1. Boolean Algebra and Logic Gates

Boolean algebra is the mathematical foundation of digital logic design. It provides the rules and notation used to analyze and simplify the switching (logic) circuits that make up every digital computer. This module introduces binary codes, binary storage devices, binary logic operations, and the basic digital logic gates.

1.1 Binary Codes

A binary code represents each piece of information — decimal digits, letters, or special symbols — as a group of bits (0s and 1s). Because computers operate only on binary signals, all data must eventually be encoded in binary form. A code that represents decimal digits is called a binary-coded-decimal (BCD) code.

1.1.1 Binary-Coded Decimal (BCD)

BCD represents each decimal digit (0–9) using a 4-bit binary pattern. It is also known as the 8421 code because the weights of the four bits are 8, 4, 2, and 1.

Decimal	BCD (8421)	Decimal	BCD (8421)
0	0000	5	0101
1	0001	6	0110
2	0010	7	0111
3	0011	8	1000
4	0100	9	1001

Note: Combinations 1010 through 1111 (10–15) are invalid in BCD and are never used.

1.1.2 Other Common Binary Codes

- Excess-3 Code: Obtained by adding 3 (0011) to the BCD value of each decimal digit; it is self-complementing, which simplifies decimal arithmetic circuits.
- Gray Code: A reflected code in which only one bit changes between successive values; used to minimize errors in mechanical/optical position encoders.
- ASCII (American Standard Code for Information Interchange): A 7-bit (or 8-bit extended) code used to represent letters, digits, punctuation, and control characters in text.

- Unicode: A modern character-encoding standard supporting characters from virtually all world languages, using 16 or 32 bits per character.
- Parity Bit Codes: An extra bit added to a binary code to enable simple error detection (even parity or odd parity).

1.1.3 Why Binary Codes Matter

Binary codes allow decimal numbers, alphanumeric characters, and instructions to be processed uniformly by digital hardware that recognizes only two voltage levels (representing 0 and 1). Choosing an appropriate code affects the complexity of the arithmetic and control circuits that operate on it.

1.2 Binary Storage and Registers

Digital circuits need a way to store binary information temporarily or permanently. The basic storage element in digital systems is called a binary cell, and a group of binary cells is called a register.

1.2.1 Binary Cell

A binary cell is a device that possesses two stable states and is capable of storing one bit of information. Examples include a flip-flop, a magnetic core, or a small region on a storage medium. At any given time, a binary cell holds either logic 0 or logic 1, and this state can be sensed (read) or altered (written).

1.2.2 Register

A register is a group of binary cells. An n-bit register consists of n binary cells and is capable of storing any binary information of n bits. A register's state — the collection of bit values stored in each cell — represents the binary information held by the register at that time.

- Registers may be used to hold data, addresses, or control information inside a CPU.
- A transfer of new information into a register is referred to as a load or update operation.
- If each cell in the register changes state independently, the process is called parallel loading; if the bits are entered one at a time, it is called serial loading.

1.2.3 Register Transfer

A digital system is best characterized by the registers it contains and the sequence of microoperations performed on the binary information stored in them. A microoperation is an elementary operation (such as shift, count, clear, or load) performed on the data stored in one or more registers.

1.3 Binary Logic

Binary logic deals with binary variables that take only two discrete values (0 and 1) and with the operations performed on these variables. Binary logic is used to describe, in algebraic form, the manipulation and processing of binary information.

1.3.1 Binary Variables and Logical Operations

A binary variable can hold a value of 1 or 0, where each value represents a distinct logical state — for example, true/false, on/off, or high/low voltage. The three fundamental logical operations from which all others can be derived are AND, OR, and NOT (also called complementation).

Operation	Symbol	Description
AND	$x \cdot y$ or xy	Output is 1 only when both x and y are 1.
OR	$x + y$	Output is 1 when x, y, or both are 1.
NOT	x' or $\bar{x}$	Output is the complement (inverse) of x.

1.3.2 Truth Tables

A truth table lists every possible combination of input values along with the corresponding output for a logic operation or expression. It is the primary tool for defining and verifying the behavior of a logic function.

1.4 Digital Logic Gates

A logic gate is an electronic circuit that operates on one or more binary inputs to produce a single binary output, implementing a specific Boolean function. Gates are the physical building blocks used to construct all digital circuits.

