Group Assignment-01

EE1201: Digital Systems

Submitted by

Group-3

Submitted to

Siva Vanjari Sir
Indian Institute of Technology Hyderabad

Date of Submission: February 3, 2025

1 Intro To Digital Systems

1.1 Role of Digital Systems

- Digital systems are integral to various fields, including communication, business, traffic control, medical applications, weather monitoring, and scientific research.
- They are embedded in everyday devices such as digital telephones, televisions, cameras, and handheld touchscreen devices.

1.2 Binary Representation in Digital Systems

- Digital systems operate on discrete elements of information, represented using **binary digits (bits)**—0 and 1.
- Groups of bits, called **binary codes**, represent numbers, characters, and symbols.
- The decimal number system can be converted into binary, where each number is expressed using a sequence of bits (e.g., $7_{10} = 0111_2$).

1.3 Core Components of a Digital Computer

- **Memory Unit**: Stores programs, data, and intermediate results.
- **Central Processing Unit (CPU)**: Executes arithmetic, logic, and data processing operations.
- **Input/Output Devices**: Allow data entry (e.g., keyboards, touchscreens) and result display (e.g., printers, monitors).
- **Communication Unit**: Enables data exchange through networks, such as the Internet.

1.4 Binary Logic and Digital Circuits

- Digital circuits process binary signals using **logic gates** (AND, OR, NOT, etc.).
- **Flip-flops** store binary data and are used in memory and sequential circuits.

1.5 Advantages of Digital Systems

- **High Speed**: Modern digital circuits perform operations at millions of cycles per second.
- **Programmability**: Many digital devices can be reprogrammed for different tasks, making them versatile.
- **Reliability**: Error-correcting codes ensure accurate data storage and transmission.
- **Cost Efficiency**: Advances in **integrated circuit (IC) technology** reduce manufacturing costs while increasing performance.

1.6 Quantization of Data

- Digital systems process both inherently discrete data (e.g., payroll records) and **quantized** continuous data (e.g., temperature readings).
- **Analog-to-Digital Converters (ADCs)** transform continuous signals into digital form (e.g., in digital cameras).

1.7 Digital System Design Using HDL

- Modern digital systems are designed using **Hardware Description Languages (HDLs)**, which describe circuit functionality in textual form.
- HDLs enable simulation, verification, and synthesis of digital circuits before fabrication.
- Proper HDL-based design ensures efficient and functional digital hardware.

1.8 Conclusion

- Digital systems form the foundation of modern computing and information processing.
- Their ability to represent and manipulate binary data makes them highly efficient and widely used.
- Understanding binary representation, logic circuits, and digital design methodologies is essential for working with digital technology.

2 Binary Numbers

2.1 Decimal Number System (Base 10)

A decimal number consists of digits from 0 to 9, with each digit's place value determined by powers of 10. For example, the number 7392 can be expressed as:

$$7 \times 10^3 + 3 \times 10^2 + 9 \times 10^1 + 2 \times 10^0 \tag{1}$$

2.2 Number Systems and Radix (Base)

A number system's radix (or base) determines the set of digits and the positional values of those digits:

- **Decimal System (Base 10):** Uses digits 0-9.
- **Binary System (Base 2):** Uses only two digits: 0 and 1.
- **Octal System (Base 8):** Uses digits 0-7.
- **Hexadecimal System (Base 16):** Uses digits 0-9 and letters A-F.

2.3 Binary Number System (Base 2)

In the binary system, each digit is either 0 or 1 and is multiplied by a power of 2. For example, converting 1100.11_2 to decimal:

$$(1 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (0 \times 2^0) + (1 \times 2^{-1}) + (1 \times 2^{-2}) = 12.75_{10} \tag{2}$$

2.4 Octal Number System (Base 8)

The octal system has eight digits (0 to 7). For example, converting $(127.4)_8$ to decimal:

$$(1 \times 8^2) + (2 \times 8^1) + (7 \times 8^0) + (4 \times 8^{-1}) = 87.5_{10} \tag{3}$$

2.5 Hexadecimal Number System (Base 16)

The hexadecimal system has sixteen symbols (0-9 and A-F). For example, converting $(B65F)_{16}$ to decimal:

$$(11 \times 16^3) + (6 \times 16^2) + (5 \times 16^1) + (15 \times 16^0) = 46687_{10} \tag{4}$$

2.6 Powers of Two and Storage in Computers

The binary system is widely used in computing. Common memory sizes:

- $2^{10} = 1024$ bytes = 1 KB
- $2^{20} = 1,048,576$ bytes = 1 MB
- $2^{30} = 1,073,741,824$ bytes = 1 GB
- $2^{40} = 1,099,511,627,776$ bytes = 1 TB

2.7 Arithmetic Operations in Binary

Binary arithmetic follows similar principles as decimal arithmetic but uses only 0s and 1s.

