data types are also converted into binary. Images are broken down into pixels, and sound into numerical samples, all of which are stored as sequences of 0s and 1s.

b) Why This Single Form?

Technically, it is easy to encode two different values using electrical voltage. Electronic circuits are designed to distinguish between two clear states:

- **High Voltage (e.g., 5V):** Represents the value **1**. 
- **Low Voltage (e.g., 0V):** Represents the value **0**. 

By using only two states, the computer becomes highly resistant to "noise" or slight fluctuations in electricity. If the system used ten different voltages to represent the decimal digits (0–9), a small drop in power might cause the computer to mistake a "9" for an "8," leading to massive errors. With binary, a signal is either clearly "on" or "off."

This uniformity allows the **Central Processing Unit (CPU)** to use the same basic circuits (the **ALU** and **Control Unit**) to process any type of information, whether it's a mathematical calculation or a line of text.

5. Information Encoding

Encoding is essentially the "translation" layer of computing. Since computers operate using billions of tiny electronic switches (transistors) that are either **on** or **off**, every piece of data whether it's a photo, a song, or a number must be converted into a sequence of bits (0s and 1s).

a) How Numeric Encoding Works

When you enter a decimal number like **36**, the computer doesn't "see" the digits 3 and 6. Instead, it uses a positional number system based on powers of 2.

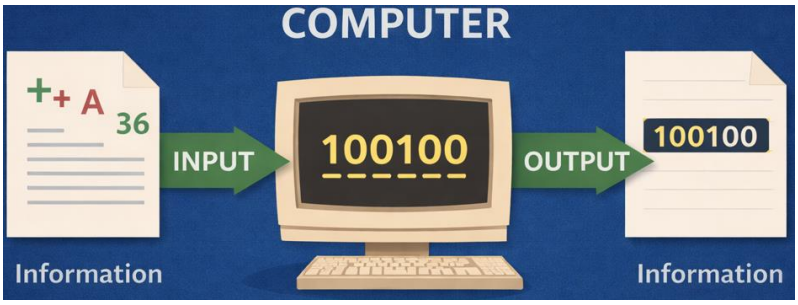


Figure 5-2 Information Encoding and Processing in a Computer

Breaking Down the Example: 36_{10}

To get the binary equivalent **1**00**1**00, the computer calculates which powers of 2 sum up to 36:

- **32** (2^5) = 1
- **16** (2^4) = 0
- **8** (2^3) = 0
- **4** (2^2) = 1
- **2** (2^1) = 0
- **1** (2^0) = 0

Summing them up: 32 + 4 = 36.

Memorizing at least the first ten powers of 2 can significantly reduce the time and effort required for calculations;

6. Units of Measurement

a) Capacity / Size

These units are mainly used to measure the size of memory (cache, RAM) and storage devices (hard disks, SSDs, USB drives).

- bit (b) is the smallest unit of information and can be either 0 or 1.
- Byte (B) equals 8 bits.

The byte is commonly used with multiples:

Unit	Value
Byte (B)	8 bits
Kilobyte (KB)	1024 Bytes (2^{10})
Megabyte (MB)	1024 KB (2^{20})
Gigabyte (GB)	1024 MB (2^{30})
Terabyte (TB)	1024 GB (2^{40})

Table 5-1 Binary Digital Storage Units

Remember: **B = Byte** (octet) ,**b = bit** ,They are different.

b) Large Data Storage Units (High-Capacity Units)

As global data continues to grow, new terms are used to describe massive scales of information. While most users stop at the Terabyte (TB), computer science also recognizes much larger units.

- **Brontobyte (BB):** This unit is not yet officially standardized but is used to describe massive data sets.
- **Scale of a Brontobyte:** * **Decimal Scale:** Approximately 10^{27} bytes, or 1,000,000,000,000,000,000,000,000,000 bytes.

- **Binary Scale:** Exactly 2^{90} bytes, which equals 1,237,940,039,285,380,274,899,124,224 bytes.

Table 5-2 presents large scale digital storage units in both decimal and binary forms

Unit Name	Symbol	Decimal Value (Base 10)	Binary Value (Base 2)
Terabyte	TB	10^{12} bytes	2^{40} bytes
Petabyte	PB	10^{15} bytes	2^{50} bytes
Exabyte	EB	10^{18} bytes	2^{60} bytes
Zettabyte	ZB	10^{21} bytes	2^{70} bytes
Yottabyte	YB	10^{24} bytes	2^{80} bytes
Brontobyte	**BB**	**10^{27}bytes**	**2^{90} bytes**

Table 5-2 Large Scale Digital Storage Units, Decimal and Binary Representations

c) Data Transfer Rate / Speed

Speed is measured in bits per second (bps). It is used for network connections, modems, and Internet speeds (see Table 5-3).

Unit	Meaning	Value
bps	bit per second	base unit
Kbps	kilobit/s	1024 bps (≈ 10^3)
Mbps	megabit/s	1024 Kbps
Gbps	gigabit/s	1024 Mbps

Table 5-3 Data Transmission Rate Units and Their Binary Multiples

You may also see **Bps** (Bytes per second).

1 Byte/s = 8 bit/s.

Example: A connection of **100 Mbps** corresponds to about **12.5 MB/s**.

d) Frequency

Frequency represents the number of cycles or events per second.

It is used to measure CPU clock speed, bus frequency, RAM frequency, and screen refresh rate. (see Table 5-4).

Unit	Value
Hertz (Hz)	1 per second
Kilohertz (KHz)	1,000 Hz (10^3)
Megahertz (MHz)	1,000 KHz = 10^6 Hz
Gigahertz (GHz)	1,000 MHz = 10^9 Hz

Table 5-4 Frequency Units and Decimal Multiples

7. Conclusion

Chapter 1 has established the fundamental principles that enable a computer to function as an automated information-processing machine. By introducing the core concepts of information, computers, hardware, and software, this chapter has clarified how data is represented, processed, and stored within a digital system. Particular emphasis was placed on binary representation and numeric encoding, highlighting why computers rely on base-2 systems and how information is transformed from human-readable form into internal binary structures.

This introductory chapter provides the essential vocabulary and structural understanding required for further study in computer science. With these foundations in place, the reader is now prepared to explore the technical aspects of numbering systems in greater depth. Chapter 2 will build directly on these concepts by focusing on the mathematical representation of numbers and mastering the conversions between decimal, binary, octal, and hexadecimal systems.

6. Chapter 2: Numbering Systems

Base Systems, Conversions, and Basic Binary Arithmetic

1. Introduction

In our daily lives, we use the Decimal system (Base 10), likely because humans have ten fingers. However, electronic circuits and computer processors operate using electrical signals that are either "ON" or "OFF." This physical reality requires a different mathematical approach: the Binary system (Base 2).

Understanding how data is structured at this fundamental level is the first step in mastering computer architecture. This chapter explores how we represent values using different bases, how to translate between them, and how the machine performs calculations using only two digits.

2. Fundamental Definitions

A numbering system is a mathematical way of representing numbers using a specific set of symbols. The base (or radix) determines how many unique symbols are used and the value of each position in a number.[1]

a) General Representation of a Number

A real number N expressed in base b is written as:

$$N = [a_{n-1}\ a_{n-2} \dots a_1\ a_0\ ,\ a_{-1}\ a_{-2} \dots a_{-m}]_{(\beta)}$$

This representation is divided into two parts:

• Integer part: a_{n-1} to a_0

• Fractional part: a_{-1} to a_{-m}

b) Mathematical Value of the Number

The numerical value of N is given by:

$$N = a_{n-1} \times b^{n-1} + a_{n-2} \times b^{n-2} + \ldots + a_1 \times b^1 + a_0 \times b^0$$
$$+ a_{-1} \times b^{-1} + a_{-2} \times b^{-2} + \ldots + a_{-m} \times b^{-m}$$

b : base of the numbering system

a_i : digit such that $0 \leq a_i < b$

b^i : positional weight of the digit a_i

Example (Decimal System)

$$\mathbf{(245.37)_{10}} = 2 \times 10^2 + 4 \times 10^1 + 5 \times 10^0 + 3 \times 10^{-1} + 7 \times 10^{-2}$$
$$= 245.37$$

3. Main Base Systems

In computer science, **four primary** numbering systems are used to represent data. Each system serves a specific purpose, ranging from human interaction to low-level hardware communication.

1) Decimal (Base 10)

The Human Standard: This system is used in everyday life.

Symbols: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}

Principle: Once the digit 9 is reached, a carry is generated to the next higher position (for example: 9 → 10, 19 → 20).

2) Binary (Base 2)

The Machine Standard: The fundamental language of computers.

Symbols: {0, 1}

Principle: Binary represents the two physical states of a transistor: 0 (Off or Low Voltage) and 1 (On or High Voltage).

Table 6.1 displays the first 16 positive binary integers (expressed as 4-bit values including leading zeros) and extends to the first 32, as these are frequently used in our work.

Decimal	Binary	4-bit	Decimal	Binary
0	0	0000	**16**	10000
1	1	0001	**17**	10001
2	10	0010	**18**	10010
3	11	0011	**19**	10011
4	100	0100	**20**	10100
5	101	0101	**21**	10101
6	110	0110	**22**	10110
7	111	0111	**23**	10111
8	1000	1000	**24**	11000
9	1001	1001	**25**	11001
10	1010	1010	**26**	11010
11	1011	1011	**27**	11011
12	1100	1100	**28**	11100
13	1101	1101	**29**	11101
14	1110	1110	**30**	11110
15	1111	1111	**31**	11111

Table 6-1 First 32 Binary Integers

3) Octal (Base 8)

Compact Representation: Used to shorten long binary strings.

Symbols: {0, 1, 2, 3, 4, 5, 6, 7}

Relationship: One octal digit corresponds exactly to three binary bits, since $2^3 = 8$.

4) Hexadecimal (Base 16)

Technical Standard: Commonly used for memory addresses, color codes (HTML/CSS), and debugging purposes.

Symbols: {0–9, A, B, C, D, E, F}

Note: Letters are used to represent values from 10 to 15:

A = 10, B = 11, C = 12, D = 13, E = 14, F = 15

Relationship: One hexadecimal digit corresponds exactly to four binary bits, since $2^4 = 16$.

Summary Reference Table

System	Base	Symbols Used	Power of 2	Typical Usage
Binary	2	0, 1	2^1	Logic gates, CPU registers
Octal	8	0 – 7	2^3	File permissions (Unix)
Decimal	10	0 – 9	N/A	Human interface
Hexadecimal	16	0 – 9, A – F	2^4	RAM addresses, IPv6

Table 6-2 Number Systems, Bases, Binary Powers, and Practical Applications

4. Conversions Between Numbering Systems

1) Any Base to Decimal (Base 10)

Method: Polynomial Expansion. Each digit is multiplied by the base B raised to the power of its position.

a) Binary to Decimal Example:

$(110.1)_2 = (1 \times 2^2) + (1 \times 2^1) + (0 \times 2^0) + (1 \times 2^{-1})$

$= 4 + 2 + 0 + 0.5 = 6.5_{10}$

b) Octal to Decimal Example:

$(17)_8 = (1 \times 8^1) + (7 \times 8^0)$

$= 8 + 7 = 15_{10}$

c) Hexadecimal to Decimal Example:

$(1A)_{16} = (1 \times 16^1) + (10 \times 16^0)$

$= 16 + 10 = 26_{10}$

2) Decimal to Any Base

Method: Successive Division for the integer part and Successive Multiplication for the fractional part.

a) Integer Part (Division)

The decimal number is divided by the target base B. The remainders obtained at each step are recorded and read from bottom to top.

Example: 13_{10} to Binary

13 ÷ 2 = 6 remainder 1 (LSB)

6 ÷ 2 = 3 remainder 0

3 ÷ 2 = 1 remainder 1

1 ÷ 2 = 0 remainder 1 (MSB)

Result: 1101_2

b) Fractional Part (Multiplication)

The fractional part is multiplied by the target base B. The integer part of the result is taken, and the process is repeated with the remaining fraction. The digits are read from top to bottom.

Example: 0.375_{10} to Binary

0.375 × 2 = 0.75 → 0

0.75 × 2 = 1.5 → 1

0.5 × 2 = 1.0 → 1 (stop)

Result: 0.011_2

3) The "Power of 2" Shortcuts (Binary, Octal, Hexadecimal)

Since $8 = 2^3$ and $16 = 2^4$, conversions between binary, octal, and hexadecimal can be performed directly without converting through the decimal system. This technique is known as the Grouping Method.

a) Binary ↔ Octal (Groups of 3)

Binary to Octal:

Binary digits are grouped into sets of three starting from the right. Each group is converted to its octal equivalent.

$(110111)_2 \rightarrow 110\ 111 \rightarrow 6\ 7 \rightarrow 67_8$

Octal to Binary:

Each octal digit is replaced by its 3-bit binary equivalent.

$52_8 \rightarrow 5 = 101_2\ ,\ 2 = 010_2 \rightarrow 101010_2$

b) Binary ↔ Hexadecimal (Groups of 4)

Binary to Hexadecimal:

Binary digits are grouped into sets of four starting from the right. Each group is converted to its hexadecimal equivalent.

$(10111100)_2 \rightarrow 1011\ 1100 \rightarrow B\ C \rightarrow BC_{16}$

Hexadecimal to Binary:

Each hexadecimal digit is replaced by its 4-bit binary equivalent.

$A1_{16} \rightarrow A = 1010,\ 1 = 0001 \rightarrow 10100001_2$

c) Octal ↔ Hexadecimal

Method: Always pass through the binary system first. Binary acts as a reliable intermediate representation between octal and hexadecimal.

Step 1: Convert octal to binary using 3-bit groups.
Step 2: Regroup the binary digits into 4-bit groups and convert to hexadecimal.

Base conversion techniques are fundamental in digital systems. Mastery of these methods allows efficient translation between human-readable numbers and machine-level representations.

5. Basic Binary Arithmetic

Binary arithmetic follows the same fundamental principles as decimal arithmetic. However, it is simpler because it uses only two digits: 0 and 1.

1) Binary Addition

Binary addition is based on a small set of rules. The most important case is 1 + 1, which produces a carry to the next higher-order bit.

Basic Rules:

0 + 0 = 0

0 + 1 = 1

1 + 0 = 1

1 + 1 = 0 (carry 1)

1 + 1 + 1 = 1 (carry 1)

Example: $1101_2 + 1011_2$ (13 + 11 in decimal)

```
  1 1 1      <-- Carries
  1 1 0 1    (13)
+ 1 0 1 1    (11)
----------
1 1 0 0 0    (24)
```

2) Binary Subtraction

Binary subtraction uses borrowing when subtracting 1 from 0.

Basic Rules:

0 – 0 = 0

1 – 0 = 1

1 – 1 = 0

0 – 1 = 1 (borrow 1)

Example: $1101_2 - 0110_2$ (13 − 6 in decimal)

```
  1 1 0 1    (13)
- 0 1 1 0    (6)
----------
  0 1 1 1    (7)
```

Note: In computer architecture, subtraction is typically implemented using the two's complement method.

3) Binary Multiplication

Binary multiplication is straightforward because digits are multiplied only by 0 or 1. The process consists of shifts and additions.

Basic Rules:

0 × 0 = 0

0 × 1 = 0

1 × 0 = 0

1 × 1 = 1

Example: $101_2 \times 11_2$ (5 × 3 in decimal)

```
    1 0 1
  x   1 1
  -------
    1 0 1    (101 x 1)
+ 1 0 1 .    (101 x 1, shifted left)
  -------
  1 1 1 1    (15)
```

4) Binary Division

Binary division follows the same long-division procedure used in decimal arithmetic. At each step, the divisor is compared with the current portion of the dividend. If subtraction is possible, the divisor is subtracted and a 1 is placed in the quotient; otherwise, a 0 is placed in the quotient.

Example: $110_2 \div 10_2$ (6 ÷ 2 in decimal)

Step 1: Compare 10_2 with the leftmost bit 1_2 → not possible.

Step 2: Compare 10_2 with 11_2 → possible.

$11_2 - 10_2 = 1_2$.

Step 3: Bring down the next bit (0), forming 10_2.

Step 4: $10_2 - 10_2 = 0_2$.

Quotient: 11_2

Remainder: 0

Therefore: $110_2 \div 10_2 = 11_2$

which corresponds to: $6 \div 2 = 3$ in decimal.

6. Conclusion

The study of Numbering Systems is the bridge between human logic and machine reality. While we perceive numbers as abstract values, the computer views them as a series of physical states (high or low voltage).

This chapter provides the essential tools needed to progress to Chapter 3: Information Representation. In the next section, we will explore how these binary strings are used to represent more complex data, such as negative numbers, real numbers (fractions), and text characters.

7. Chapter 3: Information Representation

Binary Coding, Character Representation, and Number Representation

1. Introduction

Information representation is a foundational topic in computer science. All data processed, stored, and transmitted by digital systems must ultimately be encoded in a form that electronic hardware can manipulate. This chapter presents the principal techniques used to represent information in digital computers, including binary coding schemes, character representation standards, and numeric representation methods for both integers and fractional values.

2. Binary Coding Systems

Binary coding techniques represent information using combinations of binary digits, or bits. Computers don't just use one type of binary; they use different "dialects" depending on the task.

Each coding scheme is designed to satisfy specific requirements such as simplicity of arithmetic operations, error reduction, or compatibility with decimal systems[2].

Pure Binary Code

Pure binary, also known as natural binary, is the most fundamental representation system. Each number is expressed as a weighted sum of powers of two.

a) 1-How it Works

Each bit in a Pure Binary number represents a power of 2, starting from the right (Least Significant Bit or LSB).

The value of a binary number is calculated as:

$$\text{Value} = \sum \left(\text{bit}_{\,i} \times 2^{i} \right)$$

Example: Converting Binary to Decimal

Let's take the binary number **1011**:

- $1 \times 2^3 = 8$
- $0 \times 2^2 = 0$
- $1 \times 2^1 = 2$
- $1 \times 2^0 = 1$
- **Total:** 8 + 0 + 2 + 1 = 11 (Decimal)

Pure binary is efficient for arithmetic operations and is natively supported by digital hardware. However, it is not always optimal for representing decimal data or minimizing transition errors.