1.4.1 Basic Gates

- AND Gate: Produces logic 1 only when all inputs are 1. Boolean expression: $Y = A \cdot B$
- OR Gate: Produces logic 1 when at least one input is 1. Boolean expression: $Y = A + B$
- NOT Gate (Inverter): Produces the complement of a single input. Boolean expression: $Y = A'$

1.4.2 Universal Gates

- NAND Gate: The complement of AND; output is 0 only when all inputs are 1. $Y = (A \cdot B)'$
- NOR Gate: The complement of OR; output is 1 only when all inputs are 0. $Y = (A + B)'$

NAND and NOR are called universal gates because either one, by itself, can be used to implement any Boolean function — including AND, OR, and NOT.

1.4.3 Exclusive Gates

- XOR (Exclusive-OR): Output is 1 when the inputs differ. $Y = A \oplus B$
- XNOR (Exclusive-NOR): Output is 1 when the inputs are the same. $Y = (A \oplus B)'$

1.4.4 Truth Tables of Basic Gates

A	B	AND	OR	NAND	NOR	XOR	XNOR
0	0	0	0	1	1	0	1
0	1	0	1	1	0	1	0
1	0	0	1	1	0	1	0
1	1	1	1	0	0	0	1

1.4.5 Gate Symbols and Standards

Standard graphic symbols for logic gates are defined by ANSI/IEEE Standard 91-1984. The NOT operation is represented by a small circle (bubble) at the gate output (or input), indicating signal inversion. Multiple-input versions of AND, OR, NAND, and NOR gates are common in practical circuit design.

1.4.6 Applications of Logic Gates

- Combinational circuits: adders, multiplexers, decoders, and encoders.
- Sequential circuits: flip-flops, counters, and registers built from basic gates.
- Control logic: implementing conditions and decision-making in digital systems and processors.

2. Data Representation

Data representation refers to the methods used by digital computers to internally store and manipulate different kinds of data — numbers, characters, and instructions — using binary digits. Correct representation schemes are essential for accurate arithmetic, efficient storage, and correct interpretation of information.

2.1 Data Types

A data type defines the nature of the information stored in a computer's memory or registers and how that information should be interpreted and processed by hardware and software.

- Numeric Data: Represented using number systems (binary, octal, decimal, hexadecimal) and further classified as fixed-point (integers) or floating-point (real numbers).
- Non-Numeric (Alphanumeric) Data: Letters, symbols, and control characters, typically represented using character codes such as ASCII or Unicode.
- Logical (Boolean) Data: Represents true/false or 1/0 values used in decision-making operations.

2.1.1 Number Systems Used in Data Representation

Number System	Base (Radix)	Digits Used
Binary	2	0, 1
Octal	8	0–7
Decimal	10	0–9
Hexadecimal	16	0–9, A–F

Computers internally use the binary number system because binary digits map directly onto the two stable states of electronic circuits (on/off, high/low voltage).

2.2 Complements

Complements are used in digital computers to simplify the subtraction operation and to represent negative numbers. There are two types of complements for each base-r system: the radix complement (r's complement) and the diminished radix complement ((r−1)'s complement).

2.2.1 (r − 1)'s Complement — 1's Complement (Binary)

The 1's complement of a binary number is obtained by inverting every bit of the number — changing every 1 to 0 and every 0 to 1.

Example: The 1's complement of 1010 0110 is 0101 1001.

- Simple to compute using inverter (NOT) gates.
- Has two representations of zero: +0 (0000) and −0 (1111), which complicates arithmetic circuit design.

2.2.2 r's Complement — 2's Complement (Binary)

The 2's complement of a binary number is obtained by taking the 1's complement and adding 1 to the least significant bit.

Example: For 1010 0110 → 1's complement = 0101 1001 → add 1 → 2's complement = 0101 1010.

- Used almost universally to represent signed integers in modern computers.
- Has a single representation for zero, simplifying arithmetic hardware.
- Subtraction can be performed as addition of the 2's complement, eliminating the need for a separate subtractor circuit.

