# Logical Organization of Computer
# CHAPTER 1

## DIGITAL SYSTEMS AND INFORMATION

Developed By:
Dr. Vivek Vyas
Department of MCA
DDU

# 1. INFORMATION REPRESENTATION

- The information represents a broad range of phenomena from the physical and man-made world.
- The physical world is characterized by parameters such as weight, temperature, pressure, velocity, flow, and sound intensity and frequency.
- Most physical parameters are continuous, typically capable of taking on all possible values over a defined range.
- In contrast, in the man-made world, parameters can be discrete in nature, such as business records using words, quantities, and currencies.

# 1. INFORMATION REPRESENTATION

- In general, information systems must be able to represent both continuous and discrete information.
- We refer to such a continuous voltage as an analog signal and discrete voltage as digital signal.
- The signals in most present-day electronic digital systems use just two discrete values and are therefore said to be binary.
- The two discrete values used are often called 0 and 1, the digits for the binary number system.

# 1. INFORMATION REPRESENTATION

(a) Example voltage ranges

(b) Time-dependent voltage

(c) Binary model of time-dependent voltage

**FIGURE 1**
Examples of Voltage Ranges and Waveforms for Binary Signals

# 1. INFORMATION REPRESENTATION

- Since 0 and 1 are associated with the binary number system, they are the preferred names for the signal ranges. A binary digit is called a bit. Information is represented in digital computers by groups of bits.
- Why is binary used? In contrast to the situation in Diagram 1, consider a system with 10 values representing the decimal digits.
- In such a system, the voltages available—say, 0 to 1.0 volts—could be divided into 10 ranges, each of length 0.1 volt. A circuit would provide an output voltage within each of these 10 ranges. An input of a circuit would need to determine in which of the 10 ranges an applied voltage lies.

# ❑ The Digital Computer

❑ **FIGURE 2**
Block Diagram of a Digital Computer

## ❑ The Digital Computer

- The memory stores programs as well as input, output, and intermediate data.
- The datapath performs arithmetic and other data-processing operations as specified by the program.
- The control unit supervises the flow of information between the various units.
- A datapath, when combined with the control unit, forms a component referred to as a *central processing unit, or CPU.*

## ❑ The Digital Computer

- The program and data prepared by the user are transferred into memory by means of an input device such as a keyboard.
- An output device, such as an LCD (liquid crystal display), displays the results of the computations and presents them to the user.

# Logical Organization of Computer
# CHAPTER 1

## DIGITAL SYSTEMS AND INFORMATION

Developed By:
Dr. Vivek Vyas
Department of MCA
DDU

# 2. NUMBER SYSTEMS

- The decimal number system is employed in everyday arithmetic to represent numbers by strings of digits. Depending on its position in the string, each digit has an associated value of an integer raised to the power of 10.
- For example, the decimal number 724.5 is interpreted to represent 7 hundreds plus 2 tens plus 4 units plus 5 tenths.
- The hundreds, tens, units, and tenths are powers of 10 implied by the position of the digits. The value of the number is computed as follows:

$$724.5 = 7 \times 10^2 + 2 \times 10^1 + 4 \times 10^0 + 5 \times 10^{-1}$$

# 2. NUMBER SYSTEMS

- The convention is to write only the digits and infer the corresponding powers of 10 from their positions.
- In general, a decimal number with $n$ digits to the left of the decimal point and $m$ digits to the right of the decimal point is represented by a string of coefficients:

$$A_{n-1}A_{n-2}\cdots A_1A_0.A_{-1}A_{-2}\cdots A_{-m+1}A_{-m}$$

# 2. NUMBER SYSTEMS

- In general, the "." is called the *radix point*. *An – 1* is referred to as the *most significant digit* (*msd*) and *Am* as the *least significant digit* (*lsd*) of the number.
- The following illustrates a base 5 number with $n$ = 3 and $m$ = 1 and its conversion to decimal:

$$(312.4)_5 = 3 \times 5^2 + 1 \times 5^1 + 2 \times 5^0 + 4 \times 5^{-1}$$
$$= 75 + 5 + 2 + 0.8 = (82.8)_{10}$$

# ❑ Binary Numbers

**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 | 18 | 262,144 |
| 3 | 8 | 11 | 2,048 | 19 | 524,288 |
| 4 | 16 | 12 | 4,096 | 20 | 1,048,576 |
| 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 |

## ❑ Binary Numbers

- In digital systems, we refer to $2^{10}$ as K (kilo), $2^{20}$ as M (mega), $2^{30}$ as G (giga), and $2^{40}$ as T (tera). Thus,

$$4K = 2^2 \times 2^{10} = 2^{12} = 4096$$

$$16M = 2^4 \times 2^{20} = 2^{24} = 16{,}777{,}216$$

# ❑ Binary Numbers

**Numbers with Different Bases**

| Decimal (base 10) | Binary (base 2) | Octal (base 8) | Hexadecimal (base 16) |
|---|---|---|---|
| 00 | 0000 | 00 | 0 |
| 01 | 0001 | 01 | 1 |
| 02 | 0010 | 02 | 2 |
| 03 | 0011 | 03 | 3 |
| 04 | 0100 | 04 | 4 |
| 05 | 0101 | 05 | 5 |
| 06 | 0110 | 06 | 6 |
| 07 | 0111 | 07 | 7 |
| 08 | 1000 | 10 | 8 |
| 09 | 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 |

# Logical Organization of Computer
# CHAPTER 1

## DIGITAL SYSTEMS AND INFORMATION

Developed By:
Dr. Vivek Vyas
Department of MCA
DDU

# ❑ Binary Numbers

- **Binary to Decimal Conversion**

$$(11010)_2 = 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = (26)_{10}$$

$$(110101.11)_2 = 32 + 16 + 4 + 1 + 0.5 + 0.25 = (53.75)_{10}$$

# ❑ Binary Numbers

- **Binary to Octal Conversion**

$$(010\ 110\ 001\ 101\ 011.\ 111\ 100\ 000\ 110)_2 = (26153.7406)_8$$

# ❑ Binary Numbers

- **Binary to Hexadecimal Conversion**

$$(0010\ 1100\ 0110\ 1011.\ 1111\ 0000\ 0110)_2 = (2C6B.F06)_{16}$$

# ❑ Decimal Numbers

- **Decimal to Binary conversion**
- $(625)_{10} = ?$

$$625 - 512 = 113 = N_1 \qquad 512 = 2^9$$
$$113 - 64 = 49 = N_2 \qquad 64 = 2^6$$
$$49 - 32 = 17 = N_3 \qquad 32 = 2^5$$
$$17 - 16 = 1 = N_4 \qquad 16 = 2^4$$
$$1 - 1 = 0 = N_5 \qquad 1 = 2^0$$
$$(625)_{10} = 2^9 + 2^6 + 2^5 + 2^4 + 2^0 = (1001110001)_2$$

# Logical Organization of Computer
# CHAPTER 1

## DIGITAL SYSTEMS AND INFORMATION

Developed By:
Dr. Vivek Vyas
Department of MCA
DDU

# ❑ Binary Numbers

- **Conversion of Decimal Integers to Binary**

Convert decimal 41 to binary:

| | | |
|---|---|---|
| $41/2 = 20 + 1/2$ | Remainder $= 1$ | Least significant digit |
| $20/2 = 10$ | $= 0$ | |
| $10/2 = 5$ | $= 0$ | |
| $5/2 = 2 + 1/2$ | $= 1$ | |
| $2/2 = 1$ | $= 0$ | |
| $1/2 = 0 + 1/2$ | $= 1$ | Most significant digit |

$(41)_{10} = (101001)_2$

## ❑ Binary Numbers

- **Conversion of Decimal Fractions to Binary**

| | | |
|---|---|---|
| $0.6875 \times 2 = 1.3750$ | Integer $= 1$ | Most significant digit |
| $0.3750 \times 2 = 0.7500$ | $= 0$ | |
| $0.7500 \times 2 = 1.5000$ | $= 1$ | |
| $0.5000 \times 2 = 1.0000$ | $= 1$ | Least significant digit |

$$(0.6875)_{10} = (0.1011)_2$$

## ❑ Conversion from Decimal to Other Bases

- **Conversion of Decimal Integers to Octal**

| | | |
|---|---|---|
| $153/8 = 19 + 1/8$ | Remainder $= 1$ | Least significant digit |
| $19/8 = 2 + 3/8$ | $= 3$ | |
| $2/8 = 0 + 2/8$ | $= 2$ | Most significant digit |

$(153)_{10} = (231)_8$

## ❑ Conversion from Decimal to Other Bases

- **Conversion of Decimal Fractions to Octal**

| | | |
|---|---|---|
| $0.513 \times 8 = 4.104$ | Integer $= 4$ | Most significant digit |
| $0.104 \times 8 = 0.832$ | $= 0$ | |
| $0.832 \times 8 = 6.656$ | $= 6$ | |
| $0.656 \times 8 = 5.248$ | $= 5$ | Least significant digit |

$$(0.513)_{10} = (0.407)_8$$

# ❑ Octal and Hexadecimal Numbers

**Numbers with Different Bases**

| Decimal (base 10) | Binary (base 2) | Octal (base 8) | Hexadecimal (base 16) |
|---|---|---|---|
| 00 | 0000 | 00 | 0 |
| 01 | 0001 | 01 | 1 |
| 02 | 0010 | 02 | 2 |
| 03 | 0011 | 03 | 3 |
| 04 | 0100 | 04 | 4 |
| 05 | 0101 | 05 | 5 |
| 06 | 0110 | 06 | 6 |
| 07 | 0111 | 07 | 7 |
| 08 | 1000 | 10 | 8 |
| 09 | 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 |

# ❑ Octal and Hexadecimal Numbers

- **Octal to Decimal Conversion**

$$(127.4)_8 = 1 \times 8^2 + 2 \times 8^1 + 7 \times 8^0 + 4 \times 8^{-1} = (87.5)_{10}$$

# Logical Organization of Computer
# CHAPTER 1

## DIGITAL SYSTEMS AND INFORMATION

Developed By:
Dr. Vivek Vyas
Department of MCA
DDU

# ❑ Octal and Hexadecimal Numbers

- **Octal to Binary Conversion**

$$(673.12)_8 = 110\ \ 111\ \ 011.\ \ 001\ \ 010 = (110111011.00101)_2$$

# ❑ Octal and Hexadecimal Numbers

- **Hexadecimal to Decimal Conversion**

$$(B65F)_{16} = 11 \times 16^3 + 6 \times 16^2 + 5 \times 16^1 + 15 \times 16^0 = (46687)_{10}$$

# ❑ Octal and Hexadecimal Numbers

- **Hexadecimal to Binary Conversion**

$$(3A6.C)_{16} = 0011\ 1010\ 0110.\ 1100 = (1110100110.11)_2$$

# 3. ARITHMETIC OPERATIONS

- Binary Addition

| | |
|---|---|
| 00000 | 101100 |
| 01100 | 10110 |
| +10001 | +10111 |
| 11101 | 101101 |

# 3. ARITHMETIC OPERATIONS

- Binary Subtraction

$$\begin{array}{r} 00000 \\ 10110 \\ -10010 \\ \hline 00100 \end{array} \qquad \begin{array}{r} 00110 \\ 10110 \\ -10011 \\ \hline 00011 \end{array}$$

# Logical Organization of Computer
# CHAPTER 1

## DIGITAL SYSTEMS AND INFORMATION

Developed By:
Dr. Vivek Vyas
Department of MCA
DDU

# 3. ARITHMETIC OPERATIONS

- Binary Subtraction

$$\begin{array}{r} 00000 \\ 10110 \\ -10010 \\ \hline 00100 \end{array} \qquad \begin{array}{r} 00110 \\ 10110 \\ -10011 \\ \hline 00011 \end{array}$$

# 3. ARITHMETIC OPERATIONS

- Binary Multiplication

```
    1011
  × 101
 ------
    1011
   0000
  1011
 ------
  110111
```

# ❑ 1's & 2's complement

## ❑ r's & (r-1)'s complement

- Example: Find 3's complement and 2's complement of $121_3$?
- 2's complement of $121_3$
- = 222 – 121
- = $101_3$
- And 3's complement of $121_3$
- = 2's complement + 1
- = $101_3$+ 1
- = $102_3$

## ❑ r's & (r-1)'s complement

- Example: Find the 3's complement and 4's complement of $130_4$?
- 3's complement of $130_4$
- = 333 – 130
- = $203_4$
- 4's complement of $130_4$
- = 3's complement of $130_4$ + 1
- = $203_4$+ 1
- = $210_4$

## ❑ r's & (r-1)'s complement

- Example: Find the 4's complement and 5's complement of $224_5$?
- 4's complement of $224_5$
- = 444 – 224
- = $220_5$
- 5's complement of $224_5$
- = 4's complement of $224_5$ + 1
- = $220_5$+ 1
- = $221_5$

# Logical Organization of Computer
# CHAPTER 1

## DIGITAL SYSTEMS AND INFORMATION

Developed By:
Dr. Vivek Vyas
Department of MCA
DDU

## ❑ r's & (r-1)'s complement

- Example: Find 10's complement and 11's complement of $1A1_{11}$?
- 10's complement of $1A1_{11}$
- = AAA – 1A1
- = $909_{11}$
- 11's complement of $1A1_{11}$
- = 10's complement of $1A1_{11}$ + 1
- = $909_{11}$+ 1
- = $90A_{11}$

## ❑ r's & (r-1)'s complement

- Example: Find 11's complement and 12's complement of $AB0_{12}$?
- 11's complement of $AB0_{12}$
- = BBB – AB0
- = $10B_{12}$
- 12's complement of $AB0_{12}$
- = 11's complement of $AB0_{12}$ + 1
- = $10B_{12}$+ 1
- = $110_{12}$

# 4. DECIMAL CODES

**TABLE 4**
**Binary-Coded Decimal (BCD)**

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

# ❑ BCD Addition

| | | | | |
|---|---|---|---|---|
| 110 | BCD carry | 1 | 1 | |
| 448 | | 0100 | 0100 | 1000 |
| +489 | | +0100 | +1000 | +1001 |
| 937 | Binary sum | 1001 | 1101 | 1 0001 |
| | Add 6 | | +0110 | +0110 |
| | BCD sum | | 1 0011 | 1 0111 |
| | BCD result | 1001 | 0011 | 0111 |

# Logical Organization of Computer
# CHAPTER 1

## DIGITAL SYSTEMS AND INFORMATION

Developed By:
Dr. Vivek Vyas
Department of MCA
DDU

# 5. ALPHANUMERIC CODES

- Many applications of digital computers require the handling of data consisting not only of numbers, but also of letters.
- For instance, an insurance company with thousands of policyholders uses a computer to process its files.
- To represent the names and other pertinent information, it is necessary to formulate a binary code for the letters of the alphabet.
- In addition, the same binary code must represent numerals and special characters such as $.

## ❑ ASCII Character Code

- The standard binary code for the alphanumeric characters is called ASCII (American Standard Code for Information Interchange). It uses seven bits to code 128 characters.
- The ASCII code contains 94 characters that can be printed and 34 nonprinting characters used for various control functions.
- The printing characters consist of the 26 uppercase letters, the 26 lowercase letters, the 10 numerals, and 32 special printable characters such as %, @, and $.

# ❑ ASCII Character Code

**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 | NULL | 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 |

## ❑ Parity Bit

| | With Even Parity | With Odd Parity |
|---|---|---|
| 1000001 | 01000001 | 11000001 |
| 1010100 | 11010100 | 01010100 |

# ❑ Hamming Code

- 1001000
- $M+r+1<2^r$

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| R1 | R2 | 1 | R3 | 0 | 0 | 1 | R4 | 0 | 0 | 0 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |

# Logical Organization of Computer
# CHAPTER 1

## DIGITAL SYSTEMS AND INFORMATION

Developed By:
Dr. Vivek Vyas
Department of MCA
DDU

# ❑ Hamming Code

- 1001000
- $M+r+1<2^r$

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| R1 | R2 | 1 | R3 | 0 | 0 | 1 | R4 | 0 | 0 | 0 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |

## ❑ Hamming Code

- R1 = BIT NO. 1+3+5+7+9+11
- R2 = BIT NO. 2+3+6+7+10+11
- R3 = BIT NO. 4+5+6+7
- R4 = BIT NO. 8+9+10+11

## ❑ EBCDIC

- *Extended Binary Coded Decimal Interchange Code* (EBCDIC) is an 8-bit character encoding used on IBM mainframe and minicomputer operating systems.
- All IBM mainframe peripherals and operating systems still support EBCDIC, although the operating systems also provide ASCII and Unicode modes to allow translation between different encodings.

# ❑ EBCDIC

| Character | EBDIC Bit Configuration | Character | EBDIC Bit Configuration |
|---|---|---|---|
| A | 1100 0001 | S | 1110 0010 |
| B | 1100 0010 | T | 1110 0011 |
| C | 1100 0011 | U | 1110 0100 |
| D | 1100 0100 | V | 1110 0101 |
| E | 1100 0101 | W | 1110 0110 |
| F | 1100 0110 | X | 1110 0111 |
| G | 1100 0111 | Y | 1110 1000 |
| H | 1100 1000 | Z | 1110 1001 |
| I | 1100 1001 | 0 | 1111 0000 |
| J | 1101 0001 | 1 | 1111 0001 |
| K | 1101 0010 | 2 | 1111 0010 |
| L | 1101 0011 | 3 | 1111 0011 |
| M | 1101 0100 | 4 | 1111 0100 |
| N | 1101 0101 | 5 | 1111 0101 |
| O | 1101 0110 | 6 | 1111 0110 |
| P | 1101 0111 | 7 | 1111 0111 |
| Q | 1101 1000 | 8 | 1111 1000 |
| R | 1101 1001 | 9 | 1111 1001 |

## ❑ UNICODE

- The Unicode character set has the capacity to support over one million characters, and is being developed with an aim to have a single character set that supports all characters from all scripts, as well as many symbols, that are in common use around the world today or in the past.
- Currently, the Standard supports over 96,000 characters representing a large number of scripts. UTF-8, UTF-16 and UTF-32 are part of Unicode.

# Logical Organization of Computer
# CHAPTER 2

## COMBINATIONAL LOGIC CIRCUITS

Developed By:
Dr. Vivek Vyas
Department of MCA
DDU

# 1. BINARY LOGIC AND GATES

- Digital circuits are hardware components that manipulate binary information. The circuits are implemented using transistors and interconnections in complex semiconductor devices called *integrated circuits*. Each basic circuit is referred to as a *logic gate*.
- Each gate performs a specific logical operation. The outputs of gates are applied to the inputs of other gates to form a digital circuit.
- In order to describe the operational properties of digital circuits, we need to introduce a mathematical notation that specifies the operation of each gate and that can be used to analyze and design circuits known as *Boolean algebras*.

## ❑ Binary Logic

- Binary logic deals with binary variables, which take on two discrete values, and with the operations of mathematical logic applied to these variables.
- Associated with the binary variables are three basic logical operations called AND, OR, and NOT.
- **1. AND.** This operation is represented by a dot or by the absence of an operator. For example, $Z = X \cdot Y$ or $Z = XY$ is read "$Z$ is equal to $X$ AND $Y$." The logical operation AND is interpreted to mean that $Z = 1$ if and only if $X = 1$ and $Y = 1$; otherwise $Z = 0$.

$$0 \cdot 0 = 0$$
$$0 \cdot 1 = 0$$
$$1 \cdot 0 = 0$$
$$1 \cdot 1 = 1$$

## ❑ Binary Logic

2. OR. This operation is represented by a plus symbol. For example, $Z = X + Y$ is read "Z is equal to $X$ OR $Y$," meaning that $Z = 1$ if $X = 1$ or if $Y = 1$, or if both $X = 1$ and $Y = 1$. $Z = 0$ if and only if $X = 0$ and $Y = 0$.

$$0 + 0 = 0$$
$$0 + 1 = 1$$
$$1 + 0 = 1$$
$$1 + 1 = 1$$

3. NOT. This operation is represented by a bar over the variable. For example, $Z = \overline{X}$ is read "Z is equal to NOT $X$," meaning that $Z$ is what $X$ is not. In other words, if $X = 1$, then $Z = 0$; but if $X = 0$, then $Z = 1$. The NOT operation is also referred to as the *complement* operation, since it changes a 1 to 0 and a 0 to 1.

