Fundamentals of Microprocessor and Microcontroller Course Material (EC-223)

Prepared by:
Er. Vivek Harshey
Assistant Professor (ECE)

Department of Electronics and Communications Engineering
Sant Longowal Institute of Engineering and Technology
(Deemed to be University)
Longowal-148106, Sangrur, Punjab

CONTENTS

PREFACE

In the rapidly evolving field of electronics and embedded systems, a solid foundation in microprocessors and microcontrollers is essential for students in diploma and engineering disciplines. The 8085μp and 8051μc have been fundamental to studying digital electronics, providing students with a deep understanding of processor architecture, programming techniques, and interfacing concepts. Their simplicity and ease of implementation make them an ideal starting point for students stepping into microprocessor-based system design.

This course material and question bank have been designed as a learning resource, enabling students to grasp core concepts, strengthen problem-solving skills, and reinforce theoretical knowledge through structured questions. The document is intended to support students in preparing for academic exams, tests, and practical applications by covering the essential topics in a well-organized manner. This course material and question bank will function as a self-study guide, a reference document, and a revision tool for students. I sincerely appreciate feedback and suggestions for improvement, as continuous enhancement is key to better learning experiences.

Happy Learning!

Vivek Harshey
Asst Professor
ECE Deptt.

Syllabus

EC-223
Fundamentals of Microprocessor & Microcontroller

	L	T	P	Credits
	3	1	4	6
	Sessional Marks			50
	End Semester Examination Marks			50

Course Objectives:	The objective of the course is to expose the students to the evolution of microprocessors, the architecture and instruction set of typical 8-bit microprocessor 8085. It also deals with Assembly Language Programming and input-output techniques. Next focus is to introduce the architecture, programming and interfacing of 8051 microcontrollers.
Course Outcomes:	1. Understand the evolution of computers. 2. Analyze the architecture of the Intel 8085 microprocessor and 8051 microcontroller for its various applications. 3. Apply the programming techniques in designing simple assembly language programs for solving simple problems by using instruction sets of microprocessor and microcontroller. 4. Use the addressing modes and timing diagram for executing program efficiently.

Mapping of course outcomes with program outcomes

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO1		✓		✓	✓							
CO2		✓	✓	✓								
CO3		✓	✓	✓	✓							
CO4		✓	✓		✓							

Unit-I **14 hrs.**

Introduction: Typical organization of a microcomputer system and functions of its various blocks, Microprocessor, its evolution, function and its applications

Introduction to 8-bit Microprocessor Architecture: Concept of Bus, bus organization of 8085, functional block diagram of 8085, functions of each block of 8085 architecture, in details of 8085 and related signals.

Unit-Il **8hrs.**

Memories and I/O Interfacing: Memory organization, concept of memory mapping, partitioning of total memory space, address decoding, concept of I/O–mapped I/O and memory mapped I/O. Basic Concept of RAM, ROM, PROM, EPROM and EEPROM.

Unit-Ill **12hrs.**

Programming using 8085 Microprocessor: 8085 programming model, brief ideas of machine and assembly languages, machines and mnemonic codes, basic idea of instruction format and addressing modes, basic concept of instruction set for data transfer group, arithmetic group, logic group, stack, subroutine, I/O and machine control group, writing assembly language programs

Unit-IV **14hrs**.

Introduction: Difference between Microprocessor & Microcontroller, Concept of Embedded System.

Architecture of 8051 Microcontroller: Architecture of 8051, I/O ports in 8051, basic concept of memory in 8051, basic idea of addressing Modes in 8051, basic idea of instructions in 8051, applications of microcontroller.

BOOKS RECOMMENDED		
Title	**Author**	**Publisher**
l. Microprocessor Architecture- Programming & Applications with 8085/8080A	Ramesh S Gaonkar	5th Edition, Penram International Publishing
2. Introduction of Microprocessors & Microcomputers	Ram B	4th Edition, Dhanpat Rai Publisher (P) Ltd.
3. The 8051 Microcontroller	Kenneth J. Ayala	3rd Edition, Cengage Learning, 2004

Chapter 1. Introduction to Microprocessors

A microprocessor is a programmable electronic chip with computing and decision-making capabilities similar to a computer's central processing unit (CPU). A system built around a microprocessor with limited resources is referred to as a microcomputer. Today, microprocessors are embedded in nearly all electronic devices, including mobile phones, printers, washing machines, and home automation systems. Additionally, they play a critical role in advanced applications such as radar systems, satellites, and aerospace technology. The rapid advancement of the electronics industry and the development of large-scale integration (LSI) technology have led to significant cost reductions and increased adoption of microprocessors across various fields. A microprocessor is an integrated circuit (IC) that incorporates arithmetic, logic, and control circuitry, enabling it to interpret and execute instructions. As the core processing unit of a computing system, it facilitates the execution of computational and control tasks by coordinating with memory, input/output devices, and other peripherals. These interactions allow the microprocessor to execute operations as defined by user programs or system software, making it a fundamental component of modern digital and embedded systems.

Advancements in microprocessor technology have resulted in improved clock speeds, reduced power consumption, and increased integration of functionalities such as floating-point arithmetic, cache memory, and multiple cores. Modern microprocessors now support parallel processing and artificial intelligence capabilities, making them integral to various applications, from personal computing to industrial automation and embedded systems. One of the significant advantages of the 8085 microprocessor is its built-in serial communication capability, which facilitates data exchange between the processor and peripheral devices. It also includes five hardware interrupt signals and eight software interrupts, enabling efficient handling of external and internal events. Additionally, the 8085 supports both memory-mapped and I/O-mapped input/output operations, making it versatile for various interfacing applications.

The 8085 microprocessor is an 8-bit microprocessor developed by Intel in 1976. It has a 16-bit address bus, allowing it to address 64KB of memory. It operates on a +5V power supply and has a clock speed of 3 MHz. 8085 follows the von Neumann architecture, meaning both data and instructions share the same memory. It includes 74 instructions and supports five hardware and eight software interrupts.

1. Relevance of 8-bit Microprocessors in Academic Education

Despite the availability of 32-bit and 64-bit microprocessors, the 8-bit microprocessor remains a highly suitable choice for teaching microprocessor concepts, even in the twenty-first century. A significant reason for this is the widespread use of 8-bit microprocessors in industrial applications. Over 90% of microprocessor sales worldwide are accounted for by 8-bit processors and single-chip microcontrollers. These processors have firmly established themselves in industrial control systems, including machine control, process control, instrumentation, and consumer appliances. Such systems, incorporating microprocessors, are classified as embedded systems or microprocessor-based products.

In contrast, 32-bit and 64-bit microprocessors are primarily utilized in microcomputers, workstations, and high-performance computing environments. Their processing power is well-suited for applications such as high-speed data processing, CAD/CAM operations, multitasking, and multiuser systems. However, in industrial control applications, 8-bit microprocessors continue to dominate and are unlikely to be replaced by their 32-bit and 64-bit counterparts soon.

i Why High-End Processors Are Not Ideal for Teaching Basic Concepts?