Table 7-1 presents the complete 4-bit pure binary code representation for the decimal digits from 0 through 15. Each decimal value is encoded using a fixed-length 4-bit binary word, consistent with the natural binary numbering system.

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

Table 7-1 4-Bit Pure Binary Code Table

1) Reflected Binary Code (Gray Code)

commonly known as Gray Code, is a non-weighted binary system where two successive values differ by only one bit.

In standard "Pure" binary, jumping from 3 $(011)_2$ to 4 $(100)_2$ requires three bits to change simultaneously. If a hardware sensor reads those changes at slightly different speeds, the computer might briefly see a completely wrong number. Gray Code prevents this "switching noise.

Although Gray code is not direct arithmetic operations, it is valuable in applications where minimizing bit change errors is critical.

Why is it called "Reflected"?

The code is built using a reflection (mirror) technique. To create a 3-bit Gray Code from a 2-bit one, you "reflect" the sequence like a mirror and prefix the top half with 0s and the bottom half with 1s.

a. Conversion Methods

You will often be asked to convert between Pure Binary and Gray Code. Here is the logic:

a) Binary to Gray Code

To convert a Pure Binary number to its corresponding Gray Code, the following rule is used:

1. Keep the **Most Significant Bit (MSB)** the same.
2. Perform an **XOR** operation on each pair of adjacent bits in the binary number.
    - *XOR Rule:* If bits are the same = 0. If bits are different = 1.

Example: Convert Binary 1011 to Gray

- **Step 1:** Keep the first bit: **1**
- **Step 2:** $1 \oplus 0 = \mathbf{1}$
- **Step 3:** $0 \oplus 1 = \mathbf{1}$
- **Step 4:** $1 \oplus 1 = \mathbf{0}$
- **Result:** 1110

b) Gray Code to Binary

To convert a Gray code number to its corresponding binary value, the following rule is used:

- The **most significant bit (MSB)** of the binary number is the same as the MSB of the Gray code.
- Each subsequent binary bit is obtained by performing **exclusive OR (XOR)** between the previous binary bit and the current Gray code bit.

Example: Convert Gray 1110 back to Binary

- **Step 1:** Keep the first bit: **1**
- **Step 2:** (Binary 1) ⊕ (Gray 1) = **0**
- **Step 3:** (Binary 0) ⊕ (Gray 1) = **1**
- **Step 4:** (Binary 1) ⊕ (Gray 0) = **1**
- **Result:** Binary equivalent=1011

b. Comparison Table

Table 7.2 presents the 4-bit Gray code representation alongside the corresponding decimal values and pure binary codes for the range 0 to 15.

Decimal	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

Table 7-2 4-Bit Gray Code Table

2) Binary Coded Decimal (BCD)

Binary Coded Decimal (BCD) is a "hybrid" system. It's binary, but it thinks like a human. Instead of converting a whole number into binary, BCD converts each individual decimal digit into its own 4-bit binary equivalent.

BCD simplifies the conversion between decimal and binary representations and is commonly used in financial and commercial applications. The main disadvantage is inefficient use of storage compared to pure binary.

a. The 4-Bit Rule

In BCD, every decimal digit (0–9) is represented by a 4-bit As shown in Table 7.3.

Crucial Rule: The binary codes for 10 through 15 (1010, 1011, 1100, 1101, 1110, 1111) are **invalid** in BCD. If they appear, it's an error!

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

Table 7-3 BCD Code Table

Example: Decimal to BCD Conversion

Let's convert the decimal number **395** into BCD:

- **3** →0011
- **9** →1001
- **5** →0101

Result (395 in BCD): 0011 1001 0101

b. BCD vs. Pure Binary

It is important not to confuse the two. BCD usually takes up **more space** than pure binary.

- **Decimal:** 15
- **Pure Binary:** 1111 (only 4 bits)
- **BCD:** 0001 0101 (8 bits)

c. BCD Addition (The +6 Rule)

When adding BCD numbers, if the result of a nibble is greater than 9 (invalid) or generates a carry, you must **add 6** (0110) to that nibble to "skip" the 6 invalid states and get back to a proper BCD value.

Example: Add 5 + 7 in BCD

1. 0101 (5) + 0111 (7) = 1100 (**12** in binary).
2. 1100 is greater than 9, so it is **Invalid BCD**.
3. Add 6 (0110):
    - 1100 + 0110 = 1 0010
4. The 1 carries to the next nibble: 0001 0010.
5. **Result:** 1 2 (which is 12 in BCD).

3) Excess-3 Code

Excess-3 Code (often abbreviated as **XS-3**) is a non-weighted digital code used to represent decimal numbers. It is closely related to BCD (Binary Coded Decimal), but as the name suggests, it is always "3 more" than the standard BCD value. This property simplifies the implementation of subtraction operations using complements.

Excess-3 is primarily of theoretical and educational interest, though it illustrates important concepts in code design.

a. How to Generate Excess-3

To convert a decimal digit to Excess-3, you follow two simple steps:

1. Add **3** to the decimal digit.
2. Convert that sum into a **4-bit binary** number.

Example: Convert Decimal 5 to Excess-3

- 5 + 3 = 8
- 8 in binary is 1000.
- **Result:** 1000

Example: Convert Decimal 19 to Excess-3

- **1** + 3 = 4 →0100
- **9** + 3 = 12 → 1100
- **Result:** 0100 1100

Table 7-4 shows the Excess 3 code for decimal digits 0 to 9, obtained by adding three to each digit before converting it to binary.

Decimal Digit	Decimal + 3	Excess-3 Code (Binary)
0	3	0011
1	4	0100
2	5	0101
3	6	0110
4	7	0111
5	8	1000
6	9	1001
7	10	1010
8	11	1011
9	12	1100

Table 7-4 The Excess-3 Table (0–9)

Note: Just like BCD, the remaining six codes (0–2 and 13–15) are **invalid** in Excess-3.

3. Character Representation

Character representation defines how letters, digits, punctuation marks, and control symbols are encoded as binary patterns so that they can be stored, processed, and transmitted by

digital computers. Several standardized character encoding schemes, such as ASCII and Unicode, have been developed to ensure consistency and interoperability across computer systems.

a. ASCII Code

ASCII (American Standard Code for Information Interchange) is the most famous and widely used character set in the history of computing.

Characteristics

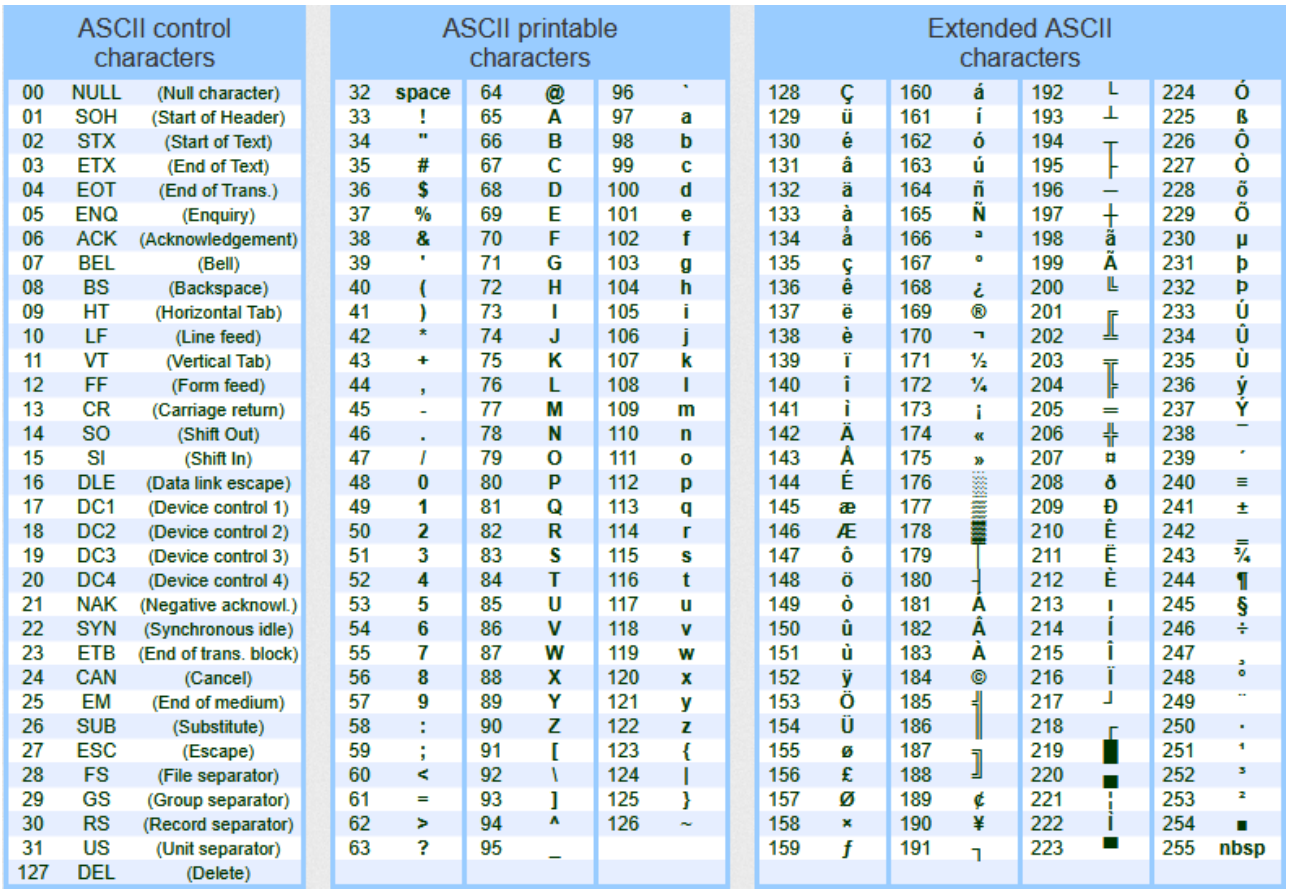
- **Bits:** Originally **7-bit** (128 characters), but modern systems use **8-bit Extended ASCII** (256 characters).
- **Characters:** Includes English alphabets (A-Z, a-z), digits (0-9), punctuation, and control characters (like "Enter" or "Backspace").
- **Example:**
    - The letter **'A'** is decimal **65**, which is binary 0100 0001.
    - The letter **'B'** is decimal **66**, which is binary 0100 0010.

Advantages

- Simple and efficient.
- Widely supported across hardware and software platforms.
- Forms the foundation of many modern encoding schemes.

Extended ASCII Codes

Extended ASCII refers to an 8-bit character encoding scheme that expands the original 7-bit ASCII standard. While Standard ASCII defines codes from 0 to 127, Extended ASCII uses 8 bits, allowing values from 128 to 255. This extension was introduced to support additional symbols, accented characters, and graphical elements required by different languages and computing environments. Figure 7.1 shows the Extended ASCII 8-bit table, which expands the standard ASCII set to 256 characters by including additional symbols and accented characters.



ASCII control characters			ASCII printable characters						Extended ASCII characters							
00	NULL	(Null character)	32	space	64	@	96	`	128	Ç	160	á	192	└	224	Ó
01	SOH	(Start of Header)	33	!	65	A	97	a	129	ü	161	í	193	┴	225	ß
02	STX	(Start of Text)	34	"	66	B	98	b	130	é	162	ó	194	┬	226	Ô
03	ETX	(End of Text)	35	#	67	C	99	c	131	â	163	ú	195	├	227	Ò
04	EOT	(End of Trans.)	36	$	68	D	100	d	132	ä	164	ñ	196	─	228	õ
05	ENQ	(Enquiry)	37	%	69	E	101	e	133	à	165	Ñ	197	┼	229	Õ
06	ACK	(Acknowledgement)	38	&	70	F	102	f	134	å	166	ª	198	ã	230	µ
07	BEL	(Bell)	39	'	71	G	103	g	135	ç	167	º	199	Ã	231	þ
08	BS	(Backspace)	40	(	72	H	104	h	136	ê	168	¿	200	╚	232	Þ
09	HT	(Horizontal Tab)	41	)	73	I	105	i	137	ë	169	®	201	╔	233	Ú
10	LF	(Line feed)	42	*	74	J	106	j	138	è	170	¬	202	╩	234	Û
11	VT	(Vertical Tab)	43	+	75	K	107	k	139	ï	171	½	203	╦	235	Ù
12	FF	(Form feed)	44	,	76	L	108	l	140	î	172	¼	204	╠	236	ý
13	CR	(Carriage return)	45	-	77	M	109	m	141	ì	173	¡	205	═	237	Ý
14	SO	(Shift Out)	46	.	78	N	110	n	142	Ä	174	«	206	╬	238	¯
15	SI	(Shift In)	47	/	79	O	111	o	143	Å	175	»	207	¤	239	´
16	DLE	(Data link escape)	48	0	80	P	112	p	144	É	176	░	208	ð	240	≡
17	DC1	(Device control 1)	49	1	81	Q	113	q	145	æ	177	▒	209	Ð	241	±
18	DC2	(Device control 2)	50	2	82	R	114	r	146	Æ	178	▓	210	Ê	242	‗
19	DC3	(Device control 3)	51	3	83	S	115	s	147	ô	179	│	211	Ë	243	¾
20	DC4	(Device control 4)	52	4	84	T	116	t	148	ö	180	┤	212	È	244	¶
21	NAK	(Negative acknowl.)	53	5	85	U	117	u	149	ò	181	Á	213	ı	245	§
22	SYN	(Synchronous idle)	54	6	86	V	118	v	150	û	182	Â	214	Í	246	÷
23	ETB	(End of trans. block)	55	7	87	W	119	w	151	ù	183	À	215	Î	247	¸
24	CAN	(Cancel)	56	8	88	X	120	x	152	ÿ	184	©	216	Ï	248	°
25	EM	(End of medium)	57	9	89	Y	121	y	153	Ö	185	╣	217	┘	249	¨
26	SUB	(Substitute)	58	:	90	Z	122	z	154	Ü	186	║	218	┌	250	·
27	ESC	(Escape)	59	;	91	[	123	{	155	ø	187	╗	219	█	251	¹
28	FS	(File separator)	60	<	92	\	124	|	156	£	188	╝	220	▄	252	³
29	GS	(Group separator)	61	=	93	]	125	}	157	Ø	189	¢	221	¦	253	²
30	RS	(Record separator)	62	>	94	^	126	~	158	×	190	¥	222	Ì	254	■
31	US	(Unit separator)	63	?	95	_			159	ƒ	191	┐	223	▀	255	nbsp
127	DEL	(Delete)														

Figure 7-1 Extended ASCII (8-bit) table[3]

b. EBCDIC Code

EBCDIC (Extended Binary Coded Decimal Interchange Code) is an 8-bit character encoding scheme developed by IBM for use in mainframe and midrange computer systems.

Characteristics

- Uses 8 bits, allowing 256 possible character combinations.
- Supports uppercase letters, lowercase letters, digits, special characters, and control codes.
- Character codes are non-sequential, making sorting and comparison operations complex.
- Primarily used in legacy IBM mainframe systems.

Example

In EBCDIC, the character:

- 'A' is represented as **11000001**
- '0' is represented as **11110000**

Limitations

- Not compatible with ASCII. For example, in ASCII, 'A' is **65**; in EBCDIC, 'A' is **193**.
- Limited support for international character sets.
- Rarely used in modern computing environments.

c. UTF (Unicode) Codes

As computing went global, 256 characters (ASCII/EBCDIC) weren't enough to cover languages like Chinese, Arabic, or even Emojis. **Unicode** was created to solve this.

Common UTF Formats

- **UTF-8:** The most popular version. It is variable-width (uses 1 to 4 bytes).
    - *Why it's smart:* It is backwards compatible with ASCII. The first 128 characters of UTF-8 are identical to ASCII.
- **UTF-16:** Uses 2 or 4 bytes per character. Commonly used in Windows and Java environments.
- **UTF-32:** Uses a fixed 4 bytes for every single character. It's simple but wastes a lot of memory.

Unicode and UTF-8 Representation Examples

Character	Unicode	UTF-8 Binary
A	U+0041	01000001
€	U+20AC	11100010 10000010 10101100
ع	U+0639	11011000 10111001

Advantages

- Supports **multilingual and international text**.
- Eliminates ambiguity across platforms.

- Standard encoding for modern software systems.

4. Number Representation

Number representation defines how numerical values are encoded in binary form so that they can be processed by digital computers. Because computer hardware operates on a finite number of bits, different representation schemes are used to balance range, precision, simplicity of arithmetic operations, and hardware efficiency. Numerical data is broadly classified into integers and fractional numbers[4].

a. Integer Representation

Integer representation methods define how whole numbers, both positive and negative, are encoded in binary.

1) Unsigned Representation

In unsigned representation, all bits are used to represent the magnitude of the number. There is no provision for negative values.

Characteristics

- Represents only non-negative integers.
- For an n-bit number, the range is:

$$0 \; to \; 2^n - 1$$

Example

Using 8 bits: $11111111_2 = 255_{10}$

Unsigned representation is simple and efficient but unsuitable for signed arithmetic.

2) Sign and Magnitude Representation (Signed-Magnitude)

In sign and magnitude representation, the most significant bit (MSB) indicates the sign of the number:

- 0 → positive
- 1 → negative

The remaining bits represent the magnitude.

Example

Using 8 bits:

$+25 = 00011001_2$

$-25 = 10011001_2$

Limitations

- Two representations of zero (+0 and −0).
- Arithmetic operations are complex to implement.

3) One's Complement Representation (CA1)

One's complement represents negative numbers by inverting all bits of the corresponding positive number.

Example

Using 8 bits:

$+25_{10} = 00011001_2$

$-25_{10} = 11100110_2$

Range

For an n-bit word, the representable interval is: $-(2^{n-1}-1)$ to $+(2^{n-1}-1)$

Characteristics

- Two representations of zero.
- Requires end-around carry in addition.
- Rarely used in modern systems.

Arithmetic in One's Complement (CA1)

In CA1, negative numbers are obtained by inverting all bits. Arithmetic operations require special handling of the carry.

a. Addition in CA1

Procedure

1- Add the two binary numbers normally.
2- If a carry is generated from the most significant bit, perform end-around carry, meaning: Add the carry bit to the least significant bit.