# Logical Organization of Computer
# CHAPTER 2

## COMBINATIONAL LOGIC CIRCUITS

Developed By:
Dr. Vivek Vyas
Department of MCA
DDU

## ❑ Binary Logic

- A *truth table* for an operation is a table of combinations of the binary variables showing the relationship between the values that the variables take on and the values of the result of the operation.
- The truth tables for the operations AND, OR, and NOT are shown below.

# ❑ Binary Logic

❑ **TABLE 1**
**Truth Tables for the Three Basic Logical Operations**

AND

| X | Y | $Z = X \cdot Y$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

OR

| X | Y | $Z = X + Y$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

NOT

| X | $Z = \overline{X}$ |
|---|---|
| 0 | 1 |
| 1 | 0 |

## ❑ Logic Gates

- Logic gates are electronic circuits that operate on one or more input signals to produce an output signal.
- The input terminals of logic gates accept binary signals within the allowable range and respond at the output terminals with binary signals that fall within a specified range.
- The intermediate regions between the allowed ranges in the figure are crossed only during changes from 1 to 0 or from 0 to 1.
- These changes are called *transitions*, and the intermediate regions are called the *transition regions*.

## ❑ Logic Gates

- The graphics symbols used to designate the three types of gates—AND, OR, and NOT—are shown below.

# ❑ Logic Gates

(b) Timing diagram

# 2. BOOLEAN ALGEBRA

- A *Boolean expression* is an algebraic expression formed by using binary variables, the constants 0 and 1, the logic operation symbols, and parentheses.
- *Boolean function* can be described by a Boolean equation consisting of a binary variable identifying the function followed by an equals sign and a Boolean expression.

# ❑ Basic Identities of Boolean Algebra

<u>Annulment Law</u> – A term AND´ed with a "0" equals 0 or OR´ed with a "1" will equal 1

| | |
|---|---|
| $A . 0 = 0$ | A variable AND'ed with 0 is always equal to 0 |
| $A + 1 = 1$ | A variable OR'ed with 1 is always equal to 1 |

<u>Identity Law</u> – A term OR´ed with a "0" or AND´ed with a "1" will always equal that term

| | |
|---|---|
| $A + 0 = A$ | A variable OR'ed with 0 is always equal to the variable |
| $A . 1 = A$ | A variable AND'ed with 1 is always equal to the variable |

# Logical Organization of Computer
# CHAPTER 2

## COMBINATIONAL LOGIC CIRCUITS

Developed By:
Dr. Vivek Vyas
Department of MCA
DDU

# ❑ Basic Identities of Boolean Algebra

<u>Idempotent Law</u> – An input that is AND´ed or OR´ed with itself is equal to that input

| | |
|---|---|
| $A + A = A$ | A variable OR'ed with itself is always equal to the variable |
| $A . A = A$ | A variable AND'ed with itself is always equal to the variable |

<u>Complement Law</u> – A term AND´ed with its complement equals "0" and a term OR´ed with its complement equals "1"

| | |
|---|---|
| $A . \overline{A} = 0$ | A variable AND'ed with its complement is always equal to 0 |
| $A + \overline{A} = 1$ | A variable OR'ed with its complement is always equal to 1 |

# ❑ Basic Identities of Boolean Algebra

<u>Commutative Law</u> – The order of application of two separate terms is not important

$A . B = B . A$ The order in which two variables are AND'ed makes no difference

$A + B = B + A$ The order in which two variables are OR'ed makes no difference

<u>Double Negation Law</u> – A term that is inverted twice is equal to the original term

$\overline{\overline{A}} = A$ A double complement of a variable is always equal to the variable

# ❑ Basic Identities of Boolean Algebra

Distributive Law – This law permits the multiplying or factoring out of an expression.

$A(B + C) = A.B + A.C$ (OR Distributive Law)

$A + (B.C) = (A + B).(A + C)$ (AND Distributive Law)

Absorptive Law – This law enables a reduction in a complicated expression to a simpler one by absorbing like terms.

$A + (A.B) = A$ (OR Absorption Law)

$A(A + B) = A$ (AND Absorption Law)

# ❑ Basic Identities of Boolean Algebra

<u>Associative Law</u> – This law allows the removal of brackets from an expression and regrouping of the variables.

| |
|---|
| $A + (B + C) = (A + B) + C = A + B + C$ (OR Associate Law) |
| $A(B.C) = (A.B)C = A . B . C$ (AND Associate Law) |

<u>de Morgan´s Theorem</u> – There are two "de Morgan´s" rules or theorems,

(1) Two separate terms NOR´ed together is the same as the two terms inverted (Complement) and AND´ed for example: $\overline{A+B} = \overline{A} . \overline{B}$

(2) Two separate terms NAND´ed together is the same as the two terms inverted (Complement) and OR´ed for example: $\overline{A.B} = \overline{A} + \overline{B}$

# ❑ Basic Identities of Boolean Algebra

**Truth Tables to Verify DeMorgan's Theorem**

| (a) X | Y | X + Y | $\overline{X+Y}$ |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 |

| (b) X | Y | $\overline{X}$ | $\overline{Y}$ | $\overline{X} \cdot \overline{Y}$ |
|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 |

## ❑ Algebraic Manipulation

- Boolean algebra is a useful tool for simplifying digital circuits. Consider, for example, the Boolean function represented by

$$F = \overline{X}YZ + \overline{X}Y\overline{Z} + XZ$$

$$\begin{aligned} F &= \overline{X}YZ + \overline{X}Y\overline{Z} + XZ \\ &= \overline{X}Y(Z + \overline{Z}) + XZ && \text{Distributive Law} \\ &= \overline{X}Y \cdot 1 + XZ && \text{Complement Law} \\ &= \overline{X}Y + XZ && \text{Identity Law} \end{aligned}$$

# ❑ Algebraic Manipulation

(a) $F = \overline{X}YZ + \overline{X}Y\overline{Z} + XZ$

# ❑ Algebraic Manipulation

(b) $F = \overline{X}Y + XZ$

# Logical Organization of Computer
# CHAPTER 2

## COMBINATIONAL LOGIC CIRCUITS

Developed By:
Dr. Vivek Vyas
Department of MCA
DDU

# ❑ Algebraic Manipulation

**Truth Table for Boolean Function**

| X | Y | Z | (a) F | (b) F |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 |

# 3. STANDARD FORMS

- A Boolean function expressed algebraically can be written in a variety of ways.
- There are, however, specific ways of writing algebraic equations that are considered to be standard forms.
- The standard forms facilitate the simplification procedures for Boolean expressions and, in some cases, may result in more desirable expressions for implementing logic circuits.

# 3. STANDARD FORMS

The standard forms contain *product terms* and *sum terms*. An example of a product term is $X\overline{Y}Z$. This is a logical product consisting of an AND operation among three literals. An example of a sum term is $X + Y + \overline{Z}$. This is a logical sum consisting of an OR operation among the literals. In Boolean algebra, the words "product" and "sum" do not imply arithmetic operations; instead, they specify the logical operations AND and OR, respectively.

## ❑ Minterms and Maxterms

- A product term in which all the variables appear exactly once, either complemented or uncomplemented, is called a *minterm*.
- Its characteristic property is that it represents exactly one combination of binary variable values in the truth table. It has the value 1 for that combination and 0 for all others.

# ❑ Minterms and Maxterms

**Minterms for Three Variables**

| X | Y | Z | Product Term | Symbol | $m_0$ | $m_1$ | $m_2$ | $m_3$ | $m_4$ | $m_5$ | $m_6$ | $m_7$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | $\overline{X}\,\overline{Y}\,\overline{Z}$ | $m_0$ | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | $\overline{X}\,\overline{Y}Z$ | $m_1$ | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | $\overline{X}Y\overline{Z}$ | $m_2$ | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | $\overline{X}YZ$ | $m_3$ | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | $X\overline{Y}\,\overline{Z}$ | $m_4$ | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | $X\overline{Y}Z$ | $m_5$ | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | $XY\overline{Z}$ | $m_6$ | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | 1 | $XYZ$ | $m_7$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

## ❑ Minterms and Maxterms

- A sum term that contains all the variables in complemented or uncomplemented form is called a *maxterm*.
- Again, it is possible to formulate $2^n$ maxterms with $n$ variables.
- Each maxterm is a logical sum of the three variables, with each variable being complemented if the corresponding bit of the binary number is 1 and uncomplemented if it is 0.

# ❑ Minterms and Maxterms

**Maxterms for Three Variables**

| X | Y | Z | Sum Term | Symbol | $M_0$ | $M_1$ | $M_2$ | $M_3$ | $M_4$ | $M_5$ | $M_6$ | $M_7$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | $X+Y+Z$ | $M_0$ | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 0 | 1 | $X+Y+\overline{Z}$ | $M_1$ | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 0 | $X+\overline{Y}+Z$ | $M_2$ | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 1 | $X+\overline{Y}+\overline{Z}$ | $M_3$ | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | $\overline{X}+Y+Z$ | $M_4$ | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | $\overline{X}+Y+\overline{Z}$ | $M_5$ | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | $\overline{X}+\overline{Y}+Z$ | $M_6$ | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | $\overline{X}+\overline{Y}+\overline{Z}$ | $M_7$ | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |

# ❑ Minterms and Maxterms

- Consider the Boolean function $F$ in Table

**Boolean Functions of Three Variables**

| (a) X | Y | Z | F | $\overline{F}$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 |

$$F = \overline{X}\,\overline{Y}\,\overline{Z} + \overline{X}Y\overline{Z} + X\overline{Y}Z + XYZ = m_0 + m_2 + m_5 + m_7$$

$$\overline{F}(X,Y,Z) = \overline{X}\,\overline{Y}Z + \overline{X}YZ + X\overline{Y}\,\overline{Z} + XY\overline{Z} = m_1 + m_3 + m_4 + m_6$$

# Logical Organization of Computer
# CHAPTER 2

## COMBINATIONAL LOGIC CIRCUITS

Developed By:
Dr. Vivek Vyas
Department of MCA
DDU

## ❑ Minterms and Maxterms

The following is a summary of the most important properties of minterms:

1. There are $2^n$ minterms for $n$ Boolean variables. These minterms can be generated from the binary numbers from 0 to $2^n - 1$.
2. Any Boolean function can be expressed as a logical sum of minterms.
3. The complement of a function contains those minterms not included in the original function.
4. A function that includes all the $2^n$ minterms is equal to logic 1.

## ❑ Sum of Products (SOP)

- A boolean expression consisting purely of Minterms (product terms) is said to be in canonical sum of products form.
- Example: lets say, we have a boolean function F defined on two variables A and B. So, A and B are the inputs for F and lets say, output of F is true i.e., F = 1 when any one of the input is true or 1. Now we draw the truth table for F.

# ❑ Sum of Products (SOP)

| A | B | F |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

Now we will create a column for the minterm using the variables A and B. If input is 0 we take the complement of the variable and if input is 1 we take the variable as is.

| A | B | F | Minterm |
|---|---|---|---|
| 0 | 0 | 0 | A'B' |
| 0 | 1 | 1 | A'B |
| 1 | 0 | 1 | AB' |
| 1 | 1 | 1 | AB |

## ❑ Sum of Products (SOP)

To get the desired canonical SOP expression we will add the **minterms** (product terms) for which the **output is 1**.

$F = A'B + AB' + AB$

SOP expression implements 2 level AND-OR design in which the 1st level gate is AND gate following the 2nd level gate which is OR gate.

# Logical Organization of Computer
# CHAPTER 2

## COMBINATIONAL LOGIC CIRCUITS

Developed By:
Dr. Vivek Vyas
Department of MCA
DDU

## ❑ Product of Sums (POS)

- A boolean expression consisting purely of Maxterms (sum terms) is said to be in canonical product of sums form.
- Example: Lets say, we have a boolean function F defined on two variables A and B. So, A and B are the inputs for F and lets say, output of F is true i.e., F = 1 when only one of the input is true or 1. now we draw the truth table for F .

# ❑ Product of Sums (POS)

| A | B | F |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

Now we will create a column for the maxterm using the variables A and B. If input is 1 we take the complement of the variable and if input is 0 we take the variable as is.

| A | B | F | Maxterm |
|---|---|---|---|
| 0 | 0 | 0 | A+B |
| 0 | 1 | 1 | A+B’ |
| 1 | 0 | 1 | A’+B |
| 1 | 1 | 0 | A’+B’ |

## ❑ Product of Sums (POS)

To get the desired canonical POS expression we will multiply the maxterms (sum terms) for which the output is 0.

$F = (A+B) . (A'+B')$

POS expression implements 2 level OR-AND design in which the 1st level gate is OR gate following the 2nd level gate which is AND gate.

## ❑ Karnaugh Map Boolean Algebraic Simplification Technique

- Karnaugh map (K-map), introduced by Maurice Karnaugh in 1953.
- The K-map method of solving the logical expressions is referred to as the graphical technique of simplifying Boolean expressions. K-maps are also referred to as 2D truth tables .
- K-maps basically deal with the technique of inserting the values of the output variable in cells within a rectangle or square grid according to a definite pattern.
- The number of cells in the K-map is determined by the number of input variables and is mathematically expressed as two raised to the power of the number of input variables, i.e., $2^n$, where the number of input variables is $n$.

## ❑ Karnaugh Map Boolean Algebraic Simplification Technique

- **Gray Coding**
- Each cell within a K-map has a definite place-value which is obtained by using an encoding technique known as Gray code.
- The specialty of this code is the fact that the adjacent code values differ only by a single bit. That is, if the given code-word is 01, then the previous and the next code-words can be 11 or 00, in any order, but cannot be 10 in any case.
- In K-maps, the rows and the columns of the table use Gray code-labeling which in turn represent the values of the corresponding input variables. This means that each K-map cell can be addressed using a unique Gray Code-Word.

# ❑ Karnaugh Map Boolean Algebraic Simplification Technique

- These concepts are further emphasized by a typical 16-celled K-map shown in below diagram, which can be used to simplify a logical expression comprising of 4-variables (A, B, C and D mentioned at its top-left corner).

# ❑ Karnaugh Map Boolean Algebraic Simplification Technique

For example, the grey colored cell of the K-map shown can be addressed using the code-word "0101" which is equivalent to 5 in decimal (shown as the green number in the figure) and corresponds to the input variable combination $\bar{A}B\bar{C}D$ or $A+\bar{B}+C+\bar{D}$, depending on whether the input–output relationship is expressed in SOP (sum of products) form or POS (product of sums) form, respectively.

Similarly, $A\bar{B}CD$ or $\bar{A}+B+\bar{C}+\bar{D}$ refers to the Gray code-word of "1011", equivalent to 11 in decimal (again, shown in green in the figure), which in turn means that we are addressing the pink-colored K-map cell in the figure.

## ❑ Karnaugh Map Boolean Algebraic Simplification Technique

- With this general idea of K-maps, let us now move on to the procedure employed in designing an optimal (in terms of the number of gates used to realize the logic) digital system. We'll start with a given problem statement.
- **Example 1:**
- **Design a digital system whose output is defined as logically low if the 4-bit input binary number is a multiple of 3; otherwise, the output will be logically high. The output is defined if and only if the input binary number is greater than 2.**

## ❑ Karnaugh Map Boolean Algebraic Simplification Technique

- **Step 1: Truth Table**
- In the given example:
- Number of input variables = 4, which we will call A, B, C and D.
- Number of output variables = 1, which we will call Y
- where
- Y = Don't Care, if the input number is less than 3 (orange entries in the truth table)
- Y = 0, if the input number is an integral multiple of 3 (green entries in the truth table)
- Y = 1, if the input number is not an integral multiple of 3 (blue entries in the truth table)

# ❑ Karnaugh Map Boolean Algebraic Simplification Technique

**Truth Table**

| Inputs | | | | Decimal Equivalent | Output |
|---|---|---|---|---|---|
| A | B | C | D | | Y |
| 0 | 0 | 0 | 0 | 0 | X |
| 0 | 0 | 0 | 1 | 1 | X |
| 0 | 0 | 1 | 0 | 2 | X |
| 0 | 0 | 1 | 1 | 3 | 0 |
| 0 | 1 | 0 | 0 | 4 | 1 |
| 0 | 1 | 0 | 1 | 5 | 1 |
| 0 | 1 | 1 | 0 | 6 | 0 |
| 0 | 1 | 1 | 1 | 7 | 1 |
| 1 | 0 | 0 | 0 | 8 | 1 |
| 1 | 0 | 0 | 1 | 9 | 0 |
| 1 | 0 | 1 | 0 | 10 | 1 |
| 1 | 0 | 1 | 1 | 11 | 1 |
| 1 | 1 | 0 | 0 | 12 | 0 |
| 1 | 1 | 0 | 1 | 13 | 1 |
| 1 | 1 | 1 | 0 | 14 | 1 |
| 1 | 1 | 1 | 1 | 15 | 0 |

where X indicates Don't Care Condition

# Logical Organization of Computer
# CHAPTER 2

## COMBINATIONAL LOGIC CIRCUITS

Developed By:
Dr. Vivek Vyas
Department of MCA
DDU

## ❑ Karnaugh Map Boolean Algebraic Simplification Technique

- Karnaugh map (K-map), introduced by Maurice Karnaugh in 1953.
- The K-map method of solving the logical expressions is referred to as the graphical technique of simplifying Boolean expressions. K-maps are also referred to as 2D truth tables .
- K-maps basically deal with the technique of inserting the values of the output variable in cells within a rectangle or square grid according to a definite pattern.
- The number of cells in the K-map is determined by the number of input variables and is mathematically expressed as two raised to the power of the number of input variables, i.e., $2^n$, where the number of input variables is $n$.

## ❑ Karnaugh Map Boolean Algebraic Simplification Technique

- **Gray Coding**
- Each cell within a K-map has a definite place-value which is obtained by using an encoding technique known as Gray code.
- The specialty of this code is the fact that the adjacent code values differ only by a single bit. That is, if the given code-word is 01, then the previous and the next code-words can be 11 or 00, in any order, but cannot be 10 in any case.
- In K-maps, the rows and the columns of the table use Gray code-labeling which in turn represent the values of the corresponding input variables. This means that each K-map cell can be addressed using a unique Gray Code-Word.

# ❑ Karnaugh Map Boolean Algebraic Simplification Technique

- These concepts are further emphasized by a typical 16-celled K-map shown in below diagram, which can be used to simplify a logical expression comprising of 4-variables (A, B, C and D mentioned at its top-left corner).

# ❑ Karnaugh Map Boolean Algebraic Simplification Technique

For example, the grey colored cell of the K-map shown can be addressed using the code-word "0101" which is equivalent to 5 in decimal (shown as the green number in the figure) and corresponds to the input variable combination $\bar{A}B\bar{C}D$ or $A+\bar{B}+C+\bar{D}$, depending on whether the input–output relationship is expressed in SOP (sum of products) form or POS (product of sums) form, respectively.

Similarly, $A\bar{B}CD$ or $\bar{A}+B+\bar{C}+\bar{D}$ refers to the Gray code-word of "1011", equivalent to 11 in decimal (again, shown in green in the figure), which in turn means that we are addressing the pink-colored K-map cell in the figure.