2.2.3 Subtraction Using Complements

To subtract B from A using complements: take the r's complement of B, add it to A, and discard any carry out of the most significant bit (for r's complement arithmetic). This technique allows a single adder circuit to perform both addition and subtraction.

2.3 Fixed Point Representation

In fixed-point representation, the position of the binary (radix) point is fixed and implied — it is not stored explicitly. Typically the point is assumed to be either to the right of the least significant bit (representing integers) or after a fixed number of bits (representing a fraction).

2.3.1 Representing Integers

A fixed-point number is generally divided into two parts: the sign and the magnitude.

- Sign bit: The most significant bit (MSB); 0 indicates a positive number and 1 indicates a negative number.
- Magnitude bits: The remaining bits represent the absolute value of the number.

2.3.2 Signed Number Representations

Method	Description	Range for n bits
Sign-Magnitude	MSB is the sign; remaining bits store magnitude directly.	−(2^(n−1) − 1) to +(2^(n−1) − 1)
1's Complement	Negative numbers stored as the bitwise complement of the positive value.	−(2^(n−1) − 1) to +(2^(n−1) − 1)
2's Complement	Negative numbers stored as the 2's complement of the positive value.	−2^(n−1) to +(2^(n−1) − 1)

Note: 2's complement is the standard scheme used by modern processors because it supports a single representation of zero and simplifies addition/subtraction hardware.

2.3.3 Advantages and Limitations of Fixed-Point Representation

- Advantage: Simple and fast arithmetic circuits; predictable precision.
- Advantage: Efficient for representing whole numbers and counters.
- Limitation: Limited range — cannot efficiently represent very large or very small (fractional) values.
- Limitation: Not suitable for scientific or engineering computations that require a wide dynamic range.

2.4 Floating Point Representation

Floating-point representation is used to store real numbers that require a very large range of magnitudes, including very large and very small fractional values. It represents a number in a form analogous to scientific notation: Mantissa × Base^Exponent.

2.4.1 Structure of a Floating-Point Number

A floating-point number is generally divided into three fields:

- Sign bit (S): Indicates whether the number is positive (0) or negative (1).
- Exponent (E): Represents the power to which the base is raised; usually stored in a biased form so that both large and small exponents can be represented using unsigned bits.
- Mantissa / Significand (M): Represents the significant digits (precision) of the number.

General form: N = (−1)^S × 1.M × 2^(E − bias)

2.4.2 IEEE 754 Standard

The IEEE 754 standard is the most widely used format for floating-point representation in modern computers, defining both single-precision and double-precision formats.

Format	Total Bits	Sign	Exponent	Mantissa	Bias
Single Precision	32	1	8	23	127
Double Precision	64	1	11	52	1023

2.4.3 Normalization

A floating-point number is normalized when the mantissa is adjusted so that there is exactly one non-zero digit before the radix point (in binary, this digit is always 1, so it is often not stored explicitly — the implicit or hidden bit). Normalization maximizes precision for a given number of mantissa bits.

2.4.4 Special Values in IEEE 754

- Zero: Represented when both the exponent and mantissa fields are all zero (with the sign bit indicating +0 or −0).
- Infinity: Represented when the exponent field is all 1s and the mantissa is all zeros.
- NaN (Not a Number): Represented when the exponent field is all 1s and the mantissa is non-zero; used to flag invalid operations such as 0/0.
- Denormalized (subnormal) numbers: Used to represent values smaller than the smallest normalized number, allowing gradual underflow.

2.4.5 Fixed-Point vs. Floating-Point — Comparison

Aspect	Fixed-Point	Floating-Point
Radix point	Fixed / implied position	Variable; tracked via exponent
Range	Limited	Very large (large and small magnitudes)
Precision	Uniform across range	Varies with magnitude
Hardware complexity	Simple, fast	More complex circuitry required
Typical use	Counters, integers, embedded systems	Scientific computation, graphics, engineering

3. Digital Computers

A digital computer is an electronic device that accepts data as input, processes it according to a set of stored instructions (a program), and produces the processed data as output. Unlike analog systems, which represent information as continuously varying physical quantities, digital computers represent all information — numbers, text, images, instructions — in binary (two-state) form.

Introduction

Digital computers have become indispensable across virtually every field of modern life — science and engineering, business, communication, entertainment, and government — because they can perform enormous numbers of arithmetic and logical operations at extremely high speed, reliably and repeatably, according to a stored program.