Focusing on high-end Intel 32-bit and 64-bit processors for introductory microprocessor education is impractical. This situation can be compared to using large-scale integrated (LSI) devices to teach fundamental logic gate concepts such as AND, NAND, and OR. A simpler processor with a well-defined instruction set is required to grasp the fundamental principles of microprocessor architecture and programming.

The Intel high-end processors are too complex for introductory courses due to their intricate architecture and extensive instruction set. These processors are primarily designed to support high-level programming languages, handle large databases, and process graphics-intensive applications. Their primary use is in personal computers (PCs) and network servers, where advanced computing power is essential. As a result, for fundamental microprocessor education, a simpler, more structured processor such as the 8-bit microprocessor is a far better choice.

2. The Microprocessor as a Programmable Device

A microprocessor is a programmable device, meaning it can be instructed to perform specific tasks within its operational limits. Much like a piano, which produces different tones depending on the keys pressed, a microprocessor processes and executes binary instructions according to a predefined set of rules. A musician selects keys based on a musical score, while a programmer selects and sequences instructions to control the microprocessor. The microprocessor is a versatile computing unit, capable of executing both complex computational functions and simple control tasks, such as turning devices on or off. By programming appropriate instructions, users can direct the microprocessor to process data and perform designated operations efficiently.

The tasks a microprocessor can execute are determined by its architecture and instruction set, which are defined by hardware engineers during the design phase. Engineers develop the logic circuits necessary for executing specific operations and establish a predefined instruction set that the processor understands. For example, a basic instruction for adding two numbers may be represented as an 8-bit binary code, such as 1000 0000. These instructions, composed of 0s and 1s, form the fundamental commands that a programmer selects and sequences to execute a given task. The execution order of these instructions is stored in memory, allowing the microprocessor to retrieve, interpret, and process them as needed.

In a computing system, software consists of a set of instructions or commands that enable a programmable device to perform a specific task. Hardware, on the other hand, refers to the physical components of a system that execute and support software operations. While

hardware provides the foundation for computational processes, it remains inactive without software. Software acts as the bridge between the user and the machine, ensuring that hardware resources are utilized effectively. A computing system requires integration of both software and hardware to function efficiently.

i The microprocessor as a CPU (MPU)

The microprocessor serves as the core component of a computer system. Traditionally, computers are represented by a block diagram comprising four fundamental components: memory, input, output, and the central processing unit (CPU). The CPU includes key functional elements such as the arithmetic and logic unit (ALU), control unit, registers, instruction decoders, counters, and control lines. The CPU retrieves instructions from memory, processes them, and executes tasks accordingly. It interacts with input/output devices (peripherals) to either receive or transmit data. While the CPU is responsible for managing communication between different components, the control unit coordinates the timing and execution of these operations.

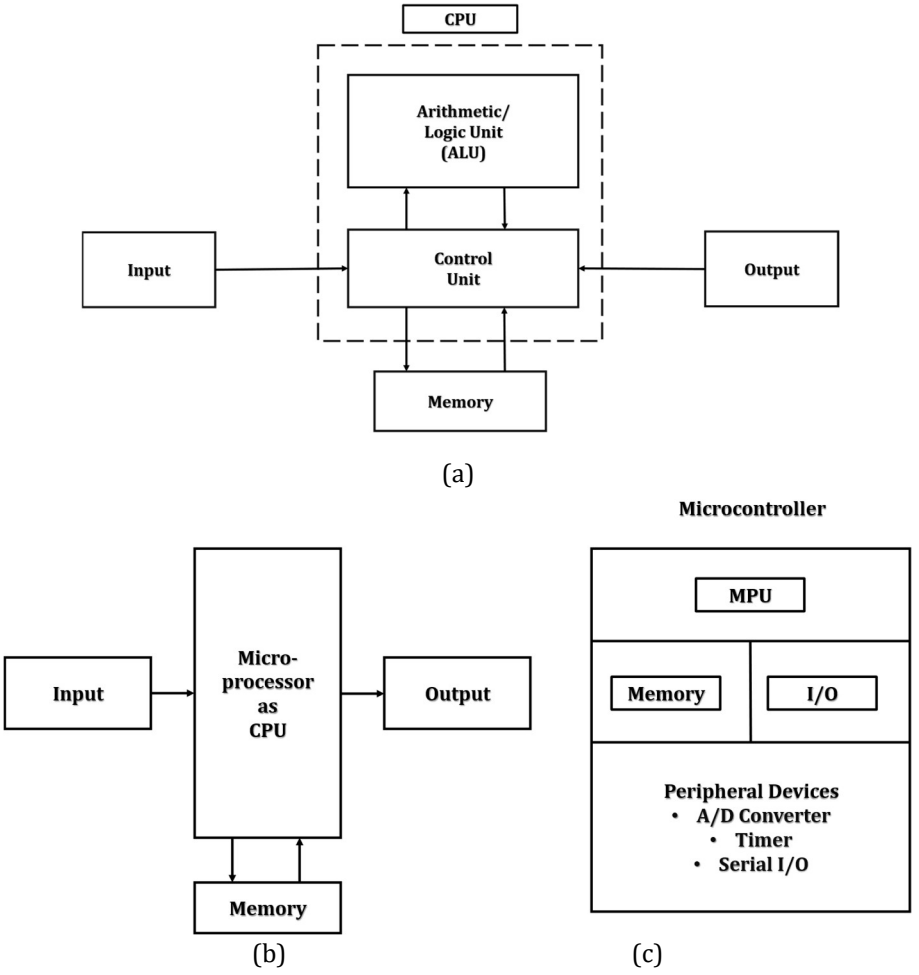

Figure 1. Block Diagram of Microcomputer.

During the late 1960s, CPUs were built using discrete components mounted on multiple circuit boards. The advent of integrated circuit (IC) technology revolutionized CPU design, allowing it to be integrated into a single chip, leading to the development of the microprocessor. This advancement replaced the traditional multi-board CPU architecture with a compact microprocessor-based system. A computer system utilizing a

microprocessor as its CPU is called a microcomputer. The terms microprocessor and microprocessor unit (MPU) are often used interchangeably. The MPU functions as a complete processing unit with built-in control signals. However, due to the limited number of pins on a microprocessor chip, some control and multiplexed signals must be generated using additional external circuits to create a fully functional MPU.

ii Evolution of Microprocessors

Microprocessors have undergone remarkable advancements since their inception. The first commercially available microprocessor, the Intel 4004, was introduced in 1971. It was a 4-bit processor primarily designed for calculators and other fundamental computing applications. This was soon followed by the Intel 8008, an 8-bit microprocessor, which expanded the scope of microprocessor applications beyond simple arithmetic processing. The introduction of the Intel 8080 and its enhanced version, the Intel 8085, represented a significant milestone in microprocessor development. These processors significantly improved computing capabilities, enabling microprocessors to be widely adopted in various computer systems. As technology progressed, the industry moved towards 16-bit, 32-bit, and 64-bit microprocessors, each offering greater processing power, enhanced memory addressing, and more efficient instruction execution.

iii Technological Advancements in Microprocessors

Continuous improvements in microprocessor technology have resulted in higher clock speeds, lower power consumption, and greater functional integration. Modern microprocessors incorporate floating-point arithmetic units, cache memory, and multiple processing cores, allowing more efficient and high-speed computation. The introduction of parallel processing, artificial intelligence acceleration, and advanced power management has further expanded the capabilities of microprocessors. Today, they play a crucial role in diverse fields, from personal computing and industrial automation to embedded systems and artificial intelligence applications. The evolution of microprocessors continues, driving innovation in computing and digital systems worldwide.