Example : Let's solve **(+5) + (-3)** using 8-bit 1's Complement.

Step 1: Represent numbers

+5=0000 0101 +3=0000 0011 −3=1111 1100 (invert bits)

Step 2: Add

```
  0000 0101
+ 1111 1100
-----------
  0000 0001  (with carry 1)
```

Step 3: End-around carry

```
0000 0001
+         1
-----------
0000 0010
```

Result: +2

b. Subtraction in CA1

Subtraction is performed by adding the complement:

$$A - B = A + (\text{CA1 of } B)$$

After addition, if a carry is produced from the most significant bit, an end-around carry must be added to the least significant bit.

Example (8-bit System)

Compute: 7−5

Step 1: Represent the Numbers in Binary +7=00000111 +5=00000101

Step 2: Find CA1 of 5 Invert all bits of (0000 0101): 1111 1010

This represents −5 in CA1.

Step 3: Add

```
  0000 0111
+ 1111 1010
-------------
  0000 0001   (carry = 1)
```

Step 4: Add the carry to the result:

```
00000001
+       1
-------------
00000010
```

Final Result $00000010_2 = 2_{10}$

4) Two's Complement Representation (CA2)

Two's complement is the most widely used signed integer representation. A negative number is obtained by taking the one's complement and adding 1.

Example

Using 8 bits:

$+25 = 00011001_2$

One's complement $= 11100110_2$

$+1$

Two's complement $= 11100111_2$

Range

For n bits: -2^{n-1} to $+(2^{n-1}-1)$

Advantages

- Single representation of zero.
- Simplifies arithmetic operations.
- Standard representation in modern processors.

2. Arithmetic in Two's Complement (CA2)

In CA2, negative numbers are formed by inverting all bits and adding 1. Arithmetic operations are simpler than CA1 because no end-around carry is required.

A. Addition in CA2

Procedure

1. Add the two numbers normally.
2. Ignore any carry beyond the most significant bit.
3. Detect overflow using sign rules.

Example (8-bit CA2)

Compute: (+5)+(−3)

Step 1: Represent numbers +5=0000 0101 +3=0000 0011

Invert bits: 11111100

Add 1: 11111101

Step 2: Add

```
  0000 0101
+ 1111 1101
-----------
 0000 0010  (carry ignored)
```

Result: +2

B. Subtraction in CA2

Subtraction is performed as:

$$A - B = A + (\text{CA2 of } B)$$

No special carry adjustment is required.

Example

Compute: **5 − 3**

5 = 0000 0101 −3 = 1111 1101

Add:

```
  0000 0101
+ 1111 1101
-----------
  0000 0010
```

Result**: + 2**

Overflow Detection in CA2

Overflow occurs when:

1. Adding two positive numbers yields a negative result.
2. Adding two negative numbers yields a positive result.

It can also be detected when the carry into the sign bit differs from the carry out.

b. Fractional Number Representation

Fractional numbers include values with a fractional component. Their representation must account for both magnitude and precision.

1) Fixed-Point Representation

In fixed-point representation, the position of the binary point is fixed and predetermined. A specific number of bits is allocated to the integer part and the fractional part.

Example

Assume 4 bits for integer and 4 bits for fraction: 0101.1100_2

Value: $5 + \frac{1}{2} + \frac{1}{4} = 5.75_{10}$

Characteristics

- Simple and fast to implement.
- Limited range and precision.
- Suitable for embedded and real-time systems.

2) Floating-Point Representation

A real number can be written in several equivalent forms when expressed in binary scientific notation. For example:

$$0.110 \times 2^5 = 110 \times 2^2 = 0.0110 \times 2^6$$

All three expressions represent exactly the same numerical value. This occurs because the binary point can be shifted left or right, provided the exponent is adjusted accordingly.

Mantissa Normalization

To avoid multiple representations of the same number, the mantissa is normalized.

In the most common convention, a nonzero normalized binary number has the form:

$$\pm 1.b_1 b_2 b_3 \ldots b_k \times 2^{\pm e}$$

Explanation

- The sign $\pm$ indicates whether the number is positive or negative.
- The mantissa always begins with 1.
- The bits $b_1, b_2, \ldots, b_k$ represent the fractional part.
- e is an integer exponent.

Why Normalization Is Necessary?

In base 2, any nonzero real number can be uniquely expressed in the form:

$$1.xxx \times 2^{e}$$

This normalization ensures:

- Uniqueness of representation
- Maximum precision, since the leading bit is always 1

IEEE 754 Standard

Floating-point representation expresses numbers in scientific notation, allowing a wide dynamic range[4].

A floating-point number is represented as:

$$(-1)^{s} \times 1.m \times 2^{(e-\text{bias})}$$

The two types of IEEE 754 floating-point representations commonly used in computer science courses are Single-Precision (32-bit) and Double-Precision (64-bit).

1-IEEE 754 Single Precision Floating-Point Representation (32 bits)

Field	Number of Bits	Description
Sign (S)	1 bit	Determines the sign of the number
Exponent (E)	8 bits	Stores the biased exponent
Mantissa (Fraction)	23 bits	Stores the significant digits

For double precision, the **bias** value is **127**.

Example

Decimal number: $5.75_{10}=101.11_{2}=1.0111\times2^{2}$

Encoded as:

- Sign = 0
- Exponent = 2 + 127 = 129 → $10000001_{(2)}$

- Mantissa = 01110000000000000000000

5.75_{10}= **01000000101110000000000000000000** **(2)**

5.75_{10}= **40B80000** **(16) (32 bits)**

Characteristics

- Very large range.
- Supports very small and very large numbers.
- Introduces rounding errors.
- Essential for scientific and engineering computations.

2-IEEE 754 Double Precision Floating-Point Representation (64 bits)

A double-precision floating-point number is stored using 64 bits, divided into three fields:

Field	Number of Bits	Description
Sign (S)	1 bit	Determines the sign of the number
Exponent (E)	11 bits	Stores the biased exponent
Mantissa (Fraction)	52 bits	Stores the significant digits

For double precision, the **bias value is 1023**.

Example: Representing 5.75 in Double Precision

Step 1: Convert to binary $5.75_{10}=101.11_2$

Step 2: Normalize $101.11_2=1.0111_2\times2^2$

Step 3: Determine fields

- Sign bit: 0 (positive)
- Exponent: $2+\mathbf{1023}=1025_{(10)}=10000000001_2$
- Mantissa: 0111 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000

5.75_{10}= **0**10000000001 0111 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 **(2)**

5.75_{10}= 4017000000000000 **(16)** (64 bits)

Special Values ($\pm\mathbf{0}$, $\pm\infty$, NaN)

IEEE 754 double precision also supports:

1. Zero ($\pm\mathbf{0}$)

In the IEEE 754 standard, zero is not just all bits off. Because there is a dedicated sign bit, we actually have two zeros.

- **Representation:** Exponent = **All 0s**, Mantissa = **All 0s.**
- **Sign Bit:** If 0, it's **+0**. If 1, it's **-0**.
- **Why?** Having a signed zero helps maintain precision when approaching limits in calculus or complex number calculations.

2. Infinity ($\pm\infty$)

Infinity occurs when a number exceeds the maximum representable value (overflow), or when you divide a non-zero number by zero.

- **Representation**: Exponent = All 1s, Mantissa = All 0s.
- **Sign Bit**: Determines if it is $+\infty$ or $-\infty$.
- **Example**: $1.0 / 0.0$ results in $+\infty$

3. NaN (Not a Number)

NaN is used for mathematically undefined or "illegal" operations.

Representation: Exponent = **All 1s**, Mantissa = **Non-zero**.

Cases that produce NaN:

- $0 / 0$
- $\infty - \infty$
- $\sqrt{-1}$ (Square root of a negative number)

5. Conclusion

In this chapter, we have presented the fundamental concepts of information representation in digital computer systems, emphasizing that all data—whether numeric or textual—must ultimately

be encoded as structured binary patterns. We examined binary coding schemes such as Pure Binary, Gray Code, BCD, and Excess-3, identifying their distinct properties and real-world applications in digital logic. We also analyzed character representation standards, including ASCII, EBCDIC, and Unicode (UTF), to demonstrate how standardized encoding guarantees system interoperability. Furthermore, we investigated integer representations ranging from unsigned and signed magnitude to the critical 1's and 2's complement methods alongside fractional representations like fixed-point and the IEEE 754 floating-point standard.

8. Chapter 4: Binary Boolean Algebra

Basic Definitions, Basic Logic Operators, Truth Tables, SOP and POS, Universal Gates, and Units of Measurement Logic Function Simplification

1. Introduction

Binary Boolean algebra provides the mathematical foundation for digital systems, switching theory, and modern computing architectures. It formalizes reasoning over binary variables that assume values in the set {0, 1}, where 0 typically represents the logical false state and 1 represents the logical true state. Unlike classical algebra over real numbers, Boolean algebra operates under axiomatic rules tailored to logical operations and binary variables.[5]

This chapter presents the formal structure of Boolean algebra, its operators, canonical representations, circuit realizations, and systematic techniques for logic function simplification.

2. Basic Definitions and Axioms of Boolean Algebra

a. Basic Definitions

A Boolean algebra is an algebraic structure defined as a set B = {0, 1} equipped with two binary operations:

- Logical **OR**, denoted by **+**
- Logical **AND**, denoted by **·**

and one unary operation:

- Logical complement (NOT), denoted by ( )'

Logic (Boolean) Variable

A Boolean variable is a symbol (usually an uppercase letter like A, B, C) that represents a quantity whose value can only be one of two states.

- **Binary States:** These states are typically represented as **1** (True, High, or Closed) and **0** (False, Low, or Open).
- **Complement:** Every variable A has a corresponding complement A' (or $\bar{A}$), which always holds the opposite value.

Boolean Function

A Boolean function is an algebraic expression formed using binary variables, logic operators (AND, OR, NOT), and parentheses.

- **Input and Output:** It describes the relationship between a set of inputs and a specific output.
- **Mapping:** For every combination of input variables, the function produces a single output value (0 or 1).

Example of a Boolean Function:

If we have a function $F(A, B, C) = (A \cdot B) + C'$:

Inputs: 3 (A, B, C).

Operations: AND (·), OR (+), and NOT (′).

b. Axioms of Boolean Algebra

The axiomatic system of Boolean algebra includes[6]:

1) Closure

- a + b ∈ B
- a · b ∈ B

2) Identity Elements

- a + 0 = a
- a · 1 = a

3) Commutativity

- a + b = b + a

- $a \cdot b = b \cdot a$

4) Associativity

- $(a + b) + c = a + (b + c)$
- $(a \cdot b) \cdot c = a \cdot (b \cdot c)$

5) Distributivity

- $a \cdot (b + c) = a \cdot b + a \cdot c$
- $a + (b \cdot c) = (a + b)(a + c)$

6) Complementarity

- $a + a' = 1$
- $a \cdot a' = 0$

7) Existence of Complements

For every $a \in B$, there exists a' such that:

- $a + a' = 1$
- $a \cdot a' = 0$

These axioms distinguish Boolean algebra from conventional arithmetic.

3. Theorems and Properties

Several important theorems are derived from the axioms[6].

1) Idempotent Laws

- $a + a = a$
- $a \cdot a = a$

2) Domination Laws

- $a + 1 = 1$
- $a \cdot 0 = 0$

3) Double Negation

- $(a')' = a$

4) Absorption Laws

- a + a·b = a
- a · (a + b) = a

5) De Morgan's Laws

- (a + b)' = a' · b'
- (a · b)' = a' + b'

De Morgan's laws are fundamental in digital logic design and gate transformations.

4. Truth Tables.

1) Definition A truth table identifies the output of a logic circuit based on every possible combination of input values[6].

Columns: Each column in the table corresponds to one input variable or output function. Therefore, the total number of columns equals the number of inputs plus the number of outputs.

Rows: The total number of rows in a truth table is determined by the number of input variables. For a function with n inputs, the number of possible input combinations is $\mathbf{2^n}$. Consequently, the truth table contains $\mathbf{2^n}$ rows.

2) Example Case

a Boolean function with **3** input variables and **1 output** requires **4 columns**, corresponding to the three inputs and one output. Since there are 3 inputs, the number of possible input combinations is $\mathbf{2^3}$**=8**. Therefore, the truth table contains **8 rows.**

Columns: 4 (3 inputs + 1 output).

Rows: 8 (calculated as $2^3 = 8$).

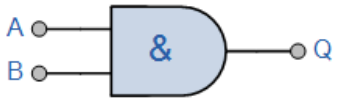
A	B	C	Output
0	0	0	$\bar{A}\bar{B}\bar{C}$

0	0	1	$\bar{A}\bar{B}$C
0	1	0	$\bar{A}$B$\bar{C}$
0	1	1	$\bar{A}$BC
1	0	0	A$\bar{B}\bar{C}$
1	0	1	A$\bar{B}$C
1	1	0	AB$\bar{C}$
1	1	1	ABC

5. Basic Logic Operators:

a. AND Operator

Symbol: · , * , ∧



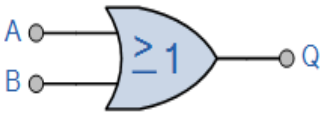
Definition: The AND operation yields 1 only when all inputs are 1.

Truth Table:

A	B	A·B
0	0	0
0	1	0
1	0	0
1	1	1

b. OR Operator

Symbol: +, ∨



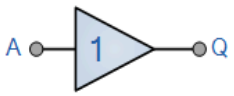
Definition: The OR operation yields 1 if at least one input is 1.

Truth Table:

A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1

c. NOT Operator

Symbol: ( )' , $^{-}$



Definition: The NOT operation complements the input.

Truth Table: Boolean Expression Q = not A or $\overline{A}$

A	A'
0	1
1	0

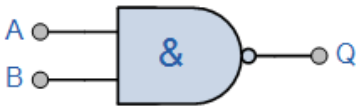
Throughout this guide, we will represent the **NOT** operation using two common symbols: (¬) and (′).

6. Other Logic Operators (Logic Gates):

a. NAND

Definition:

Q= A NAND B = (A·B)' , A NAND B = $\overline{(A \cdot B)}$



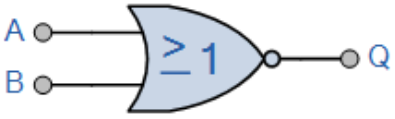
Truth Table:

A	B	NAND
0	0	1
0	1	1
1	0	1
1	1	0

b. NOR

Definition:

Q= A NOR B = (A + B)' , A NOR B = $\overline{(A + B)}$



Truth Table:

A	B	NOR
0	0	1
0	1	0

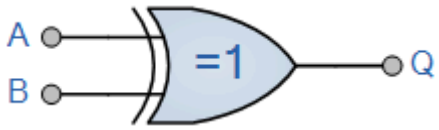
1	0	0
1	1	0

c. XOR (Exclusive OR)

Definition:

$A \oplus B = A'B + AB'$, $A \oplus B = \bar{A}B + A\bar{B}$

XOR produces 1 only when inputs differ.



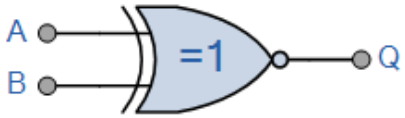
Truth Table:

A	B	XOR
0	0	0
0	1	1
1	0	1
1	1	0

d. The Logic Exclusive-NOR Gate (Ex-NOR)

Definition:

Boolean Expression Q = $\overline{A \oplus B}$



Truth Table:

A	B	XNOR
0	0	1
0	1	0
1	0	0
1	1	1

e. Implication.

Definition:

$A \rightarrow B = A' + B$

A	B	$A \rightarrow B$
0	0	1
0	1	1
1	0	0
1	1	1

7. Logic Expressions and Functions.

A logic expression is a symbolic representation of a Boolean function using variables and operators.

Example: F(A,B,C) = A·B + A'·C

This expression defines a combinational logic function with three inputs and one output.

a. Minterms (Sum of Products)

A Minterm is a product (AND) of all variables in the function. Each Minterm represents a single row in the truth table where the output is **1**.

- ✓ **Rule:** If a variable is 0, use its complement ($\bar{A}$). If it is 1, use the normal variable (A).
- ✓ **Symbol:** Lowercase "m" followed by the decimal equivalent of the binary row.
- ✓ **Goal:** To describe the function as $F = \sum(m_x, m_y, \dots)$.

b. Maxterms (Product of Sums)

A Maxterm is a sum (OR) of all variables. Each Maxterm represents a single row where the output is **0**.

- ✓ **Rule:** This is the opposite of minterms. If a variable is 1, use its complement ($\bar{A}$). If it is 0, use the normal variable (A).
- ✓ **Symbol:** Uppercase "M" followed by the decimal equivalent.
- ✓ **Goal:** To describe the function as $F = \prod(M_x, M_y, \dots)$.

Example : $\boldsymbol{F}\,(A, B, C) = \boldsymbol{A} \cdot \boldsymbol{B} + \boldsymbol{C}$

Truth Table:

A	B	C	$\boldsymbol{F}$	Minterms / MAXterms
0	0	0	0	$(\boldsymbol{A} + \boldsymbol{B} + \boldsymbol{C})$
0	0	1	1	$(\bar{\boldsymbol{A}} \cdot \bar{\boldsymbol{B}} \cdot \boldsymbol{C})$

0	1	0	0	$(A + \overline{B} + C)$
0	1	1	1	$(\overline{A} \cdot B \cdot C)$
1	0	0	0	$(\overline{A} + B + C)$
1	0	1	1	$(A \cdot \overline{B} \cdot C)$
1	1	0	1	$(A \cdot B \cdot \overline{C})$
1	1	1	1	$(A \cdot B \cdot C)$

From our previous expansion, the output is "**1**" for rows 1, 3, 5, 6, and 7.

Minterm Notation: $F(A, B, C) = \sum m(1,3,5,6,7)$

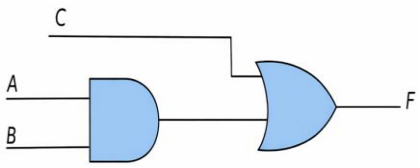
Maxterm Notation: $F(A, B, C) = \prod M(0,2,4)$ (these are the remaining rows where the output is 0).