2.7.1 Binary Addition Example

$$\begin{array}{r} 101011 + \\ \hline 1010000 \end{array} \qquad 100111$$

2.7.2 Binary Subtraction Example

$$\begin{array}{rr} & 101010 \\ - & 100111 \\ \hline & 000011 \end{array}$$

2.7.3 Binary Multiplication Example

$$\begin{array}{rr} & 1011 \\ \times & 101 \\ \hline & 1011 \\ & 0000 \\ + & 1011 \\ \hline & 110111 \end{array}$$

3 Number-Base Conversions

Key Concepts

- Two number representations are **equivalent** if they have the same decimal value (e.g., $(0011)_8$ and $(1001)_2$ both represent 9).
- Conversion from base r to decimal involves expanding the number into a power series and summing the terms.
- Conversion from decimal to base r requires separating the number into its integer and fractional parts, as each part is converted differently.

Decimal to Base-r Conversion

- **Integer Part**: Divide the number by r repeatedly, accumulating remainders. The remainders (in reverse order) form the base-r representation.
- **Fractional Part**: Multiply the fraction by r repeatedly, accumulating integers. The integers form the base-r representation.

Examples

- **Decimal to Binary**:
  - Convert $(41)_{10}$ to binary:

$$\begin{aligned} 41 \div 2 &= 20 \quad \text{remainder } 1 \\ 20 \div 2 &= 10 \quad \text{remainder } 0 \\ 10 \div 2 &= 5 \quad \text{remainder } 0 \\ 5 \div 2 &= 2 \quad \text{remainder } 1 \\ 2 \div 2 &= 1 \quad \text{remainder } 0 \\ 1 \div 2 &= 0 \quad \text{remainder } 1 \end{aligned}$$

    Result: $(41)_{10} = (101001)_2$.

- **Decimal to Octal**:

- Convert $(153)_{10}$ to octal:

$$\begin{aligned} 153 \div 8 &= 19 \quad \text{remainder } 1 \\ 19 \div 8 &= 2 \quad \text{remainder } 3 \\ 2 \div 8 &= 0 \quad \text{remainder } 2 \end{aligned}$$

Result: $(153)_{10} = (231)_8$.

- **Decimal Fraction to Binary**:
  - Convert $(0.6875)_{10}$ to binary:

$$\begin{aligned} 0.6875 \times 2 &= 1.375 \quad \text{integer } 1 \\ 0.375 \times 2 &= 0.75 \quad \text{integer } 0 \\ 0.75 \times 2 &= 1.5 \quad \text{integer } 1 \\ 0.5 \times 2 &= 1.0 \quad \text{integer } 1 \end{aligned}$$

Result: $(0.6875)_{10} = (0.1011)_2$.

- **Decimal Fraction to Octal**:
  - Convert $(0.513)_{10}$ to octal:

$$\begin{aligned} 0.513 \times 8 &= 4.104 \quad \text{integer } 4 \\ 0.104 \times 8 &= 0.832 \quad \text{integer } 0 \\ 0.832 \times 8 &= 6.656 \quad \text{integer } 6 \\ 0.656 \times 8 &= 5.248 \quad \text{integer } 5 \\ 0.248 \times 8 &= 1.984 \quad \text{integer } 1 \\ 0.984 \times 8 &= 7.872 \quad \text{integer } 7 \end{aligned}$$

Result: $(0.513)_{10} = (0.406517)_8$.

Combining Integer and Fractional Parts

- For numbers with both integer and fractional parts, convert each part separately and combine the results.
- Example: $(41.6875)_{10} = (101001.1011)_2$.
- Example: $(153.513)_{10} = (231.406517)_8$.