## ❑ Karnaugh Map Boolean Algebraic Simplification Technique

- With this general idea of K-maps, let us now move on to the procedure employed in designing an optimal (in terms of the number of gates used to realize the logic) digital system. We'll start with a given problem statement.
- **Example 1:**
- **Design a digital system whose output is defined as logically low if the 4-bit input binary number is a multiple of 3; otherwise, the output will be logically high. The output is defined if and only if the input binary number is greater than 2.**

## ❑ Karnaugh Map Boolean Algebraic Simplification Technique

- **Step 1: Truth Table**
- In the given example:
- Number of input variables = 4, which we will call A, B, C and D.
- Number of output variables = 1, which we will call Y
- where
- Y = Don't Care, if the input number is less than 3 (orange entries in the truth table)
- Y = 0, if the input number is an integral multiple of 3 (green entries in the truth table)
- Y = 1, if the input number is not an integral multiple of 3 (blue entries in the truth table)

# ❑ Karnaugh Map Boolean Algebraic Simplification Technique

**Truth Table**

| Inputs | | | | Decimal Equivalent | Output |
|---|---|---|---|---|---|
| A | B | C | D | | Y |
| 0 | 0 | 0 | 0 | 0 | X |
| 0 | 0 | 0 | 1 | 1 | X |
| 0 | 0 | 1 | 0 | 2 | X |
| 0 | 0 | 1 | 1 | 3 | 0 |
| 0 | 1 | 0 | 0 | 4 | 1 |
| 0 | 1 | 0 | 1 | 5 | 1 |
| 0 | 1 | 1 | 0 | 6 | 0 |
| 0 | 1 | 1 | 1 | 7 | 1 |
| 1 | 0 | 0 | 0 | 8 | 1 |
| 1 | 0 | 0 | 1 | 9 | 0 |
| 1 | 0 | 1 | 0 | 10 | 1 |
| 1 | 0 | 1 | 1 | 11 | 1 |
| 1 | 1 | 0 | 0 | 12 | 0 |
| 1 | 1 | 0 | 1 | 13 | 1 |
| 1 | 1 | 1 | 0 | 14 | 1 |
| 1 | 1 | 1 | 1 | 15 | 0 |

where X indicates Don't Care Condition

## ❑ Karnaugh Map Boolean Algebraic Simplification Technique

- **Step 2: Select and Populate K-Map**
- From Step 1, we know the number of input variables involved in the logical expression from which size of the K-map required will be decided.
- Further, we also know the number of such K-maps required to design the desired system as the number of output variables would also be known definitely.
- This means that, for the example considered, we require a single (due to one output variable) K-map with 16 cells (as there are four input variables).

## ❑ Karnaugh Map Boolean Algebraic Simplification Technique

- Next, we have to fill the K-map cells with one for each minterm, zero for each maxterm, and X for Don't Care terms. The procedure is to be repeated for every single output variable.
- Hence for this example, we get the K-map as shown in diagram.

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

## ❑ Karnaugh Map Boolean Algebraic Simplification Technique

- **Step 3: Form the Groups**
- K-map simplification can also be referred to as the "simplification by grouping" technique as it solely relies on the formation of clusters. That is, the main aim of the entire process is together as many ones (for SOP solution) or zeros (for POS solution) under one roof for each of the output variables in the problem stated.
- However, while doing so we have to strictly abide by certain rules and regulations:

## ❑ Karnaugh Map Boolean Algebraic Simplification Technique

- 1. The process has to be initiated by grouping the bits which lie in adjacent cells such that the group formed contains the maximum number of selected bits. This means that for an *n*-variable K-map with $2^n$ cells, try to group for $2^n$ cells first, then for $2^{n-1}$ cells, next for $2^{n-2}$ cells, and so on until the "group" contains only $2^0$ cells, i.e., isolated bits (if any). Note that the number of cells in the group must be equal to an integer power to 2, i.e., 1, 2, 4, 8. . . .
- 2. The procedure must be applied for all adjacent cells of the K-map, even when they appear to be not adjacent—the top row is considered to be adjacent to the bottom row and the rightmost column is considered to be adjacent to the leftmost column, as if the K-map wraps around from top to bottom and right to left.

## ❑ Karnaugh Map Boolean Algebraic Simplification Technique

- 3. A bit appearing in one group can be repeated in another group provided that this leads to the increase in the resulting group-size.
- 4. Don't Care conditions are to be considered for the grouping activity if and only if they help in obtaining a larger group. Otherwise, they are to be neglected.

# ❑ Karnaugh Map Boolean Algebraic Simplification Technique

| | | SOP Form Solution | |
|---|---|---|---|
| Number of groups having 16 cells | | 0 | |
| Number of groups having 8 cells | | 0 | |
| Number of groups having 4 cells (Blue Enclosures in Figure 3) | 2 | Group 1 (Cells 0,2,8,10) | |
| | | Group 2 (Cells 0,1,4,5) | |
| Number of groups having 2 cells (Orange Enclosures in Figure 3) | 4 | Group 3 (Cells 5,7) | Group 4 (Cells 5,13) |
| | | Group 5 (Cells 10,11) | Group 6 (Cells 10,14) |

# Logical Organization of Computer
# CHAPTER 2

## COMBINATIONAL LOGIC CIRCUITS

Developed By:
Dr. Vivek Vyas
Department of MCA
DDU

## ❑ Karnaugh Map Boolean Algebraic Simplification Technique

- With this general idea of K-maps, let us now move on to the procedure employed in designing an optimal (in terms of the number of gates used to realize the logic) digital system. We'll start with a given problem statement.
- **Example 1:**
- **Design a digital system whose output is defined as logically low if the 4-bit input binary number is a multiple of 3; otherwise, the output will be logically high. The output is defined if and only if the input binary number is greater than 2.**

## ❑ Karnaugh Map Boolean Algebraic Simplification Technique

- **Step 1: Truth Table**
- In the given example:
- Number of input variables = 4, which we will call A, B, C and D.
- Number of output variables = 1, which we will call Y
- where
- Y = Don't Care, if the input number is less than 3 (orange entries in the truth table)
- Y = 0, if the input number is an integral multiple of 3 (green entries in the truth table)
- Y = 1, if the input number is not an integral multiple of 3 (blue entries in the truth table)

# ❑ Karnaugh Map Boolean Algebraic Simplification Technique

**Truth Table**

| Inputs | | | | Decimal Equivalent | Output |
|---|---|---|---|---|---|
| A | B | C | D | | Y |
| 0 | 0 | 0 | 0 | 0 | X |
| 0 | 0 | 0 | 1 | 1 | X |
| 0 | 0 | 1 | 0 | 2 | X |
| 0 | 0 | 1 | 1 | 3 | 0 |
| 0 | 1 | 0 | 0 | 4 | 1 |
| 0 | 1 | 0 | 1 | 5 | 1 |
| 0 | 1 | 1 | 0 | 6 | 0 |
| 0 | 1 | 1 | 1 | 7 | 1 |
| 1 | 0 | 0 | 0 | 8 | 1 |
| 1 | 0 | 0 | 1 | 9 | 0 |
| 1 | 0 | 1 | 0 | 10 | 1 |
| 1 | 0 | 1 | 1 | 11 | 1 |
| 1 | 1 | 0 | 0 | 12 | 0 |
| 1 | 1 | 0 | 1 | 13 | 1 |
| 1 | 1 | 1 | 0 | 14 | 1 |
| 1 | 1 | 1 | 1 | 15 | 0 |

where X indicates Don't Care Condition

## ❑ Karnaugh Map Boolean Algebraic Simplification Technique

- **Step 2: Select and Populate K-Map**
- From Step 1, we know the number of input variables involved in the logical expression from which size of the K-map required will be decided.
- Further, we also know the number of such K-maps required to design the desired system as the number of output variables would also be known definitely.
- This means that, for the example considered, we require a single (due to one output variable) K-map with 16 cells (as there are four input variables).

## ❑ Karnaugh Map Boolean Algebraic Simplification Technique

- Next, we have to fill the K-map cells with one for each minterm, zero for each maxterm, and X for Don't Care terms. The procedure is to be repeated for every single output variable.
- Hence for this example, we get the K-map as shown in diagram.

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

## ❑ Karnaugh Map Boolean Algebraic Simplification Technique

- **Step 3: Form the Groups**
- K-map simplification can also be referred to as the "simplification by grouping" technique as it solely relies on the formation of clusters. That is, the main aim of the entire process is together as many ones (for SOP solution) or zeros (for POS solution) under one roof for each of the output variables in the problem stated.
- However, while doing so we have to strictly abide by certain rules and regulations:

## ❑ Karnaugh Map Boolean Algebraic Simplification Technique

- 1. The process has to be initiated by grouping the bits which lie in adjacent cells such that the group formed contains the maximum number of selected bits. This means that for an *n*-variable K-map with $2^n$ cells, try to group for $2^n$ cells first, then for $2^{n-1}$ cells, next for $2^{n-2}$ cells, and so on until the "group" contains only $2^0$ cells, i.e., isolated bits (if any). Note that the number of cells in the group must be equal to an integer power to 2, i.e., 1, 2, 4, 8. . . .
- 2. The procedure must be applied for all adjacent cells of the K-map, even when they appear to be not adjacent—the top row is considered to be adjacent to the bottom row and the rightmost column is considered to be adjacent to the leftmost column, as if the K-map wraps around from top to bottom and right to left.

## ❑ Karnaugh Map Boolean Algebraic Simplification Technique

- 3. A bit appearing in one group can be repeated in another group provided that this leads to the increase in the resulting group-size.
- 4. Don't Care conditions are to be considered for the grouping activity if and only if they help in obtaining a larger group. Otherwise, they are to be neglected.

# ❑ Karnaugh Map Boolean Algebraic Simplification Technique

| | | SOP Form Solution | |
|---|---|---|---|
| Number of groups having 16 cells | | 0 | |
| Number of groups having 8 cells | | 0 | |
| Number of groups having 4 cells (Blue Enclosures in Figure 3) | 2 | Group 1 (Cells 0,2,8,10) | |
| | | Group 2 (Cells 0,1,4,5) | |
| Number of groups having 2 cells (Orange Enclosures in Figure 3) | 4 | Group 3 (Cells 5,7) | Group 4 (Cells 5,13) |
| | | Group 5 (Cells 10,11) | Group 6 (Cells 10,14) |

## ❑ Karnaugh Map Boolean Algebraic Simplification Technique

- **Step 4: Simplified Logical Expression**
- For each of the resulting groups, we have to obtain the corresponding logical expression in terms of the input-variables. This can be done by expressing the bits which are common amongst the Gray code-words which represent the cells contained within the considered group.
- Finally, all these group-wise logical expressions need to be combined appropriately to form the simplified Boolean equation for the output variable. The same procedure must be repeated for every output variable of the given problem.

# ❑ Karnaugh Map Boolean Algebraic Simplification Technique

| SOP Form Solution | |
|---|---|
| **Groups** | **Logical Expression** |
| Group 1 | $\bar{B}\bar{D}$ |
| Group 2 | $\bar{A}\bar{C}$ |
| Group 3 | $\bar{A}BD$ |
| Group 4 | $B\bar{C}D$ |
| Group 5 | $A\bar{B}C$ |
| Group 6 | $AC\bar{D}$ |

Thus, $Y = \bar{B}\bar{D} + \bar{A}\bar{C} + \bar{A}BD + B\bar{C}D + A\bar{B}C + AC\bar{D}$

# ❑ Karnaugh Map Boolean Algebraic Simplification Technique

- **Step 5: System Design**

# Logical Organization of Computer
# CHAPTER 2

## COMBINATIONAL LOGIC CIRCUITS

Developed By:
Dr. Vivek Vyas
Department of MCA
DDU

# ❑ Karnaugh Map Boolean Algebraic Simplification Technique

| POS Form Solution | |
|---|---|
| **Groups** | **Logical Expression** |
| Group 1 | $A+B$ |
| Group 2 | $B+C+\overline{D}$ |
| Group 3 | $A+\overline{C}+D$ |
| Group 4 | $\overline{A}+\overline{B}+C+D$ |
| Group 5 | $\overline{A}+\overline{B}+\overline{C}+\overline{D}$ |
| | |

**Thus, $Y = (A+B)\ (B+C+\overline{D})\ (A+\overline{C}+D)\ (\overline{A}+\overline{B}+C+D)\ (\overline{A}+\overline{B}+\overline{C}+\overline{D})$**

## ❑ Exercise

- **Design Problem:** Design a 3 input, 1 output digital logic circuit which will take all the octal digits (0, 1, … 7) as its input and produce the even parity bit for the corresponding octal digit.

# ❑ Exercise

| A | B | C | O |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 |

# Exercise

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

$$O = A'B'C + A'BC' + AB'C' + ABC$$

# ❑ OTHER GATE TYPES

AND

$F = XY$

| X | Y | F |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

OR

$F = X + Y$

| X | Y | F |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

NOT (inverter)

$F = \overline{X}$

| X | F |
|---|---|
| 0 | 1 |
| 1 | 0 |

Buffer

$F = X$

| X | F |
|---|---|
| 0 | 0 |
| 1 | 1 |

# ❑ OTHER GATE TYPES

NAND

$F = \overline{X \cdot Y}$

| X | Y | F |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

NOR

$F = \overline{X + Y}$

| X | Y | F |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

Exclusive-OR (XOR)

$F = X\overline{Y} + \overline{X}Y$
$= X \oplus Y$

| X | Y | F |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

# ❑ OTHER GATE TYPES

Exclusive-NOR
(XNOR)

$$F = XY + \overline{X}\,\overline{Y}$$
$$= \overline{X \oplus Y}$$

| X | Y | F |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

AND-OR-INVERT
(AOI)

$$F = \overline{WX + YZ}$$

OR-AND -INVERT
(OAI)

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

# ❑ OTHER GATE TYPES

AND-OR
(AO)

$F = WX + YZ$

OR-AND
(OA)

$F = (W + X)(Y + Z)$

## ❑ Exercise

$$G = ABC + ABD + E + ACF + ADF$$

$$G = (AB + AF)(C + D) + E$$

$$G = AB(C + D) + E + A(C + D)F$$

$$G = A(B + F)(C + D) + E$$

# Computer Organization and Architecture
# CHAPTER 3

## Combinational Logic Design

Developed By:
Dr. Vivek Vyas
Department of MCA
DDU

## ❑ Encoder and Decoder

- **Encoder**
- An Encoder is a **combinational circuit** that performs the reverse operation of Decoder.
- It has maximum of **2^n input lines** and **‘n’ output lines**, hence it encodes the information from 2^n inputs into an n-bit code.
- It will produce a binary code equivalent to the input. Therefore, the encoder encodes 2^n input lines with ‘n’ bits.
- The encoders and decoders play an essential role in digital electronics projects; encoders & decoders are used to convert data from one form to another form.

## ❑ Encoder and Decoder

- These are frequently used in communication system such as telecommunication, networking, etc..to transfer data from one end to the other end.

# ❑ Encoder and Decoder

- **4 to 2 Encoder**
- 4 to 2 Encoder has four inputs $Y_3$, $Y_2$, $Y_1$ & $Y_0$ and two outputs $A_1$ & $A_0$. The **block diagram** of 4 to 2 Encoder is shown in the following figure.

## ❑ Encoder and Decoder

- At any time, only one of these 4 inputs can be '1' in order to get the respective binary code at the output. The **Truth table** of 4 to 2 encoder is shown below.

| Inputs | | | | Outputs | |
|---|---|---|---|---|---|
| $Y_3$ | $Y_2$ | $Y_1$ | $Y_0$ | $A_1$ | $A_0$ |
| 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 | 1 |

# ❑ Encoder and Decoder

From Truth table, we can write the **Boolean functions** for each output as

$$A_1 = Y_3 + Y_2$$

$$A_0 = Y_3 + Y_1$$

We can implement the above two Boolean functions by using two input OR gates. The **circuit diagram** of 4 to 2 encoder is shown in the following figure.

# ❑ Encoder and Decoder

# ❑ Encoder and Decoder

- **Octal to Binary Encoder**
- Octal to binary Encoder has eight inputs, $Y_7$ to $Y_0$ and three outputs $A_2$, $A_1$ & $A_0$. Octal to binary encoder is nothing but 8 to 3 encoder. The **block diagram** of octal to binary Encoder is shown in the following figure.

# ❑ Encoder and Decoder

- At any time, only one of these eight inputs can be '1' in order to get the respective binary code. The **Truth table** of octal to binary encoder is shown below.

| Inputs | | | | | | | | Outputs | | |
|---|---|---|---|---|---|---|---|---|---|---|
| $Y_7$ | $Y_6$ | $Y_5$ | $Y_4$ | $Y_3$ | $Y_2$ | $Y_1$ | $Y_0$ | $A_2$ | $A_1$ | $A_0$ |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |

# ❑ Encoder and Decoder

From Truth table, we can write the **Boolean functions** for each output as

$$A_2 = Y_7 + Y_6 + Y_5 + Y_4$$

$$A_1 = Y_7 + Y_6 + Y_3 + Y_2$$

$$A_0 = Y_7 + Y_5 + Y_3 + Y_1$$

# ❑ Encoder and Decoder

## ❑ Encoder and Decoder

- **Drawbacks of Encoder**
- There is an ambiguity, when all outputs of encoder are equal to zero. Because, it could be the code corresponding to the inputs, when only least significant input is one or when all inputs are zero.
- If more than one input is active High, then the encoder produces an output, which may not be the correct code. For **example**, if both $Y_3$ and $Y_6$ are '1', then the encoder produces 111 at the output. This is neither equivalent code corresponding to $Y_3$, when it is '1' nor the equivalent code corresponding to $Y_6$, when it is '1'.

## ❑ Encoder and Decoder

- So, to overcome these difficulties, we should assign priorities to each input of encoder. Then, the output of encoder will be the (binary) code corresponding to the active High input(s), which has higher priority. This encoder is called as **priority encoder**.

# Computer Organization and Architecture
# CHAPTER 3

## Combinational Logic Design

Developed By:
Dr. Vivek Vyas
Department of MCA
DDU

## ❑ Encoder and Decoder

- **Priority Encoder**
- A 4 to 2 priority encoder has four inputs $Y_3$, $Y_2$, $Y_1$ & $Y_0$ and two outputs $A_1$ & $A_0$. Here, the input, $Y_3$ has the highest priority, whereas the input, $Y_0$ has the lowest priority. In this case, even if more than one input is '1' at the same time, the output will be the (binary) code corresponding to the input, which is having **higher priority**.

## ❑ Encoder and Decoder

- We considered one more **output, V** in order to know, whether the code available at outputs is valid or not.
  - If at least one input of the encoder is ‘1’, then the code available at outputs is a valid one. In this case, the output, V will be equal to 1.
  - If all the inputs of encoder are ‘0’, then the code available at outputs is not a valid one. In this case, the output, V will be equal to 0.

## ❑ Encoder and Decoder

- **Truth table** of 4 to 2 priority encoder is shown below.

| Inputs | | | | Outputs | | |
|---|---|---|---|---|---|---|
| $Y_3$ | $Y_2$ | $Y_1$ | $Y_0$ | $A_1$ | $A_0$ | V |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 1 | x | 0 | 1 | 1 |
| 0 | 1 | x | x | 1 | 0 | 1 |
| 1 | x | x | x | 1 | 1 | 1 |

# ❑ Encoder and Decoder

- Use **4 variable K-maps** for getting simplified expressions for each output.

# ❑ Encoder and Decoder

The simplified **Boolean functions** are

$$A_1 = Y_3 + Y_2$$

$$A_0 = Y_3 + Y_2'Y_1$$

Similarly, we will get the Boolean function of output, V as

$$V = Y_3 + Y_2 + Y_1 + Y_0$$

# ❑ Encoder and Decoder

## ❑ Encoder and Decoder

- The above circuit diagram contains two 2-input OR gates, one 4-input OR gate, one 2input AND gate & an inverter.
- Here AND gate & inverter combination are used for producing a valid code at the outputs, even when multiple inputs are equal to ‘1’ at the same time.
- Hence, this circuit encodes the four inputs with two bits based on the **priority** assigned to each input.