8. Logic Diagrams

a. Definition

Logic diagrams are graphical representations of Boolean functions using standardized logic gate symbols. They provide a structural view of digital circuits and directly correspond to hardware implementation.

Example 1: F = A·B + C

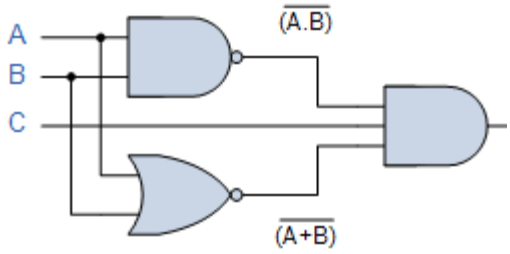


This is represented by:

- One **AND gate** for A and B
- One **OR gate** combining output of AND and C

Example 2:

F = $\overline{(A.B)}$. $\overline{(A + B)}$. c



Logic diagrams are essential in digital circuit synthesis and hardware realization.

9. Canonical Forms

In Boolean algebra, the **Canonical Form** (also known as the Standard Form) is a way of expressing a logic function so that **every term contains all the variables** of that function.

There are two primary types of canonical forms[5]:

1) Sum of Products (SOP) – Canonical Disjunctive Form

This is a logical **OR** of several **Minterms**. A Minterm is a product (**AND**) where every variable appears exactly once (either in its normal form A or its complemented form $\bar{A}$).

- **When to use:** You look at the rows in a Truth Table where the output is **1**.
- **Example1:** For a function $F(A, B, C)$, a canonical term would look like $A \cdot B \cdot \bar{C}$. It cannot be just $A \cdot B$.
- **Example2:** F(A,B) = A'.B + A.B' This is a canonical SOP representation.

2) Product of Sums (POS) – Canonical Conjunctive Form

This is a logical **AND** of several **Maxterms**. A Maxterm is a sum (**OR**) where every variable appears exactly once.

- **When to use:** You look at the rows in a Truth Table where the output is **0**.
- **Example1:** A canonical term would look like $(A + \bar{B} + C)$.
- **Example2:** F(A,B) = (A + B)*(A' + B')

3) Converting to Canonical Form

If you have the function $F(A, B) = A + B$, it is **not** in canonical form because the first term is missing B and the second is missing A.

To make it canonical (SOP), we expand it:

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

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

Canonical Form: $F = AB + A\bar{B} + \bar{A}B$

To convert $F = A \cdot B + C$ into its **Canonical Sum of Products (SOP)** form, we must ensure every term contains all three variables (A, B, and C).

The Conversion Process

We use the Boolean identity $X + \bar{X} = 1$ to "expand" the missing variables into each term.

Step 1: Expand the first term ($A \cdot B$)

This term is missing the variable C. We multiply it by $(C + \bar{C})$:

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

Step 2: Expand the second term (C)

This term is missing both A and B. We multiply it by $(A + \bar{A})$ and then by $(B + \bar{B})$:

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

Now, multiply both results by $(B + \bar{B})$:

$$\left(AC \cdot (B + \bar{B})\right) + \left(\bar{A}C \cdot (B + \bar{B})\right) = ABC + A\bar{B}C + \bar{A}BC + \boldsymbol{\bar{A}\bar{B}C}$$

Step 3: Combine and Remove Duplicates

Now we bring all the resulting terms together:

$$F = (ABC + AB\bar{C}) + (ABC + A\bar{B}C + \bar{A}BC + \boldsymbol{\bar{A}\bar{B}C})$$

Since $ABC + ABC = ABC$ in Boolean algebra, the final **Canonical Form** is:

$$F = ABC + AB\overline{C} + A\overline{B}C + \overline{A}BC + \overline{A}\overline{B}C$$

10. Universal Gates

In digital electronics, **Universal Gates** are a crucial concept for streamlining the design and manufacturing of circuits.

A universal gate is a logic gate that is **functionally complete**. This means it can be used to implement **any** Boolean function without needing any other type of gate. To be considered functionally complete, a single gate type must be able to replicate all fundamental operations: **AND**, **OR**, and **NOT**.

There are two primary universal gates used in digital logic: NAND Gate, NOR Gate

1) NAND as Universal Gate

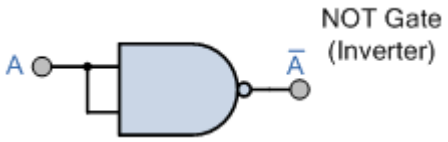
To demonstrate that any Boolean function can be implemented using only NAND gates, we must establish that the three fundamental logic operations AND, OR, and NOT can be reconstructed exclusively using NAND configurations.

1. Implementing an Inverter (NOT) Using Only NAND

To create a NOT gate, you simply connect all inputs of the NAND gate together to a single input signal. Because a NAND gate outputs the inverse of the AND operation, feeding it the same signal on both inputs results in the inverse of that signal.

Logic: $F = (A \cdot A)' = A'$

Configuration: A single NAND gate with inputs tied together.

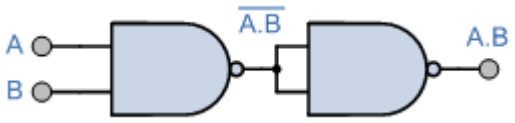


2. Implementing AND Using Only NAND Gates

Since a NAND gate is an "Inverted AND," the standard AND function is realized by passing the signal through a NAND gate and then inverting the result. In practice, this means placing a second NAND gate (configured as an inverter) after the first one.

Logic: $F = ((A \cdot B)')' = A \cdot B$

Configuration: Two NAND gates in series; the first performs the NAND, and the second acts as a NOT gate.

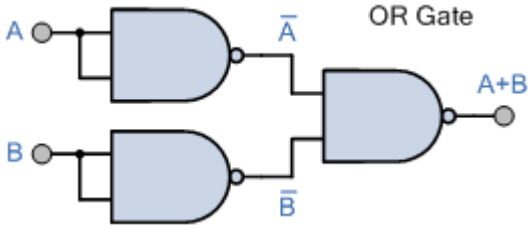


3. Implementing OR Using Only NAND Gates

According to **De Morgan's Law**, the OR operation is equivalent to the NAND of inverted inputs. To achieve this, you first invert the individual inputs using NAND-based inverters and then feed those signals into a final NAND gate.

Logic: $F = (A' \cdot B')' = (A')' + (B')' = A + B$

Configuration: Three NAND gates total; two are used as inverters for the inputs, and the third combines them.



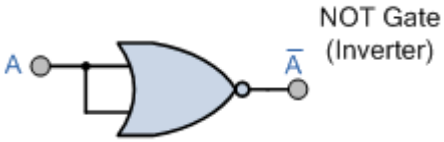
2) NOR as Universal Gate

1. Implementing an Inverter (NOT) Using Only NOR

To create a **NOT** gate, you connect all inputs of the NOR gate together. Since the gate outputs the inverse of an OR operation, feeding the same signal to both inputs results in the complement of that signal.

Logic: $F = (A + A)' = A'$

Configuration: A single NOR gate with inputs tied together.

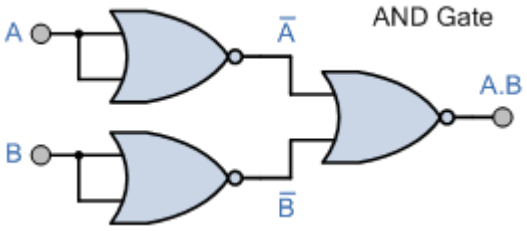


2. Implementing AND Using Only NOR Gates

According to De Morgan's Laws, an AND function is equivalent to the NOR of inverted inputs. To build this, you first invert each input using NOR-based inverters and then feed those signals into a final NOR gate.

Logic: $F = (A' + B')' = (A')' \cdot (B')' = A \cdot B$

Configuration: Three NOR gates total; two function as inverters for the inputs, and the third combines them.

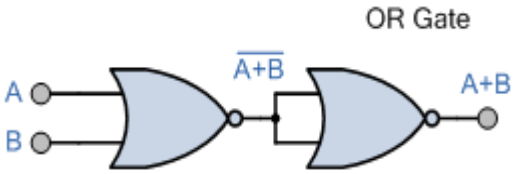


3. Implementing OR Using Only NOR Gates

Because a NOR gate is an "Inverted OR," you can achieve the standard **OR** function by following a NOR gate with a NOR-based inverter (NOT).

Logic: $F = ((A + B)')' = A + B$

Configuration: Two NOR gates in series; the first performs the NOR operation, and the second acts as an inverter to flip the result back.



11. Logic Function Simplification:

Simplification reduces gate count, propagation delay, power consumption, and hardware cost.

1) Algebraic method.

Algebraic simplification consists of systematically applying the fundamental laws of Boolean algebra in order to reduce a logical function to a more compact form. The objective is structural optimization, reduction in the number of logic gates, and minimization of literal count.

a) General Simplification Strategy

1. Eliminate null or dominant terms.
2. Apply idempotent and complement laws.
3. Factor expressions when it reduces the total number of literals.
4. Apply De Morgan's theorems if a global negation appears.
5. Check for absorption opportunities.

Examples

1	2	3
F=A+AB Using absorption: F=A	**F=A$\bar{B}$+AB** Factorization: F=A($\bar{B}$+B) Since: $\bar{B}$+B=1 Therefore: F=A	$\mathbf{F = A\bar{B} + AB + \bar{A}B}$ First grouping: $AB + A\bar{B} = A(B + \bar{B})$=A So: $A + \bar{A}B$ Apply absorption: $A + \bar{A}B = A + B$ Final result: $F = A + B$
4	**5**	**6**
F=(A+B) (A+$\bar{B}$) Expansion: F=A+B$\bar{B}$	$\mathbf{F = b + a.\bar{b}.c}$ $F = (b + \bar{b}).(b + a.c)$ $F = 1.(b + a.c)$	$\mathbf{F = \bar{a}\bar{b}\bar{c} + \bar{a}\bar{b}c + \bar{a}bc}$ $F = \bar{a}\bar{b}.(\bar{c} + c) + \bar{a}bc$ $F = \bar{a}\bar{b}.1 + \bar{a}bc$

Since: B$\bar{B}$=0 Therefore: F=A	$F = b + ac$	$\mathrm{F} = \bar{a}\bar{b} + \bar{a}bc$ $\mathrm{F} = \bar{a}.(\bar{b} + bc)$ $\mathrm{F} = \bar{a}.\left((\bar{b} + b).(\bar{b} + c)\right)$ $\mathrm{F} = \bar{a}.(\bar{b} + c)$ Therefore: F= $\bar{a}\bar{b} + \bar{a}c$

b) Optimization Guidelines

1. Reduce the Number of Product Terms

Search for common factors.

2 Minimize Literal Count

Compare expanded and factored forms.

3 Frequently Used Identities

Common pattern: $XY + X\bar{Y} = X$

- **Proof:** $X(Y + \bar{Y}) \rightarrow X(1) \rightarrow X$
- **Example:** If you have $ABC + AB\bar{C}$, it simplifies instantly to AB.

The Sum Pattern (Dual Form) : $(X + Y)(X + \bar{Y}) = X$

- **Proof:** Using FOIL: $XX + X\bar{Y} + YX + Y\bar{Y}$. Since $XX = X$ and $Y\bar{Y} = 0$, we get $X + X\bar{Y} + XY$. Factor out X: $X(1 + \bar{Y} + Y)$. Since $(1 + \text{anything}) = 1$, the result is X.
- **Use Case:** This is incredibly helpful when simplifying **Product of Sums (POS)** circuits.

Algebraic simplification typically transforms a canonical form into a minimal non canonical representation.

While algebraic simplification is great for small functions, it can get messy with 4 or more variables. For those cases, we usually use Karnaugh Maps (K-Maps).

2) Karnaugh Maps (K-Maps).

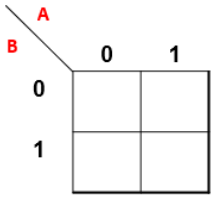
A Karnaugh Map, often abbreviated as K map, is a graphical minimization technique used to obtain a minimal Boolean expression from a truth table or canonical form. It is particularly efficient for functions containing up to four or five variables.

The method relies on spatial adjacency of minterms arranged in Gray code order so that adjacent cells differ by exactly one variable. Grouping adjacent 1 cells allows elimination of variables and direct derivation of prime implicants.

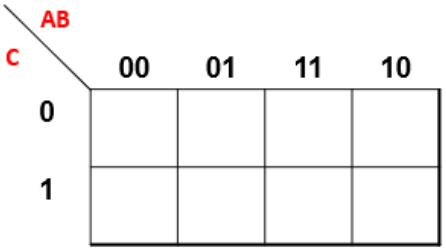
a) 1 Structural Organization of a Karnaugh Map

For an n variable function, the map contains 2^n cells.

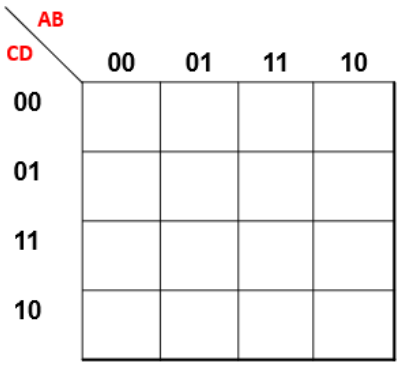
2 variables: 2 × 2 grid



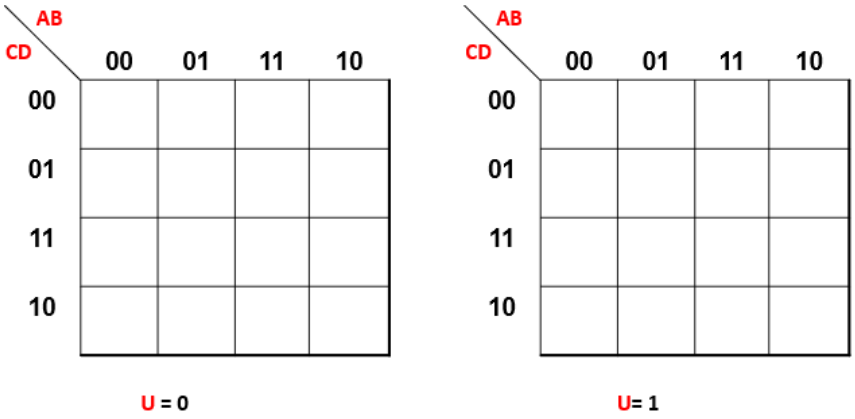
3 variables: 2 × 4 grid



4 variables: 4 × 4 grid



5 variables: 2 × 4 × 4 grid



Each cell corresponds to one minterm.

b) Procedure for Simplification

Step 1: Construct the Map

Fill each cell with 1 where the function equals 1. Enter 0 elsewhere. If don't care terms exist, mark them as X.

Step 2: Form Groups

Create rectangular groups of adjacent 1 cells. Each group must contain:

- 1, 2, 4, 8, or 16 cells

- A power of two number of cells
- The largest possible grouping
- Wrap around adjacency is allowed

Groups may overlap if this leads to fewer literals.

Step 3: Derive Simplified Terms

For each group:

- Identify variables that remain constant across the group
- Eliminate variables that change within the group

Each group yields one product term.

Step 4: Combine Terms

The final simplified expression is the OR sum of all derived product terms.

c) Worked Example

Given: $F(A,B,C) = \sum(3,5,6,7)$

To simplify the function $F(A,B,C) = \sum(3,5,6,7)$ using a Karnaugh Map, we follow these steps:

1. Fill the K-Map

We place a 1 in the cells corresponding to the decimal minterms 3, 5, 6, and 7.

C \ AB	00	01	11	10
0			1	
1		1	1	1

2. Identify the Groups

The goal is to find the largest power-of-2 groups.

- **Group 1 (Horizontal Pair of 4 - Not possible)**: We have a row of three 1s (m_5, m_7, m_6), but groups must be powers of 2. So, we look for pairs of 2.

C \ AB	00	01	11	10
0			1	
1		1	1	1

- **Group 2 (Horizontal Pair)**: Covers m_7 and m_6 in the bottom row.
  - Variables: $A = 1$. $B = 1$ for both. C changes from 1 to 0 (eliminated).
  - Term: $\boldsymbol{AB}$

C \ AB	00	01	11	10
0			1	
1		1	1	1

- **Group 3 (Horizontal Pair)**: Covers m_5 and m_7 in the bottom row.
  - Variables: $A = 1$. $C = 1$ for both. B changes from 0 to 1 (eliminated).
  - Term: $\boldsymbol{AC}$

C \ AB	00	01	11	10
0			1	
1		1	1	1

- **Group 4 (Vertical Pair):** Covers m_3 and m_7 in the $BC = 11$ column.
  - Variables: A changes from 0 to 1 (eliminated). $B = 1$ and $C = 1$.
  - Term: $\boldsymbol{BC}$

C \ AB	00	01	11	10
0			1	
1		1	1	1

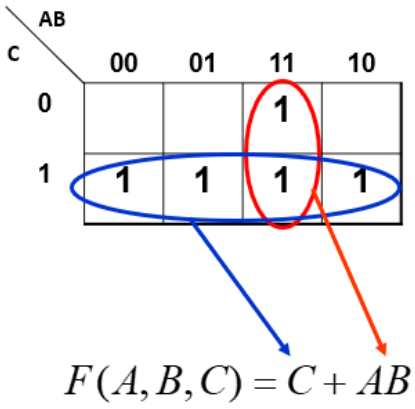
- **Final Simplified Equation**

C \ AB	00	01	11	10
0			1	
1		1	1	1

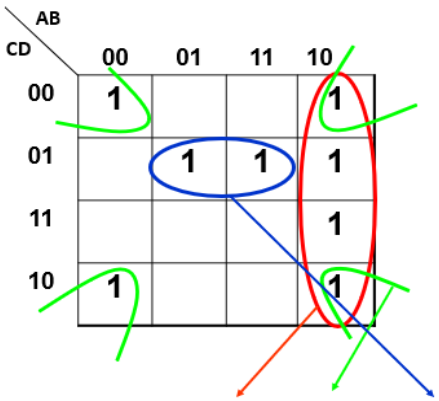
By combining the terms from the three groups, we get the minimized expression:

$$\boldsymbol{F = AB + AC + BC}$$

Exemple 2 : 3 variables

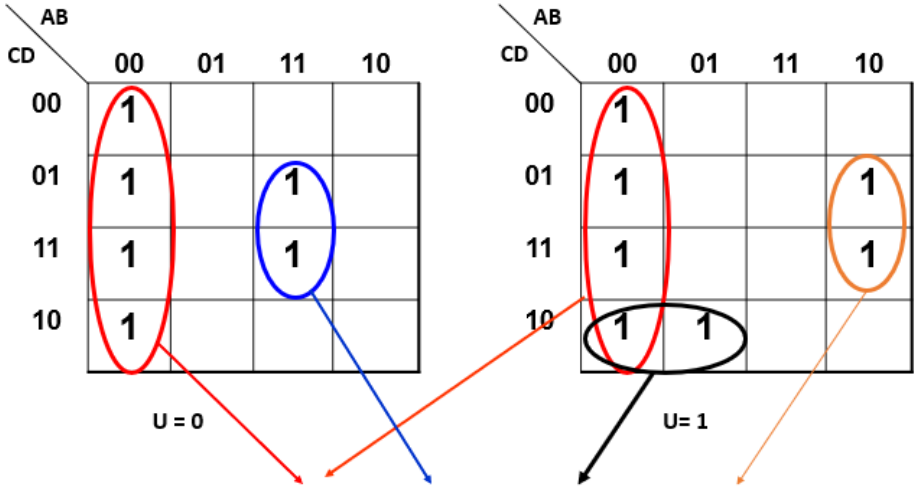


Exemple 3 : 4 variables



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

Exemple 4 : 5 variables



$$F(A,B,C,D,U) = \overline{A}\,\overline{B} + A.B.D.\overline{U} + \overline{A}.C.\overline{D}.U + A.\overline{B}.D.U$$

d) The "Don’t Care" Conditions X in Karnaugh Maps

In some designs, certain input combinations are impossible or their output doesn't matter (e.g., BCD codes only go from 0-9; 10-15 are "Don't Cares").