Table of Powers of Two

n	2^n	n	2^n	n	2^n
0	1	8	256	16	65,536
1	2	9	512	17	131,072
2	4	10	1,024 (1K)	18	262,144
3	8	11	2,048	19	524,288
4	16	12	4,096 (4K)	20	1,048,576 (1M)
5	32	13	8,192	21	2,097,152
6	64	14	16,384	22	4,194,304
7	128	15	32,768	23	8,388,608

Table 1: Powers of Two

4 octal and Hexadecimal Numbers

Key Concepts

- Octal (base-8) and hexadecimal (base-16) systems are widely used in digital systems because they provide a compact representation of binary numbers.
- Each octal digit corresponds to **3 binary digits**, and each hexadecimal digit corresponds to **4 binary digits**.
- Conversion between binary, octal, and hexadecimal is straightforward due to the direct relationship between their bases.

Binary to Octal Conversion

- Partition the binary number into groups of **3 digits** (starting from the binary point).
- Convert each group to its corresponding octal digit.
- Example:

$$(10\,110\,001\,101\,011.\,111\,100\,000\,110)_2 = (26153.7406)_8$$

Binary to Hexadecimal Conversion

- Partition the binary number into groups of **4 digits** (starting from the binary point).
- Convert each group to its corresponding hexadecimal digit.
- Example:

$$(10\,1100\,0110\,1011.\,1111\,0010)_2 = (2C6B.F2)_{16}$$

Octal/Hexadecimal to Binary Conversion

- Convert each octal digit to its **3-digit binary equivalent**.
- Convert each hexadecimal digit to its **4-digit binary equivalent**.
- Examples:

- Octal to Binary:

$$(673.124)_8 = (110\,111\,011.\,001\,010\,100)_2$$

- Hexadecimal to Binary:

$$(306.D)_{16} = (0011\,0000\,0110.\,1101)_2$$

Advantages of Octal and Hexadecimal

- Binary numbers are long and difficult to work with, but octal and hexadecimal provide a compact representation.
- Example: The binary number 111111111111 (12 digits) can be represented as:
  - Octal: 7777 (4 digits)
  - Hexadecimal: FFF (3 digits)
- Hexadecimal is particularly useful for representing bytes (8 bits) with just 2 digits.

Table of Numbers with Different Bases

Decimal (base 10)	Binary (base 2)	Octal (base 8)	Hexadecimal (base 16)
0	0000	00	0
1	0001	01	1
2	0010	02	2
3	0011	03	3
4	0100	04	4
5	0101	05	5
6	0110	06	6
7	0111	07	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

Table 2: Numbers with Different Bases

5 Complement of Numbers

- Complements are used in digital computers to simplify subtraction operation and for logical manipulation
- There are two types of complment for base-r system
  1. Diminished radix complement-(r-1)'s complement
  2. Radix complement-r's Complement

5.1 Diminished Radix Complement

- Given a number N in base r having n digits, the (r-1)'s complement of N is its diminished radix complement , is defined as $(r^n - 1) - N$.
- For example in decimal system,

$$\text{The 9's complement of 546700 is } 999999 - 546700 = 453299. \quad (5)$$
$$\text{The 9's complement of 012398 is } 999999 - 012398 = 987601. \quad (6)$$

From the above example it is clear that 9's complement can be obtained by subtracting each digit with 9

- In binary

$$\text{The 1's complement of 1011000 is 0100111.} \tag{7}$$
$$\text{The 1's complement of 0101101 is 1010010.} \tag{8}$$

  From the above example it is clear that the 1's complement of a binary number is formed by changing 1's to 0's and 0's to 1's.

- Generally, in any base-r system (r-1)'s complement is obtained by subtracting each digit in (r-1)

5.2 Radix Complement

- The r's complement of an n-digit number N in base r is defined as $r^n - N$ for N $\neq$ 0 and as 0 for N= 0.

- Notice, r's complement = (r-1)'s complement + 1. i,e

$$r^n - N = (r^n - 1) - N + 1 \tag{9}$$

- For example, in decimal system

$$\text{the 10's complement of 012398 is 987602} \tag{10}$$
$$\text{the 10's complement of 246700 is 753300} \tag{11}$$

- In binary

$$\text{the 2's complement of 1101100 is 0010100} \tag{12}$$
$$\text{the 2's complement of 0110111 is 1001001} \tag{13}$$

- The original number N contains a radix point, the point should be removed temporarily in order to form the r's or (r- 1)>s complement. The radix point is then restored to the complemented number in the same relative position.

5.3 Subtraction with complement

- When we subtract borrow carry when the minuend is smaller than the subtrahend. This works when using pen and paper but very inefficient than using complements.