## ❑ Encoder and Decoder

- **Decoder**
- **Decoder** is a combinational circuit that has ‘n’ input lines and maximum of $2^n$ output lines.
- One of these outputs will be active High based on the combination of inputs present, when the decoder is enabled. That means decoder detects a particular code.
- The outputs of the decoder are nothing but the **min terms** of ‘n’ input variables (lines), when it is enabled.

# ❑ Encoder and Decoder

- **2 to 4 Decoder**
- Let 2 to 4 Decoder has two inputs $A_1$ & $A_0$ and four outputs $Y_3$, $Y_2$, $Y_1$ & $Y_0$. The **block diagram** of 2 to 4 decoder is shown in the following figure.

## ❑ Encoder and Decoder

- One of these four outputs will be '1' for each combination of inputs when enable, E is '1'. The **Truth table** of 2 to 4 decoder is shown below.

| Enable | Inputs | | Outputs | | | |
|---|---|---|---|---|---|---|
| E | $A_1$ | $A_0$ | $Y_3$ | $Y_2$ | $Y_1$ | $Y_0$ |
| 0 | x | x | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 | 0 | 0 |

# ❑ Encoder and Decoder

From Truth table, we can write the **Boolean functions** for each output as

$$Y_3 = E.A_1.A_0$$

$$Y_2 = E.A_1.A_0'$$

$$Y_1 = E.A_1'.A_0$$

$$Y_0 = E.A_1'.A_0'$$

# ❑ Encoder and Decoder

## ❑ Encoder and Decoder

- Therefore, the outputs of 2 to 4 decoder are nothing but the **min terms** of two input variables $A_1$ & $A_0$, when enable, E is equal to one. If enable, E is zero, then all the outputs of decoder will be equal to zero.

# Computer Organization and Architecture
# CHAPTER 3

## Combinational Logic Design

Developed By:
Dr. Vivek Vyas
Department of MCA
DDU

## ❑ Encoder and Decoder

- **3 to 8 Decoder**
- We know that 2 to 4 Decoder has two inputs, $A_1$ & $A_0$ and four outputs, $Y_3$ to $Y_0$. Whereas, 3 to 8 Decoder has three inputs $A_2$, $A_1$ & $A_0$ and eight outputs, $Y_7$ to $Y_0$.
- We can find the number of lower order decoders required for implementing higher order decoder using the following formula.
  - Required number of lower order decoders = m2/m1
- Where, m1 is the number of outputs of lower order decoder.
- m2 is the number of outputs of higher order decoder.

## ❑ Encoder and Decoder

- Here, m1= 4 and m2 = 8. Substitute, these two values in the above formula.
  - Required number of 2 to 4 decoders = 8/4 = 2
- Therefore, we require two 2 to 4 decoders for implementing one 3 to 8 decoder. The **block diagram** of 3 to 8 decoder using 2 to 4 decoders is shown in the following figure.

# ❑ Encoder and Decoder

# ❑ Encoder and Decoder

| Inputs | | | | | | | | Outputs | | |
|---|---|---|---|---|---|---|---|---|---|---|
| $Y_7$ | $Y_6$ | $Y_5$ | $Y_4$ | $Y_3$ | $Y_2$ | $Y_1$ | $Y_0$ | $A_2$ | $A_1$ | $A_0$ |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |

## ❑ Encoder and Decoder

- The parallel inputs $A_1$ & $A_0$ are applied to each 2 to 4 decoder. The complement of input $A_2$ is connected to Enable, E of lower 2 to 4 decoder in order to get the outputs, $Y_3$ to $Y_0$.
- These are the **lower four min terms**. The input, $A_2$ is directly connected to Enable, E of upper 2 to 4 decoder in order to get the outputs, $Y_7$ to $Y_4$. These are the **higher four min terms**.

## ❑ Multiplexers and De-Multiplexers

- **Multiplexer** is a combinational circuit that has maximum of $2^n$ data inputs, 'n' selection lines and single output line. One of these data inputs will be connected to the output based on the values of selection lines.
- Since there are 'n' selection lines, there will be $2^n$ possible combinations of zeros and ones. So, each combination will select only one data input. Multiplexer is also called as **Mux**.

# ❑ Multiplexers and De-Multiplexers

- **4x1 Multiplexer**
- 4x1 Multiplexer has four data inputs $I_3$, $I_2$, $I_1$ & $I_0$, two selection lines $s_1$ & $s_0$ and one output Y. The **block diagram** of 4x1 Multiplexer is shown in the following diagram.

# ❑ Multiplexers and De-Multiplexers

- One of these 4 inputs will be connected to the output based on the combination of inputs present at these two selection lines. **Truth table** of 4x1 Multiplexer is shown below.

| Selection Lines | | Output |
|---|---|---|
| $S_1$ | $S_0$ | Y |
| 0 | 0 | $I_0$ |
| 0 | 1 | $I_1$ |
| 1 | 0 | $I_2$ |
| 1 | 1 | $I_3$ |

# ❑ Multiplexers and De-Multiplexers

From Truth table, we can directly write the **Boolean function** for output, Y as

$$Y = S_1'S_0'I_0 + S_1'S_0I_1 + S_1S_0'I_2 + S_1S_0I_2$$

## ❑ Multiplexers and De-Multiplexers

- **8x1 Multiplexer**
- We know that 4x1 Multiplexer has 4 data inputs, 2 selection lines and one output. Whereas, 8x1 Multiplexer has 8 data inputs, 3 selection lines and one output.
- So, we require two **4x1 Multiplexers** in first stage in order to get the 8 data inputs. Since, each 4x1 Multiplexer produces one output, we require a **2x1 Multiplexer** in second stage by considering the outputs of first stage as inputs and to produce the final output.

# ❑ Multiplexers and De-Multiplexers

- Let the 8x1 Multiplexer has eight data inputs $I_7$ to $I_0$, three selection lines $s_2$, $s_1$ & s0 and one output Y. The **Truth table** of 8x1 Multiplexer is shown below.

| Selection Inputs | | | Output |
|---|---|---|---|
| $S_2$ | $S_1$ | $S_0$ | Y |
| 0 | 0 | 0 | $I_0$ |
| 0 | 0 | 1 | $I_1$ |
| 0 | 1 | 0 | $I_2$ |
| 0 | 1 | 1 | $I_3$ |
| 1 | 0 | 0 | $I_4$ |
| 1 | 0 | 1 | $I_5$ |
| 1 | 1 | 0 | $I_6$ |
| 1 | 1 | 1 | $I_7$ |

# ❑ Multiplexers and De-Multiplexers

## ❑ Multiplexers and De-Multiplexers

- The same **selection lines, $s_1$ & $s_0$** are applied to both 4x1 Multiplexers. The data inputs of upper 4x1 Multiplexer are $I_7$ to $I_4$ and the data inputs of lower 4x1 Multiplexer are $I_3$ to $I_0$. Therefore, each 4x1 Multiplexer produces an output based on the values of selection lines, $s_1$ & $s_0$.
- The outputs of first stage 4x1 Multiplexers are applied as inputs of 2x1 Multiplexer that is present in second stage. The other **selection line, $s_2$** is applied to 2x1 Multiplexer.

## ❑ Multiplexers and De-Multiplexers

- If $s_2$ is zero, then the output of 2x1 Multiplexer will be one of the 4 inputs $I_3$ to $I_0$ based on the values of selection lines $s_1$ & $s_0$.
- If $s_2$ is one, then the output of 2x1 Multiplexer will be one of the 4 inputs $I_7$ to $I_4$ based on the values of selection lines $s_1$ & $s_0$.

# ❑ Multiplexers and De-Multiplexers

# ❑ Multiplexers and De-Multiplexers

- **16x1 Multiplexer**
- let us implement 16x1 Multiplexer using 8x1 Multiplexers and 2x1 Multiplexer. We know that 8x1 Multiplexer has 8 data inputs, 3 selection lines and one output. Whereas, 16x1 Multiplexer has 16 data inputs, 4 selection lines and one output.
- So, we require two **8x1 Multiplexers** in first stage in order to get the 16 data inputs. Since, each 8x1 Multiplexer produces one output, we require a 2x1 Multiplexer in second stage by considering the outputs of first stage as inputs and to produce the final output.
- Let the 16x1 Multiplexer has sixteen data inputs $I_{15}$ to $I_0$, four selection lines $s_3$ to $s_0$ and one output Y. The **Truth table** of 16x1 Multiplexer is shown below.

# ❑ Multiplexers and De-Multiplexers

| Selection Inputs | | | | Output |
|---|---|---|---|---|
| $S_3$ | $S_2$ | $S_1$ | $S_0$ | Y |
| 0 | 0 | 0 | 0 | $I_0$ |
| 0 | 0 | 0 | 1 | $I_1$ |
| 0 | 0 | 1 | 0 | $I_2$ |
| 0 | 0 | 1 | 1 | $I_3$ |
| 0 | 1 | 0 | 0 | $I_4$ |
| 0 | 1 | 0 | 1 | $I_5$ |
| 0 | 1 | 1 | 0 | $I_6$ |
| 0 | 1 | 1 | 1 | $I_7$ |
| 1 | 0 | 0 | 0 | $I_8$ |
| 1 | 0 | 0 | 1 | $I_9$ |
| 1 | 0 | 1 | 0 | $I_{10}$ |
| 1 | 0 | 1 | 1 | $I_{11}$ |
| 1 | 1 | 0 | 0 | $I_{12}$ |
| 1 | 1 | 0 | 1 | $I_{13}$ |
| 1 | 1 | 1 | 0 | $I_{14}$ |
| 1 | 1 | 1 | 1 | $I_{15}$ |

# ❑ Multiplexers and De-Multiplexers

## ❑ Multiplexers and De-Multiplexers

- The **same selection lines, $s_2$, $s_1$ & $s_0$** are applied to both 8x1 Multiplexers. The data inputs of upper 8x1 Multiplexer are $I_{15}$ to $I_8$ and the data inputs of lower 8x1 Multiplexer are $I_7$ to $I_0$. Therefore, each 8x1 Multiplexer produces an output based on the values of selection lines, $s_2$, $s_1$ & $s_0$.
- The outputs of first stage 8x1 Multiplexers are applied as inputs of 2x1 Multiplexer that is present in second stage. The other **selection line, $s_3$** is applied to 2x1 Multiplexer.

## ❑ Multiplexers and De-Multiplexers

- If $s_3$ is zero, then the output of 2x1 Multiplexer will be one of the 8 inputs $Is_7$ to $I_0$ based on the values of selection lines $s_2$, $s_1$ & $s_0$.
- If $s_3$ is one, then the output of 2x1 Multiplexer will be one of the 8 inputs $I_{15}$ to $I_8$ based on the values of selection lines $s_2$, $s_1$ & $s_0$.

# Computer Organization and Architecture
# CHAPTER 3

## Combinational Logic Design

Developed By:
Dr. Vivek Vyas
Department of MCA
DDU

## ❑ Multiplexers and De-Multiplexers

- **De-Multiplexers**
- **De-Multiplexer** is a combinational circuit that performs the reverse operation of Multiplexer. It has single input, 'n' selection lines and maximum of $2^n$ outputs. The input will be connected to one of these outputs based on the values of selection lines.
- Since there are 'n' selection lines, there will be $2^n$ possible combinations of zeros and ones. So, each combination can select only one output. De-Multiplexer is also called as **De-Mux**.

# ❑ Multiplexers and De-Multiplexers

- **1x4 De-Multiplexer**
- 1x4 De-Multiplexer has one input I, two selection lines, $s_1$ & $s_0$ and four outputs $Y_3$, $Y_2$, $Y_1$ &$Y_0$. The **block diagram** of 1x4 De-Multiplexer is shown in the following diagram.

# ❑ Multiplexers and De-Multiplexers

- The single input 'I' will be connected to one of the four outputs, $Y_3$ to $Y_0$ based on the values of selection lines $s_1$ & s0. The **Truth table** of 1x4 De-Multiplexer is shown below.

| Selection Inputs | | Outputs | | | |
|---|---|---|---|---|---|
| $S_1$ | $S_0$ | $Y_3$ | $Y_2$ | $Y_1$ | $Y_0$ |
| 0 | 0 | 0 | 0 | 0 | I |
| 0 | 1 | 0 | 0 | I | 0 |
| 1 | 0 | 0 | I | 0 | 0 |
| 1 | 1 | I | 0 | 0 | 0 |

From the above Truth table, we can directly write the **Boolean functions** for each output as

$$Y_3 = s_1 s_0 I$$

$$Y_2 = s_1 s_0' I$$

$$Y_1 = s_1' s_0 I$$

$$Y_0 = s_1' s_0' I$$

# ❑ Multiplexers and De-Multiplexers

## ❑ Multiplexers and De-Multiplexers

- **1x8 De-Multiplexer**
- let us implement 1x8 De-Multiplexer using 1x4 De-Multiplexers and 1x2 De-Multiplexer. We know that 1x4 De-Multiplexer has single input, two selection lines and four outputs. Whereas, 1x8 De-Multiplexer has single input, three selection lines and eight outputs.
- So, we require two **1x4 De-Multiplexers** in second stage in order to get the final eight outputs. Since, the number of inputs in second stage is two, we require **1x2 DeMultiplexer** in first stage so that the outputs of first stage will be the inputs of second stage. Input of this 1x2 De-Multiplexer will be the overall input of 1x8 De-Multiplexer.

# ❑ Multiplexers and De-Multiplexers

- Let the 1x8 De-Multiplexer has one input I, three selection lines $s_2$, $s_1$ & $s_0$ and outputs $Y_7$ to $Y_0$. The **Truth table** of 1x8 De-Multiplexer is shown below.

| Selection Inputs | | | Outputs | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| $s_2$ | $s_1$ | $s_0$ | $Y_7$ | $Y_6$ | $Y_5$ | $Y_4$ | $Y_3$ | $Y_2$ | $Y_1$ | $Y_0$ |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | I |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | I | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | I | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | I | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | I | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | I | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | I | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | I | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## ❑ Multiplexers and De-Multiplexers

$$Y0 = D\,\overline{S2}\,\overline{S1}\,\overline{S0}$$

$$Y1 = D\,\overline{S2}\,\overline{S1}\,S0$$

$$Y2 = D\,\overline{S2}\,S1\,\overline{S0}$$

$$Y3 = D\,\overline{S2}\,S1\,S0$$

$$Y4 = D\,S2\,\overline{S1}\,\overline{S0}$$

$$Y5 = D\,S2\,\overline{S1}\,S0$$

$$Y6 = D\,S2\,S1\,\overline{S0}$$

$$Y7 = D\,S2\,S1\,S0$$

# ❑ Multiplexers and De-Multiplexers

# ❑ Multiplexers and De-Multiplexers

## ❑ Multiplexers and De-Multiplexers

- The common **selection lines, $s_1$ & $s_0$** are applied to both 1x4 De-Multiplexers. The outputs of upper 1x4 De-Multiplexer are $Y_7$ to $Y_4$ and the outputs of lower 1x4 De-Multiplexer are $Y_3$ to $Y_0$.
- The other **selection line, $s_2$** is applied to 1x2 De-Multiplexer. If $s_2$ is zero, then one of the four outputs of lower 1x4 De-Multiplexer will be equal to input, I based on the values of selection lines $s_1$ & $s_0$. Similarly, if $s_2$ is one, then one of the four outputs of upper 1x4 DeMultiplexer will be equal to input, I based on the values of selection lines $s_1$ & $s_0$.

## ❑ Multiplexers and De-Multiplexers

- **1x16 De-Multiplexer**
- let us implement 1x16 De-Multiplexer using 1x8 De-Multiplexers and 1x2 De-Multiplexer. We know that 1x8 De-Multiplexer has single input, three selection lines and eight outputs. Whereas, 1x16 De-Multiplexer has single input, four selection lines and sixteen outputs.
- So, we require two **1x8 De-Multiplexers** in second stage in order to get the final sixteen outputs. Since, the number of inputs in second stage is two, we require **1x2 DeMultiplexer** in first stage so that the outputs of first stage will be the inputs of second stage. Input of this 1x2 De-Multiplexer will be the overall input of 1x16 De-Multiplexer.

# ❑ Multiplexers and De-Multiplexers

- Let the 1x16 De-Multiplexer has one input I, four selection lines $s_3$, $s_2$, $s_1$ & $s_0$ and outputs $Y_{15}$ to $Y_0$. The **block diagram** of 1x16 De-Multiplexer using lower order Multiplexers is shown in the following figure.

## ❑ Multiplexers and De-Multiplexers

- The common **selection lines $s_2$, $s_1$ & $s_0$** are applied to both 1x8 De-Multiplexers. The outputs of upper 1x8 De-Multiplexer are $Y_{15}$ to $Y_8$ and the outputs of lower 1x8 DeMultiplexer are $Y_7$ to $Y_0$.
- The other **selection line, $s_3$** is applied to 1x2 De-Multiplexer. If $s_3$ is zero, then one of the eight outputs of lower 1x8 De-Multiplexer will be equal to input, I based on the values of selection lines $s_2$, $s_1$ & $s_0$. Similarly, if s3 is one, then one of the 8 outputs of upper 1x8 De-Multiplexer will be equal to input, I based on the values of selection lines $s_2$, $s_1$ & $s_0$.

# Computer Organization and Architecture
# CHAPTER 4

## Arithmetic Functions and HDLs

Developed By:
Dr. Vivek Vyas
Department of MCA
DDU

# 1. ARITHMETIC LOGIC UNIT

- An **arithmetic logic unit (ALU)** is a digital circuit used to perform arithmetic and logic operations. It represents the fundamental building block of the **central processing unit (CPU)** of a computer. Modern CPUs contain very powerful and complex ALUs. In addition to ALUs, modern CPUs contain a control unit (CU).
- Most of the operations of a CPU are performed by one or more ALUs, which load data from input registers. A **register** is a small amount of storage available as part of a CPU. The control unit tells the ALU what operation to perform on that data, and the ALU stores the result in an output register. The control unit moves the data between these registers, the ALU, and memory.

# 1. ARITHMETIC LOGIC UNIT

- An ALU performs basic arithmetic and logic operations. Examples of arithmetic operations are addition, subtraction, multiplication, and division. Examples of logic operations are comparisons of values such as NOT, AND, and OR.

# Computer Organization and Architecture
# CHAPTER 4

## Arithmetic Functions and HDLs

Developed By:
Dr. Vivek Vyas
Department of MCA
DDU

# 2. BINARY ADDERS

- Combinational circuit that performs the addition of two bits is called a *half adder*.
- One that performs the addition of three bits is called a *full adder*.

# ❑ Half Adder

- By using half adder, you can design simple addition with the help of logic gates.
- Let's see an addition of single bits.

# ❑ Half Adder

$0+0 = 0$

$0+1 = 1$

$1+0 = 1$

$1+1 = 10$

These are the least possible single-bit combinations. But the result for 1+1 is 10, the sum result must be re-written as a 2-bit output. Thus, the equations can be written as

$0+0 = 00$

$0+1 = 01$

$1+0 = 01$

$1+1 = 10$

# ❑ Half Adder

- Half Adder Truth Table

| TURTH TABLE | | | |
|---|---|---|---|
| INPUT | | OUTPUT | |
| A | B | $S = A \oplus B$ | $C = AB$ |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

Now it has been cleared that 1-bit adder can be easily implemented with the help of the XOR Gate for the output 'SUM' and an AND Gate for the 'Carry'.