You can treat an **X** as a 1 if it helps you make a larger group.

You can treat an **X** as a 0 if you don't need it.

Don’t care conditions represent input combinations for which the output value is irrelevant or unspecified. They are denoted by X and may be treated as either 0 or 1 during minimization.

They commonly arise from:

- Unused input combinations, such as invalid BCD codes
- Unreachable states in sequential circuits
- Design constraints where certain inputs never occur

In a Karnaugh map:

- X cells may be included in a group if doing so increases the group size.
- Groups must still contain at least one actual 1.
- A group cannot consist solely of X cells.

The primary objective is to form the largest possible power of two groupings. Larger groups eliminate more variables, producing simpler Boolean expressions.

Proper use of don’t care terms can significantly reduce:

- The number of product terms
- The number of literals
- Hardware implementation complexity

Thus, don’t care conditions provide design flexibility and enable more aggressive Boolean minimization without affecting required system behavior.

Exemple 1 : Consider: F(A,B,C)=∑m(1,3,7)+d(5)

C \ AB	00	01	11	10
0				
1	1	1	1	**x**

Without the don't care(X):

- You would have one group for (m_1, m_3) which equals $\bar{A}C$.
- You would have a separate group for m_7 (likely paired with m_3) which equals BC.
- Result: $F = \bar{A}C + BC$

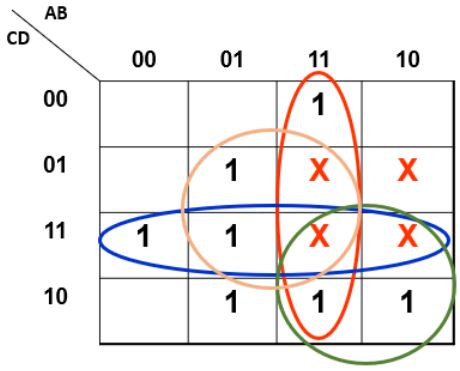
With the don't care m_5 (With the X):

- The entire $C = 1$ column is filled.
- The variables A and B both change states $(0 \rightarrow 1)$ within that column, so they drop out.
- Result: $F = C$

C \ AB	00	01	11	10
0				
1	1	1	1	**1**

The don't care condition enabled elimination of additional variables.

Exemple 2 :



While K-Maps are great for humans to visualize patterns, they become a nightmare once you hit 5 or 6 variables (imagine trying to "wrap" a 3D or 4D cube in your head). The Quine-McCluskey (QM) algorithm is the tabular version of the K-Map. It's systematic, step-by-step, and perfect for computers.

3) Quine-McCluskey Algorithm.

The **Quine-McCluskey (QM) algorithm** is a systematic, tabular method used to simplify Boolean expressions. It is essentially the "computer-logic" version of a Karnaugh Map, designed to handle more than 4 variables without the visual confusion.

a) Procedure for Simplification

Here are the 5 general steps to perform the algorithm:

Step 1: **Binary Grouping**

First, convert all decimal minterms (the "1s" of your function) into binary. Then, organize them into groups based on the **number of 1s** in their binary representation.

- **Group 0:** Zero 1s (e.g., `0000`)
- **Group 1:** One 1 (e.g., `0001`, `0010`)
- **Group 2:** Two 1s (e.g., `1100`, `1010`)
- ...and so on.

Step 2: **Form Pairs (Reduction Table)**

Compare each term in Group n with every term in Group $n + 1$.

1. **Check for 1-bit difference:** If two terms are identical except for exactly one bit position, combine them.
2. **Use a Dash:** Replace the changing bit with a dash (-).
3. **Checkmark:** Place a checkmark next to every original term you just used. This indicates it is "covered" by a larger group.

Step 3: **Repeat for Quads and Octets**

Repeat the process using the newly created dashed terms.

- You can only combine terms if their **dashes are in the same position**.
- Continue this process until no more combinations can be made.
- **Crucial:** Any term that **does not** have a checkmark at the end of the entire process is a **Prime Implicant (PI)**. These are your final candidates for the simplified equation.

Step 4: **The Prime Implicant Chart**

Now you must choose the smallest set of PIs to cover all original minterms.

1. Create a grid: Rows are your Prime Implicants; Columns are your original minterms.
2. Mark Coverage: Put an 'X' where a PI covers a minterm.
3. Find Essential Prime Implicants (EPIs): Look for columns that contain only one 'X'. Circle that 'X'—the PI in that row is "Essential" and must be in your final answer.

Step 5: **Write the Final Expression**

Once you have identified your **Essential Prime Implicants** (and any additional Prime Implicants needed to cover the remaining minterms), you convert the binary strings (with dashes) back into algebraic variables (A, B, C, D).

The Conversion Rules:

- **1** = The variable is **Normal** (e.g., A)
- **0** = The variable is **Complemented** (e.g., $\bar{A}$)
- **-(Dash)** = The variable is **Eliminated** (it has no effect on the outcome)

b) Worked Example

Consider: F(A,B,C,D)= ∑m(0,1,2,5,6,7,8,9,10,14)

Step 1: Group by Number of 1s

We list the minterms in binary and group them based on how many "1" bits they contain.

Group G0	0 0000
Group G1 (One 1)	1 0001 2 0010 8 1000
Group G2 (Two 1s)	5 0101 6 0110 9 1001 10 1010
Group G3 (Three 1s)	7 0111 14 1110

Step 2: Combine Groups (First Iteration)

We compare terms in adjacent groups (e.g., G0 with G1) and combine them if they differ by only one bit.

Combination	Binary		Combination	Binary
(0, 1)	000-		(2, 6)	0-10
(0, 2)	00-0		(2, 10)	-010
(0, 8)	-000		(8, 9)	100-
(1, 5)	0-01		(8, 10)	10-0
(1, 9)	-001		(5, 7)	01-1
(5, 7) is $01-1$			(6, 7)	011-
(6, 14)	-110		(10, 14)	1-10

Step 3: Combine Groups (Second Iteration)

Now we combine the results from Step 2. Again, we only combine if the dashes match and only one other bit differs.

- **(0, 1, 8, 9)** → `-00-` (Matches 0,1 with 8,9 and 0,8 with 1,9)
- **(0, 2, 8, 10)** → `-0-0` (Matches 0,2 with 8,10 and 0,8 with 2,10)
- **(2, 6, 10, 14)** → `- -10` (Wait, checking: (2,6) is $0-10$ and (10,14) is $1-10$, so → `- -10`)

Remaining Prime Implicants (PIs): Terms that couldn't be combined further:

1. **A** = `-00-` (Minterms: 0, 1, 8, 9) →$\bar{B}\,\bar{C}$

2. **B** = `-0-0` (Minterms: 0, 2, 8, 10) → $\bar{B}\,\bar{D}$
3. **C** = `--10` (Minterms: 2, 6, 10, 14) → $C\bar{D}$
4. **D** = `01--` (Minterms: 5, 6, 7) → let's re-check 5, 6, 7.

- (5,7) is $01-1$ and (6,7) is $011-$. These don't combine further into a group of 4. So **01-1** ($\bar{A}BD$) and **011-** ($\bar{A}BC$) are PIs.

Step 4: Prime Implicant Chart

We check which PIs are "Essential" to cover all minterms: $\{0, 1, 2, 5, 6, 7, 8, 9, 10, 14\}$.

PI	Minterms	0	1	2	5	6	7	8	9	10	14
$\bar{B}\,\bar{C}$	0,1,8,9	X	**X**					X	**X**		
$\bar{B}\,\bar{D}$	0,2,8,10	X		X				X		X	
$C\bar{D}$	2,6,10,14			X		X				X	**X**
$\bar{A}BD$	5,7				**X**		X				
$\bar{A}BC$	6,7					X	X				

1-Identify Essential Prime Implicants (EPIs): Look for columns with only one "X".

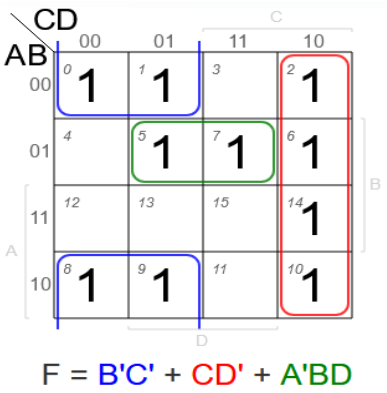
- **Minterm 1, 9** are only covered by $\bar{B}\,\bar{C}$ (Essential).
- **Minterm 14** is only covered by $C\bar{D}$ (Essential).
- **Minterm 5** is only covered by $\bar{A}BD$ (Essential).

2-Verify Coverage: These three EPIs cover all minterms: $\{0, 1, 2, 5, 6, 7, 8, 9, 10, 14\}$. (Note: Minterm 6 is covered by $C\bar{D}$ and Minterm 7 is covered by $\bar{A}BD$).

Final Simplified Expression:

$$F = \bar{B}\,\bar{C} + \bar{A}BD + C\bar{D}$$

we can use Karnaugh Map (K-Map) to check the Quine-McCluskey (QM) result



The Quine–McCluskey method is a formal minimization algorithm based on systematic combination of minterms and structured selection of prime implicants. It provides an exact minimal solution but may require significant computational effort for large scale problems.

12. Conclusion

This chapter establishes Binary Boolean algebra as the mathematical foundation of digital logic design by defining binary variables, logical operations, and a consistent axiomatic framework. It explains how fundamental theorems and properties support systematic manipulation and transformation of logic expressions.

The chapter also presents canonical forms such as Sum of Products and Product of Sums, demonstrates the universality of NAND and NOR gates, and emphasizes the role of truth tables and logic diagrams in precise functional representation. Finally, it highlights simplification techniques including algebraic methods, Karnaugh maps, and the Quine-McCluskey algorithm, which reduce hardware complexity and form the basis of efficient digital system synthesis.

Second Part
Exercises and Solutions

9. Chapter 1 Exercises

Introduction

1. Exercise 1: Units of Information Measurement

a) Arrange these units of measurement in ascending order:

1 YO, 1 KO, 1 MO, 1 GO, 1 O, 1 BIT, 1 TO, 1 ZO

b) Fill in the blanks:

2,048 bytes =..... KO

5,120 GO =..... TO

2. Exercise 2: Byte Representation

How many different values can be represented using a single byte?

3. Exercise 3: Storage Capacity Planning

We have a 320 GB hard drive with 100 GB of free space. We want to copy all the contents of the hard drive onto 64 GB USB flash drives.

- How many USB flash drives are needed to complete this task?

4. Exercise 4: Maximum Downloadable Data Calculation

What is the maximum size, in kilobytes, that can be downloaded in one second with a 1 Mbps connection?

5. Exercise 5: Bytes and Bits Calculation

A memory chip has a capacity of 16 GB.

a) How many bytes does it contain?

b) How many bits does it contain?

Use binary units for storage conversion.

6. Exercise 6: Download Time Calculation

A video file has a size of 2.4 GB.

a) How long will it take to download the file using a 20 Mbps connection?

b) Express the final answer in minutes and seconds.

Assume: 1 byte = 8 bits

10. Chapter 2 Exercises

Numbering Systems

1. Exercise 1: Number Base Conversions

1) Conversion from base b to base 10

a) From base 2 to base 10

A = $1101_{(2)}$ B = $1001001_{(2)}$ C = $1001_{(2)}$ D = $110000_{(2)}$

b) From base 8 to base 10

E = $36_{(8)}$ F = $15_{(8)}$ G = $67_{(8)}$

c) From base 16 to base 10

H = $9A_{(16)}$ I = $20_{(16)}$ J = $10_{(16)}$ K = $AB_{(16)}$ L = $101_{(16)}$

2) Conversion from base 10 to base b

In these questions, the two methods presented in the course will be used.

a) From base 10 to base 2

M = $20_{(10)}$ N = $31_{(10)}$ O = $256_{(10)}$ P = $400_{(10)}$

b) From base 10 to base 8

Q = $20_{(10)}$ R = $13_{(10)}$ S = $208_{(10)}$

c) From base 10 to base 16

T = $70_{(10)}$ U = $512_{(10)}$ V = $228_{(10)}$ W = $450_{(10)}$

3) Binary to hexadecimal conversion

In these questions, convert directly from one base to another without passing through base 10.

a) From binary to hexadecimal

Write the base 16 representation of the following numbers:

X = $101110_{(2)}$ Y = $1010010110_{(2)}$ Z = $10010011101_{(2)}$

b) From hexadecimal to binary

Write the base 2 representation of the following numbers:

A = $10_{(16)}$ B = $7C_{(16)}$ C = $ABC_{(16)}$ D = $1E6_{(16)}$

2. Exercise 2 : Base-to-Base Conversion

Conversion from base X to base Y

a) $(18)_9$ = (.....)$_{10}$

b) $(112)_5$ = (.....)$_6$

c) $(121)_3$ = (.....)$_2$

3. Exercise 3: Determining an Unknown Base

1) Find the base B such that the following equality holds: $(36)^B = (27)_{10}$
2) Determine x and y such that: $(1x5,y3)_6 = (71.25)_{10}$

4. Exercise 4: Arithmetic Operations in Binary

Perform the following operations and verify the results by carrying out the necessary conversions.

a. 1000 + 0110

b. 1101001 + 10110

c. 1001000 + 1011001 , What happens if only 7 bits are available?

d. 101101 – 10010

e. 10100 – 1001

f. 11001101 × 1011

g. 1100101010 ÷ 1001

h. 101010 ÷ 101

11. Chapter 3 Exercises

Information Representation

1. **Exercise 1:** Signed Integer Representation

We consider an 8 bit representation of signed integers.

1) Complete the following table:

Decimal	Binary	SVA	CA1 one's complement	CA2 two's complement
-20				
-14				
20				
-120				
-128				

2) Give the encoding ranges on 8 bits and on 16 bits for the different representations of signed integers: S+VA, CA1, and CA2.
3) How do we write -512 in S+VA? What is the minimum number of bytes required to encode this value?

2. **Exercise 2:** SVA, One's Complement, Two's Complement

Give the signed decimal value of each of the following representations:

$(0001\ 0000)_{SVA}$	$(00011010)_{CA1}$	$(0001\ 0001)_{CA2}$
$(1001\ 1010)_{SVA}$	$(11100100)_{CA1}$	$(1110\ 0101)_{CA2}$

3. **Exercise 3:** Arithmetic Operations in One's Complement

Using 8 bit integers in one's complement, perform the following calculations:

a) +10 – 12

b) +50 – 7

c) +18 +(−5)

d) −34 + (−12)

4. **Exercise 4:** Arithmetic Operations in Two's Complement

Perform the following additions in two's complement:

a) 0110 1011 + 1011 1101

b) 1001 0110 + 1111 1011

c) 0110 1111 + 0001 1001

d) 1000 0010 + 1010 1011

5. **Exercise 5:** IEEE 754 Single Precision Floating Point(Encoding)

Represent the following real numbers in IEEE 754 single precision floating point format. Express the result in hexadecimal:

a) +128.0

b) −32.625

c) +18.125

d) +∞

e) −8.75

6. **Exercise 6:** IEEE 754 Single Precision Floating Point(Decoding)

Determine the decimal value of each of the following numbers encoded in IEEE 754 single precision format:

a) 01000001 11001100 00000000 00000000

b) 10111111 11000000 00000000 00000000

c) (C1BC0000) base 16

d) (3D800000) base 16

7. **Exercise 7:** Binary Code / Gray Code

a) Convert the following binary numbers into Gray code:

- $(100101)_2$
- $(10110)_2$
- $(10011001)_2$

b) Give the binary code of the following numbers expressed in Gray code:

- $(10111)_{Gray}$
- $(101101)_{Gray}$
- $(101101)_{Gray}$

8. **Exercise 8:** ASCII Code Encoding and Decoding

a) Using ASCII code, decode the following sentence given in hexadecimal:

54 64 20 33 20 73 74 72 75 63 74 75 72 65 20 6d 61 63 68 69 6e 65 20 73 31

b) Using ASCII code, decode the following sentence:

85 110 105 118 101 114 115 105 116 195 32 65 109 97 114 32 84 101 108 105 100 106 105

12. Chapter 4 Exercises

Boolean Algebra

1. **Exercise 1:** Truth Table Construction

Find the truth table of the function $F(A,B,C) = \overline{(A \cdot B)} \cdot (C + B) + A \cdot \bar{B} \cdot C$

2. **Exercise 2:** Logic Diagram

Give the logic diagram of the following functions:

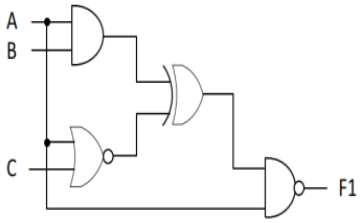
$$F(x,y,z) = (\bar{x} + y)\overline{(y + x\bar{z})} + x\bar{y}z$$

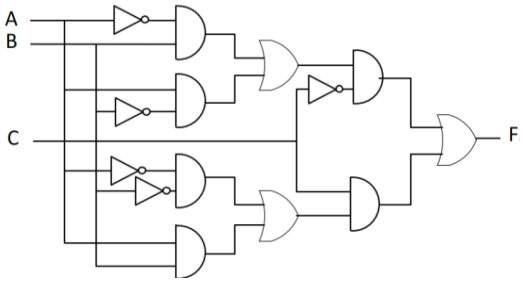
$$F(x_1,x_2,x_3) = x_1\overline{x_2}x_3 + \overline{x_1}\,\overline{x_2}\,x_3 + \overline{x_1}x_2\overline{x_3}$$

$$G(a,b,c,d) = \overline{(a \oplus b) \cdot \overline{(c + d)}}$$

3. **Exercise 3:** Derivation of Boolean Expressions

Determine the logical expressions corresponding to the following logic diagrams:





4. **Exercise 4:** Universal gates

a) Rewrite the following functions using **NAND** gates only.