- The subtraction of two n-digit unsigned numbers M- N in base r can be done as follows:

1. Add the minuend M to the r's complement of the subtrahend N. Mathematically,$M + (r^n - N) = M - N + r^n$
2. If $M \geq N$, the sum the sum will produce an end carry r^n , which can be discarded; what is left is the result M- N.
3. if $M < N$, the sum does not produce an end carry and is equal to r^n - (N- M), which is the r's complement of (N- M). To obtain the answer in a familiar form, take the r's complement of the sum and place a negative sign in front.

6 Signed Binary Numbers

1. Representation of Signed and Unsigned Binary Numbers

- **Unsigned Numbers:** Represent only positive integers (including zero). All bits are used to represent the magnitude.
- **Signed Numbers:** Represent both positive and negative integers.
  - The **leftmost bit (MSB)** represents the sign:
    - 0 indicates a positive number.
    - 1 indicates a negative number.
- **Example:**
  - 01001 (Unsigned) = 9 (Signed) = +9
  - 11001 (Unsigned) = 25 (Signed) = −9

2. Signed Number Representations

(a) Signed-Magnitude Representation

- Leftmost bit is the **sign bit**.
- Remaining bits represent the **magnitude**.
- Example (8 bits):

$$+9 = 00001001$$
$$-9 = 10001001$$

(b) 1's Complement Representation

- Negative numbers are represented by **flipping all bits** of the positive number.
- Example:

$$+9 = 00001001$$
$$-9 = 11110110$$

(c) 2's Complement Representation

- Obtain by adding 1 to the 1's complement.
- Simplifies arithmetic operations.
- Example:

$$+9 = 00001001$$
$$-9 = 11110111$$

3. Arithmetic Operations

(a) Addition in 2's Complement

- Add numbers including the sign bit.
- Discard carry out of the MSB.
- Example:

$$(+6) + (-13):$$
$$00000110 + 11110011 = 11111001 \quad (Result = -7)$$

(b) Subtraction in 2's Complement

- Take 2's complement of the subtrahend and add it to the minuend.
- Discard carry out of the MSB.
- Example:

$$(-6) - (-13):$$
$$11111010 + 00001101 = 00000111 \quad (Result = +7)$$

4. Overflow Conditions

- Occurs when the result exceeds the range that can be represented with the given number of bits.
- Detection Rule:
  - Adding two positive numbers gives a negative result.
  - Adding two negative numbers gives a positive result.

7 Binary Codes

Digital systems operate using binary signals (0 and 1) and circuit elements that have two stable states. Binary numbers and other discrete information are represented using binary codes, which are patterns of 0s and 1s. These codes do not change the meaning of the information but provide a way for digital circuits to process it efficiently.

An **n-bit binary code** consists of n bits, allowing for 2^n distinct combinations, with each combination representing a unique element. For example, a two-bit code can represent four elements:

$$\{00, 01, 10, 11\}$$

while a three-bit code can represent eight elements. To avoid ambiguity, each element must have a unique binary combination.

While the minimum number of bits required to represent 2^n elements is n, there is no maximum limit. For instance, decimal digits (0-9) can be represented using a 10-bit code, where each digit is assigned a unique combination with a single 1 among nine 0s. For example, the digit 6 can be represented as:

$$0001000000$$

This illustrates how binary coding is essential for digital systems to function effectively.

7.1 Binary Coded Decimal Code

7.1.1 Binary vs. Decimal in Computers

- Computers use the **binary number system** because it is compatible with electronic technology.
- Humans are more familiar with the **decimal system** (base 10).
- Conversion between **decimal and binary** is often required for calculations.

7.1.2 Storing Decimal Numbers in Binary Form

- Since computers accept only binary values, decimal numbers must be represented using **binary-coded forms**.
- **Binary-Coded Decimal (BCD)** is one method of storing decimal numbers in binary.

7.1.3 Structure of BCD

- Each decimal digit (0–9) is represented using **4 bits**.
- A decimal number with k digits requires **4k bits** in BCD.
- Example: Decimal 396 in BCD:

$$396_{10} = 0011\ 1001\ 0110_{BCD}$$

7.1.4 Comparison: BCD vs. Binary Representation

- BCD is different from standard binary representation.
- Example:

$$185_{10} = 0001\ 1000\ 0101_{BCD} \quad (12 \text{ bits})$$
$$185_{10} = 10111001_2 \quad (8 \text{ bits})$$

- BCD requires **more bits** than standard binary encoding.