# ❑ Half Adder

# ❑ Half Adder

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

# ❑ Half Adder

- **Half Adder Using NAND Gate**

# ❑ Half Adder

- **Half Adder Using NOR Gate**

## ❑ Full Adder

- This adder is difficult to implement than a half-adder. The difference between a half-adder and a full-adder is that the full-adder has three inputs and two outputs, whereas half adder has only two inputs and two outputs. The first two inputs are A and B and the third input is an input carry as C-IN.

# ❑ Full Adder

- Full Adder Truth Table

| TURTH TABLE | | | | |
|---|---|---|---|---|
| INPUT | | | OUTPUT | |
| A | B | C | $S = A \oplus B$ | $C = AB + BC + AC$ |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

## ❑ Full Adder

- With the truth-table, the full adder logic can be implemented. You can see that the output S is an XOR between the input A and the half-adder, SUM output with B and C-IN inputs. We take C-OUT will only be true if any of the two inputs out of the three are HIGH.
- So, we can implement a full adder circuit with the help of two half adder circuits. At first, half adder will be used to add A and B to produce a partial Sum and a second half adder logic can be used to add C-IN to the Sum produced by the first half adder to get the final S output.

# ❑ Full Adder

# ❑ Full Adder

# ❑ Full Adder

# ❑ Full Adder

- **Full Adder Using NAND Gate**

# ❑ Full Adder

- **Full Adder Using NOR Gate**

## ❑ Half Subtractor

- The **Half Subtractor** is a digital circuit which processes the subtraction of two 1-bit numbers.
- The circuit of Half subtractor consists of two inputs and two outputs.
- The word **"HALF"** before the subtractor signifies that it deals with only **two 1-bit numbers**, it has nothing to do with the borrow from the previous stage.

# ❑ Half Subtractor

- **Procedure of Subtraction**

$$\begin{array}{r} 1010 \\ -\,1001 \\ \hline 1 \end{array} \qquad \begin{array}{r} 1010 \\ -\,1001 \\ \hline 01 \end{array} \qquad \begin{array}{r} 1010 \\ -\,1001 \\ \hline 001 \end{array} \qquad \begin{array}{r} 1010 \\ -\,1001 \\ \hline 0001 \end{array}$$

# ❑ Half Subtractor

- **Truth Table of Half Subtractor**

| TURTH TABLE | | | |
|---|---|---|---|
| INPUT | | OUTPUT | |
| A | B | D = A ⊕ B | B = AB |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |

## ❑ Half Subtractor

- If we observe carefully, it is quite evident that the difference operation performed by half subtractor is exactly similar to the operation of EX-OR gate.
- Thus, we can easily utilize the EX-OR gate for generating difference bit.
- Similarly, the borrow generated by half subtractor can be easily obtained by using the combination of NOT gate and AND gate.

## ❑ Half Subtractor

# ❑ Half Subtractor

# ❑ Half Subtractor

- **Half Subtractor using NAND gates**

# ❑ Half Subtractor

- **Half Subtractor using NOR gates**

# Computer Organization and Architecture
# CHAPTER 4

## Arithmetic Functions and HDLs

Developed By:
Dr. Vivek Vyas
Department of MCA
DDU

## ❑ Full Subtractor

- A full subtractor is a combinational circuit that performs a subtraction between two bits, taking into account borrow of the lower significant stage.
- This circuit has three inputs and two outputs. The three inputs are A, B and C.
- The two outputs, D and B represent the difference and output borrow, respectively.

# ❑ Full Subtractor

- **Truth Table of Full Subtractor**

| INPUT | | | OUTPUT | |
|---|---|---|---|---|
| A | B | Bin | D | Bout |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 |

## ❑ Full Subtractor

- The construction of **full subtractor circuit diagram** involves two half subtractor joined by an OR gate.
- The two borrow bits generated by two separate half subtractor are fed to the OR gate which produces the final borrow bit.
- The final difference bit is the combination of the difference output of the first half adder and the next higher order pair of bits.

# ❑ Full Subtractor

# ❑ Full Subtractor

# ❑ Full Subtractor

# ❑ Full Subtractor

- **Full Subtractor using NAND gates**

# ❑ Full Subtractor

- **Full Subtractor using NOR gates**

# Computer Organization and Architecture
# CHAPTER 4

## Arithmetic Functions and HDLs

Developed By:
Dr. Vivek Vyas
Department of MCA
DDU

## ❑ Binary Parallel adder

- The **Parallel binary adder** is a **combinational circuit** consists of various full adders in parallel structure so that when more than 1-bit numbers are to be added, then there can be full adder for every column for the addition.
- The number of full adders in a parallel binary adder depends on the number of bits present in the number for the addition.
- If 4-bits numbers are to be added, then there will be 4-full adder in the parallel binary adder.

# ❑ Binary Parallel adder

- **Working of Binary Parallel adder**

# ❑ Binary Parallel adder

- **Logic circuit of 2-Bit Parallel Binary Adder**

Logic Diagram for 2-bit Parallel Adder

# ❑ Binary Parallel adder

- **Working of 5 bit Binary Parallel adder**

5 – Bit Parallel Adder Circuit Using Full Adders

# ❑ Decimal Adder and BCD Adder

- Decimal Adder – The digital systems handles the decimal number in the form of binary coded decimal numbers (BCD).
- A BCD adder is a circuit that adds two BCD digits and produces a sum digit also in BCD.
- BCD numbers use 10 digits, 0 to 9 which are represented in the binary form 0 0 0 0 to **1 0 0 1,** i.e. each BCD digit is represented as a 4-bit binary number.
- Here, we should note that BCD cannot be greater than 9.
- When we write BCD number say 526, it can be represented as

| 5 | 2 | 6 |
|---|---|---|
| ↓ | ↓ | ↓ |
| 0 1 0 1 | 0 0 1 0 | 0 1 1 0 |

## ❑ Decimal Adder and BCD Adder

- If the sum of two number is less then or equal to 9, then the value of BCD sum and binary sum will be same otherwise they will differ by 6(0110 in binary).
- Now, lets move to the table and find out the logic when we are going to add “0110”.

# ❑ Decimal Adder and BCD Adder

- **Truth Table Decimal Adder**
- **/ BCD Adder**

| Decimal | Binary Sum | | | | | BCD Sum | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | C' | S3' | S2' | S1' | S0' | C | S3 | S2 | S1 | S0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| 2 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 3 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 |
| 4 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 5 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 |
| 6 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 |
| 7 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 |
| 8 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 9 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 |
| 10 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| 11 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 |
| 12 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 13 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 14 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| 15 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 |
| 16 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 |
| 17 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 |
| 18 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 |
| 19 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 |

# ❑ Decimal Adder and BCD Adder

We are adding "0110" (=6) only to the second half of the table.

**The conditions are:**

1. If C' = 1 (Satisfies 16-19)
2. If S3'.S2' = 1 (Satisfies 12-15)
3. If S3'.S1' = 1 (Satisfies 10 and 11)

So, our logic is

```
C' + S3'.S2' + S3'.S1' = 1
```

# ❑ Decimal Adder and BCD Adder

# Computer Organization and Architecture
# CHAPTER 5

## Sequential Circuits

Developed By:
Dr. Vivek Vyas
Department of MCA
DDU

## ❑ Introduction

- The combinational circuit does not use any memory. Hence the previous state of input does not have any effect on the present state of the circuit. But sequential circuit has memory so output can vary based on input. This type of circuits uses previous input, output, clock and a memory element.

## ❑ Introduction

- A **Sequential circuit** combinational logic circuit that consists of inputs variable (X), logic gates (Computational circuit), and output variable (Z).

Figure: Combinational Circuits

## ❑ Introduction

- Combinational circuit produces an output based on input variable only, but **Sequential circuit** produces an output based on **current input and previous input variables**.
- That means sequential circuits include memory elements which are capable of storing binary information. That binary information defines the state of the sequential circuit at that time. A latch capable of storing one bit of information.

# ❑ Introduction

**Figure:** Sequential Circuit

## ❑ Introduction

- **Types of Sequential Circuits –** There are two types of sequential circuit :
- **Asynchronous sequential circuit –** These circuit **do not use a clock signal** but uses the pulses of the inputs. These circuits are **faster** than synchronous sequential circuits because there is clock pulse and change their state immediately when there is a change in the input signal.
- We use asynchronous sequential circuits when speed of operation is important and **independent** of internal clock pulse. But these circuits are more **difficult** to design and their output is **uncertain**.

# ❑ Introduction

**Figure:** Asynchronous Sequential Circuit

## ❑ Introduction

- **Synchronous sequential circuit –** These circuit **uses clock signal** and level inputs (or pulsed). The output pulse is the same duration as the clock pulse for the clocked sequential circuits.
- Since they wait for the next clock pulse to arrive to perform the next operation, so these circuits are bit **slower** compared to asynchronous.
- We use sequential circuits to design Counters, Registers, RAM.

# ❑ Introduction

**Figure:** Synchronous Sequential Circuit

# 1. Latch

- The basic storage element is called latch. As the name suggest it latches 0 or 1.
- SR NAND Latch

| Symbol | Truth Table | | |
|---|---|---|---|
| 2-input NAND Gate | B | A | Q |
| | 0 | 0 | 1 |
| | 0 | 1 | 1 |
| | 1 | 0 | 1 |
| | 1 | 1 | 0 |
| Boolean Expression $Q = \overline{A.B}$ | Read as A **AND** B gives **NOT** Q | | |

# 1. Latch

- SR NAND Latch Truth Table

| S | R | Q | $\overline{Q}$ | |
|---|---|---|---|---|
| 0 | 1 | 1 | 0 | Set state |
| 1 | 1 | 1 | 0 | |
| 1 | 0 | 0 | 1 | Reset state |
| 1 | 1 | 0 | 1 | |
| 0 | 0 | 1 | 1 | Undefined |

# Computer Organization and Architecture
# CHAPTER 5

## Sequential Circuits

Developed By:
Dr. Vivek Vyas
Department of MCA
DDU

# 2. S-R Flip Flop

- It is basically S-R latch using NAND gates with an additional **enable** input. It is also called as level triggered SR-FF. For this, circuit in output will take place if and only if the enable input (E) is made active. In short this circuit will operate as an S-R latch if E = 1 but there is no change in the output if E = 0.

# 2. S-R Flip Flop

- Truth Table

| Inputs | | | Outputs | | Comments |
|---|---|---|---|---|---|
| E | S | R | $Q_{n+1}$ | $\overline{Q}_{n+1}$ | |
| 1 | 0 | 0 | $Q_n$ | $\overline{Q}_n$ | No change |
| 1 | 0 | 1 | 0 | 1 | Rset |
| 1 | 1 | 0 | 1 | 0 | Set |
| 1 | 1 | 1 | x | x | Indeterminate |

# 2. S-R Flip Flop

- **Circuit Diagram**

# 2. S-R Flip Flop

- **Operation**
- **1. S = R = 0 : No change**
  - If S = R = 0 then output of NAND gates 3 and 4 are forced to become 1.
  - Hence R' and S' both will be equal to 1. Since S' and R' are the input of the basic S-R latch using NAND gates, there will be no change in the state of outputs.
- **2. S = 0, R = 1, E = 1**
  - Since S = 0, output of NAND-3 i.e. R' = 1 and E = 1 the output of NAND-4 i.e. S' = 0.
  - Hence $Q_{n+1}$ = 0 and $Q_{n+1}$ bar = 1. This is reset condition.

# 2. S-R Flip Flop

- **3. S = 1, R = 0, E = 1**
  - Output of NAND-3 i.e. R' = 0 and output of NAND-4 i.e. S' = 1.
  - Hence output of S-R NAND latch is $Q_{n+1}$ = 1 and $Q_{n+1}$ bar = 0. This is the reset condition.
- **4. S = 1, R = 1, E = 1**
  - As S = 1, R = 1 and E = 1, the output of NAND gates 3 and 4 both are 0 i.e. S' = R' = 0.
  - Hence the **Race** condition will occur in the basic NAND latch.

# 3. D Flip Flop

- Data Flip Flop or D Flip Flop is the simple gated S-R latch with a NAND inverter connected between S and R inputs.
- Hence S = R = 0 or S = R = 1, these input condition will never appear. This problem is avoid by SR = 00 and SR = 1 conditions.

# 3. D Flip Flop

| Inputs | | Outputs | | Comments |
|---|---|---|---|---|
| E | D | $Q_{n+1}$ | $\overline{Q}_{n+1}$ | |
| 1 | 0 | 0 | 1 | Rset |
| 1 | 1 | 1 | 0 | Set |

# 3. D Flip Flop

# 3. D Flip Flop

- Operation
- **1. E = 0:** Latch is disabled. Hence no change in output.
- **2 E = 1 and D = 0:** If E = 1 and D = 0 then S = 0 and R = 1. Hence irrespective of the present state, the next state is Qn+1 = 0 and Qn+1 bar = 1. This is the reset condition.
- **3 E = 1 and D = 1:** If E = 1 and D = 1, then S = 1 and R = 0. This will set the latch and Qn+1 = 1 and Qn+1 bar = 0 irrespective of the present state.

# Computer Organization and Architecture
# CHAPTER 5

## Sequential Circuits

Developed By:
Dr. Vivek Vyas
Department of MCA
DDU

# 4. J K Flip Flop

- The **JK Flip Flop** is the most widely used flip flop. It is considered to be a universal flip-flop circuit.
- The sequential operation of the JK Flip Flop is same as for the SR flip-flop with the same **SET** and **RESET** input.
- The difference is that the JK Flip Flop does not have the invalid input states of the SR Latch (when S and R are both 1).
- The JK Flip Flop name has been kept on the inventor name of the circuit known as **Jack Kilby.** The basic **symbol** of the JK Flip Flop is shown below.

# 4. J K Flip Flop

# 4. J K Flip Flop

# 4. J K Flip Flop

Truth Table

| J | K | CLK | Q |
|---|---|---|---|
| 0 | 0 | ↑ | $Q_0$ (no change) |
| 1 | 0 | ↑ | 1 |
| 0 | 1 | ↑ | 0 |
| 1 | 1 | ↑ | $\overline{Q}_0$ (toggles) |

# 5. Master Slave J K Flip Flop

- **Race Around Condition In JK Flip-flop**
- For J-K flip-flop, if J=K=1, and if clk=1 for a long period of time, then Q output will toggle as long as CLK is high, which makes the output of the flip-flop unstable or uncertain. This problem is called race around condition in J-K flip-flop.
- This problem (Race Around Condition) can be avoided by ensuring that the clock input is at logic “1” only for a very short time. This introduced the concept of **Master Slave JK** flip flop.

# 5. Master Slave J K Flip Flop

- The Master-Slave Flip-Flop is basically a combination of two JK flip-flops connected together in a series configuration.
- Out of these, one acts as the **"master"** and the other as a **"slave"**. The output from the master flip flop is connected to the two inputs of the slave flip flop whose output is fed back to inputs of the master flip flop.
- In addition to these two flip-flops, the circuit also includes an **inverter**. The inverter is connected to clock pulse in such a way that the inverted clock pulse is given to the slave flip-flop.
- In other words if CP=0 for a master flip-flop, then CP=1 for a slave flip-flop and if CP=1 for master flip flop then it becomes 0 for slave flip flop.

# 5. Master Slave J K Flip Flop

# Computer Organization and Architecture
# CHAPTER 5

## Sequential Circuits

Developed By:
Dr. Vivek Vyas
Department of MCA
DDU

# 5. Master Slave J K Flip Flop

- **Race Around Condition In JK Flip-flop**
- For J-K flip-flop, if J=K=1, and if clk=1 for a long period of time, then Q output will toggle as long as CLK is high, which makes the output of the flip-flop unstable or uncertain. This problem is called race around condition in J-K flip-flop.
- This problem (Race Around Condition) can be avoided by ensuring that the clock input is at logic “1” only for a very short time. This introduced the concept of **Master Slave JK** flip flop.

# 5. Master Slave J K Flip Flop

- The Master-Slave Flip-Flop is basically a combination of two JK flip-flops connected together in a series configuration.
- Out of these, one acts as the **"master"** and the other as a **"slave"**. The output from the master flip flop is connected to the two inputs of the slave flip flop whose output is fed back to inputs of the master flip flop.
- In addition to these two flip-flops, the circuit also includes an **inverter**. The inverter is connected to clock pulse in such a way that the inverted clock pulse is given to the slave flip-flop.
- In other words if CP=0 for a master flip-flop, then CP=1 for a slave flip-flop and if CP=1 for master flip flop then it becomes 0 for slave flip flop.

# 5. Master Slave J K Flip Flop

# 5. Master Slave J K Flip Flop

## Truth Table

| Trigger | Inputs | | Output | | | | | | Inference |
|---|---|---|---|---|---|---|---|---|---|
| | | | Present State | | Intermediate | | Next State | | |
| CLK | J | K | Q | $\bar{Q}$ | $M_1$ | $M_2$ | Q | $\bar{Q}$ | |
| ↑ | 0 | 0 | 0 | 1 | 0 | 1 | Latched | | No Change |
| ↓ | | | 0 | 1 | Latched | | 0 | 1 | |
| ↑ | | | 1 | 0 | 1 | 0 | Latched | | |
| ↓ | | | 1 | 0 | Latched | | 1 | 0 | |
| ↑ | 0 | 1 | 0 | 1 | 0 | 1 | Latched | | Reset |
| ↓ | | | 0 | 1 | Latched | | 0 | 1 | |
| ↑ | | | 1 | 0 | 0 | 1 | Latched | | |
| ↓ | | | 1 | 0 | Latched | | 0 | 1 | |
| ↑ | 1 | 0 | 0 | 1 | 1 | 0 | Latched | | Set |
| ↓ | | | 0 | 1 | Latched | | 1 | 0 | |
| ↑ | | | 1 | 0 | 1 | 0 | Latched | | |
| ↓ | | | 1 | 0 | Latched | | 1 | 0 | |
| ↑ | 1 | 1 | 0 | 1 | 1 | 0 | Latched | | Toggles |
| ↓ | | | 0 | 1 | Latched | | 1 | 0 | |
| ↑ | | | 1 | 0 | 0 | 1 | Latched | | |
| ↓ | | | 1 | 0 | Latched | | 0 | 1 | |

# Computer Organization and Architecture
# CHAPTER 5

## Sequential Circuits

Developed By:
Dr. Vivek Vyas
Department of MCA
DDU

# 6. T Flip Flop

- The name T flip-flop is termed from the nature of toggling operation. The major applications of T flip-flop are counters and control circuits.
- **T flip flop is modified form of JK flip-flop** making it to operate in toggling region.
- Whenever the **clock signal is LOW, the input is never going to affect the output state**. The clock has to be high for the inputs to get active.

# 6. T Flip Flop

# 6. T Flip Flop

- **Truth Table**

| | Previous | | Next | |
|---|---|---|---|---|
| T | $Q_{Prev}$ | $Q'_{Prev}$ | $Q_{Next}$ | $Q'_{Next}$ |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 |

# 7. Shift Registers

- One flip-flop can store one-bit of information. In order to store multiple bits of information, we require multiple flip-flops.
- The group of flip-flops, which are used to hold (store) the binary data is known as **register**.
- If the register is capable of shifting bits either towards right hand side or towards left hand side is known as **shift register**.
- An 'N' bit shift register contains 'N' flip-flops. Following are the four types of shift registers based on applying inputs and accessing of outputs.

# 7. Shift Registers

# 7. Shift Registers

- Serial In - Serial Out shift register
- Serial In - Parallel Out shift register
- Parallel In - Serial Out shift register
- Parallel In - Parallel Out shift register
- Bidirectional Shift Register

## ❑ Serial In - Serial Out (SISO) Shift Register

- The shift register, which allows serial input and produces serial output is known as Serial In – Serial Out **(SISO)** shift register. The **block diagram** of 3-bit SISO shift register is shown in the following figure.