1) $x\bar{y} + \bar{x}y$
2) $xy + xz + yz$
3) $\overline{(x + z)(y + z)}$

b) Rewrite the following functions using **NOR** gates only.

1) $(x + y)(\bar{x} + \bar{y})$
2) $(x + y)(x + z)(y + z)$
3) $\overline{xy + \bar{x}z}$

5. **Exercise 5:** Algebraic Simplification of Boolean Expressions

Simplify the following expressions using the rules of Boolean algebra:

1) F(A,B)=AB+A$\overline{B}$
2) F(A,B)=(A+B)(A+$\overline{B}$)
3) F(A,B)=A+AB
4) F(A,B)=A(A+B)
5) F(A,B)= A + B + $\overline{A}$ $\overline{B}$
6) F(A,B)=A+ $\overline{A}$B
7) F(A,B)= A($\overline{A}$+B)
8) F(A,B,C)= AB+A $\overline{B}$C
9) F(A,B)= A $\overline{B}$ + $\overline{A}$ $\overline{B}$+ $\overline{A}$ B
10) F(A, B, C)= (A+B)(A+$\overline{B}$+C)
11) F(A, B, C)= $\overline{A}$ $\overline{B}$C+ A$\overline{B}$ C+$\overline{A}$ B C+A B C
12) F(A, B, C, D)= $\overline{A}$ BC+A $\overline{B}$D+A $\overline{C}$D+BC$\overline{D}$
13) F(A, B, C)= A B + (C $\overline{B}$ + $\overline{C}$)+C$\overline{A}$

6. **Exercise 6:** SOP and POS Forms

Consider the function defined by the truth table below:

A	B	C	F(A,B,C)
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

1) Generate a corresponding Boolean (logic) expression:

a) In **sum-of-products (SOP)** form

b) In **product-of-sums (POS)** form

2) Simplify expression (a) using the rules of Boolean algebra

7. **Exercise 7:** Algebraic Simplification

Let the following function be given:

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

- Simplify F
- Find $\bar{F}$

8. **Exercise 8:** Simplification and Equivalence Relations

Prove algebraically the following relations[7]:

1) $AB + ACD + \bar{B}D = AB + \bar{B}D$
2) $(\bar{A} + B)(A + C)(B + C) = (\bar{A} + B)(A + C)$
3) $AB + \bar{B}C = (A + \bar{B})(B + C)$
4) $\overline{A\bar{B} + \bar{A}B} = AB + \bar{A}\,\bar{B}$
5) $\overline{(A + B)(\bar{A} + C)} = (A + \bar{B})(\bar{A} + \bar{C})$

9. **Exercise 9:** Karnaugh Map Minimization of Boolean Functions

Simplify the following functions using Karnaugh maps:

F1 = $AB\bar{C} + \bar{A}\bar{B}C + A\bar{B}\bar{C} + A\bar{B}C$

F2 = $ABC + \bar{A}BC + \bar{A}\bar{B}C + AB\bar{C}$

F3 = $\bar{A}\bar{B}C\bar{D} + \bar{A}B\bar{C}D + A\bar{B}C\bar{D} + \bar{A}\,\bar{B}\bar{C}\,\bar{D} + AB\bar{C}\,D + \bar{A}BCD + A\bar{B}\bar{C}\,\bar{D}$

10. **Exercise 10:** Karnaugh maps

Obtain the simplified functions using Karnaugh maps.

AB / CD	00	01	11	10
00	0	1	1	0
01	1	1	1	1
11	0	1	1	0
10	0	1	1	0

AB / CD	00	01	11	10
00	1	0	0	1
01	0	0	0	0
11	0	1	1	0
10	1	0	0	1

AB / CD	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	0	0	0	0
10	1	1	1	1

F_1 =	F_2 =	F_3 =

CD \ AB	00	01	11	10
00	0	1	1	0
01	1	1	1	1
11	1	1	1	1
10	0	1	1	0

F_4 =

CD \ AB	00	01	11	10
00	0	0	0	0
01	1	0	0	1
11	1	0	0	1
10	0	0	0	0

F_5 =

CD \ AB	00	01	11	10
00	0	1	0	1
01	1	1	1	1
11	1	1	1	1
10	0	1	0	1

F_6 =

CD \ AB	00	01	11	10
00	1	1	1	1
01	0	1	1	0
11	0	1	1	0
10	1	1	1	1

F_7 =

CD \ AB	00	01	11	10
00	1	0	0	1
01	0	1	1	0
11	0	1	1	0
10	1	0	0	1

F_8 =

CD \ AB	00	01	11	10
00	1	0	0	1
01	1	1	1	1
11	1	1	0	0
10	0	0	0	0

F_9 =

11. **Exercise 11:** Quine–McCluskey

Consider the truth table of the logical function F(A,B,C,D):

A	B	C	D	F
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1

1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	0

1) Determine the canonical disjunctive expression of F in binary form.
2) Apply the Quine–McCluskey method to minimize the Boolean function F.

Solutions to Exercises

13. Chapter 1 Solutions

Introduction

1. **Exercise 1:** Units of Information Measurement

a) **Ascending Order** In computing, we move from the smallest unit (a bit) to the basic unit (an octet/byte), and then through the prefixes. "O" stands for **Octet**, which is the French term for **Byte**.

1 BIT (Smallest unit)

1 O (Octet/Byte)

1 KO (Kilooctet / KB)

1 MO (Megaoctet / MB)

1 GO (Gigaoctet / GB)

1 TO (Teraoctet / TB)

1 ZO (Zettaoctet / ZB)

1 YO (Yottaoctet / YB)

b) Fill in the Blanks

2,048 bytes = 2 KO (Since $2{,}048 / 1{,}024 = 2$)

5,120 GO = 5 TO (Since $5{,}120 / 1{,}024 = 5$)

2. **Exercise 2:** Byte Representation

A single byte consists of **8 bits**. Since each bit can be either a 0 or a 1 (2 possibilities), we calculate the total combinations using the power of 2:

$$2^8 = 256$$

A single byte can represent **256 different values** (typically ranging from 0 to 255).

3. **Exercise 3:** Storage Capacity Planning

Storage Calculation:

First, we need to determine how much data is actually on the drive.

Calculate Used Space:

- $320\ \text{GB (Total)} - 100\ \text{GB (Free)} = 220\ \text{GB of data.}$

Calculate Number of Flash Drives:

- Divide the total data by the capacity of one flash drive:
- $220\ \text{GB}/64\ \text{GB} = 3.4375$
- Since you cannot have a fraction of a flash drive, you must round up.

  Answer: You need **4 USB flash drives.**

4. **Exercise 4:** Maximum Downloadable Data Calculation

Connection Speed:

This is a classic "bits vs. bytes" trap. Internet speeds are usually measured in **bits** (lowercase 'b'), while file sizes are measured in **Bytes** (uppercase 'B').

Convert Megabits to Kilobits:

$$1\ \text{Mbps} = 1{,}000\ \text{kilobits per second (kbps).}$$

Convert Kilobits to Kilobytes:

There are 8 bits in a byte, so we divide by 8:

$$1{,}000 / 8 = 125$$

Answer: The maximum download size is **125 KB/s**.

5. **Exercise 5:** Bytes and Bits Calculation

Since we are using **binary units** (where $1\ \text{GB} = 1{,}024\ \text{MB}$ and so on), we calculate the capacity using powers of 2 (2^{10}).

a) How many bytes does it contain?

To get from Gigabytes (GB) to Bytes (B), we multiply by 1,024 three times:

$$16 \times 1{,}024 \times 1{,}024 \times 1{,}024 = 17{,}179{,}869{,}184 \text{ bytes}$$

(Alternatively, in powers of two: $2^4 \times 2^{30} = 2^{34}$ *bytes)*

b) How many bits does it contain?

Since there are 8 bits in 1 byte, we multiply the previous result by 8:

$$17{,}179{,}869{,}184 \times 8 = 137{,}438{,}953{,}472 \text{ bits}$$

(Alternatively: $2^{34} \times 2^3 = 2^{37}$ *bits)*

6. **Exercise 6:** Download Time Calculation

Download Time Calculation

To solve this, we must ensure the file size and the connection speed are in the same unit (bits).

Step 1: Convert the video file size to Megabits (Mb)

- File size: $2.4\ \text{GB}$
- Convert to MB (Megabytes): $2.4 \times 1{,}024 = 2{,}457.6\ \text{MB}$
- Convert to Mb (Megabits): $2{,}457.6 \times 8 = 19{,}660.8\ \text{Mb}$

Step 2: Calculate the download time in seconds

Divide the total size by the connection speed ($20\ \text{Mbps}$):

$$19{,}660.8\ \text{Mb} / 20\ \text{Mbps} = 983.04\ \text{seconds}$$

Step 3: Convert to minutes and seconds

- **Minutes:** $983.04/60 = 16.384$ minutes
- **Whole minutes:** 16
- **Remaining seconds:** $0.384 \times 60 = 23.04$ seconds

Final Answer: It will take approximately **16 minutes and 23 seconds**.

14. Chapter 2 Solutions

Introduction

1. **Exercise 1:** Number Base Conversions

1) Conversion to Base 10

To convert from base b to base 10, use the positional weight method: $d_n \cdot b^n + \cdots + d_0 \cdot b^0$.

a) Base 2 to 10

A = $1101_2 : (1 \cdot 2^3) + (1 \cdot 2^2) + (0 \cdot 2^1) + (1 \cdot 2^0) = 8 + 4 + 0 + 1 = 13$

B = $1001001_2 : 64 + 8 + 1 = 73$

C = $1001_2 : 8 + 1 = 9$

D = $110000_2 : 32 + 16 = 48$

b) Base 8 to 10

E = $36_8 : (3 \cdot 8^1) + (6 \cdot 8^0) = 24 + 6 = 30$

F = $15_8 : (1 \cdot 8) + 5 = 13$

G = $67_8 : (6 \cdot 8) + 7 = 48 + 7 = 55$

c) Base 16 to 10 (Note: A=10, B=11)

H = $9A_{16}: (9 \cdot 16) + 10 = 144 + 10 = 154$

I = $20_{16} : 2 \cdot 16 = 32$

J = $10_{16} : 1 \cdot 16 = 16$

K = $AB_{16} : (10 \cdot 16) + 11 = 160 + 11 = 171$

L = $101_{16} : (1 \cdot 16^2) + (0 \cdot 16) + 1 = 256 + 1 = 257$

2) Conversion from Base 10 to Base b (Using repeated division by b).

a) Base 10 to 2

M = 20 : $20/2 = 10(R0), 10/2 = 5(R0), 5/2 = 2(R1), 2/2 = 1(R0), 1/2 = 0(R1)$ → 10100

N = 31: $2^5 - 1$ → 11111

O = 256 : 2^8 → 100000000

P = 400: $256 + 128 + 16$ → 110010000

b) Base 10 to 8

Q = 20 : $20/8 = 2$ Remainder 4 → 24

R = 13 : $13/8 = 1$ Remainder 5 → 15

S = 208 : $208/8 = 26(R0), 26/8 = 3(R2), 3/8 = 0(R3)$ → 320

c) Base 10 to 16

T = 70 : $70/16 = 4$ Remainder 6 → 46

U = 512 : $512/16 = 32(R0), 32/16 = 2(R0)$ → 200

V = 228 : $228/16 = 14$ Remainder 4. (14 is E) → E4

W = 450 : $450/16 = 28(R2), 28/16 = 1(R12)$. (12 is C) → 1C2

3) Direct Binary ↔ Hexadecimal

Group bits by 4 starting from the right.

a) Binary to Hex

X = 10 1110 : $(0010)(1110) = 2,14$ → 2E

Y = 10 1001 0110 : $(0010)(1001)(0110)$ →296

Z = 100 1001 1101 : $(0100)(1001)(1101)$ → 49D

b) Hex to Binary (Each digit becomes 4 bits)

A = 10 : $1 = 0001, 0 = 0000$ → 10000

B = 7C : $7 = 0111, C = 1100$ → 1111100

C = ABC : $A = 1010, B = 1011, C = 1100$ → 101010111100

D = 1E6 : $1 = 0001, E = 1110, 6 = 0110$ → 111100110

1. Exercise 2: Base-to-Base Conversion

a) $(18)_9$ **to base 10**: $(1 \cdot 9) + 8 = 17$

b) $(112)_5$ **to base 6**:

1) To base 10: $(1 \cdot 25) + (1 \cdot 5) + 2 = 32_{10}$
2) To base 6: $32/6 = 5(R2), 5/6 = 0(R5) \rightarrow 52$

c) $(121)_3$to **base 2**:

1) To base 10: $(1 \cdot 9) + (2 \cdot 3) + 1 = 16_{10}$
2) To base 2: 16 is 2^4 →10000

2. Exercise 3 : Determining an Unknown Base

1) Find Base B for $(36)_B = (27)_{10}$

Set up the equation: $3B + 6 = 27 \Rightarrow 3B = 21 \Rightarrow$ **B = 7**.

2) Determine x and y for $(1x6, y3)_6 = (71.25)_{10}$

1. Solving for x (Integer Part)

The integer part of the equation is: $(1x5)_6 = 71_{10}$

Using the positional weight method for base 6:

$$(1 \times 6^2) + (x \times 6^1) + (5 \times 6^0) = 71$$

Step-by-step:

1) Simplify the powers: $36 + 6x + 5 = 71$
2) Combine the constants: $41 + 6x = 71$
3) Subtract 41 from both sides: $6x = 71 - 41$
4) Calculate: $6x = 30$
5) Divide by 6: $x = 30 / 6 = 5$

   Result: $x = 5$

2. Solving for y (Fractional Part)

The fractional part of the equation is: $(0, y3)_6 = 0.25_{10}$

Using negative powers of 6: $(y \times 6^{-1}) + (3 \times 6^{-2}) = 0.25$

Step-by-step:

1) Write as fractions: $\frac{y}{6} + \frac{3}{36} = \frac{1}{4}$
2) Simplify $3/36$: $\frac{y}{6} + \frac{1}{12} = \frac{1}{4}$
3) Find a common denominator (12): $\frac{2y}{12} + \frac{1}{12} = \frac{3}{12}$
4) Subtract $1/12$ from both sides: $\frac{2y}{12} = \frac{2}{12}$
5) Solve for y: $2y = 2 \Rightarrow y = 1$

   Result: $y = 1$

3. **Exercise 4:** Arithmetic Operations in Binary

Operation	Result (Binary)	Verification (Decimal)
a. 1000 + 0110	1110	8 + 6 = 14
b. 1101001 + 10110	1111111	105 + 22 = 127
c. 1001000 + 1011001	10100001	72 + 89 = 161

Note for (c): If only 7 bits are available, an **overflow** occurs. the result would be truncated to 0100001 (33).

Operation	Result (Binary)	Verification (Decimal)
d. 101101 – 10010	11011	45 - 18 = 27
e. 10100 – 1001	1011	20 - 9 = 11

f. 11001101 × 1011	11001101 x 1011 ---------- 11001101 110011010 0000000000 + 11001101000 ------------- 100011001111	205 × 11 = 2255
g. 1100101010 ÷ 1001	1100101010 | 1001 -1001 | 1011010 111 - 0 1110 - 1001 1011 -1001 100 - 0 1001 - 1001 00 0 0	$810 / 9 = 90$ (1011010_2)
h. 101010 ÷ 101	101010 | 101 -101 | 1000.01 00 - 0 001 - 0 0010 - 0 100 - 0 1000 - 101 11	$42 / 5 = 8.4$

15. Chapter 3 Solutions

Information Representation

1. **Exercise 1:** Signed Integer Representation

We consider an 8 bit representation of signed integers.

4) Complete the following table:

Decimal	Binary	SVA	CA1 one's complement	CA2 two's complement
-20	00010100	10010100	11101011	11101100
-14	00001110	10001110	11110001	11110010
+20	00010100	00010100	00010100	00010100
-120	01111000	11111000	10000111	10001000
-128	10000000	/	/	10000000

128 cannot be represented in SVA or CA1 with 8 bits.

Encoding ranges

Method	**Interval**	**On 8 bits (n=8)**	**On 16 bits(n=16)**
S+VA	$[-(2^{(n-1)} - 1) , +(2^{(n-1)} - 1)]$	−127 to +127	−32767 to +32767
CA1	$[-(2^{(n-1)} - 1) , +(2^{(n-1)} - 1)]$	−127 to +127	−32767 to +32767
CA2	$[-2^{(n-1)} , +(2^{(n-1)} - 1)]$	−128 to +127	−32768 to +32767

On 16 bits

Writing −512 in SVA

512_{10} = 10 0000 0000_2 (10 bits magnitude)

SVA requires 1 sign bit + magnitude bits

Thus minimum bits = 11 bits

Minimum full bytes = 16 bits

Therefore at least 2 bytes are required.