7.1.5 Unused Bit Combinations in BCD

- A **4-bit binary** system provides **16 combinations** (0000 to 1111).
- Only **10 combinations** (0000 to 1001) are used for decimal digits.
- Combinations **1010 to 1111 are not used** in BCD.

7.1.6 Advantages & Disadvantages of BCD

- Advantage:
  - Easier for humans since input/output data remains in **decimal format**.
- Disadvantage:
  - BCD requires **more storage space** compared to binary representation.

7.1.7 BCD vs. Decimal Representation

- Decimal numbers use symbols **0–9**.
- BCD represents each decimal digit using a **4-bit binary code**.
- Example:

$$10_{10} = 0001\ 0000_{BCD} \quad \text{(8 bits)}$$
$$10_{10} = 1010_2 \quad \text{(4 bits)}$$

7.1.8 BCD Addition

- When adding two BCD digits, the sum can range from 0 to 19 (including a carry from a previous operation).
- If the binary sum is greater than or equal to 1010, the result is invalid in BCD and requires correction.
- Adding 6 (0110) to the binary sum corrects the digit and adjusts the carry.
- Example calculations:

$$\begin{aligned}
&4 + 5 = 9 \quad (0100 + 0101 = 1001) \\
&4 + 8 = 12 \quad (0100 + 1000 = 1100) \quad \text{(Invalid BCD, add 0110)} \\
&\qquad = 0010 \quad \text{(Correct BCD sum) with a carry} \\
&8 + 9 = 17 \quad (1000 + 1001 = 10001) \\
&\qquad\quad \text{(Requires correction, add 0110)} \\
&\qquad = 0111 \quad \text{(Correct BCD sum) with a carry}
\end{aligned}$$

- The same method applies to multi-digit BCD addition, ensuring correct decimal results.
- Example: Adding 184 + 576 = 760 in BCD follows the same process.

7.1.9 Decimal Arithmetic in BCD

The representation of signed decimal numbers in BCD (Binary-Coded Decimal) follows a similar approach to signed binary numbers. Two main systems are used:

- Signed-magnitude system (rarely used in computers).

- Signed-complement system, which includes 9's complement and 10's complement (the latter being more common).

To compute the 10's complement of a BCD number:

1. Compute the 9's complement by subtracting each digit from 9.
2. Add 1 to the least significant digit.

For addition in the 10's complement system:

- Sum all digits, including the sign digit.
- Discard the end carry.

For subtraction, take the 10's complement of the subtrahend and add it to the minuend, similar to binary arithmetic. Many computers incorporate hardware to directly perform BCD arithmetic, allowing programmed instructions to handle decimal calculations without conversion to binary.

7.2 Weighted Codes

In weighted codes, each digit position is assigned a specific weight.

Code Type	**Weights**	**Example (Decimal 7)**
8421 Code (BCD)	8, 4, 2, 1	0111
2421 Code	2, 4, 2, 1	1100
5211 Code	5, 2, 1, 1	0111
8, 4, -2, -1 Code	8, 4, -2, -1	0110

Table 3: Comparison of Weighted Codes

7.2.1 Excess-3 (XS-3) Code

A non-weighted binary code where each decimal digit is represented by adding 3 before converting to binary.

Decimal	Binary	Excess-3 Code
0	0000	0011
1	0001	0100
2	0010	0101
3	0011	0110
4	0100	0111
5	0101	1000
6	0110	1001
7	0111	1010
8	1000	1011
9	1001	1100

Table 4: Excess-3 Code Table

7.2.2 Gray Code

A special binary code where only one bit changes between successive values.

Decimal	Binary	Gray Code
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100

Table 5: Gray Code vs Binary Code

7.3 ASCII code

7.3.1 Introduction

Many digital computer applications require handling not only numerical data but also alphanumeric characters and symbols. To represent these characters, a binary encoding system is needed. One such system is the American Standard Code for Information Interchange (ASCII), which is widely used for encoding text.

7.3.2 ASCII Encoding

ASCII is a 7-bit character encoding system that can represent 128 characters. The encoding includes:

- 26 uppercase letters (A-Z)
- 26 lowercase letters (a-z)
- 10 decimal digits (0-9)
- 32 special printable characters (e.g., %, *, $)
- 34 control characters for formatting and communication

Each character in ASCII is represented by a unique 7-bit binary number. For example, the letter 'A' is represented as `1000001`.