Table 1 Intel Microprocessors Historical Perspective

Processor	Year of Introduction	Number of Transistors	Initial Clock Speed	Address Bus	Data Bus	Addressable Memory
4004	1971	2,300	108 kHz	10-bit	4-bit	640 bytes
8008	1972	3,500	200 kHz	14-bit	8-bit	16 K
8080	1974	6,000	2 MHz	16-bit	8-bit	64 K
8085	1976	6,500	5 MHz	16-bit	8-bit	64 K
8086	1978	29,000	5 MHz	20-bit	16-bit	1 M
8088	1979	29,000	5 MHz	20-bit	8-bit	1 M
80286	1982	134,000	8 MHz	24-bit	16-bit	16 M

80386	1985	275,000	16 MHz	32-bit	32-bit	4 G
80486	1989	1.2 M	25 MHz	32-bit	32-bit	4 G
Pentium	1993	3.1 M	60 MHz	32-bit	32/64-bit	4 G
Pentium Pro	1995	5.5 M	150 MHz	36-bit	32/64-bit	64 G
Pentium II	1997	8.8 M	233 MHz	36-bit	64-bit	64 G
Pentium III	1999	9.5 M	650 MHz	36-bit	64-bit	64 G
Pentium 4	2000	42 M	1.4 GHz	36-bit	64-bit	64 G

3. Organization of a Microprocessor-Based System

A microprocessor-based system consists of essential components that work together to execute computing and control tasks efficiently. Figure 2 illustrates a simplified yet formal structure of such a system. Since a microcomputer is a type of microprocessor-based system, it follows the same fundamental architecture. The core components of this system include the microprocessor, input/output (I/O) devices, and memory (both read/write memory and read-only memory). These elements are interconnected through a shared communication channel known as the system bus, which facilitates the transfer of data, addresses, and control signals. The entire system, comprising these interconnected components, is referred to as a microcomputer system, with each individual component serving as a subsystem within it.

It is important to distinguish between the terms microprocessor and microcomputer, as they are often used interchangeably in popular literature. The microprocessor is a single component within a system, functioning as the central processing unit (CPU). It performs computations, logic operations, and control functions necessary for system execution. In contrast, a microcomputer is a complete computing system, integrating the microprocessor along with memory, input/output interfaces, and supporting circuits to form a fully operational computing unit.

Additionally, the term peripheral refers to input/output (I/O) devices connected to the microcomputer. These devices, such as keyboards, displays, storage units, and sensors, enable interaction between the user and the computing system. The organization of these components in a microprocessor-based system or microcomputer is depicted in Figure 2, providing a structural representation of how they interact to execute programmed instructions. Each component within this structure plays a crucial role in ensuring efficient data processing, system control, and external communication, making microprocessor-based systems essential in modern computing and automation applications.

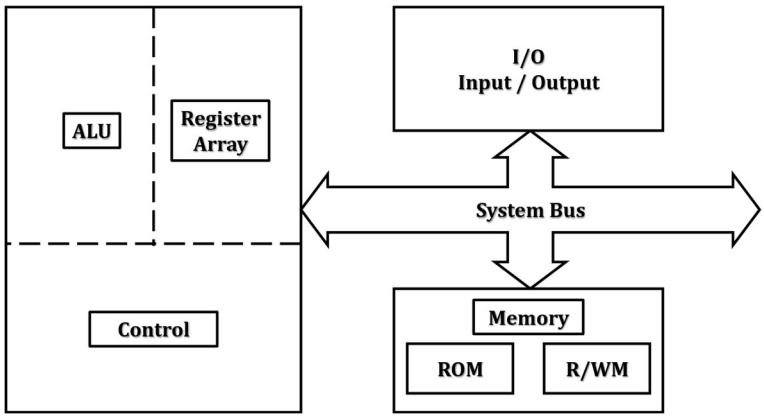

Figure 2 Microprocessor-Based System with Bus Architecture.

i Microprocessor Internal blocks

A microprocessor is a clock-driven semiconductor device composed of electronic logic circuits fabricated using Large-Scale Integration (LSI) or Very-Large-Scale Integration (VLSI) technology. It serves as the processing core of a system, performing various computing operations and executing instructions to make decisions that alter the sequence of program execution. In large computing systems, the CPU consists of multiple components spread across one or more circuit boards. However, in a microprocessor, all essential logic circuits, including the Control Unit (CU), Arithmetic Logic Unit (ALU), and Register Array, are integrated onto a single chip, making it a compact and efficient computing device. For better understanding, the internal architecture of a microprocessor can be divided into three main functional units, as depicted in Figure 2.

Arithmetic and Logic Unit (ALU): The Arithmetic and Logic Unit (ALU) is the computational core of the microprocessor. It executes arithmetic operations such as addition, subtraction, increment, and decrement and performs logical operations including AND, OR, XOR, and bitwise shifts. The ALU interacts with registers and memory to process data, and the results of operations affect specific status flags in the flag register, indicating conditions such as zero, carry, parity, and sign.

Register Array: The register array consists of multiple temporary storage locations, known as registers, that facilitate fast data access and manipulation. These registers are classified into general-purpose registers and special-purpose registers:

General-Purpose Registers (B, C, D, E, H, and L) – Used for storing intermediate results and operands during instruction execution.

Accumulator (A Register) – A dedicated register where most arithmetic and logical operations take place.

Flag Register – Holds status flags to indicate the outcome of ALU operations.

Program Counter (PC) – Keeps track of the next instruction to be executed.

Stack Pointer (SP) – Points to the top of the stack, used for temporary data storage and function calls.

Control Unit (CU): The Control Unit (CU) acts as the brain of the microprocessor, generating timing and control signals to coordinate all system operations. It ensures

proper data flow between the ALU, register array, memory, and peripheral devices. The control unit manages instruction decoding, execution sequencing, and communication with external components through control signals such as Read ($R\bar{D}$), Write ($W\bar{R}$), and Address Latch Enable (ALE).

ii Memory

Memory is an essential component of a microprocessor-based system, responsible for storing binary information, including instructions and data, and supplying it to the microprocessor when needed. The microprocessor retrieves instructions and data from memory to execute programs and performs the necessary computing operations in its Arithmetic and Logic Unit (ALU). The results of these operations are either sent to the output devices for display or stored back in memory for future use. As shown in Figure 3, the memory block is typically divided into two main sections:

1. Read-Only Memory (ROM): Read-Only Memory (ROM) is used for storing programs that do not require modifications. Programs stored in ROM are permanent and can only be read, not altered. A common example is the monitor program in a single-board microcomputer, which interprets user inputs from a keyboard and converts them into binary instructions for the microprocessor. Since ROM retains its contents even when power is turned off, it is classified as non-volatile memory.