2. **Exercise 2:** SVA, One's Complement, Two's Complement

Give the signed decimal value of each of the following representations:

(**0**001 0000) $_{SVA}$ Sign = 0 Magnitude = 16 Result = +16	(**0**0011010) $_{CA1}$ MSB = 0 Positive Result = 26	(**0**001 0001) $_{CA2}$ Positive Result = 17
(1001 1010) $_{SVA}$ Sign = 1 Magnitude = 0011010_2 = 26 **Result = −26**	(11100100) $_{CA1}$ Sign = 1 Invert → 00011011 = 27 **Result = −27**	(1110 0101) $_{CA2}$ Sign = 1 Invert → 00011010 +1 → 00011011 = 27 **Result = −27**

3. **Exercise 3:** Arithmetic Operations in One's Complement

Using 8 bit integers in one's complement, perform the following calculations:

a) +10-12	**b) 50-7**	**c) 18 +(-5)**	**d) -34 + (-12)**
0000 1010 + 1111 0011 = 1111 1101**(ca1)** = 0000 0010**(2)** **= - 2(10)**	0011 0010 + 1111 1000 = 0010 1010 + 1 = 0010 1011**(2)**	0001 0010 + 1111 1010 = 0000 1100 + 1 = 0000 1101**(2)**	1101 1101 +1111 0010 =1100 1111(ca1) =0011 0000**(2)**

	= + **43**$_{(10)}$	= + **13**$_{(10)}$	=- **48**$_{(10)}$

4. **Exercise 4:** Arithmetic Operations in Two's Complement

Perform the following additions in two's complement:

(a) **0110 1011 + 1011 1101**	**(b)** **1001 0110 + 1111 1011**	**(c)** **0110 1111 + 0001 1001**	**(d)** **1000 0010 + 1010 1011**
0110 1011 (107) + 1011 1101 (-67)	1001 0110 (-106) + 1111 1011 (-5)	0110 1111 (111) + 0001 1001 (25)	1000 0010 (-126) + 1010 1011 (-85)
= 0010 1000 $_{(ca2)}$ = 0010 1000 $_{(2)}$ = +40$_{(10)}$	= 1001 0001$_{(ca2)}$ = 0110 1111 $_{(2)}$ =-111$_{(10)}$	= 1000 1000$_{(ca2)}$ = 0111 1000 $_{(2)}$ = - 120$_{(10)}$	= 0010 1101$_{(ca2)}$ = 0010 1101 $_{(2)}$ = + 45$_{(10)}$

5. **Exercise 5:** IEEE 754 Single Precision Floating Point(Encoding)

Represent the following real numbers in IEEE 754 single precision floating point format. Express the result in hexadecimal:

a) **+128.0**

The real number 128 is positive, therefore the sign bit is **0**.

$128_{10} = 10000000.0_2$

Normalized form: $1.0 \times 2^{+7}$

Mantissa (23 bits): **00000000000000000000000**

Exponent: 7 + 127 = 134 = **10000110_2**

IEEE 754 (binary): 0**100 0011 0**000 0000 0000 0000 0000 0000

Hexadecimal: **43000000**

b) **−32.625**

The real number −32.625 is negative, therefore the sign bit is **1**.

$32_{10} = 100000_2$

$0.625_{10} = 0.101_2$

$32.625_{10} = 100000.101_2$

Normalized form: $1.00000101 \times 2^{+5}$

Mantissa (23 bits): **00000101000000000000000**

Exponent: 5 + 127 = 132 = $\mathbf{10000100_2}$

IEEE 754 (binary):**1100 0010 0000 0010 1000 0000 0000 0000**

Hexadecimal: **C2028000**

c) **+18.125**

The real number 18.125 is positive, therefore the sign bit is **0**.

$18_{10} = 10010_2$

$0.125_{10} = 0.001_2$

$18.125_{10} = 10010.001_2$

Normalized form: $1.0010001 \times 2^{+4}$

Mantissa (23 bits):**00100010000000000000000**

Exponent: 4 + 127 = 131 = $\mathbf{10000011_2}$

IEEE 754 (binary):

0100 0001 1001 0001 0000 0000 0000 0000

Hexadecimal: **41910000**

d) **+∞**

Sign bit: **0**

Exponent: **11111111**

Mantissa: **00000000000000000000000**

IEEE 754 (binary): **0111 1111 1000 0000 0000 0000 0000 0000**

Hexadecimal: **7F800000**

e) −8.75

The real number −8.75 is negative, therefore the sign bit is **1**.

$8_{10} = 1000_2$

$0.75_{10} = 0.11_2$

$8.75_{10} = 1000.11_2$

Normalized form: $1.00011 \times 2^{+3}$

Mantissa (23 bits): **00011000000000000000000**

Exponent: 3 + 127 = 130 = **10000010_2**

IEEE 754 (binary):**1100 0001 0000 1100 0000 0000 0000 0000**

Hexadecimal: C10C0000

6. **Exercise 6:** IEEE 754 Single Precision Floating Point(Decoding)

Determine the decimal value of each of the following numbers encoded in IEEE 754 single precision format:

a) 0100 0001 1100 1100 00000000 00000000

<table>
<tr><td colspan="2">Hexadecimal</td><td colspan="2">4</td><td>1</td><td colspan="2">C</td><td>C</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td colspan="2">Binary</td><td>0</td><td>100</td><td>0001</td><td>1</td><td>100</td><td>1100</td><td>0000</td><td>0000</td><td>0000</td><td>0000</td></tr>
<tr><td>IEEE 774</td><td>Sign: +</td><td colspan="2">Biased exponent: 131
Unbiased exponent :
131–127 = +4</td><td colspan="8">Mantissa :
10011000000000000000000</td></tr>
<tr><td colspan="12">So the normalized significand is: = + 1, 10011x 2^{+4} ,Final decimal value: +25.5</td></tr>
</table>

b) **1011 1111 11000000 00000000 00000000**

Hexadecimal	B	F	C	0	0	0	0	0
Binary	1 011	1111	1 100	0000	0000	0000	0000	0000

IEEE 774	Sign : -	Biased exponent: 127 Unbiased exponent : 127–127 = 0	Mantissa : **100 0000 0000 0000 0000 0000**

So the normalized significand is: **- 1, 10 x 2^0** , Final decimal value: **-1.5**

c) (**C1BC0000**) 16

Hexadecimal	C	1	B	C	0	0	0	0
Binary	1 100	0001	1 011	1100	0000	0000	0000	0000

IEEE 774	Sign: -	Biased exponent: 131 Unbiased exponent : 131–127 = +4	Mantissa : **011 1100 0000 0000 0000 0000**

So the normalized significand is: **- 1, 0111100x 2^{+4}** , Final decimal value: **-23,5**

d) (**3D800000**)16

Hexadecimal	3	D	8	0	0	0	0	0
Binary	0 011	1101	1 000	0000	0000	0000	0000	0000

IEEE 774	Sign : **+**	Biased exponent: 123 Unbiased exponent : 123–127 = -4	Mantissa : **000 0000 0000 0000 0000 0000**
	the normalized significand is: **+ 1,0 x 2^{-4}** =0.0001 , Final decimal value: **+ 0.0625**		

7. **Exercise 7:** Binary Code / Gray Code

c) Convert the following binary numbers into Gray code:

Binary	GRAY
$(100101)_2$	110111
$(10110)_2$	11101
$(10011001)_2$	11010101

d) Give the binary code of the following numbers expressed in Gray code:

GRAY	Binary
$(10111)_{gray}$	11010
$(101101)_{gray}$	110110
$(10100101)_{gray}$	11000110

8. **Exercise 8:** ASCII Code Encoding and Decoding

c) Using ASCII code, decode the following sentence given in hexadecimal:

54	64	20	33	20	73	74	72	75	63	74	75	72	65	20	6d	61	63	68	69	6e	65	20	73	31
T	d		3		s	t	r	u	c	t	u	r	e		m	a	c	h	i	n	e		s	1

d) Using ASCII code, decode the following sentence:

85	110	105	118	101	114	115	105	116	195	169	32	65	109	97	114	32	84	101	108	105	100	106	105
U	n	i	v	e	r	s	i	t	é			A	m	a	r		T	e	l	i	d	j	i

16. Chapter 4 Solutions

Boolean Algebra

1. **Exercise 1:** Truth Table Construction

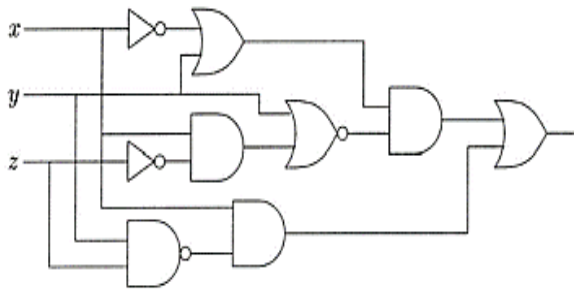
Here is the truth table for $F(A,B,C) = \overline{(A \cdot B)} \cdot (C + B) + A \cdot \bar{B} \cdot C$

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

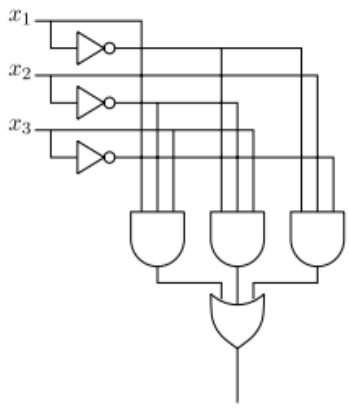
2. **Exercise 2:** Logic Diagram

Give the logic diagram of the following functions:

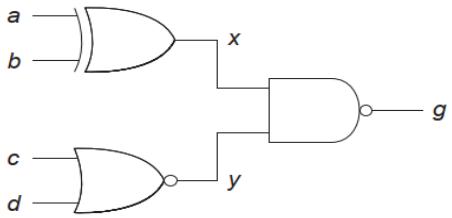
$\mathbf{F(x,y,z) = (\bar{x} + y)\overline{(y + x\bar{z})} + x\bar{y}z}$



$\mathbf{F(x_1,x_2,x_3) = x_1\overline{x_2}x_3 + \overline{x_1}\,\overline{x_2}\,x_3 + \overline{x_1}x_2\overline{x_3}}$

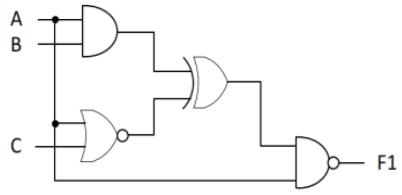


$\mathbf{G(a,b,c,d) = \overline{(a \oplus b) \cdot (\overline{c + d})}}$

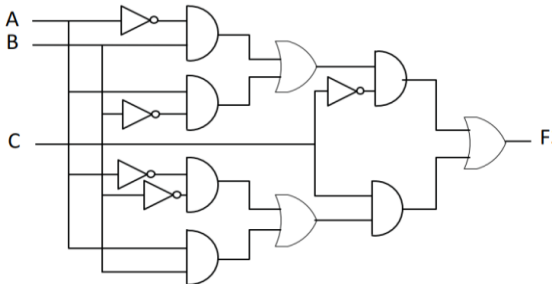


3. **Exercise 3:** Derivation of Boolean Expressions

To find the logical expression for **F1**, we can break the circuit down by identifying the output of each individual gate from left to right.



$$F1 = \overline{\left((A \cdot B) \oplus \left(\overline{A + C}\right)\right) \cdot A}$$



$$F2 = \left((A \oplus B) \cdot \overline{C}\right) + \left(\left(\overline{A \oplus B}\right) \cdot C\right)$$

Simplification: $F = A \oplus B \oplus C$

4. **Exercise 4:** Universal gates

c) Rewrite the following functions using **NAND** gates only.

1) $x\overline{y} + \overline{x}y = \overline{\overline{x\overline{y} + \overline{x}y}} = \overline{\overline{x\overline{y}} \cdot \overline{\overline{x}y}}$

2) $xy + xz + yz = \overline{\overline{xy + xz + yz}} = \overline{\overline{xy} \cdot \overline{xz} \cdot \overline{yz}}$

3) $\overline{(x + z)(y + z)} = \overline{\overline{(x + z)} \cdot \overline{(y + z)}} = \overline{\overline{x} \cdot \overline{z} \cdot \overline{y} \cdot \overline{z}}$

d) Rewrite the following functions using **NOR** gates only.

1) $(x + y)(\overline{x} + \overline{y}) = \overline{\overline{(x + y)(\overline{x} + \overline{y})}} = \overline{\overline{(x + y)} + \overline{(\overline{x} + \overline{y})}}$

2) $(x + y)(x + z)(y + z) = \overline{\overline{(x + y)(x + z)(y + z)}} = \overline{\overline{(x + y)} + \overline{(x + z)} + \overline{(y + z)}}$

3) $\overline{xy + \overline{x}z} = \overline{\overline{xy}} + \overline{\overline{\overline{x}}z} = \overline{\overline{\overline{x} + \overline{y}} + \overline{x + \overline{z}}}$

5. **Exercise 5:** Algebraic Simplification of Boolean Expressions

Simplified Expressions

1) $\boldsymbol{F(A,B) = AB + A\overline{B}}$

- Factor out A: $A(B + \overline{B})$
- Since $(B + \overline{B}) = 1$, the result is $\boldsymbol{A}$.

2) $\boldsymbol{F(A,B) = (A + B)(A + \overline{B})}$

- Apply the Distributive Law: $A + (B \cdot \overline{B})$
- Since $(B \cdot \overline{B}) = 0$, the result is $\boldsymbol{A}$.

3) $\boldsymbol{F(A,B) = A + AB}$

- Apply the Absorption Law: $A(1 + B)$
- Since $(1 + B) = 1$, the result is $\boldsymbol{A}$.

4) $\boldsymbol{F(A,B) = A(A + B)}$

- Apply the Absorption Law: $AA + AB = A + AB$
- The result is $\boldsymbol{A}$.

5) $\boldsymbol{F(A,B) = A + B + \overline{AB}}$

- Use De Morgan's on the last term: $A + B + (\overline{A + B})$
- Let $X = (A + B)$. The expression is $X + \overline{X} = 1$. The result is $\mathbf{1}$.

6) $\boldsymbol{F(A,B) = A + \overline{A}B}$

- Apply the Redundancy Law: $(A + \overline{A})(A + B)$
- Since $(A + \overline{A}) = 1$, the result is $\boldsymbol{A + B}$.

7) $\boldsymbol{F(A,B) = A(\overline{A} + B)}$

- Distribute A: $A\overline{A} + AB$
- Since $A\overline{A} = 0$, the result is $\boldsymbol{AB}$.

8) $\boldsymbol{F(A,B,C) = AB + A\overline{B}C}$

- Factor out A: $A(B + \overline{B}C)$
- Apply Redundancy Law inside: $A(B + C)$. The result is $\boldsymbol{AB + AC}$.

9) $\boldsymbol{F(A,B) = A\overline{B} + \overline{A}\,\overline{B} + \overline{A}B}$

- Combine first two terms: $\overline{B}(A + \overline{A}) + \overline{A}B = \overline{B} + \overline{A}B$
- Apply Redundancy Law: $(\overline{B} + \overline{A})(\overline{B} + B) = \overline{B} + \overline{A}$. The result is $\boldsymbol{\overline{A} + \overline{B}}$.

10) $\boldsymbol{F(A,B,C) = (A + B)(A + \overline{B} + C)}$

- Apply Distributive Law (A is common): $A + B(\overline{B} + C)$
- Expand: $A + B\overline{B} + BC$. The result is $\boldsymbol{A + BC}$.

11) $\boldsymbol{F(A,B,C) = \overline{A\overline{B}}C + A\overline{B}C + \overline{A}BC + ABC}$

- Group terms: $\overline{B}C(\overline{A} + A) + BC(\overline{A} + A)$
- Simplify: $\overline{B}C + BC = C(\overline{B} + B)$. The result is $\boldsymbol{C}$.

12) $\boldsymbol{F(A,B,C,D) = \overline{A}BC + A\overline{B}D + A\overline{C}D + BC\overline{D}}$

- Group the terms by common variables:

  Group the first and last terms: $BC(\overline{A} + \overline{D})$
  Group the middle two terms: $AD(\overline{B} + \overline{C})$

- Apply De Morgan's Law to the terms in parentheses:

  $\overline{A} + \overline{D}$ becomes $\boldsymbol{\overline{AD}}$

  $\overline{B} + \overline{C}$ becomes $\boldsymbol{\overline{BC}}$

- Combine the results:

$$F = BC(\overline{AD}) + AD(\overline{BC})$$

- The Final Result: This matches the standard form of the XOR gate, where $X\overline{Y} + \overline{X}Y = X \oplus Y$. If we let $X = BC$ and $Y = AD$: $\boldsymbol{F = BC \oplus AD}$

13) $\boldsymbol{F(A,B,C) = AB + (C\overline{B} + \overline{C}) + C\overline{A}}$

- Simplify inner term $(C\overline{B} + \overline{C})$ using Redundancy Law: $(\overline{C} + \overline{B})$
- Expression: $AB + \overline{C} + \overline{B} + C\overline{A}$
- Combine $AB + \overline{B}$ to get $A + \overline{B}$. Combine $\overline{C} + C\overline{A}$ to get $\overline{C} + \overline{A}$.
- $A + \overline{B} + \overline{C} + \overline{A}$. Since $A + \overline{A} = 1$, the result is $\mathbf{1}$.

6. **Exercise 6:** SOP and POS Forms

1) Generate a corresponding Boolean (logic) expression:

a) Sum-of-Products (SOP) Form:

To find the SOP form, we look at the rows where the output $F(A, B, C)$ is 1

SOP Expression: $F(A, B, C) = \overline{A}BC + A\overline{B}C + AB\overline{C} + ABC$

b) Product-of-Sums (POS) Form

To find the POS form, we look at the rows where the output $F(A, B, C)$ is 0.