Key Characteristics of Digital Computers

- Speed: Capable of performing millions to billions of operations per second.
- Accuracy: Produces highly precise and repeatable results, limited mainly by the precision of the data representation used (see fixed-point and floating-point representation).
- Storage: Can store vast amounts of data and instructions for immediate or later use, across the memory hierarchy.
- Automation (Stored-Program Concept): Once a program is loaded into memory, the computer executes its instructions automatically, without requiring manual intervention for each step.
- Versatility: A general-purpose digital computer can be reprogrammed to perform an essentially unlimited range of different tasks.

The Stored-Program Concept

Modern digital computers are built on the stored-program concept, first articulated in the von Neumann architecture: both program instructions and data are stored together in the same memory unit, and instructions are fetched, decoded, and executed automatically in sequence (except when explicitly redirected by control-transfer instructions such as branches and calls).

Note: The stored-program concept is what allows a single piece of hardware to run entirely different programs without being physically rewired — a defining feature that separates general-purpose digital computers from fixed-function digital devices.

Basic Functional Units

Regardless of size or purpose, every digital computer is built from the same basic functional units, which together carry out the process of accepting, processing, storing, and delivering information: an input unit, a memory unit, an arithmetic and logic unit, a control unit, and an output unit. Together, the control unit and the arithmetic and logic unit form the Central Processing Unit (CPU), often called the 'brain' of the computer.

Block Diagram of a Digital Computer

The relationship between the basic functional units of a digital computer, and the flow of information and control between them, is most commonly illustrated with a block diagram, as shown below.

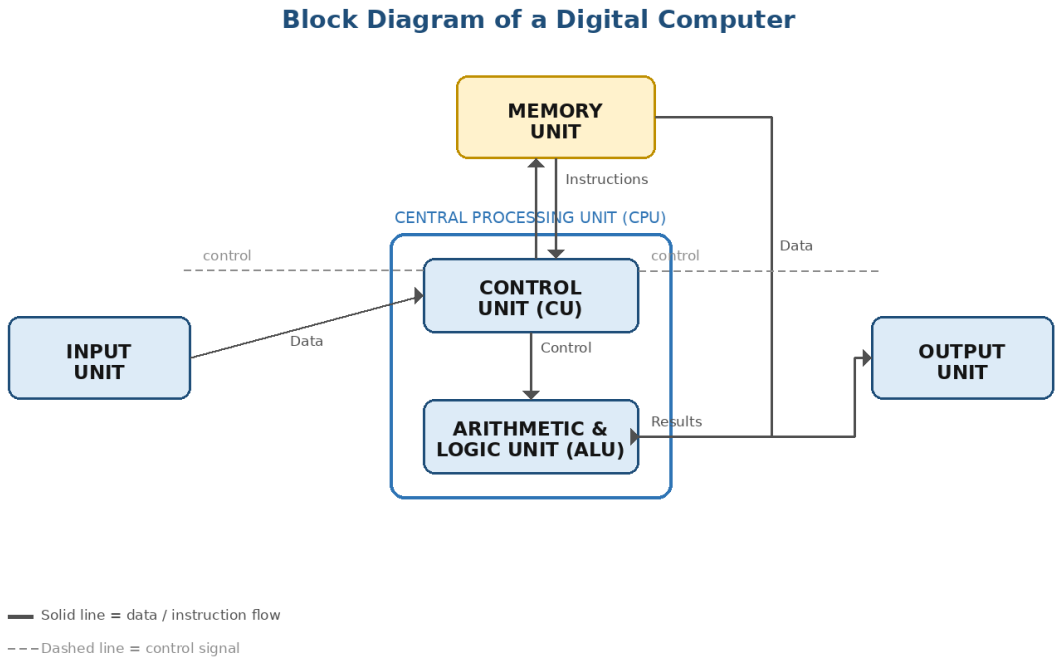

Figure 1.1 — Block diagram of a digital computer, showing data/instruction flow (solid lines) and control signal flow (dashed lines).