2. Random-Access Memory (RAM): Read/Write Memory (R/WM), commonly referred to as Random-Access Memory (RAM), is used for storing user programs and data during execution. Unlike ROM, the contents of RAM can be read and modified. This memory is often called user memory because it temporarily holds instructions and data entered by the user. In single-board microcomputers, a monitor program tracks inputs from a hexadecimal keypad and stores the corresponding instructions and data in RAM. Since RAM is volatile memory, its contents are lost when the power supply is turned off.

iii Input/Output devices

The Input/Output (I/O) subsystem is the third essential component of a microprocessor-based system, responsible for enabling communication between the microprocessor and the external world. I/O devices, also known as peripherals, serve as an interface between the user and the system, allowing data and instructions to be transferred to and from the microprocessor.

Input Devices: Input devices allow the transfer of binary information, such as data and instructions, from external sources into the microprocessor. Common input devices include:

Keyboards – Used to enter data and commands.

Switches – Simple input mechanisms that provide binary signals to the microprocessor.

Analog-to-Digital (A/D) Converters – Convert analog signals (e.g., temperature, pressure) into digital form for processing.

In laboratory microcomputers, input is typically provided through a hexadecimal (Hex) keyboard or an ASCII keyboard:

Hexadecimal Keyboard – A 16-key keypad (digits 0-9 and A-F) with additional function keys for storing data and executing programs.

ASCII Keyboard – Similar to a typewriter keyboard, used for entering programs in an English-like language. While ASCII keyboards are common in personal computers (PCs), single-board microcomputers often use Hex keyboards. Microprocessor-based consumer products, such as microwave ovens, frequently feature decimal keypads for user input.

Output Devices: Output devices transfer processed data from the microprocessor to the external world. These devices include:

Light-Emitting Diodes (LEDs) – Used for basic binary indication and status displays.

Seven-Segment Displays – Display numerical values and simple characters.

Cathode-Ray Tube (CRT) or Video Screen – Found in traditional computing systems for visual output.

Printers – Used to produce hard copies of processed data.

Digital-to-Analog (D/A) Converters – Convert digital signals from the microprocessor into analog signals for real-world applications.

iv System Bus

The system bus is a communication path between the microprocessor and peripherals; it is nothing but a group of wires that carry bits. Several buses in the system will be discussed in the next chapter. All peripherals (and memory) share the same bus; however, the microprocessor communicates with only one peripheral at a time. The control unit of the microprocessor provides timing.

4. Microcomputer Organization:

A microprocessor combined with memory and input/output devices forms a microcomputer. These components are shown in the figure below:

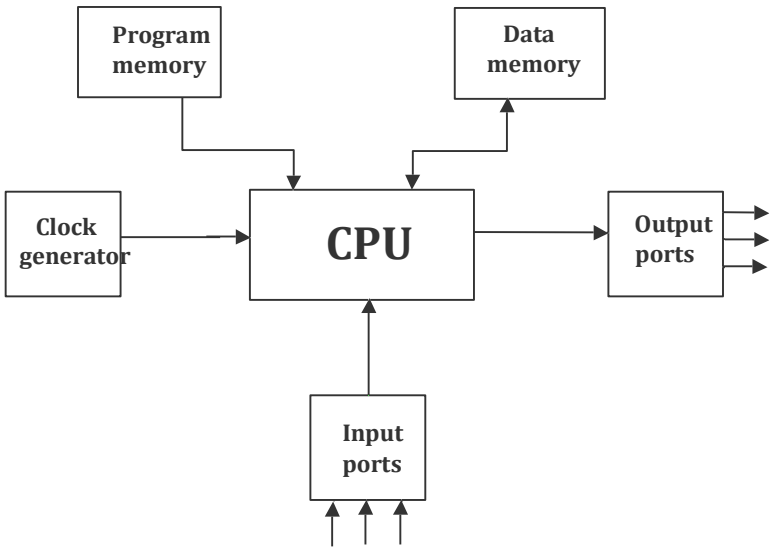

Figure 3 Basic Components of Microcomputer

Central Processing Unit (CPU) and Program Execution in a Microprocessor-Based System

The Central Processing Unit (CPU) is the core of a microprocessor-based system and is responsible for executing instructions and processing data. It consists of three major components: Arithmetic and Logic Unit (ALU): Performs arithmetic operations such as addition, subtraction, multiplication, and division, as well as logical operations like AND, OR, XOR, and NOT. Register Unit: Contains various temporary storage locations (registers) used for storing data, intermediate results, and special-purpose information such as the program counter (PC), stack pointer (SP), and flag registers.

Control Unit (CU): Generates necessary timing and control signals to coordinate operations between the CPU, memory, and peripheral devices. The CPU fetches instructions from program memory (ROM or RAM), retrieves data from data memory or an input device, processes the data using the ALU, and then stores the result in memory or sends it to an output device.

System Boot-Up and Reset Mechanism: When power is turned on, the monitor program stored in EPROM or ROM initializes the system. The Reset key is used to clear the program counter (PC), which then holds the memory address 0000H. Some systems feature automatic power-on reset, which ensures the system starts from a predefined state every time it is powered on. Upon resetting, the program counter (PC) places address 0000H on the address bus, and the instruction stored at that location is fetched and executed. This marks the beginning of the Key Monitor program, which is usually stored on page 00H of memory.

Low-Level Languages

Low-level languages are programming languages that are closer to machine language and provide direct hardware control.

Machine Language: Uses binary code (0s and 1s) directly understood by the microprocessor. Difficult to write, read, and debug. Example: 10110000 01100001 (Binary representation of MOV A, 61H in 8085).

Assembly Language: Uses mnemonics instead of binary code. It is easier to understand than machine language but still hardware dependent. Requires an assembler for conversion to machine code.

High-Level Languages

High-level languages (HLLs) are programming languages that use English-like syntax and abstraction, making them easier for humans to understand. They are hardware-independent and require a compiler or interpreter to convert them into machine code. Examples: C, Python, Java, FORTRAN

Advantages: Easy to learn and use. Portable across different hardware architectures. Requires less knowledge of hardware details.

int a = 5, b = 10;

int sum = a + b;

This code is much easier to understand than low-level assembly instructions.

Table 2 Comparison of languages

Feature	**Machine Language**	**Assembly Language**	**High-Level Language**
Readability	Difficult (binary)	Moderate (mnemonics)	Easy (English-like)
Execution Speed	Fastest	Faster	Slower (due to compilation)
Hardware Dependence	Highly dependent	Dependent	Independent
Ease of Debugging	Difficult	Moderate	Easy

Steps in writing and executing an assembly language program

Step 1: Writing Instructions in Mnemonics

The assembly language program is written using mnemonics obtained from the instruction set of the microprocessor. Mnemonics are human-readable representations of machine instructions.

Step 2: Finding Hexadecimal Machine Code

Each mnemonic corresponds to a hexadecimal opcode in the microprocessor's instruction set. The assembler or manual lookup can be used to convert mnemonics into machine code.