POS Expression: $F(A, B, C) = (A + B + C)\left(A + B + \overline{C}\right)\left(A + \overline{B} + C\right)\left(\overline{A} + B + C\right)$

2) Simplification of Expression (a)

We can simplify the SOP expression using the rules of Boolean algebra (specifically the Distributive and Complement laws):

Original: $F = \overline{A}BC + A\overline{B}C + AB\overline{C} + ABC$

Rearrange and group terms: Group the last term ABC with others to simplify multiple parts (since $X + X = X$):

$$F = \left(\overline{A}BC + ABC\right) + \left(A\overline{B}C + ABC\right) + \left(AB\overline{C} + ABC\right)$$

Factor out common terms:

$$F = BC\left(\overline{A} + A\right) + AC\left(\overline{B} + B\right) + AB\left(\overline{C} + C\right)$$

Apply the Complement Law ($X + \overline{X} = 1$):

$$F = BC(1) + AC(1) + AB(1)$$

Simplified Expression:

$$F(A, B, C) = AB + AC + BC$$

7. **Exercise 7:** Algebraic Simplification

Let the following function be given:

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

1. Simplification of F

Step 1: Group terms with $\bar{C}$

Isolating terms containing $\bar{C}$: $(\bar{A}\,\bar{B}\bar{C} + \bar{A}B\bar{C} + AB\bar{C} + A\bar{B}\bar{C})$

Factoring out $\bar{C}$: $\bar{C}(\bar{A}\,\bar{B} + \bar{A}B + AB + A\bar{B})$

Within the parentheses, group by A:

$$\bar{C}[\bar{A}(\bar{B} + B) + A(B + \bar{B})]$$

Since $(B + \bar{B}) = 1$:

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

Step 2: Remaining terms

The terms left over from the original expression are:

$$\bar{A}BC + A\bar{B}C$$

Factoring out C:

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

Step 3: Combine and Final Simplification

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

Using the Absorption Identity ($X + \bar{X}Y = X + Y$), where $X = \bar{C}$:

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

2. **Find $\bar{F}$**

To find the complement, we apply De Morgan's Laws to the simplified expression.

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

Apply De Morgan's to the main OR gate:

$$\bar{F} = \left(\overline{\bar{C}}\right) \cdot \left(\overline{\bar{A}B + A\bar{B}}\right)$$

Simplify the first term: $\overline{\bar{C}} = C$

Apply De Morgan's to the second term: $\overline{\bar{A}B + A\bar{B}} = \left(\overline{\bar{A}B}\right) \cdot \left(\overline{A\bar{B}}\right)$

Expanding those: $(A + \bar{B}) \cdot (\bar{A} + B)$

Combine the results: $\bar{\boldsymbol{F}} = \boldsymbol{C}(\boldsymbol{A} + \bar{\boldsymbol{B}})(\bar{\boldsymbol{A}} + \boldsymbol{B})$

Simplified $\boldsymbol{F}$: $\bar{\boldsymbol{C}} + \boldsymbol{A} \oplus \boldsymbol{B}$

Complement $\bar{F}$: $C(A \odot B)$ (Note: $A \odot B$ is the XNOR operation)

8. **Exercise 8:** Simplification and Equivalence Relations

Prove algebraically the following relations:

To prove these Boolean algebra relations, we rely on fundamental laws such as De Morgan's Theorems, the Distributive Law, and Identity/Complement Laws.

1. Prove: $\boldsymbol{AB + ACD + \bar{B}D = AB + \bar{B}D}$

Step 1: Expand the middle term. We introduce $(B + \bar{B})$ because it equals 1.

$$AB + ACD(B + \bar{B}) + \bar{B}D = AB + ABCD + A\bar{B}CD + \bar{B}D$$

Step 2: Group and factorize. Group the first two terms and the last two terms.

$$AB(1 + CD) + \bar{B}D(AC + 1)$$

Step 3: Simplify using the Identity Law. Since $(1 + X) = 1$:

$$AB(1) + \bar{B}D(1) = AB + \bar{B}D$$

2. Prove: $(\boldsymbol{\bar{A} + B})(\boldsymbol{A + C})(\boldsymbol{B + C}) = (\boldsymbol{\bar{A} + B})(\boldsymbol{A + C})$
   (This is a variation of the Consensus Theorem)

Step 1: Expand the $(\boldsymbol{B + C})$ term. Add $A\bar{A}$ (which is 0) to the third term.

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

Step 2: Use the Distributive Law $(X + YZ = (X + Y)(X + Z))$ on the last term:

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

Step 3: Rearrange and identify redundancies.

$$[(\bar{A} + B)(\bar{A} + B + C)] \cdot [(A + C)(A + C + B)]$$

Step 4: Use the Absorption Law $(X(X + Y) = X)$:

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

3. Prove: $\boldsymbol{AB + \bar{B}C = (A + \bar{B})(B + C)}$

Step 1: Expand terms.

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

Step 2: Rearrange.

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

Step 3: Add **0** in the form of $\boldsymbol{B\bar{B}}$ and factorize.

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

Step 4: Final factorization.

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

4. Prove: $\overline{\boldsymbol{A\bar{B} + \bar{A}B}} = \boldsymbol{AB + \bar{A}\,\bar{B}}$

Step 1: Apply De Morgan's First Law De Morgan's law states that $\overline{X + Y} = \bar{X} \cdot \bar{Y}$. Applying this to the entire expression: $\overline{A\bar{B} + \bar{A}B} = \left(\overline{A\bar{B}}\right) \cdot \left(\overline{\bar{A}B}\right)$

Step 2: Apply De Morgan's Second Law

Now, apply $\overline{X \cdot Y} = \bar{X} + \bar{Y}$ to each of the two terms inside the product:

$$\left(\overline{A\bar{B}}\right) = (\bar{A} + B)$$

$$\left(\overline{\bar{A}B}\right) = (A + \bar{B})$$

Resulting Expression: $(\bar{A} + B)(A + \bar{B})$

Step 3: Expand using the Distributive Law (FOIL)

Multiply the terms together:

$$(\bar{A} \cdot A) + (\bar{A} \cdot \bar{B}) + (B \cdot A) + (B \cdot \bar{B})$$

Step 4: Apply the Complement Law

In Boolean algebra, any variable ANDed with its complement is 0 ($X \cdot \bar{X} = 0$):

$$\left(\underbrace{\bar{A}A}_{=0}\right) + \overline{A\bar{B}} + BA + \left(\underbrace{B\bar{B}}_{=0}\right)$$

Step 5: Final Simplification

Remove the zeros and rearrange the terms (Commutative Law):

$$0 + \overline{A\bar{B}} + AB + 0 = AB + \bar{A}\,\boldsymbol{\bar{B}}$$

5. Prove: $\overline{(\boldsymbol{A}+\boldsymbol{B})(\bar{\boldsymbol{A}}+\boldsymbol{C})} = (\boldsymbol{A}+\bar{\boldsymbol{B}})(\bar{\boldsymbol{A}}+\bar{\boldsymbol{C}})$

Step-by-Step Derivation:

Step 1: Apply De Morgan's Law ($\overline{X \cdot Y} = \bar{X} + \bar{Y}$):

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

Step 2: Apply De Morgan's Law again ($\overline{X+Y} = \bar{X} \cdot \bar{Y}$):

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

Step 3:Expand using the Distributive Law:

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

Step 4:Simplify Identity ($\bar{A}+A=1$):

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

Step 5: Apply Consensus Theorem (Redundancy):

Adding 0 in the form of $\bar{A}A$ and $\bar{C}C$ to the terms helps demonstrate that $(\bar{B}+\bar{C})$ is redundant:

$$(\bar{A}+\bar{C})(A+\bar{B}) \quad \text{(See Prove 2)}$$

Final Result: $(A+\bar{B})(\bar{A}+\bar{C})$

9. **Exercise 9:** Karnaugh Map Minimization of Boolean Functions

Simplify the following functions using Karnaugh maps:

F1 = AB$\bar{C}$ +$\bar{A}\bar{B}$C + A$\bar{B}\bar{C}$+A$\bar{B}$C

C \ AB	00	01	11	10
0	0	0	1	1
1	1	0	0	1

F2 = ABC+$\bar{A}$BC+$\bar{A}\bar{B}$C+AB$\bar{C}$

C \ AB	00	01	11	10
00	0	0	1	0
01	1	1	1	0

F3=$\bar{A}\bar{B}$C$\bar{D}$+$\bar{A}$B$\bar{C}$D+A$\bar{B}$C$\bar{D}$+$\bar{A}$ $\bar{B}\bar{C}$ $\bar{D}$+AB$\bar{C}$ D
+$\bar{A}$BCD+A$\bar{B}\bar{C}$ $\bar{D}$

CD \ AB	00	01	11	10
00	1	0	0	1
01	0	1	1	0
11	0	1	0	0
10	1	0	0	1

$F_1 = A\bar{C} + \bar{B}C$	$F_2 = \bar{A}C + AB$	$F_3 = \bar{B}\bar{D} + B\bar{C}D + \bar{A}BD$

10. **Exercise 10:** Karnaugh maps

Obtain the simplified functions using Karnaugh maps.

CD \ AB	00	01	11	10
00	0	1	1	0
01	1	1	1	1
11	0	1	1	0
10	0	1	1	0

CD \ AB	00	01	11	10
00	1	0	0	1
01	0	0	0	0
11	0	1	1	0
10	1	0	0	1

CD \ AB	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	0	0	0	0
10	1	1	1	1

$F_1 = F(A,B) = B + \bar{C}D$	$F_2 = F(A,B) = \bar{B}\,\bar{D} + CDB$	$F_3 = F(A,B) = \bar{C}D + C\bar{D}$

CD \ AB	00	01	11	10
00	0	1	1	0
01	1	1	1	1
11	1	1	1	1
10	0	1	1	0

CD \ AB	00	01	11	10
00	0	0	0	0
01	1	0	0	1
11	1	0	0	1
10	0	0	0	0

CD \ AB	00	01	11	10
00	0	1	0	1
01	1	1	1	1
11	1	1	1	1
10	0	1	0	1

$F_4 =$ $B+D$	$F_5 =$ $\bar{B}D$	$F_6 =$ $D + \bar{A}B + A\bar{B}$

CD \ AB	00	01	11	10
00	1	1	1	1
01	0	1	1	0
11	0	1	1	0
10	1	1	1	1

CD \ AB	00	01	11	10
00	1	0	0	1
01	0	1	1	0
11	0	1	1	0
10	1	0	0	1

CD \ AB	00	01	11	10
00	1	0	0	1
01	1	1	1	1
11	1	1	0	0
10	0	0	0	0

$F_7 =$ $B + \bar{D}$	$F_8 = \bar{B}\bar{D} + BD$	$F_9 =$ $\bar{C}\bar{B} + \bar{A}D + C\bar{D}$

11. Exercise 11: Quine–McCluskey

Consider the truth table of the logical function F(A,B,C,D):

A	B	C	D	F
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1

1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	0

3) Determine the canonical disjunctive expression of F in binary form.

To find the canonical disjunctive normal form (also known as the Sum of Minterms) in binary form for the function $F = \sum m(0,1,2,5,6,7,8,9,10,14)$, we convert each decimal minterm into its 4-bit binary equivalent ($ABCD$).

Minterm (mi)	Binary Form (ABCD)
m_0	0000
m_1	0001
m_2	0010
m_5	0101
m_6	0110
m_7	0111
m_8	1000
m_9	1001
m_{10}	1010
m_{14}	1110

4) Apply the Quine–McCluskey method to minimize the Boolean function F.

Step 1: Group Minterms by Number of 1s

We list the binary representation of each minterm and group them by the count of "1" bits.

Group A1		0	0000	*
Group A2		1	0001	*
		2	0010	*
		8	1000	*
Group A3		5	0101	*
		6	0110	*
		9	1001	*
		10	1010	*
Group A4		7	0111	*
		14	1110	*

Step 2: Combine Groups (Find Prime Implicants)
We combine terms that differ by only one bit. A dash (-) represents the eliminated variable.

Group B1 (A1,A2)		0,1	000-	*
		0,2	00-0	*
		0,8	-000	*
Group B2 (A2,A3)		1,5	0-01	✓
		1,9	-001	*
		2,6	0-10	*
		2,10	-010	*
		8,9	100-	*
		8,10	10-0	*
Group B3 (A3,A4)		5,7	01-1	✓
		6,7	011-	✓
		6,14	-110	*
		10,14	1-10	*

3. merging of min term pairs

Group C1 (B1,B2)		0,1,8,9	-00-	✓
		0,2,8,10	-0-0	✓
Group C2 (B2,B3)		2,6,10,14	--10	✓

Step 3: Prime Implicant Table

We identify which terms are essential to cover all original minterms.

Prime Implicants	0	1	2	5	6	7	8	9	10	14
$\bar{B}\bar{C}$ (0,1,8,9)	X	X					X	X		
$\bar{B}\ \bar{D}$ (0,2,8,10)	X		X				X		X	
$C\bar{D}$ (2,6,10,14)			X		X				X	X
$\bar{A}BD$ $01-1$(5,7)				X		X				
$\bar{A}BC$ $011-$ (6,7)					X	X				

Step 4: Final Simplified Expression

To cover all minterms:

$\bar{B}\bar{C}$ is needed for minterm 1 and 9.

$C\bar{D}$ is needed for minterm 14.

$\bar{B}\bar{D}$ is redundant if we use the others, but it's often included for 0, 2, 8, 10.

To cover 5 and 7, we use $\bar{A}BD$ (from 5,7) or $\bar{A}BC$ (from 6,7). To cover 5 specifically, we need the (5,7) grouping.

The simplified Boolean expression is:

$$F(A,B,C,D) = \bar{B}\bar{C} + C\bar{D} + \bar{A}BD$$

17. References

[1] M. M. Mano and M. Ciletti, *Digital Design: With an Introduction to the Verilog HDL, VHDL, and SystemVerilog*. NY, NY: Pearson, 2018.

[2] A. Cazes and J. Delacroix, *Architecture des machines et des systèmes informatiques*. Paris: Dunod, 2008.

[3] "How to enter ASCII codes on Windows." Accessed: JUN. 1, 2025. [Online]. Available: https://www.keepandshare.com/doc/947949/how-to-enter-ascii-codes-on-windows

[4] P. Zanella, Y. Ligier, and E. Lazard, *Architecture et technologie des ordinateurs - 6e éd. - Cours et exercices corrigés: Cours et exercices corrigés*. Paris: Dunod, 2018.

[5] A. B. Marcovitz, *Introduction to logic design*, 3. ed. Boston: McGraw-Hill, 2010.

[6] S. D. Brown and Z. G. Vranesic, *Fundamentals of digital logic with verilog design*, Internat. ed. Boston: McGraw-Hill, 2003.

[7] "Exercices_et_corriges_Boole.pdf." Accessed: Feb. 20, 2026. [Online]. Available: https://moodle.imt-atlantique.fr/pluginfile.php/18550/mod_resource/content/1/Exercices_et_corriges_Boole.pdf

18. Appendix A

List of Abbreviations

Abbreviation	Full Term	Description
ALU	Arithmetic Logic Unit	Hardware component responsible for arithmetic and logical operations inside the CPU
ASCII	American Standard Code for Information Interchange	7-bit character encoding standard for text representation
B	Byte	Unit of digital information equal to 8 bits
b	Bit	Binary digit, smallest unit of information
BB	Brontobyte	Extremely large digital storage unit
BCD	Binary Coded Decimal	Decimal digit encoded using a 4-bit binary representation
bps	Bits per Second	Unit of data transmission rate

Bps	Bytes per Second	Data transfer rate measured in bytes
CA1	One's Complement	Signed number representation obtained by bit inversion
CA2	Two's Complement	Signed number representation obtained by bit inversion plus 1
CPU	Central Processing Unit	Primary processing unit of a computer system
EB	Exabyte	10^{18} bytes or 2^{60} bytes
EBCDIC	Extended Binary Coded Decimal Interchange Code	8-bit character encoding developed by IBM
EPI	Essential Prime Implicant	Mandatory implicant in Quine McCluskey simplification
GB	Gigabyte	10^{9} bytes or 2^{30} bytes
GHz	Gigahertz	Frequency unit equal to 10^{9} hertz
Hz	Hertz	Unit of frequency, cycles per second
IEEE	Institute of Electrical and Electronics Engineers	Organization defining the IEEE 754 floating point standard

KB	Kilobyte	1024 bytes
Kbps	Kilobits per second	1024 bits per second
KHz	Kilohertz	10^3 hertz
LSB	Least Significant Bit	Lowest order bit in a binary number
MB	Megabyte	1024 kilobytes
Mbps	Megabits per second	1024 kilobits per second
MHz	Megahertz	10^6 hertz
MSB	Most Significant Bit	Highest order bit in a binary number
NAND	NOT AND	Universal logic gate equivalent to the complement of AND
NOR	NOT OR	Universal logic gate equivalent to the complement of OR
PB	Petabyte	10^{15} bytes or 2^{50} bytes
PI	Prime Implicant	Fundamental implicant in Boolean minimization
POS	Product of Sums	Canonical conjunctive Boolean form

QM	Quine McCluskey	Tabular algorithm for Boolean function minimization
RAM	Random Access Memory	Volatile primary memory
SOP	Sum of Products	Canonical disjunctive Boolean form
TB	Terabyte	10^{12} bytes or 2^{40} bytes
UTF	Unicode Transformation Format	Unicode encoding standard
UTF-8	8-bit Unicode Transformation Format	Variable width encoding compatible with ASCII
UTF-16	16-bit Unicode Transformation Format	Variable width encoding using 2 or 4 bytes
UTF-32	32-bit Unicode Transformation Format	Fixed width Unicode encoding
XNOR	Exclusive NOR	Logical equivalence operator
XOR	Exclusive OR	Logical inequality operator
XS-3	Excess-3 Code	Non weighted decimal code obtained by adding 3
ZB	Zettabyte	10^{21} bytes or 2^{70} bytes
YB	Yottabyte	10^{24} bytes or 2^{80} bytes

Machine Structure 1 is an introductory course in computer architecture that examines the fundamental principles governing the organization and operation of digital systems. It covers number representation, binary arithmetic, data encoding, memory structures, and the basic components of a computer system, including the processor and its internal architecture. The course establishes the conceptual and technical foundations necessary to understand how software interacts with hardware, thereby preparing students for more advanced studies in computer engineering and systems design.