7.3.3 Control Characters

The ASCII table includes 34 non-printable control characters, which are categorized as:

- **Format Effectors**: Control text layout (e.g., Backspace (BS), Carriage Return (CR), Horizontal Tab (HT)).
- **Information Separators**: Divide text into sections (e.g., Record Separator (RS), File Separator (FS)).
- **Communication-Control Characters**: Used in text transmission (e.g., Start of Text (STX), End of Text (ETX)).

7.3.4 ASCII and Byte Representation

Although ASCII is a 7-bit code, most computers use 8-bit bytes. The extra bit is often used for extended characters, such as Greek letters or italic fonts. Some systems set the most significant bit (MSB) to 0 for standard ASCII characters and to 1 for extended character sets.

7.3.5 Conclusion

ASCII provides a standardized method for encoding text in computers and digital communication. Its widespread adoption has made it a fundamental component of text processing and data exchange.

Table 1.7
American Standard Code for Information Interchange (ASCII)

	$b_7b_6b_5$							
$b_4b_3b_2b_1$	000	001	010	011	100	101	110	111
0000	NUL	DLE	SP	0	@	P	‘	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	“	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	‘	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L		l	|
1101	CR	GS	–	=	M	]	m	}
1110	SO	RS	.	>	N	∧	n	~
1111	SI	US	/	?	O	–	o	DEL

Control Characters

NUL	Null	DLE	Data-link escape
SOH	Start of heading	DC1	Device control 1
STX	Start of text	DC2	Device control 2
ETX	End of text	DC3	Device control 3
EOT	End of transmission	DC4	Device control 4
ENQ	Enquiry	NAK	Negative acknowledge
ACK	Acknowledge	SYN	Synchronous idle
BEL	Bell	ETB	End-of-transmission block
BS	Backspace	CAN	Cancel
HT	Horizontal tab	EM	End of medium
LF	Line feed	SUB	Substitute
VT	Vertical tab	ESC	Escape
FF	Form feed	FS	File separator
CR	Carriage return	GS	Group separator
SO	Shift out	RS	Record separator
SI	Shift in	US	Unit separator
SP	Space	DEL	Delete

Figure 1: ASCII CODE

7.4 Error Handling Code

To detect errors in data communication, an additional parity bit is added to ASCII characters, ensuring an even or odd number of 1's. Even parity is commonly used.

At the sender's end, parity bits are generated, and at the receiver's end, they are checked. If a parity error is detected, the receiver sends an NAK (Negative Acknowledge) signal:

$$\text{NAK} = 10010101$$

This prompts retransmission. If no error is found, an ACK (Acknowledge) signal is sent:

$$\text{ACK} = 00000110$$

If repeated errors occur, manual intervention is required.

8 Binary storage and registers

8.1 Introduction

A binary cell is a device that stores one bit(0 or 1) of information using two stable states, with excitation signals setting the cell to one of these states. The output is a physical quantity (e.g., voltage or charge) that distinguishes between the two states. The information stored in cell is 1 and 0 when cell is in one stable state and other stable state respectively.

8.2 Registers

Registers are contiguous group of binary cells. Register with n cells can store information about n bits. The content of a register is a function of the interpretation given to the information stored in it i.e. the meaning of the data inside a register is determined by how it is used in the system, not just by the raw binary value it contains.The content of a register can be interpreted differently depending on the data type it holds.For example, in BCD (Binary-Coded Decimal), the bit combination 1100 is not valid because it does not represent any decimal digit. This illustrates that while a register stores binary data, its meaning depends on the context or application. The same bit pattern can represent different types of information (e.g., numbers, addresses, instructions) based on how it is used.

8.3 Register Transfer

it is a basic operation that consists of a transfer of binary information from one set of registers into another set of registers ,by direct way, from one register to another, or may pass through data-processing circuits to perform an operation.

The image given below shows how a computer stores typed letters in memory. When you press a key, it turns into an 8-bit code and goes into a small storage (input register). Then, it moves to another storage (processor register) and finally gets saved in memory. This process happens for each letter, one by one. Registers help store and move data inside a computer.