## ❑ Serial In - Serial Out (SISO) Shift Register

- This block diagram consists of three D flip-flops, which are **cascaded**. That means, output of one D flip-flop is connected as the input of next D flip-flop. All these flip-flops are synchronous with each other since, the same clock signal is applied to each one.
- In this shift register, we can send the bits serially from the input of left most D flip-flop. Hence, this input is also called as **serial input**.
- For every positive edge triggering of clock signal, the data shifts from one stage to the next. So, we can receive the bits serially from the output of right most D flip-flop. Hence, this output is also called as **serial output**.

## ❑ Serial In - Serial Out (SISO) Shift Register

- **Example**
- Let us see the working of 3-bit SISO shift register by sending the binary information “011” from LSB to MSB serially at the input.
- Assume, initial status of the D flip-flops from leftmost to rightmost is $Q_2Q_1Q_0$=000
- We can understand the working of 3-bit SISO shift register from the following table.

# ❑ Serial In - Serial Out (SISO) Shift Register

| No of positive edge of Clock | Serial Input | $Q_2$ | $Q_1$ | $Q_0$ |
|---|---|---|---|---|
| 0 | - | 0 | 0 | 0 |
| 1 | 1(LSB) | 1 | 0 | 0 |
| 2 | 1 | 1 | 1 | 0 |
| 3 | 0(MSB) | 0 | 1 | 1(LSB) |
| 4 | - | - | 0 | 1 |
| 5 | - | - | - | 0(MSB) |

## ❑ Serial In - Serial Out (SISO) Shift Register

- The initial status of the D flip-flops in the absence of clock signal is $Q_2Q_1Q_0$=000.
- Here, the serial output is coming from $Q_0$So, the LSB (1) is received at 3rd positive edge of clock and the MSB (0) is received at 5th positive edge of clock.
- Therefore, the 3-bit SISO shift register requires five clock pulses in order to produce the valid output. Similarly, the N-bit SISO shift register requires 2N-1 clock pulses in order to shift 'N' bit information.

## ❑ Serial In - Parallel Out (SIPO) Shift Register

- The shift register, which allows serial input and produces parallel output is known as Serial In – Parallel Out **(SIPO)** shift register. The **block diagram** of 3-bit SIPO shift register is shown in the following figure.

## ❑ Serial In - Parallel Out (SIPO) Shift Register

- This circuit consists of three D flip-flops, which are cascaded. That means, output of one D flip-flop is connected as the input of next D flip-flop. All these flip-flops are synchronous with each other since, the same clock signal is applied to each one.
- In this shift register, we can send the bits serially from the input of left most D flip-flop. Hence, this input is also called as **serial input**.
- For every positive edge triggering of clock signal, the data shifts from one stage to the next. In this case, we can access the outputs of each D flip-flop in parallel. So, we will get **parallel outputs** from this shift register.

## ❑ Serial In - Parallel Out (SIPO) Shift Register

- **Example**
- Let us see the working of 3-bit SIPO shift register by sending the binary information “011” from LSB to MSB serially at the input.
- Assume, initial status of the D flip-flops from leftmost to rightmost is $Q_2Q_1Q_0$=000.
- Here, $Q_2$ & $Q_0$ are MSB & LSB respectively. We can understand the working of 3-bit SIPO shift register from the following table.

## ❑ Serial In - Parallel Out (SIPO) Shift Register

| No of positive edge of Clock | Serial Input | $Q_2$(MSB) | $Q_1$ | $Q_0$(LSB) |
|---|---|---|---|---|
| 0 | - | 0 | 0 | 0 |
| 1 | 1(LSB) | 1 | 0 | 0 |
| 2 | 1 | 1 | 1 | 0 |
| 3 | 0(MSB) | 0 | 1 | 1 |

## ❑ Serial In - Parallel Out (SIPO) Shift Register

- The initial status of the D flip-flops in the absence of clock signal is $Q_2Q_1Q_0$=000.
- The binary information “011” is obtained in parallel at the outputs of D flip-flops for third positive edge of clock.
- So, the 3-bit SIPO shift register requires three clock pulses in order to produce the valid output. Similarly, the N-bit SIPO shift register requires N clock pulses in order to shift ‘N’ bit information.

# Computer Organization and Architecture
# CHAPTER 5

## Sequential Circuits

Developed By:
Dr. Vivek Vyas
Department of MCA
DDU

# ❑ Parallel In - Serial Out (PISO) Shift Register

- The shift register, which allows parallel input and produces serial output is known as Parallel In – Serial Out **(PISO)** shift register. The **block diagram** of 3-bit PISO shift register is shown in the following figure.

## ❑ Parallel In - Serial Out (PISO) Shift Register

- This circuit consists of three D flip-flops, which are cascaded. That means, output of one D flip-flop is connected as the input of next D flip-flop.
- All these flip-flops are synchronous with each other since, the same clock signal is applied to each one.
- In this shift register, we can apply the **parallel inputs** to each D flip-flop by making Preset Enable to 1.
- For every positive edge triggering of clock signal, the data shifts from one stage to the next. So, we will get the **serial output** from the right most D flip-flop.

## ❑ Parallel In - Serial Out (PISO) Shift Register

- **Example**
- Let us see the working of 3-bit PISO shift register by applying the binary information “011” in parallel through preset inputs.
- Since the preset inputs are applied before positive edge of Clock, the initial status of the D flip-flops from leftmost to rightmost will be $Q_2Q_1Q_0$=011.
- We can understand the working of 3-bit PISO shift register from the following table.

# ❑ Parallel In - Serial Out (PISO) Shift Register

| No of positive edge of Clock | $Q_2$ | $Q_1$ | $Q_0$ |
|---|---|---|---|
| 0 | 0 | 1 | 1(LSB) |
| 1 | - | 0 | 1 |
| 2 | - | - | 0(LSB) |

## ❑ Parallel In - Serial Out (PISO) Shift Register

- Here, the serial output is coming from Q0. So, the LSB (1) is received before applying positive edge of clock and the MSB (0) is received at 2nd positive edge of clock.

- Therefore, the 3-bit PISO shift register requires two clock pulses in order to produce the valid output. Similarly, the N-bit PISO shift register requires N-1 clock pulses in order to shift 'N' bit information.

# ❑ Parallel In - Parallel Out (PIPO) Shift Register

- The shift register, which allows parallel input and produces parallel output is known as Parallel In – Parallel Out **(PIPO)** shift register. The **block diagram** of 3-bit PIPO shift register is shown in the following figure.

## ❑ Parallel In - Parallel Out (PIPO) Shift Register

- This circuit consists of three D flip-flops, which are cascaded. That means, output of one D flip-flop is connected as the input of next D flip-flop. All these flip-flops are synchronous with each other since, the same clock signal is applied to each one.
- In this shift register, we can apply the **parallel inputs** to each D flip-flop by making Preset Enable to 1. We can apply the parallel inputs through preset or clear.
- These two are asynchronous inputs. That means, the flip-flops produce the corresponding outputs, based on the values of asynchronous inputs. In this case, the effect of outputs is independent of clock transition. So, we will get the **parallel outputs** from each D flip-flop.

## ❑ Parallel In - Parallel Out (PIPO) Shift Register

- **Example**
- Let us see the working of 3-bit PIPO shift register by applying the binary information “011” in parallel through preset inputs.
- Since the preset inputs are applied before positive edge of Clock, the initial status of the D flip-flops from leftmost to rightmost will be $Q_2Q_1Q_0$=011.
- So, the binary information “011” is obtained in parallel at the outputs of D flip-flops before applying positive edge of clock.
- Therefore, the 3-bit PIPO shift register requires one clock pulses in order to produce the valid output.

# ❑ Bidirectional Shift Register

## ❑ Bidirectional Shift Register

- Bidirectional shift register allows shifting of data either to left or to the right side.
- It can be implemented using logic gates circuitry that enables the transfer of data from one stage to the next stage to the right or to the left, depend on the level of control line.
- The M is the control input signal which allows data shifting either towards right or towards left.
- A high on this line enables the shifting of data towards right and low enables it towards left.

## ❑ Bidirectional Shift Register

- When M is high, gates G1, G3, G5 and G7 are enabled.
- The state of Q output of each flip flop is passed through the D input of the following flip flop.
- When the pulse arrives, the data are shifted one place to the right.
- When the M signal is low, gates G2, G4, G6, G8 are enabled.
- The Q output of each flip-flop is passed through the D input of the preceding flip-flop.

# 8. Counters

- Counter is a sequential circuit. A digital circuit which is used for a counting pulses is known counter.
- Counter is the widest application of flip-flops. It is a group of flip-flops with a clock signal applied. Counters are of two types.
- Asynchronous or ripple counters.
- Synchronous counters.

## ❑ Asynchronous or ripple counters

- The logic diagram of a 2-bit ripple up counter is shown in figure. The toggle (T) flip-flop are being used. But we can use the JK flip-flop also with J and K connected permanently to logic 1.
- External clock is applied to the clock input of flip-flop A and $Q_A$ output is applied to the clock input of the next flip-flop i.e. FF-B.

## ❑ Asynchronous or ripple counters

- **Operation**
- 1. **Initially let both the FFs be in the reset state:** $Q_BQ_A = 00$ initially
- 2. **After 1st negative clock edge:** As soon as the first negative clock edge is applied, FF-A will toggle and $Q_A$ will be equal to 1.
- $Q_A$ is connected to clock input of FF-B. Since $Q_A$ has changed from 0 to 1, it is treated as the positive clock edge by FF-B. There is no change in $Q_B$ because FF-B is a negative edge triggered FF.
- $Q_BQ_A = 01$ after the first clock pulse.

## ❑ Asynchronous or ripple counters

- 3. **After 2nd negative clock edge:** On the arrival of second negative clock edge, FF-A toggles again and $Q_A = 0$.
- The change in $Q_A$ acts as a negative clock edge for FF-B. So it will also toggle, and $Q_B$ will be 1.
- $Q_BQ_A = 10$ after the second clock pulse.
- 4. **After 3rd negative clock edge:** On the arrival of 3rd negative clock edge, FF-A toggles again and $Q_A$ become 1 from 0.
- Since this is a positive going change, FF-B does not respond to it and remains inactive. So $Q_B$ does not change and continues to be equal to 1.
- $Q_BQ_A = 11$ after the third clock pulse.

## ❑ Asynchronous or ripple counters

- 5. **After 4th negative clock edge:** On the arrival of 4th negative clock edge, FF-A toggles again and $Q_A$ becomes 1 from 0.
- This negative change in $Q_A$ acts as clock pulse for FF-B. Hence it toggles to change $Q_B$ from 1 to 0.
- $Q_BQ_A$ = 00 after the fourth clock pulse.

# ❑ Asynchronous or ripple counters

## Truth Table

| Clock | Counter output | | State number | Deciimal Counter output |
|---|---|---|---|---|
| | $Q_B$ | $Q_A$ | | |
| Initially | 0 | 0 | — | 0 |
| 1st | 0 | 1 | 1 | 1 |
| 2nd | 1 | 0 | 2 | 2 |
| 3rd | 1 | 1 | 3 | 3 |
| 4th | 0 | 0 | 4 | 0 |

# Computer Organization and Architecture
# CHAPTER 5

## Sequential Circuits

Developed By:
Dr. Vivek Vyas
Department of MCA
DDU

# ❑ Synchronous counters

- If the "clock" pulses are applied to all the flip-flops in a counter simultaneously, then such a counter is called as synchronous counter.
- **2-bit Synchronous up counter**
- The $J_A$ and $K_A$ inputs of FF-A are tied to logic 1. So FF-A will work as a toggle flip-flop. The $J_B$ and $K_B$ inputs are connected to $Q_A$.

## ❑ Synchronous counters

- **Operation**
- 1. **Initially let both the FFs be in the reset state :** $Q_BQ_A$ = 00 initially.
- 2. **After 1st negative clock edge:** As soon as the first negative clock edge is applied, FF-A will toggle and $Q_A$ will change from 0 to 1.
- But at the instant of application of negative clock edge, $Q_A$ , $J_B = K_B = 0$. Hence FF-B will not change its state. So $Q_B$ will remain 0.
- $Q_BQ_A$ = 01 after the first clock pulse.

## ❑ Synchronous counters

- 3. **After 2nd negative clock edge:** On the arrival of second negative clock edge, FF-A toggles again and $Q_A$ changes from 1 to 0.
- But at this instant $Q_A$ was 1. So $J_B = K_B = 1$ and FF-B will toggle. Hence $Q_B$ changes from 0 to 1.
- $Q_BQ_A = 10$ after the second clock pulse.
- 4. **After 3rd negative clock edge:** On application of the third falling clock edge, FF-A will toggle from 0 to 1 but there is no change of state for FF-B.
- $Q_BQ_A = 11$ after the third clock pulse.

## ❑ Synchronous counters

- 5. **After 4th negative clock edge:** On application of the next clock pulse, $Q_A$ will change from 1 to 0 as $Q_B$ will also change from 1 to 0.
- $Q_BQ_A$ = 00 after the fourth clock pulse.

## ❑ Classification of counters

- Depending on the way in which the counting progresses, the synchronous or asynchronous counters are classified as follows −
- Up counters
- Down counters
- Up/Down counters

## ❑ Classification of counters

- **UP/DOWN Ripple Counters**
- In the UP/DOWN ripple counter all the FFs operate in the toggle mode. So either T flip-flops or JK flip-flops are to be used.
- **UP counting mode (M=0)** – The Q output of the preceding FF is connected to the clock of the next stage if up counting is to be achieved. For this mode, the mode select input M is at logic 0 (M=0).
- **DOWN counting mode (M=1)** – If M = 1, then the Q bar output of the preceding FF is connected to the next FF. This will operate the counter in the counting mode.

# ❑ 3 bit ripple up counter

# Computer Organization and Architecture
# CHAPTER 5

## Sequential Circuits

Developed By:
Dr. Vivek Vyas
Department of MCA
DDU

# ❑ 3 bit ripple up counter

# ❑ 3 bit ripple up counter

## ❑ 3 bit ripple up counter

# ❑ 3 bit ripple up counter

# ❑ 3 bit ripple up counter

## ❑ 3 bit ripple up counter

| clock | $Q_C$ | $Q_B$ | $Q_C$ | Decimal Eq |
|---|---|---|---|---|
| Initially | | | | |
| 1st (↓) | | | | |
| 2nd (↓) | | | | |
| 3rd (↓) | | | | |

# ❑ 3 bit ripple up counter

| clock | $Q_C$ | $Q_B$ | $Q_C$ | Decimal Eq |
|---|---|---|---|---|
| Initially | 0 | 0 | 0 | 0 |
| 1st (↓) | 0 | 0 | 1 | 1 |
| 2nd (↓) | 0 | 1 | 0 | 2 |
| 3rd (↓) | 0 | 1 | 1 | 3 |
| 4th (↓) | 1 | 0 | 0 | 4 |
| 5th (↓) | 1 | 0 | 1 | 5 |
| 6th (↓) | 1 | 1 | 0 | 6 |
| 7th (↓) | 1 | 1 | 1 | 7 |
| 8th (↓) | 0 | 0 | 0 | 0 |

no.

# Computer Organization and Architecture
# CHAPTER 5

## Sequential Circuits

Developed By:
Dr. Vivek Vyas
Department of MCA
DDU

## ❑ BCD or Decade Counter

- A binary coded decimal (BCD) is a serial digital counter that counts ten digits .And it resets for every new clock input. As it can go through 10 unique combinations of output, it is also called as "Decade counter".

# ❑ BCD or Decade Counter

**Truth Table of Decade Counter**

| Input Pulses | D | C | B | A |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 2 | 0 | 0 | 1 | 0 |
| 3 | 0 | 0 | 1 | 1 |
| 4 | 0 | 1 | 0 | 0 |
| 5 | 0 | 1 | 0 | 1 |
| 6 | 0 | 1 | 1 | 0 |
| 7 | 0 | 1 | 1 | 1 |
| 8 | 1 | 0 | 0 | 0 |
| 9 | 1 | 0 | 0 | 1 |
| 10 | 1 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 (resets) |

# ❑ BCD or Decade Counter

# Computer Organization and Architecture

## CPU Architecture

Developed By:
Dr. Vivek Vyas
Department of MCA
DDU

## ❑ Microprocessor Introduction

- Microprocessor is a controlling unit of a micro-computer, fabricated on a small chip capable of performing ALU (Arithmetic Logical Unit) operations and communicating with the other devices connected to it.
- Microprocessor consists of an ALU, register array, and a control unit. ALU performs arithmetical and logical operations on the data received from the memory or an input device.
- Register array consists of registers identified by letters like B, C, D, E, H, L and accumulator. The control unit controls the flow of data and instructions within the computer.

# ❑ Microprocessor Introduction

- **Block Diagram of a Basic Microcomputer**

## ❑ Microprocessor Introduction

- **How does a Microprocessor Work?**
- The microprocessor follows a sequence: Fetch, Decode, and then Execute.
- Initially, the instructions are stored in the memory in a sequential order. The microprocessor fetches those instructions from the memory, then decodes it and executes those instructions till STOP instruction is reached.
- Later, it sends the result in binary to the output port. Between these processes, the register stores the temporarily data and ALU performs the computing functions.

## ❑ Microprocessor Introduction

- **List of Terms Used in a Microprocessor**
- Here is a list of some of the frequently used terms in a microprocessor −
- **Instruction Set** − It is the set of instructions that the microprocessor can understand.
- **Bandwidth** − It is the number of bits processed in a single instruction.
- **Clock Speed** − It determines the number of operations per second the processor can perform. It is expressed in megahertz (MHz) or gigahertz (GHz).It is also known as Clock Rate.

## ❑ Microprocessor Introduction

- **Word Length** – It depends upon the width of internal data bus, registers, ALU, etc. An 8-bit microprocessor can process 8-bit data at a time. The word length ranges from 4 bits to 64 bits depending upon the type of the microcomputer.
- **Data Types** – The microprocessor has multiple data type formats like binary, BCD, ASCII, signed and unsigned numbers.

## ❑ Microprocessor - 8085 Architecture

- 8085 is pronounced as "eighty-eighty-five" microprocessor. It is an 8-bit microprocessor designed by Intel in 1977 using NMOS technology.
- It has the following configuration −
  - 8-bit data bus
  - 16-bit address bus, which can address upto 64KB
  - A 16-bit program counter
  - A 16-bit stack pointer
  - Six 8-bit registers arranged in pairs: BC, DE, HL
  - Requires +5V supply to operate at 3.2 MHZ single phase clock
  - It is used in washing machines, microwave ovens, mobile phones, etc.

RST RST RST
Trap 7.5 6.5 5.5 INTR INTA
Interrupt Controller
SID SOD
Serial I/O Controller
8-bit
Internal Bus
Accumulator (8)
Status Flags (8)
Temporary Register (8)
Arithmetic Logic Unit
Processing Unit
Instruction Register (8)
Instruction Decoder
B (8) C (8)
D (8) E (8)
H (8) L (8)
Stack pointer (16)
Program counter (16)
Increment/Decrement Register
Timing & Control Unit
Reset Out
CLK Out
HLDA
Ready
HOLD
ALE
WR
RD
IO/M
Instruction Unit
Address Latch
Address/Data Latch
A8 - A15
Storage and Interface Unit
Power Supply
5v

INTA
RST 6.5
TRAP
INTR
RST 5.5
RST 7.5
SID
SOD
Interrupt Control
Serial I/O Control
8-Bit Internal Data Bus
Accumulator (8)
Temp. Reg. (8)
Flag Flip-Flops (5)
Arithmetic Logic Unit (ALU) (8)
Instruction Register (8)
Instruction Decoder and Machine Cycle Encoding
Multiplexer
Reg. Select
W Temp. Reg. (8)
Z Temp. Reg. (8)
B Reg. (8)
C Reg. (8)
D Reg. (8)
E Reg. (8)
H Reg. (8)
L Reg. (8)
Stack Pointer (16)
Program Counter (16)
Incrementer/Decrementer Address Latch (16)
Register Array
Power Supply
+5 V
GND
Timing and Control
X1
X2
CLK GEN
Control
Status
DMA
Reset
CLK OUT
READY
RD
WR
ALE
S0
S1
IO/M
HOLD
HLDA
RESET IN
RESET OUT
Address Buffer (8)
Data/Address Buffer (8)
A15–A8 Address Bus
AD7–AD0 Address/Data Bus

# Computer Organization and Architecture

## CPU Architecture

Developed By:
Dr. Vivek Vyas
Department of MCA
DDU

## ❑ 8085 Microprocessor – Functional Units

- **Accumulator**
- It is an 8-bit register used to perform arithmetic, logical, I/O & LOAD/STORE operations. It is connected to internal data bus & ALU.
- **Arithmetic and logic unit**
- As the name suggests, it performs arithmetic and logical operations like Addition, Subtraction, AND, OR, etc. on 8-bit data.