Step 3: Entering the Program into Memory

The program is loaded into the user memory sequentially. A Hexadecimal Keyboard (or a system loader) is used to input the machine code into the microprocessor's memory.

Step 4: Executing the Program

The program is executed by pressing the Execute key or using a command in an emulator/software tool. The microprocessor fetches, decodes, and executes each instruction.

The result is stored in a register or displayed on LEDs/LCD (if interfaced with the microprocessor system).

i ALU (Arithmetic and Logic Unit)

This unit performs computing functions on m-bit data where ‘m’ is the bit size of the processor. These functions are arithmetic operations such as addition, subtraction and logical operation such as AND, OR, XOR, rotate, compare etc. Results are stored either in registers or in memory or sent to output devices.

ii Register Unit:

It contains various 8-bit or 16-bit registers. These registers are used primarily to store data temporarily during the execution of a program. Some of the registers are accessible to the user through instructions. It means their contents can be read and/or changed through instructions. Some of the registers are not accessible to user but they are used by the processor for the execution of an instruction. 8085A microprocessor contains 8-bit registers such as Accumulator (Reg. A), B, C, D, E, H, L etc. and 16-bit registers such as Program Counter (PC), Stack Pointer (SP).

The Program Counter (PC) is a 16-bit register that holds the address of the next instruction to be executed. The PC fetches the address of the instruction from memory. After fetching, it automatically increments to point to the next instruction. In the case of jump or call instructions, the PC is modified to point to a new address. Controls the sequence of execution in a program.

The Stack Pointer (SP) is a 16-bit register that holds the address of the top of the stack in memory. The stack is used for temporary storage of data, return addresses, and registers. The SP is decremented (decreases) when data is pushed (stored) onto the stack. The SP is incremented (increases) when data is popped (retrieved) from the stack. It works in LIFO (Last In, First Out) order. The SP decreases by 2 during PUSH instruction and increases by 2 during POP instruction.

iii Timing and Control Unit:

It provides necessary timing & control signals required for the operation of microcomputer. It controls the flow of data between the microprocessor and peripherals (input, output & memory). The control unit gets a clock signal which determines the speed of the microprocessor. In all, the CPU has the following basic functions:

5. Memory

Memory plays a crucial role in a microprocessor-based system, as it stores both program instructions and data needed for execution. It consists of two main types: Read-Only Memory (ROM) and Read/Write Memory (RWM), both of which are considered Random Access Memory (RAM) since data can be accessed from any location directly.

1. Read-Only Memory (ROM): ROM (Read-Only Memory) is a non-volatile memory, meaning its contents remain intact even when the power is turned off. Since ROM is read-only, data stored in it cannot be modified after manufacturing (except in the case of programmable ROMs like EPROM or EEPROM). ROM is typically used for storing fixed programs that do not change during system operation. One common use of ROM is in microcomputers, where it stores the monitor program—a set of instructions that initializes the system and handles user interactions. This ensures that the processor starts executing from a predetermined location when the system powers up or is reset.

2. Read/Write Memory: RWM (Read/Write Memory), often referred to as RAM (Random Access Memory), is a volatile memory that loses its contents when power is turned off.

Unlike ROM, RWM allows both reading and writing of data, making it essential for storing user programs, temporary data, and intermediate results during program execution.

During a memory read operation, the microprocessor retrieves data from a specific location without altering its contents. During a memory write operation, new data is stored in a specified location, overwriting the previous contents.

i Program Memory:

The program memory is responsible for storing the sequence of instructions that the CPU executes. When a microcomputer system starts up, either on power-up or after a reset, the processor fetches and executes instructions from a predetermined location in program memory. Since the program remains fixed and does not change during execution, it is typically stored in ROM. The first instruction of the program must be placed at the processor's predefined reset address to ensure proper execution.

ii Data Memory:

Data memory is used for storing variables, intermediate results, and temporary data needed for program execution. It enables the microprocessor to manipulate data according to the algorithm provided in the program instructions. Internal Registers: Microprocessors have small internal memory in the form of registers, which store frequently used data and improve processing speed. External Data Memory: If the storage requirement exceeds the capacity of internal registers, the system uses external RAM for additional data storage.

6. Microcomputer Bus System

A microcomputer consists of three essential buses that facilitate the transfer of address, data, and control signals required for program execution. These buses serve as communication pathways that connect the microprocessor, memory, and input/output (I/O) devices, enabling seamless data exchange and coordinated operations. The efficiency of a microcomputer heavily depends on the organization and performance of these buses, as they determine how quickly and effectively instructions and data are transmitted between system components.

i Address Bus

In a microcomputer system, the central processing unit (CPU) is the core component responsible for controlling system operations. When executing a program, the CPU determines which device (memory or an I/O peripheral) should participate in a data transfer. This selection is accomplished by placing the specific address of the target device onto the address bus. The address bus is unidirectional, meaning that the flow of address information occurs only from the microprocessor to memory or I/O devices. The CPU transmits an address to identify a specific memory location or an I/O device before initiating a data transfer. In the case of memory access, the address bus not only identifies the memory device but also specifies the exact memory location within it.

The size of the address bus determines the number of addressable memory locations. For instance, an 8-bit address bus can address 256 (2^8) memory locations, whereas a 16-bit address bus, as seen in the 8085 microprocessor, can address 64 KB (2^{16}) of memory space.

As microprocessors evolve, they incorporate wider address buses, enabling access to larger memory capacities and more complex I/O devices. In modern computing systems, memory management techniques, such as paging and segmentation, allow efficient utilization of the address bus, enabling microprocessors to address vast amounts of memory beyond their physical addressing limit. Additionally, the introduction of multiplexed address buses in some microprocessors optimizes pin usage, further enhancing the design of compact and efficient computing devices.

ii Data Bus in Microcomputer Systems

The data bus is a set of lines used to transfer data between the microprocessor, memory, and peripheral devices. In the 8085A microprocessor, the data bus consists of 8 lines (D7-D0), which allow the processor to handle 8-bit data transfers at a time. The data bus is a shared resource, meaning that multiple devices can connect to it. However, to avoid conflicts, only one device should transmit data at any given time, while all other devices must remain in a high-impedance (high-Z) state to prevent electrical interference.

Unlike the address bus, which is unidirectional, the data bus is bidirectional, meaning data can flow both to and from the microprocessor. This bidirectional nature allows the processor to read data from memory or I/O devices and write data back when required. Therefore, this is called bidirectional data bus (BDB). In some microprocessors, the data pins are also used to send other information such as address bits in addition to data. This means that the data pins are time shared or multiplexed. In Intel 8085A microprocessor lower 8-bits of the address (A7-A0) are time-multiplexed with the 8-bit data (D7-D0) and, therefore, this bus is called AD bus (AD7-AD0).