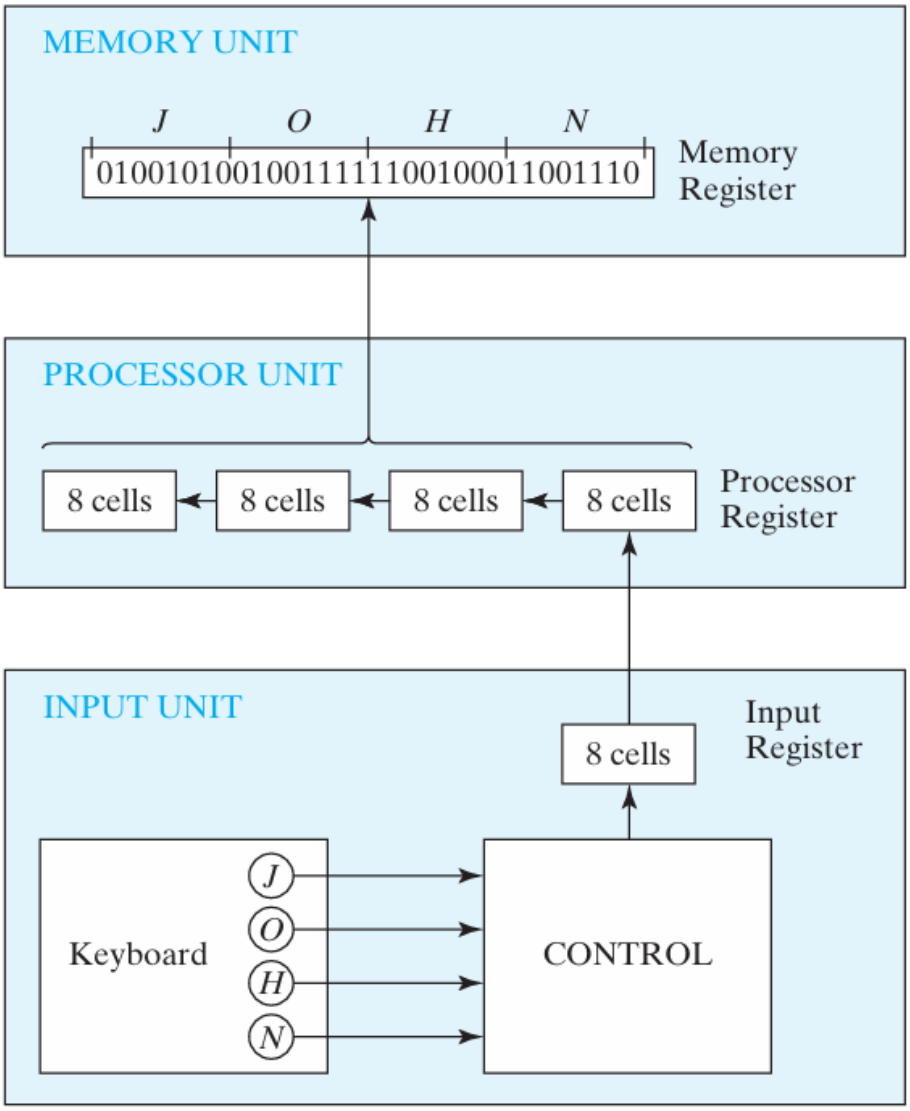

Figure 2: transfer of information among registers

To process discrete quantities of information in binary form, a computer must be provided with devices that hold the data to be processed and with circuit elements that manipulate individual bits of information. **The device most commonly used for holding data is a register.** Binary variables are manipulated by means of digital logic circuits.

As in above figure, The memory unit has many registers for storing data, while the processor unit has three registers (R1, R2, and R3) and logic circuits for processing. Memory registers store data but cannot perform calculations. Data from memory is moved to processor registers (R1 and R2), where logic circuits add them and store the result in R3. The result can then be sent back to memory for later use. The registers of the system are the basic elements for storing and holding the binary information.