## ❑ 8085 Microprocessor – Functional Units

- **General purpose register**
- There are 6 general purpose registers in 8085 processor, i.e. B, C, D, E, H & L. Each register can hold 8-bit data.
- These registers can work in pair to hold 16-bit data and their pairing combination is like B-C, D-E & H-L.
- **Program counter**
- It is a 16-bit register used to store the memory address location of the next instruction to be executed. Microprocessor increments the program whenever an instruction is being executed, so that the program counter points to the memory address of the next instruction that is going to be executed.

## ❑ 8085 Microprocessor – Functional Units

- **Stack pointer**
- It is also a 16-bit register works like stack, which is always incremented/decremented by 2 during push & pop operations.
- **Temporary register**
- It is an 8-bit register, which holds the temporary data of arithmetic and logical operations.

## ❑ 8085 Microprocessor – Functional Units

- **Flag register**
- It is an 8-bit register having five 1-bit flip-flops, which holds either 0 or 1 depending upon the result stored in the accumulator.
- These are the set of 5 flip-flops −
  - Sign (S)
  - Zero (Z)
  - Auxiliary Carry (AC)
  - Parity (P)
  - Carry (C)

## ❑ 8085 Microprocessor – Functional Units

- **Instruction register and decoder**
- It is an 8-bit register. When an instruction is fetched from memory then it is stored in the Instruction register. Instruction decoder decodes the information present in the Instruction register.
- **Timing and control unit**
- It provides timing and control signal to the microprocessor to perform operations. Following are the timing and control signals, which control external and internal circuits –

## ❑ 8085 Microprocessor – Functional Units

- Control Signals: READY, RD’, WR’, ALE
- Status Signals: S0, S1, IO/M’
- DMA Signals: HOLD, HLDA
- RESET Signals: RESET IN, RESET OUT

## ❑ 8085 Microprocessor – Functional Units

- **Interrupt control**
- As the name suggests it controls the interrupts during a process.
- When a microprocessor is executing a main program and whenever an interrupt occurs, the microprocessor shifts the control from the main program to process the incoming request.
- After the request is completed, the control goes back to the main program.
- There are 5 interrupt signals in 8085 microprocessor: INTR, RST 7.5, RST 6.5, RST 5.5, TRAP.

## ❑ 8085 Microprocessor – Functional Units

- **Serial Input/output control**
- It controls the serial data communication by using these two instructions: SID (Serial input data) and SOD (Serial output data).
- **Address buffer and address-data buffer**
- The content stored in the stack pointer and program counter is loaded into the address buffer and address-data buffer to communicate with the CPU.
- The memory and I/O chips are connected to these buses; the CPU can exchange the desired data with the memory and I/O chips.

## ❑ 8085 Microprocessor – Functional Units

- **Address bus and data bus**
- Data bus carries the data to be stored. It is bidirectional, whereas address bus carries the location to where it should be stored and it is unidirectional.
- It is used to transfer the data & Address I/O devices.

## ❑ 8085 Microprocessor – Functional Units

- **8085 instruction set consists of the following instructions:**
  - Data moving instructions.
  - Arithmetic - add, subtract, increment and decrement.
  - Logic - AND, OR, XOR and rotate.
  - Control transfer - conditional, unconditional, call subroutine, return from subroutine and restarts.
  - Input/Output instructions.
  - Other - setting/clearing flag bits, enabling/disabling interrupts, stack operations, etc

## ❑ Addressing Modes in 8085

- These are the instructions used to transfer the data from one register to another register, from the memory to the register, and from the register to the memory without any alteration in the content.
- Addressing modes in 8085 is classified into 5 groups:
  - Immediate addressing mode
  - Register addressing mode
  - Direct addressing mode
  - Indirect addressing mode
  - Implied addressing mode

## ❑ Addressing Modes in 8085

- **Immediate addressing mode**
- In this mode, the 8/16-bit data is specified in the instruction itself as one of its operand. **For example:** MVI K, 20F: means 20F is copied into register K.
- **Register addressing mode**
- In this mode, the data is copied from one register to another. **For example:** MOV K, B: means data in register B is copied to register K.

## ❑ Addressing Modes in 8085

- **Direct addressing mode**
- In this mode, the data is directly copied from the given address to the register. **For example:** LDB 5000K: means the data at address 5000K is copied to register B.
- **Indirect addressing mode**
- In this mode, the data is transferred from one register to another by using the address pointed by the register. **For example:** MOV K, B: means data is transferred from the memory address pointed by the register to the register K.

## ❑ Addressing Modes in 8085

- **Implied addressing mode**
- This mode doesn't require any operand; the data is specified by the opcode itself. **For example:** CMA (finds and stores the 1's complement of the contains of accumulator A in A).

# ❑ Microprocessor - 8085 Pin Configuration

## ❑ Microprocessor - 8085 Pin Configuration

- The pins of a 8085 microprocessor can be classified into seven groups –
- **Address bus**
- A15-A8, it carries the most significant 8-bits of memory/IO address.
- **Data bus**
- AD7-AD0, it carries the least significant 8-bit address and data bus.
- **Control and status signals**
- These signals are used to identify the nature of operation. There are 3 control signal and 3 status signals.

## ❑ Microprocessor - 8085 Pin Configuration

- Three control signals are RD, WR & ALE.
- **RD** – This signal indicates that the selected IO or memory device is to be read and is ready for accepting data available on the data bus.
- **WR** – This signal indicates that the data on the data bus is to be written into a selected memory or IO location.
- **ALE (Address Latch Enable)** – It is a positive going pulse generated when a new operation is started by the microprocessor. When the pulse goes high, it indicates address. When the pulse goes down it indicates data.

## ❑ Microprocessor - 8085 Pin Configuration

- Three status signals are IO/M, S0 & S1.
- **IO/M**
- This signal is used to differentiate between IO and Memory operations, i.e. when it is high indicates IO operation and when it is low then it indicates memory operation.
- **S1 & S0**
- These signals are used to identify the type of current operation.
- **Power supply**
- There are 2 power supply signals – VCC & VSS. VCC indicates +5v power supply and VSS indicates ground signal.

## ❑ Microprocessor - 8085 Pin Configuration

- **Clock signals**
- There are 3 clock signals, i.e. X1, X2, CLK OUT.
- **X1, X2** – It is used to set frequency of the internal clock generator. This frequency is internally divided by 2.
- **CLK OUT** – This signal is used as the system clock for devices connected with the microprocessor.

## ❑ Microprocessor - 8085 Pin Configuration

- **Interrupts & externally initiated signals**
- Interrupts are the signals generated by external devices to request the microprocessor to perform a task. There are 5 interrupt signals, i.e. TRAP, RST 7.5, RST 6.5, RST 5.5, and INTR.
- **INTA** – It is an interrupt acknowledgment signal.
- **RESET IN** – This signal is used to reset the microprocessor by setting the program counter to zero.
- **RESET OUT** – This signal is used to reset all the connected devices when the microprocessor is reset.

## ❑ Microprocessor - 8085 Pin Configuration

- **READY –** This signal indicates that the device is ready to send or receive data. If READY is low, then the CPU has to wait for READY to go high.

- **HOLD –** This signal indicates that another master is requesting the use of the address and data buses.

- **HLDA (HOLD Acknowledge) –** It indicates that the CPU has received the HOLD request and it will hand over the bus in the next clock cycle.

## ❑ Microprocessor - 8085 Pin Configuration

- **Serial I/O signals**
- There are 2 serial signals, i.e. SID and SOD and these signals are used for serial communication.
- **SOD** (Serial output data line) – The output SOD is set/reset as specified by the SIM (Set Interrupt Mask) instruction.
- **SID** (Serial input data line) – The data on this line is loaded into accumulator whenever a RIM (Read Interrupt Mask) instruction is executed.

# Computer Organization and Architecture

## CPU Architecture

Developed By:
Dr. Vivek Vyas
Department of MCA
DDU

## ❑ Interrupts in 8085

- Interrupts are the signals generated by the external devices to request the microprocessor to perform a task.
- There are 5 interrupt signals, i.e. TRAP, RST 7.5, RST 6.5, RST 5.5, and INTR.
- Interrupt are classified into following groups based on their parameter:
- **Vector interrupt** – In this type of interrupt, the interrupt address is known to the processor. **For example:** RST7.5, RST6.5, RST5.5, TRAP.
- **Non-Vector interrupt** – In this type of interrupt, the interrupt address is not known to the processor so, the interrupt address needs to be sent externally by the device to perform interrupts. **For example:** INTR.

## ❑ Interrupts in 8085

- **Maskable interrupt –** In this type of interrupt, we can disable the interrupt by writing some instructions into the program. For example: RST7.5, RST6.5, RST5.5.

- **Non-Maskable interrupt –** In this type of interrupt, we cannot disable the interrupt by writing some instructions into the program. For example: TRAP.

## ❑ Interrupts in 8085

- **Software interrupt** – In this type of interrupt, the programmer has to add the instructions into the program to execute the interrupt. There are 8 software interrupts in 8085, i.e. RST0, RST1, RST2, RST3, RST4, RST5, RST6, and RST7.

- **Hardware interrupt** – There are 5 interrupt pins in 8085 used as hardware interrupts, i.e. TRAP, RST7.5, RST6.5, RST5.5, INTA.

## ❑ Interrupts in 8085

- **Interrupt Service Routine (ISR)**
- A small program or a routine that when executed, services the corresponding interrupting source is called an ISR.
- **TRAP**
- It is a non-maskable interrupt, having the highest priority among all interrupts. By default, it is enabled until it gets acknowledged.
- In case of failure, it executes as ISR and sends the data to backup memory. This interrupt transfers the control to the location 0024H.

## ❑ Interrupts in 8085

- **RST 7.5**
- It is a maskable interrupt, having the second highest priority among all interrupts. When this interrupt is executed, the processor saves the content of the PC register into the stack and branches to 003CH address.
- **RST 6.5**
- It is a maskable interrupt, having the third highest priority among all interrupts. When this interrupt is executed, the processor saves the content of the PC register into the stack and branches to 0034H address.

## ❑ Interrupts in 8085

- **RST 5.5**
- It is a maskable interrupt. When this interrupt is executed, the processor saves the content of the PC register into the stack and branches to 002CH address.
- **INTR**
- It is a maskable interrupt, having the lowest priority among all interrupts. It can be disabled by resetting the microprocessor.

## ❑ Instruction Formats

- Computer perform task on the basis of instruction provided. A instruction in computer comprises of groups called fields.
- These field contains different information as for computers every thing is in 0 and 1 so each field has different significance on the basis of which a CPU decide what so perform. The most common fields are:
- **Operation field** which specifies the operation to be performed like addition.
- **Address field** which contain the location of operand, i.e., register or memory location.
- **Mode field** which specifies how operand is to be founded.

## ❑ Instruction Formats

- A instruction is of various length depending upon the number of addresses it contain. Generally CPU organization are of three types on the basis of number of address fields:
- Single Accumulator organization
- General register organization
- Stack organization

## ❑ Instruction Formats

- In first organization operation is done involving a special register called accumulator.
- In second on multiple registers are used for the computation purpose. In third organization the work on stack basis operation due to which it does not contain any address field.
- It is not necessary that only a single organization is applied, a blend of various organization is mostly what we see generally.
- Note that we will use X = (A+B)*(C+D) expression to showcase the procedure.
- On the basis of number of address instruction are classified as:

## ❑ Instruction Formats

- **1. Zero Address Instructions**
- A stack based computer do not use address field in instruction. To evaluate a expression first it is converted to revere Polish Notation i.e. Post fix Notation.
- Expression: X = (A+B)*(C+D)
- Postfixed : X = AB+CD+*
- TOP means top of stack
- M[X] is any memory location

# ❑ Instruction Formats

| | | |
|---|---|---|
| PUSH | A | TOP = A |
| PUSH | B | TOP = B |
| ADD | | TOP = A+B |
| PUSH | C | TOP = C |
| PUSH | D | TOP = D |
| ADD | | TOP = C+D |
| MUL | | TOP = (C+D)*(A+B) |
| POP | X | M[X] = TOP |

## ❑ Instruction Formats

- **2. One Address Instructions:**
- This use a implied ACCUMULATOR register for data manipulation.
- One operand is in accumulator and other is in register or memory location.
- Implied means that the CPU already know that one operand is in accumulator so there is no need to specify it.

| | | |
|---|---|---|
| opcode | operand/address of operand | mode |

# ❑ Instruction Formats

```
Expression: X = (A+B)*(C+D)
AC is accumulator
M[] is any memory location
M[T] is temporary location
```

| | | |
|---|---|---|
| LOAD | A | AC = M[A] |
| ADD | B | AC = A[C] + M[B] |
| STORE | T | M[T] = AC |
| LOAD | C | AC = M[C] |
| ADD | D | AC = AC + M[D] |
| MUL | T | AC = AC * M[T] |
| STORE | X | M[X] = AC |

# Computer Organization and Architecture

## CPU Architecture

Developed By:
Dr. Vivek Vyas
Department of MCA
DDU

## ❑ Instruction Formats

- **3. Two Address Instructions:**
- This is common in commercial computers. Here two address can be specified in the instruction.
- Unlike earlier in one address instruction the result was stored in accumulator here result cab be stored at different location rather than just accumulator, but require more number of bit to represent address.

| | | | |
|---|---|---|---|
| opcode | Destination address | Source address | mode |

# ❑ Instruction Formats

```
Expression: X = (A+B)*(C+D)
R1, R2 are registers
M[] is any memory location
```

| | | |
|---|---|---|
| MOV | R1, A | R1 = M[A] |
| ADD | R1, B | R1 = R1 + M[B] |
| MOV | R2, C | R2 = C |
| ADD | R2, D | R2 = R2 + D |
| MUL | R1, R2 | R1 = R1 * R2 |
| MOV | X, R1 | M[X] = R1 |

## ❑ Instruction Formats

- **4. Three Address Instructions:**
- This has three address field to specify a register or a memory location. Program created are much short in size but number of bits per instruction increase.
- These instructions make creation of program much easier but it does not mean that program will run much faster because now instruction only contain more information but each micro operation (changing content of register, loading address in address bus etc.) will be performed in one cycle only.

| | | | | |
|---|---|---|---|---|
| opcode | Destination address | Source address | Source address | mode |

# ❑ Instruction Formats

```
Expression: X = (A+B)*(C+D)
R1, R2 are registers
M[] is any memory location
```

| | | |
|---|---|---|
| ADD | R1, A, B | R1 = M[A] + M[B] |
| ADD | R2, C, D | R2 = M[C] + M[D] |
| MUL | X, R1, R2 | M[X] = R1 * R2 |

## ❑ Microprogramming

- **Microprogramming**, Process of writing microcode for a microprocessor.
- Microcode is low-level code that defines how a microprocessor should function when it executes machine-language instructions.
- Typically, one machine-language instruction translates into several microcode instructions.
- On some computers, the microcode is stored in ROM and cannot be modified; on some larger computers, it is stored in EPROM and therefore can be replaced with newer versions.

## ❑ Microprogramming

- **Microprogrammed Control Unit:**
- 1) A control unit with its binary control values stored as words in memory is called as microprogrammed control. Each word in the control memory contains microinstruction that specifies one or more microperations for the system. A sequence of microinstructions constitutes a micro program.
- 2) Microprogrammed implementation is a software approach in contrast to the hardwired approach.
- 3) It deals with various units of software but at the micro level i.e. micro-operation, micro-instruction, micro-program etc.

# ❑ Microprogramming

- 4) Different key elements used for implementation of a control unit using microprogrammed approach is shown in diagram below:

## ❑ Microprogramming

- **Control Memory:**
- The set of microinstruction is stored in control Memory (CM) also called as control store.
- **Control Address Register (CAR):**
- It contains the address of next microinstruction to be read. This is similar to the program counter(PC) which stores the address of the next instruction.

## ❑ Microprogramming

- **Control Buffer Register(CBR):**
- When microinstruction is read from the control memory, it is transferred to a control Buffer Register (CBR), which is similar to the instruction Register (IR) that stores the opcode of the instruction read from the memory.
- **Sequencing:**
- It loads the control Address register with the address of the next instruction to be read and issues a read command to control memory.

# Computer Organization and Architecture

## I/O Architecture

Developed By:
Dr. Vivek Vyas
Department of MCA
DDU

## ❑ Computer I/O

- The input and output subsystem of a computer provides an efficient mode of communication between the CPU and the outside environment.
- Programs and data must be entered into the memory for processing, and results obtained from computations must be recorded or displayed.
- To make a computer work properly, data paths must be provided that let information flow between CPU(s), RAM, and the score of I/O devices that can be connected to a personal computer.
- These data paths, which are denoted as the *buses* , act as the primary communication channels inside the computer.

## ❑ Computer I/O

- Any computer has a *system bus* that connects most of the internal hardware devices.
- A typical system bus is the PCI (*Peripheral Component Interconnect*) bus.
- Among the input and output devices commonly found in computer systems are keyboards, displays, printers, magnetic drives, and compact disk read-only memory (CD-ROM) drives.
- Other input and output devices frequently encountered are modems or other communication interfaces, scanners, and sound cards with speakers and microphones.

## ❑ SAMPLE PERIPHERALS

- Devices that the CPU controls directly are said to be connected *online*.
- These devices communicate directly with the CPU or transfer binary information into or out of the memory upon command from the CPU.
- Input or output devices attached to the computer online are called *peripherals*.
- For example: *Keyboards*, *display units* and *printers* are common peripheral devices.

## ❑ Keyboard

- The keyboard is among the simplest of the electromechanical devices attached to the typical computer.
- Since it is manually controlled, it has one of the slowest data rates of any peripheral.
- The keyboard consists of a collection of keys that can be pressed by the user.
- It is necessary to detect which of the keys have been pressed.
- To do this, a *scan matrix* that lies beneath the keys is used, as shown in diagram.

# ❑ Keyboard

## ❑ Keyboard

- The matrix shown in the figure is 8 × 16, giving 128 intersections, so it can handle up to 128 keys.
- A decoder drives the *X* lines of the matrix.
- A multiplexer is attached to the *Y* lines of the matrix.
- The decoder and the multiplexer are controlled by a microcontroller, a tiny computer that contains RAM, ROM, a timer, and simple I/O interfaces.

## ❑ Keyboard

- The microcontroller is programmed to periodically scan all intersections in the matrix by manipulating the control inputs of the decoder and multiplexer.
- If the key is pressed at an intersection, a signal path is closed from an output of the *X* decoder to an input of the *Y* multiplexer. The existence of this path is sensed at an input to the microcontroller.
- Whether a key is pressed or released, the control code at the time of the event is sensed and is translated by the microcontroller into a *K-scan code*.
- When a key is pressed, a *make code* is produced; when a key is released, a *break code* is produced.