Functional Units Explained

Unit	Function
Input Unit	Accepts data and instructions from the outside world (e.g., keyboard, mouse, scanner, network) and converts them into a binary form the computer can process.
Memory Unit	Stores program instructions and data, both before processing (input data) and after processing (results), organized across the memory hierarchy (registers, cache, main memory, auxiliary memory).
Arithmetic and Logic Unit (ALU)	Performs all arithmetic operations (addition, subtraction, multiplication, division) and logical operations (AND, OR, comparison) on data retrieved from memory.
Control Unit (CU)	Directs and coordinates the operation of all other units by generating timing and control signals; fetches instructions from memory, decodes them, and orchestrates their execution.
Output Unit	Converts the processed binary results back into a human-readable or externally usable form (e.g., display, printer, network) and delivers them to the outside world.

Information Flow

- Solid-line paths carry the actual data and instructions being processed — from the input unit into memory, between memory and the CPU, and from the ALU out to the output unit.
- Dashed-line paths carry control signals issued by the control unit to the input and output units, governing when and how each unit performs its part of an operation.

The CPU (control unit + ALU) is the central coordinating unit of the whole system: the control unit fetches instructions from memory and generates the sequence of control signals needed to carry them out, while the ALU carries out the actual data processing specified by each instruction.

Definition of Computer Organization

Computer Organization refers to the operational units and their interconnections that realize the architectural specification of a computer — that is, how a computer's architecture is actually implemented in hardware. It is concerned with structural relationships that are not visible to the programmer, such as:

- Control signals: how the control unit generates the signals needed to execute each instruction.
- Interfaces between the CPU, memory, and I/O devices: the buses, timing, and handshaking used to move data between units.
- Memory technology: the specific type of memory (e.g., SRAM vs. DRAM) and organization (e.g., cache structure, interleaving) used to implement the memory hierarchy.
- Peripheral technology: how input/output devices are connected and managed by the system.

In short, computer organization answers the question: 'How is a given architectural specification actually implemented in hardware?' Two computers with the identical architecture (i.e., the same instruction set) can have very different organizations — for instance, differing in cache size, pipeline depth, or bus width — resulting in different cost and performance, while still being able to run exactly the same machine-language programs.

Computer Design

Computer Design is concerned with the hardware design process of a computer — developing the actual hardware for a computer that satisfies given architectural specifications and organizational choices, within constraints such as cost, power, and performance targets.

- Logic design: designing the digital logic circuits (combinational and sequential) that implement each functional unit — ALU, control unit, registers, memory interface, and so on.
- Hardware component selection: choosing the specific technology and components (e.g., type of memory chips, ALU implementation strategy) used to realize each unit.
- Verification: ensuring the resulting hardware correctly implements the specified architecture and organization, through simulation and testing.

Computer design translates the abstract organizational plan into a concrete, working hardware implementation — it is the bridge between 'how the system should be organized' and 'the actual circuits that make it work.'

Computer Architecture

Computer Architecture refers to the attributes of a computer system that are visible to a programmer — that is, the conceptual structure and functional behavior of the computer as seen by someone writing machine- or assembly-language programs for it, independent of how it is actually implemented in hardware.

Computer architecture is typically concerned with specifying:

- The instruction set (opcodes, instruction formats, and addressing modes) that the machine supports.
- The data types and data representation methods the machine natively supports (e.g., fixed-point, floating-point, character encodings).
- The number and type of programmer-visible registers.
- The input–output mechanisms visible to software (e.g., I/O instructions or memory-mapped I/O addresses).
- Memory addressing techniques and the size of the addressable memory space.

- **Note:** A useful way to remember the distinction: Architecture defines what the computer does (visible to the programmer, e.g., the instruction set), while Organization defines how it is done (the hardware implementation, e.g., control signals, buses, and memory technology). Computer design is the practical engineering process that produces the hardware realizing both.

- **Architecture vs. Organization — Comparison**

Aspect	Computer Architecture	Computer Organization
Focus	Programmer-visible behavior (what the system does)	Hardware implementation (how it is done)
Includes	Instruction set, addressing modes, data types, registers	Control signals, buses, memory technology, interfaces
Visibility	Visible to programmers / software	Hidden from programmers; visible to hardware designers
Example question	'Does the ISA support a MUL instruction?'	'Is multiplication done by an array multiplier or shift-and-add?'