In some microprocessors, data pins serve multiple purposes, such as transmitting both address and data signals. This technique is known as multiplexing and helps optimize the number of available pins on the microprocessor. For example, in the Intel 8085A microprocessor, the lower 8 bits of the address bus (A7-A0) are multiplexed with the 8-bit data bus (D7-D0), forming the AD bus (AD7-AD0). During the first phase of an operation, these lines carry the address, and in the second phase, they are used for data transfer. To separate address and data signals, external latching circuits are employed, such as the 74LS373 latch.

iii Control Bus

The control bus consists of a set of dedicated control signals that coordinate operations between the microprocessor, memory, and I/O devices. These signals synchronize data transfers, ensuring that all system components operate in a coordinated manner. Every operation performed by the microprocessor, such as reading from memory, writing to memory, or communicating with I/O devices, is controlled through signals transmitted via the control bus. Some of the signals of the control bus are issued by the processor and some of the signals are received by the processor. Therefore, the control bus is called bidirectional control bus (BCB). The difference between BDB and BCB is that in BDB all data lines are either in input mode or in output mode whereas in BCB the direction of signal flow on a line is fixed.

Common control signals in the 8085 microprocessor include:

$R\bar{D}$ (Read signal): Indicates that the microprocessor is reading data from memory or an I/O device.

$W\bar{R}$ (Write signal): Indicates that the microprocessor is writing data to memory or an I/O device.

$IO/\bar{M}$ (Memory or I/O selection): Differentiates between memory ($IO/\bar{M}$ = 0) and I/O operations ($IO/\bar{M}$ = 1).

The control bus is bidirectional, meaning that some signals are generated by the processor, while others are received from external devices. Unlike the bidirectional data bus, where all lines function as either inputs or outputs at a given time, the control bus features fixed-direction signals, ensuring consistent communication between system components. As microprocessor technology has advanced, additional control signals have been introduced to manage interrupt handling, direct memory access (DMA), and power-saving features, enhancing the efficiency of modern computing systems.

Chapter 2. 8085 Architecture and Pin Diagram

The 8085 microprocessor, developed by Intel in 1976, is an 8-bit microprocessor designed for general-purpose computing and embedded applications. It follows the Von Neumann architecture, meaning that both data and instructions share the same memory space. The processor operates on a +5V power supply and runs at a clock speed of 3 MHz with a 16-bit address bus, it can access 64 KB of memory, while its 8-bit data bus allows processing of 8-bit data at a time.

The 8085 architecture is divided into several functional units, including the Arithmetic and Logic Unit (ALU), Register Array, Control Unit, Address and Data Bus, and Interrupt System. These components work together to execute instructions efficiently.

1. Internal Architecture of 8085

The internal structure of the 8085 microprocessor consists of the following key units:

i Arithmetic and Logic Unit (ALU)

The Arithmetic and Logic Unit (ALU) is responsible for performing arithmetic operations such as addition, subtraction, and logical operations such as AND, OR, and XOR. The ALU interacts with the accumulator and other registers to execute these operations efficiently. It also affects the status flags stored in the flag register.

ii Register Array

The 8085 microprocessor contains a set of registers that store data temporarily during execution. These include six general-purpose registers (B, C, D, E, H, and L), an accumulator (A), a program counter (PC), and a stack pointer (SP). The registers can be used individually or in pairs (BC, DE, HL) for 16-bit operations. The accumulator is an essential register used for arithmetic and logical operations.

Different Types of Registers in 8085

The 8085 microprocessor has several registers used for data storage, processing, and control operations. These registers can be classified based on their size, function, and accessibility.

1. *Classification Based on Size*

8-bit Registers: Accumulator (A), General-purpose registers (B, C, D, E, H, L), Flag register

16-bit Registers: Program Counter (PC), Stack Pointer (SP)

2. *Classification Based on Functionality*

General-Purpose Registers

Registers B, C, D, E, H, and L are used for temporary data storage and arithmetic operations.

They can be used as register pairs (BC, DE, HL) for 16-bit operations.

Special-Purpose Registers

Accumulator (A): An 8-bit register used for arithmetic and logic operations. It stores the final result of computations.

Flag Register: Stores the status of operations with five flags – Sign (S), Zero (Z), Auxiliary Carry (AC), Parity (P), and Carry (CY).

Control and Addressing Registers

Program Counter (PC): A 16-bit register that holds the memory address of the next instruction to be executed. It is automatically incremented after each instruction fetch.

Stack Pointer (SP): A 16-bit register that keeps track of the top of the stack during subroutine calls and interrupts.

3. *Classification Based on User Accessibility*

User-Accessible Registers: Accumulator (A), General-purpose registers (B, C, D, E, H, L)

System-Controlled Registers: Program Counter (PC), Stack Pointer (SP), Flag Register

iii Control Unit

The control unit is responsible for generating control signals to manage data flow between the microprocessor, memory, and peripherals. It interprets instructions fetched from memory and coordinates the operations of the ALU and registers. Control signals include RD̄ (Read), WR̄ (Write), IO/M̄ (Memory or I/O selection), and ALE (Address Latch Enable).