## ❑ Keyboard

- Thus, there are two codes for each key, one for when the key is pressed and one for when it is released.
- Note that the scanning of the entire keyboard occurs hundreds of times per second, so there is no danger of missing any press or release of a key.

# Computer Organization and Architecture

## I/O Architecture

Developed By:
Dr. Vivek Vyas
Department of MCA
DDU

## ❑ Hard Drive

- The hard drive is the primary intermediate-speed, nonvolatile, writable storage medium for most computers.
- The typical hard drive stores information serially on a nonremovable disk.
- There are one or more read/writer heads per recording surface.
- Each disk is divided into tracks. The set of tracks that are at the same distance from the center of all disk surfaces is referred to as a *cylinder*.

# ❑ Hard Drive

## ❑ Hard Drive

- Each track is divided into *sectors* containing a fixed number of bytes.
- The number of bytes per sector typically ranges from 256 to 4K.
- The typical byte address includes the cylinder number, head number, sector number, and word offset within the sector.
- The time required to move the heads from the current cylinder to the desired cylinder is called the *seek time*.
- The time required to rotate the disk from its current position to that having the desired sector under the heads is called the *rotational delay*.

## ❑ Hard Drive

- In addition, a certain amount of time is required by the drive controller to access and output information. This is the *controller time*.
- The time required to locate a word on the disk is the *disk access time*, which is the sum of the controller time, the seek time, and the rotational delay.
- The transfer rate for a block of words, once the block has been located, is the *disk transfer rate,* typically specified in megabytes/second (MB/s).
- The sum of the disk access time and the disk transfer rate times the number of bytes per sector gives an estimate of the time required to transfer the information in a sector to or from the hard drive.

## ❑ I/O INTERFACES

- Peripherals connected to a computer need special communication links to interface them with the CPU.
- The purpose of these links is to resolve the differences in the properties of the CPU and memory and the properties of each peripheral.
- The major differences are as follows:
- 1. Peripherals are often electromechanical devices whose manner of operation is different from that of the CPU and memory, which are electronic devices. Therefore, a conversion of signal values may be required.

## ❑ I/O INTERFACES

- **2.** The data-transfer rate of peripherals is usually different from the clock rate of the CPU. Consequently, a synchronization mechanism may be needed.
- **3.** Data codes and formats in peripherals differ from the word format in the CPU and memory.
- **4.** The operating modes of peripherals differ from each other, and each must be controlled in a way that does not disturb the operation of other peripherals connected to the CPU.

## ❑ I/O INTERFACES

- To resolve these differences, computer systems include special hardware components between the CPU and the peripherals to supervise and synchronize all input and output transfers. These components are called *interface units*.
- In addition, each device has its own controller to supervise the operations of the particular mechanism of that peripheral.
- For example, the controller in a printer attached to a computer controls the motion of the paper, the timing of the printing, and the selection of the characters to be printed.

# ❑ I/O Bus and Interface Unit

- A typical communication structure between the CPU and several peripherals is shown in diagram.

## ❑ I/O Bus and Interface Unit

- Each peripheral has an interface unit associated with it. The common bus from the CPU is attached to all peripheral interfaces.
- To communicate with a particular device, the CPU places a device address on the address bus.
- Each interface attached to the common bus contains an address decoder that monitors the address lines.
- When the interface detects its own address, it activates the path between the bus lines and the device that it controls.

## ❑ I/O Bus and Interface Unit

- All peripherals with addresses that do not correspond to the address on the bus ignore the bus activity.
- At the same time that the address is made available on the address bus, the CPU provides a function code on the control lines.
- The selected interface responds to the function code and proceeds to execute it.
- If data must be transferred, the interface communicates with both the device and the CPU data bus to synchronize the transfer.

# Computer Organization and Architecture

## I/O Architecture

Developed By:
Dr. Vivek Vyas
Department of MCA
DDU

## ❑ I/O Bus and Interface Unit

- In addition to communicating with the I/O devices, the CPU of a computer must communicate with the memory unit through an address and data bus.
- There are two ways that external computer buses communicate with memory and I/O.
- One method uses common data, address, and control buses for both memory and I/O. This is referred to this configuration as *memory-mapped I/O*.

## ❑ I/O Bus and Interface Unit

- The second alternative is to share a common address bus and data bus, but use different control lines for memory and I/O.
- This method is referred to as the *isolated I/O configuration*.

# ❑ Example of I/O Interface

| CS | RS1 | RS0 | Register selected |
|---|---|---|---|
| 0 | X | X | None: data bus in high-impedance state |
| 1 | 0 | 0 | Port A register |
| 1 | 0 | 1 | Port B register |
| 1 | 1 | 0 | Control register |
| 1 | 1 | 1 | Status register |

## ❑ Example of I/O Interface

- A typical I/O interface unit consists of two data registers called *ports*, a control register, a status register, a bidirectional data bus, and timing and control circuits.
- The function of the interface is to translate the signals between the CPU buses.
- The I/O data from the device can be transferred into either port A or port B.
- The interface may operate with an output device, with an input device, or with a device that requires both input and output.

## ❑ Example of I/O Interface

- If the interface is connected to a printer, it will only output data; if it services a scanner, it will only input data. A
- hard drive transfers data in both directions, but not at the same time; so the interface needs only one set of I/O bidirectional data lines.
- The *control register* receives control information from the CPU.
- The bits in the status register are used for status conditions and for recording errors that may occur during data transfer.

## ❑ Example of I/O Interface

- The interface registers communicate with the CPU through the bidirectional data bus.
- The circuit sets the chip select (*CS*) input when the interface is selected by the address bus.
- The two *register select inputs* *RS*1 and *RS*0 are usually connected to the two least significant lines of the address bus.
- These two inputs select one of the four registers in the interface, as specified in the table.

## ❑ Example of I/O Interface

- The CPU, interface, and I/O device are likely to have different clocks that are not synchronized with each other. Thus, these units are said to be *asynchronous* with respect to each other.

## ❑ Handshaking

- The *handshaking* method uses two control signals to deal with the timing of transfers.
- In addition to the signal from the unit initiating the transfer, there is a second control signal from the other unit involved in the transfer.
- The basic principle of a two-signal handshaking procedure for data transfer is as follows.
- One control line from the initiating unit is used to request a response from the other unit.

## ❑ Handshaking

- The second control line from the other unit is used to reply to the initiating unit that the response is occurring.
- In this way, each unit informs the other of its status, and the result is an orderly transfer through the bus.

(a) Destination-initiated transfer

## ❑ Handshaking

(b) Source-initiated transfer

## ❑ SERIAL COMMUNICATION

- The transfer of data between two units may be parallel or serial.
- In parallel data transfer, each bit of the message has its own path, and the entire message is transmitted at one time.
- This means that an *n*-bit message is transmitted in parallel through *n* separate paths.
- In serial data transmission, each bit in the message is sent in sequence, one at a time.

## ❑ SERIAL COMMUNICATION

- Parallel transmission is faster, because multiple signal lines operate in parallel.
- It is used for short distances and when speed is important.
- Serial transmission is slower, but less expensive.
- One way that computers and terminals that are remote from each other are connected is via telephone lines.
- Since telephone lines were originally designed for voice communication, but computers communicate in terms of digital signals, some form of conversion is needed.

## ❑ SERIAL COMMUNICATION

- The devices that do the conversion are called *data sets* or *modems* (modulator–demodulators).
- Serial data can be transmitted between two points in three different modes: simplex, half duplex, or full duplex.

# ❑ SERIAL COMMUNICATION

- **Simplex**

# ❑ SERIAL COMMUNICATION

- **Half Duplex**

# ❑ SERIAL COMMUNICATION

- **Full Duplex**

## ❑ Modes of I/O Data Transfer

- Data transfer between the central computer and I/O devices may be handled in a variety of modes, some of which use the CPU as an intermediate path, while others transfer the data directly to and from the memory.
- Data transfer to and from peripherals may be handled in one of three possible modes:
- **1.** Data transfer under program control.
- **2.** Interrupt-initiated data transfer.
- **3.** Direct memory access transfer.

# Computer Organization and Architecture

## I/O Architecture

Developed By:
Dr. Vivek Vyas
Department of MCA
DDU

## ❑ Modes of I/O Data Transfer

- **1. Data transfer under program control:**
- Program-controlled operations are the result of I/O instructions written in the computer program.
- Each transfer of data is initiated by an instruction in the program. Usually, the transfer is to and from a CPU register and peripheral.
- Other instructions are needed to transfer the data to and from the CPU and memory.
- Transferring data under program control requires constant monitoring of the peripheral by the CPU.

## ❑ Modes of I/O Data Transfer

- Once a data transfer is initiated, the CPU is required to monitor the interface to see when a transfer can again be made.
- In the program-controlled transfer, the CPU stays in a program loop called a busy-wait loop until the I/O unit indicates that it is ready for data transfer.
- This is a time-consuming process, since it keeps the processor busy needlessly.

# ❑ Modes of I/O Data Transfer

- **Example of Program-Controlled Transfer**

# ❑ Modes of I/O Data Transfer

## ❑ Modes of I/O Data Transfer

- **2. Interrupt-initiated data transfer:**
- An alternative to having the CPU constantly monitor the flag is to let the interface inform the computer when it is ready to transfer data. This mode of transfer uses the interrupt facility.
- While the CPU is running a program, it does not check the flag. However, when the flag is set, the computer is momentarily interrupted from proceeding with the current program and is informed that the flag has been set.
- The CPU drops what it is doing to take care of the input or output transfer.

## ❑ Modes of I/O Data Transfer

- After the transfer is completed, the computer returns to the previous program to continue what it was doing before the interrupt.
- The CPU responds to the interrupt signal by storing the return address from the program counter into a memory stack or register, and then control branches to a service routine that processes the required I/O transfer.
- The way that the processor chooses the branch address of the service routine varies from one unit to another.
- In principle, there are two methods for accomplishing this: *vectored interrupt* and *nonvectored interrupt*.

## ❑ Modes of I/O Data Transfer

- **PRIORITY INTERRUPT**
- A typical computer has a number of I/O devices attached to it that are able to originate an interrupt request.
- The first task of the interrupt system is to identify the source of the interrupt.
- There is also the possibility that several sources will request service simultaneously.
- In this case, the system must decide which device to service first.

## ❑ Modes of I/O Data Transfer

- A priority interrupt system establishes a priority over the various interrupt sources to determine which interrupt request to service first when two or more are pending simultaneously.
- Devices with high-speed transfers such as hard drives are given high priority, and slow devices such as keyboards receive the lowest priority.
- When two devices interrupt the computer at the same time, the computer services the device with the higher priority first.

## ❑ Modes of I/O Data Transfer

- **Daisy Chain Priority**
- The *daisy chain* method of establishing priority consists of a serial connection of all devices that request an interrupt.
- The device with the highest priority is placed in the first position, followed by devices of priority in descending order, down to the device with the lowest priority, which is placed last in the chain.
- This method of connection between three devices and the CPU is shown in diagram.

# ❑ Modes of I/O Data Transfer

## ❑ Modes of I/O Data Transfer

- **DIRECT MEMORY ACCESS**
- In *direct memory access* (DMA), the interface unit transfers data into and out of the memory unit through the memory bus.
- The CPU initiates the transfer by supplying the interface with the starting address and the number of words needing to be transferred and then proceeds to execute other tasks.
- When the transfer is made, the interface requests memory cycles through the memory bus.
- When the request is granted by the memory controller, the interface transfers the data directly into memory.

## ❑ Modes of I/O Data Transfer

- DMA may capture the buses in a number of ways. One common method extensively used in microprocessors is to disable the buses through special control signals.

## ❑ Modes of I/O Data Transfer

- Diagram shows two control signals in a CPU that facilitate the DMA transfer.
- The bus request (*BR*) input is used by the DMA controller to request the CPU to relinquish control of the buses.
- When *BR* input is active, the CPU places the address bus, the data bus, and the read and write lines into a high-impedance state.
- After this is done, the CPU activates the bus granted (*BG*) output to inform the external DMA that it can take control of the buses.

## ❑ Modes of I/O Data Transfer

- As long as the *BG* line is active, the CPU is unable to proceed with any operations requiring access to the buses.
- When the bus request input is reset by the DMA, the CPU returns to its normal operation, resets the *BG* output, and takes control of the buses.

## ❑ Modes of I/O Data Transfer

- **DMA Controller**
- The DMA controller needs the usual circuits of an interface to communicate with the CPU and the I/O device. In addition, it needs an address register, a word-count register, and a set of address lines.
- The address register and address lines are used for direct communication with memory.
- The word-count register specifies the number of words that must be transferred.
- The data transfer may be done directly between the device and memory under control of the DMA.

# ❑ Modes of I/O Data Transfer

## ❑ Modes of I/O Data Transfer

- The unit communicates with the CPU via the data bus and control lines.
- The registers in the DMA are selected by the CPU through the address bus by enabling the *DS* (DMA select) and *RS* (register select) inputs.
- The *RD* (read) and *WR* (write) inputs are bidirectional.
- When the *BG* (bus granted) input is 0, the CPU can communicate with the DMA registers through the data bus to read from or write to those registers.

## ❑ Modes of I/O Data Transfer

- When *BG* = 1, the CPU has relinquished the buses, and the DMA can communicate directly with memory by specifying an address on the address bus and activating the *RD* or *WR* control.
- The DMA communicates with the external peripheral through the DMA request and DMA acknowledge lines by a prescribed handshaking procedure.
- The DMA controller has three registers: an address register, a word-count register, and a control register. The address register contains an address to specify the desired location of a word in memory.

## ❑ Modes of I/O Data Transfer

- The address register is incremented after each word is transferred to memory. The word-count register holds the number of words to be transferred.
- This register is decremented by one after each word transfer and internally tested for zero. The control register specifies the mode of transfer.
- All registers in the DMA appear to the CPU as I/O interface registers.
- Thus, the CPU can read from or write to the DMA registers under program control via the data bus.

## ❑ Modes of I/O Data Transfer

- The CPU initializes the DMA by sending the following information through the data bus:
- **1.** The starting address of the memory block in which data is available (for reading) or data is to be stored (for writing).
- **2.** The word count, which is the number of words in the memory block.
- **3.** A control bit to specify the mode of transfer, such as read or write.
- **4.** A control bit to start the DMA transfer.

# ❑ Modes of Data Tran

- **DMA Transfer**

# Computer Organization and Architecture

## Memory Devices

Developed By:
Dr. Vivek Vyas
Department of MCA
DDU

## ❑ Introduction

- A memory is used to store data and instruction. Computer memory is the storage space in computer where data is to be processed and instructions required for processing are stored.
- The memory is divided into large number of small parts. Each part is called a cell. Each location or cell has a unique address which varies from zero to memory size minus one.
- For example if computer has 64k words, then this memory unit has 64 * 1024 = 65536 memory location. The address of these locations varies from 0 to 65535.

## ❑ Introduction

- Memory is primarily of two types
- **Internal Memory** – cache memory and primary/main memory
- **External Memory** – magnetic disk / optical disk etc

## ❑ Introduction

- **Characteristics of Memory Hierarchy are following when we go from top to bottom.**
- Capacity in terms of storage increases.
- Cost per bit of storage decreases.
- Frequency of access of the memory by the CPU decreases.
- Access time by the CPU increases.

## ❑ RAM

- A RAM constitutes the internal memory of the CPU for storing data, program and program result. It is read/write memory. It is called random access memory (RAM).
- Since access time in RAM is independent of the address to the word that is, each storage location inside the memory is as easy to reach as other location & takes the same amount of time.
- We can reach into the memory at random & extremely fast but can also be quite expensive.

## ❑ RAM

- RAM is volatile, i.e. data stored in it is lost when we switch off the computer or if there is a power failure.
- Hence, a backup uninterruptible power system (UPS) is often used with computers. RAM is small, both in terms of its physical size and in the amount of data it can hold.
- RAM is of two types
  - Static RAM (SRAM)
  - Dynamic RAM (DRAM)

## ❑ RAM

- **Static RAM (SRAM)**
- The word **static** indicates that the memory retains its contents as long as power remains applied. However, data is lost when the power gets down due to volatile nature.
- SRAM chips use a matrix of 6-transistors and no capacitors. Transistors do not require power to prevent leakage, so SRAM need not have to be refreshed on a regular basis.
- Because of the extra space in the matrix, SRAM uses more chips than DRAM for the same amount of storage space, thus making the manufacturing costs higher.
- Static RAM is used as cache memory needs to be very fast and small.

## ❑ RAM

- **Dynamic RAM (DRAM)**
- DRAM, unlike SRAM, must be continually **refreshed** in order for it to maintain the data.
- This is done by placing the memory on a refresh circuit that rewrites the data several hundred times per second.
- DRAM is used for most system memory because it is cheap and small. All DRAMs are made up of memory cells. These cells are composed of one capacitor and one transistor.

## ❑ ROM

- ROM stands for Read Only Memory. The memory from which we can only read but cannot write on it. This type of memory is non-volatile. The information is stored permanently in such memories during manufacture.
- A ROM, stores such instruction as are required to start computer when electricity is first turned on, this operation is referred to as bootstrap.

## ❑ ROM

- **MROM (Masked ROM)**
- The very first ROMs were hard-wired devices that contained a pre-programmed set of data or instructions. These kind of ROMs are known as masked ROMs. It is inexpensive ROM.
- **PROM (Programmable Read Only Memory)**
- PROM is read-only memory that can be modified only once by a user. The user buys a blank PROM and enters the desired contents using a PROM programmer.
- It can be programmed only once and is not erasable.

## ❑ ROM

- **EPROM (Erasable and Programmable Read Only Memory)**
- The EPROM can be erased by exposing it to ultra-violet light for a duration of upto 40 minutes.
- **EEPROM (Electrically Erasable and Programmable Read Only Memory)**
- The EEPROM is programmed and erased electrically. It can be erased and reprogrammed about ten thousand times.
- Both erasing and programming take about 4 to 10 ms (millisecond). In EEPROM, any location can be selectively erased and programmed.

## ❑ Serial Access Memory

- Sequential access means the system must search the storage device from the beginning of the memory address until it finds the required piece of data.
- Memory device which supports such access is called a Sequential Access Memory or Serial Access Memory. Magnetic tape is an example of serial access memory.

## ❑ Direct Access Memory

- Direct access memory or Random Access Memory, refers to conditions in which a system can go directly to the information that the user wants.
- Memory device which supports such access is called a Direct Access Memory. Magnetic disks, optical disks are examples of direct access memory.

## ❑ Cache Memory

- As CPU has to fetch instruction from main memory speed of CPU depending on fetching speed from main memory. CPU contains register which has fastest access but they are limited in number as well as costly.
- Cache is cheaper so we can access cache. Cache memory is a very high speed memory that is placed between the CPU and main memory, to operate at the speed of the CPU.
- It is used to reduce the average time to access data from the main memory. The cache is a smaller and faster memory which stores copies of the data from frequently used main memory locations.
- Most CPUs have different independent caches, including instruction and data.

# ❑ Cache Memory

# ❑ Cache Memory

- **Cache Performance**
- When the processor needs to read or write a location in main memory, it first checks for a corresponding entry in the cache.
  - If the processor finds that the memory location is in the cache, a **cache hit** has occurred and data is read from chache
  - If the processor **does not** find the memory location in the cache, a **cache miss** has occurred. For a cache miss, the cache allocates a new entry and copies in data from main memory, then the request is fulfilled from the contents of the cache.

## ❑ Cache Memory

- The performance of cache memory is frequently measured in terms of a quantity called Hit ratio.
  - Hit ratio = hit / (hit + miss) = no. of hits/total accesses

## ❑ Cache Mapping

- The three different types of mapping used for the purpose of cache memory are as follow,
  - Direct mapping
  - Associative mapping,
  - Set-Associative mapping.