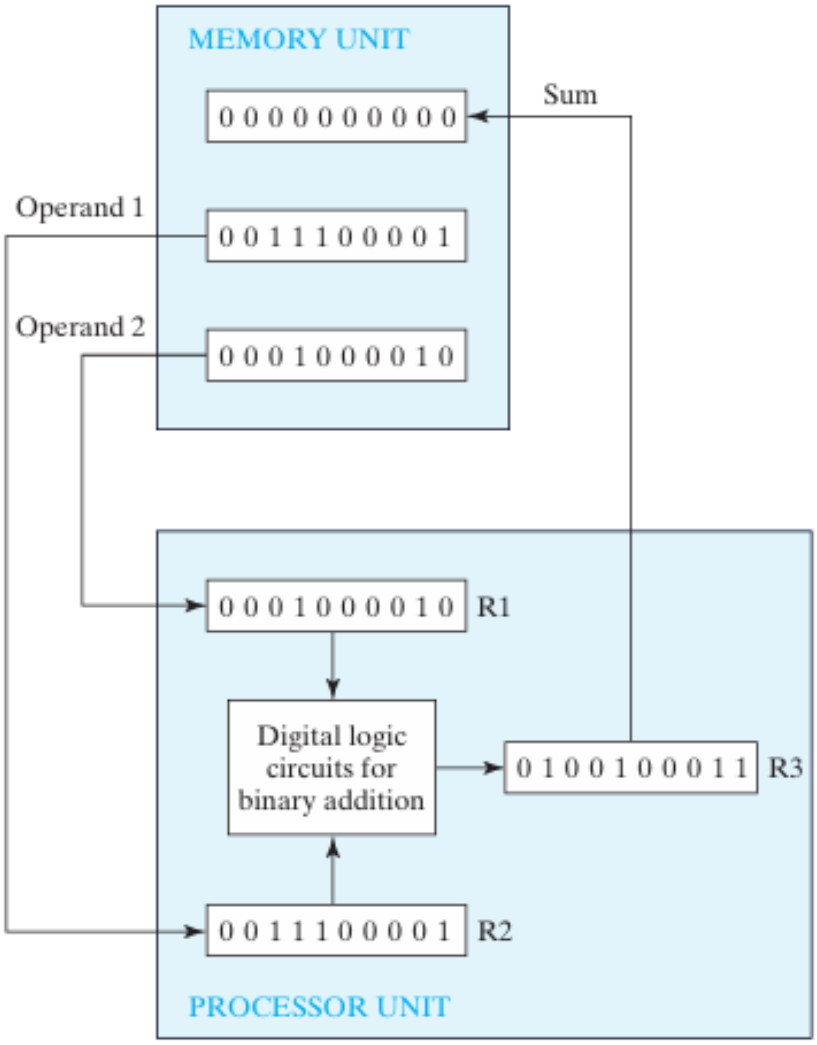

Figure 3: Example of registers in binary information processing

9 Binary Logic

Binary logic deals with variables that take on two discrete values(most commonly 0 and 1) and with operations that assume logical meaning.

The binary logic introduced in this is equivalent to an algebra called Boolean algebra.

Definition of Binary Logic
Binary logic consists of binary variables and a set of logical operations. There are three basic logical operations: AND, OR, and NOT. Each operation produces a binary result.

For each combination of the values of x and y, there is a value of z specified by the definition of the logical operation. Definitions of logical operations may be listed in a compact form called truth tables.

AND:

The logical operation AND is interpreted to mean that z = 1 if and only if x = 1 and y= 1; otherwise z = 0 (The result of the operation $x \cdot y$ is z).

- Truth table:

x	y	x · y
0	0	0
0	1	0
1	0	0
1	1	1

OR:

The output z = 1 if x = 1 or if y= 1 or if both x = 1 and y= 1. If both x = 0 and y= 0, then z = 0.This operation is represented by a plus sign.

- Truth table:

x	y	x + y
0	0	0
0	1	1
1	0	1
1	1	1

NOT:

The NOT operation is also referred to as the complement operation, since it changes a 1 to 0 and a 0 to 1, that is, the result of complementing 1 is 0, and vice versa. This operation is represented by a prime (sometimes by an overbar).

- Truth table:

x	x'
0	1
1	0

*binary logic should not be confused with binary arithmetic.

Logic Gates

Logic Gates are electronic circuits that process input signals (voltages or currents) to produce binary outputs (0 or 1).

- **Digital System:** Defines:
  - Logic 0: 0 V
  - Logic 1: High Voltage V(3V)

Logic gates interpret binary input signals and produce corresponding binary. The timing diagrams illustrate the idealized response of each gate to the four input signal combinations. The horizontal axis of the timing diagram represents the time, and the vertical axis shows the signal as it changes between the two possible voltage levels.

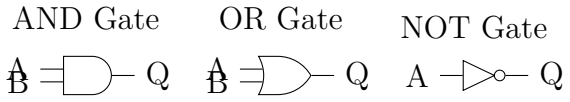

Figure 4: Basic logic gates with standard symbols and labels.

Gates with Multiple Inputs

- Three-input AND Gate: Outputs 1 if all inputs are 1.
- Four-input OR Gate: Outputs 1 if any input is 1; outputs 0 only if all inputs are 0.