2. Pin Diagram and Functions of Various Pins

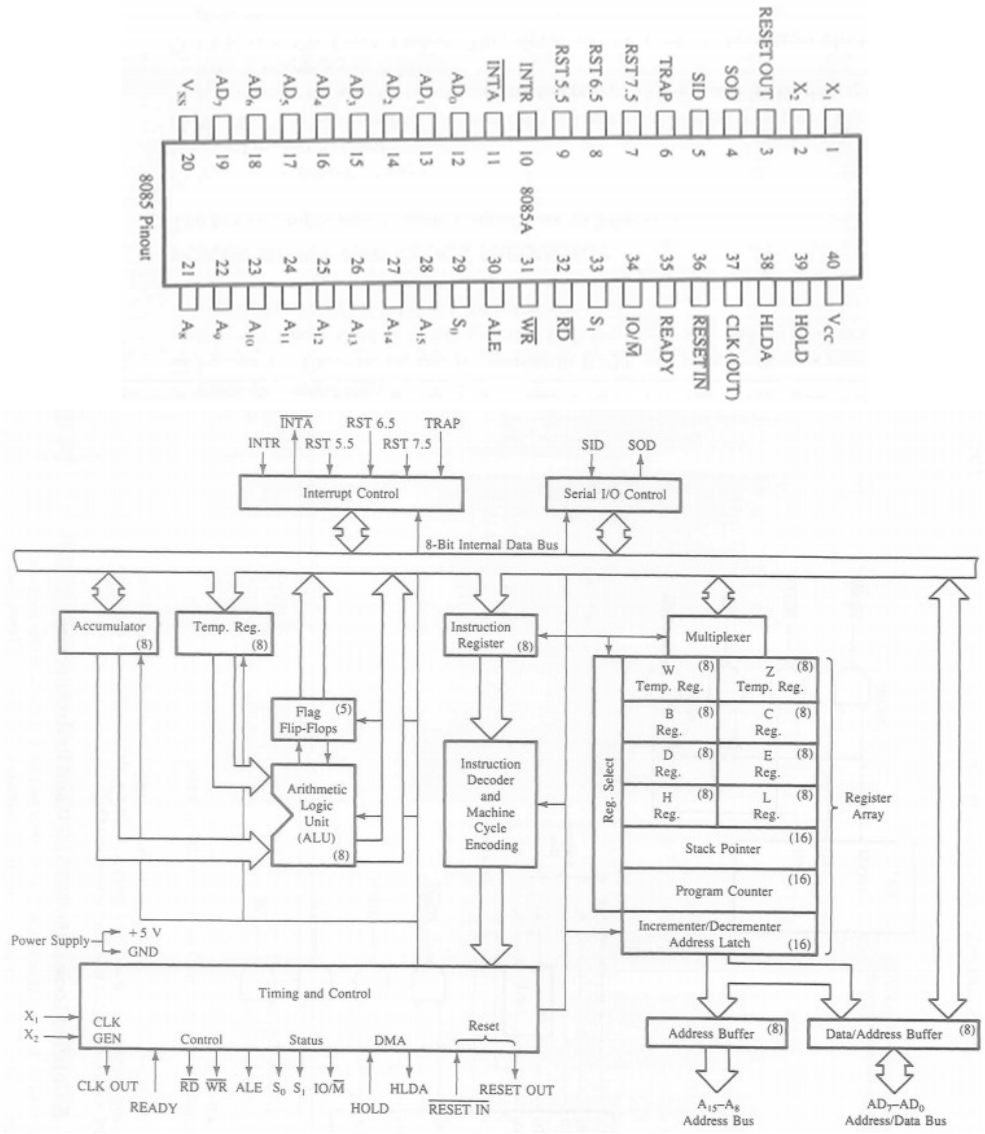

Figure 4 Pin diagram and functional block diagram.

The 8085 microprocessor has a 40-pin dual in-line package (DIP) configuration. The pins are categorized into different groups based on their functions, such as power supply, clock signals, control signals, data and address buses, and serial I/O.

i Power Supply and Clock Signals

• VCC (Pin 40): +5V power supply.
• GND (Pin 20): Ground.
• X1, X2 (Pins 1, 2): Crystal oscillator connections for clock generation.
• CLK OUT (Pin 37): Provides clock signals to external peripherals.

8085 has a clock generation circuit on the chip but an external quartz crystal or L C circuit or RC circuit should be connected at pins X1 and X2. The maximum internal clock frequency of 8085 is 3.07 MHz

ii Address and Data Buses

• Address Bus (A15–A8): The higher-order address lines (A15–A8) are used for memory addressing.
• Multiplexed Address/Data Bus (AD7–AD0): The lower-order address and data lines are multiplexed to save pins. During the address phase, these lines carry address bits, and during the data phase, they carry data.

iii Control and Status Signals

• $\overline{RD}$ (Pin 32): Read signal, active low, used to read data from memory or I/O.
• $\overline{WR}$ (Pin 31): Write signal, active low, used to write data to memory or I/O.
• IO/$\overline{M}$ (Pin 34): Distinguishes between memory (0) and I/O (1) operations.
• ALE (Pin 30): Address Latch Enable, used to demultiplex address/data lines.
• READY (Pin 35): Indicates whether the peripheral is ready for data transfer.

Pins in the Timing and Control Unit of the 8085 Microprocessor

The 8085 microprocessor has several pins associated with the timing and control unit that manage data flow, memory access, and I/O operations. These pins are crucial for coordinating the execution of instructions.

Table 3 Key Pins and Their Functionality

Pin	Type	Function
CLK (Clock Out)	Output	Provides system clock to synchronize external devices.
$\overline{RD}$ (Read Control Signal)	Output	Indicates that data is being read from memory or I/O device.
$\overline{WR}$ (Write Control Signal)	Output	Indicates that data is being written to memory or an I/O device.
ALE (Address Latch Enable)	Output	Used to separate the lower byte of address from the multiplexed address/data bus (AD0-AD7).
IO/$\overline{M}$ (Input/Output or Memory Select)	Output	Distinguishes between memory (0) and I/O (1) operations.

Pin	Type	Function
S_0 and S_1 (Status Signals)	Output	Indicate the operation type (Fetch, Memory Read, Memory Write, I/O Read, I/O Write, or Halt).
READY	Input	Ensures synchronization with slow memory or I/O devices by pausing the microprocessor until the device is ready.
$\overline{\text{RESET IN}}$	Input	When activated, resets the microprocessor and restarts execution from address 0000H.
RESET OUT	Output	Used to reset external devices upon system reset.

The timing and control unit in the 8085 microprocessor plays a critical role in data flow, memory access, and I/O operations. These control signals help in synchronizing the processor with external memory, I/O devices, and other components.

iv Interrupts and Serial I/O

• TRAP, RST7.5, RST6.5, RST5.5, INTR: Hardware interrupts.
• Serial Input/Output: SID (Serial Input Data) and SOD (Serial Output Data) for serial communication.

Interrupt Signals in 8085 Microprocessor

The 8085 microprocessor has five interrupt signals that allow external devices to interrupt the normal execution of a program. These interrupts help in handling urgent tasks, external events, and I/O operations.

Table 4 List of Interrupt Signals in 8085

Interrupt	Type	Priority	Vector Address	Maskable	Triggering Method
TRAP	Non-maskable	Highest	0024H	No	Edge and Level Triggered
RST 7.5	Maskable	Second	003CH	Yes	Edge Triggered
RST 6.5	Maskable	Third	0034H	Yes	Level Triggered
RST 5.5	Maskable	Fourth	002CH	Yes	Level Triggered
INTR	Maskable	Lowest	Any memory location (needs external hardware)	Yes	Level Triggered

Interrupt Priority Order: TRAP > RST 7.5 > RST 6.5 > RST 5.5 > INTR

The Software interrupts are program instructions. These instructions are inserted at desired locations in a program. While running a program, if software interrupt instruction is encountered then the processor executes an interrupt service routine. If an interrupt is

initiated in a processor by an appropriate signal at the interrupt pin, then the interrupt is called Hardware interrupt. The Software interrupt is initiated by the main program, but the Hardware interrupt is initiated by an external device. In 8085, the Software interrupt cannot be disabled or masked but the Hardware interrupt except TRAP can be disabled or masked. When an interrupt is accepted, if the processor control branches to a specific address defined by the manufacturer, then the interrupt is called vectored interrupt. In non-vectored interrupt there is no specific address for storing the interrupt service routine. Hence the interrupted device should give the address of the interrupt service routine.

Masking is preventing the interrupt from disturbing the current program execution. When the processor is performing an important job (process) and if the process should not be interrupted then all the interrupts should be masked or disabled. In processor with multiple interrupts, the lower priority interrupt can be masked to prevent it from interrupting, the execution of interrupt service routine of higher priority interrupt. The processor keeps on checking the interrupt pins at the second T -state of the last Machine cycle of every instruction. If the processor finds a valid interrupt signal and if the interrupt is unmasked and enabled, then the processor accepts the interrupt. The acceptance of the interrupt is acknowledged by sending an OOA signal to the interrupted device. The interrupts of 8085 except TRAP are disabled after anyone of the following operations—

1. Executing Dl instruction.

2. System or processor reset.

3. After reorganization (acceptance) of an interrupt.

For the interrupt INTR, the interrupting device has to place either an RST opcode or CALL opcode followed by l6-bit address. Instruction RST opcode is placed then the corresponding vector address is generated by the processor. In the case of CALL Opcode the given l6-bit address will be the vector address.

v Generating Control Signals

Figure below shows the $\overline{\text{RD}}$ (Read) as a control signal. Because this signal is used both for reading memory and for reading an input device, it is necessary to generate two different Read signals: one for memory and another for input. Similarly, two separate Write signals must be generated.

The figure below shows that four different control signals are generated by combining the signals $\overline{RD}$, $\overline{WR}$, and IO/$\bar{\text{M}}$. The signal IO/$\bar{\text{M}}$ goes low for the memory operation. When both input signals go low, the outputs of the gates go low and generate MEMR (Memory Read) and MEMW (Memory Write) control signals. When the $IO/\bar{M}$ signal goes high, it indicates the peripheral I/O operation.

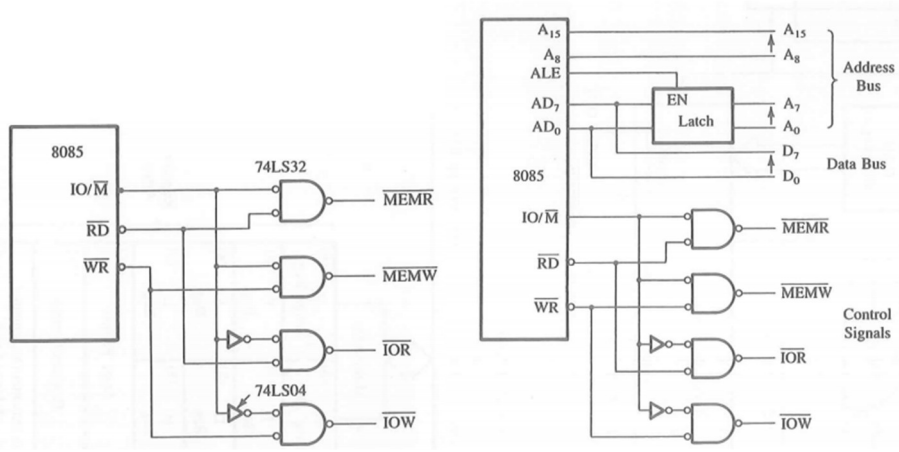

Figure 5**:** Generation of control signal for interfacing.

3. System Bus (Address Bus, Data Bus, and Control Bus)

The 8085 microprocessor communicates with memory and peripheral devices using the system bus, which consists of three main components: the Address Bus, Data Bus, and Control Bus.

i Address Bus

The address bus is used to specify the memory location or I/O device to be accessed. It is 16-bit wide, allowing the microprocessor to address up to 64 KB of memory (2^{16} locations). The address bus is unidirectional, meaning data flows only from the microprocessor to memory.

ii Data Bus

The data bus is used to transfer data between the microprocessor, memory, and I/O devices. It is 8-bit wide, meaning the microprocessor can process 8-bit data at a time. The data bus is bidirectional, allowing data to flow in both directions.

iii Control Bus

The control bus consists of signals used to control data transfer. These signals include R$\bar{D}$ (Read), W$\bar{R}$ (Write), IO/$\bar{M}$ (Memory/I/O selection), and ALE (Address Latch Enable). The control bus ensures proper synchronization between the microprocessor and external devices.

Additionally, many peripheral device operations require direct hardware interaction, which is often not supported by high-level languages. For example, operations involving serial communication, direct memory access (DMA), and device-specific control registers are typically implemented using assembly language. As a result, even in modern computing, assembly language remains an essential tool for system programming, embedded applications, and performance-critical tasks.

4. Data formats in computer

Data in digital systems is represented in different formats based on the application and processing requirements. The commonly used data formats are ASCII code, Extended ASCII, BCD code, Signed Integer, and Unsigned Integer.

1. ASCII Code (American Standard Code for Information Interchange)

ASCII (7-bit) is a standard character encoding format used to represent text on computers. It uses 7 bits to represent 128 characters (0-127), including letters, digits, punctuation, and control characters.

'A' = 65 (Decimal) = 1000001 (Binary)

'a' = 97 (Decimal) = 1100001 (Binary)

'1' = 49 (Decimal) = 0110001 (Binary)

2. Extended ASCII Code (8-bit ASCII)

Extended ASCII (8-bit) extends the standard ASCII set to 256 characters (0-255). It includes special characters, graphical symbols, and accented letters.

'Ç' = 128 (Extended ASCII)

'é' = 130 (Extended ASCII)

3. BCD (Binary-Coded Decimal)

BCD represents decimal numbers (0-9) in 4-bit binary form. Each decimal digit is converted separately into 4-bit binary.

Decimal 25 → BCD: 0010 0101

Decimal 93 → BCD: 1001 0011

4. Signed Integer Representation

Signed integers represent both positive and negative numbers. The most common method is Two's Complement Representation, where:

MSB (Most Significant Bit) is the sign bit (0 = Positive, 1 = Negative).

Positive Numbers → Stored in normal binary.

Negative Numbers → Stored in two's complement form.

+5 = 00000101, -5 = 11111011 (Two's Complement of 00000101)

Range for 8-bit Signed Integer: -128 to +127

5. Unsigned Integer Representation

Unsigned integers represent only positive numbers, using all available bits for magnitude.

Range for 8-bit Unsigned Integer: 0 to 255

Table 5 Different data format and their properties

Data Format	**Bit Size**	**Range/Usage**	**Example (Binary Representation)**

ASCII	7-bit	128 characters	'A' = 1000001, 'a' = 1100001
Extended ASCII	8-bit	256 characters	'Ç' = 10000000
BCD	4-bit per digit	0-9 per digit	25 → 0010 0101
Signed Integer (8-bit)	8-bit	-128 to +127	+5 = 00000101, -5 = 11111011
Unsigned Integer (8-bit)	8-bit	0 to 255	5 = 00000101, 255 = 11111111

5. Hardware Components in a Microcomputer-Based System

In addition to the 8085 microprocessor, a microcomputer-based system requires several supporting hardware components to ensure proper functionality and communication between system elements. These components include tri-state devices, buffers, latches, and decoders, which help in efficient data transfer and control.

Tri-state devices are essential in systems where multiple components share a common data bus. They have three output states: high (1), low (0), and high-impedance (Z). The high-impedance state effectively disconnects the device from the bus, preventing conflicts when multiple components attempt to transmit data simultaneously. **Buffers** act as intermediate circuits that amplify or regulate signals, ensuring reliable data transmission between the microprocessor and peripherals. They help in isolating circuits, preventing excessive current draw from the processor, and reducing loading effects on the bus.

Other essential components include latches, which temporarily hold data before it is transferred, and decoders, which help in memory and peripheral selection. These components collectively enhance the performance, stability, and efficiency of microcomputer-based systems by managing data flow and ensuring smooth interaction between the microprocessor, memory, and I/O devices.

i Tri-State Devices

Tri-state logic devices have three states: logic 1, logic 0, and high impedance. The term Tri-state is a trademark of National Semiconductor and is used to represent three logic states. A tri-state logic device has a third line called Enable, as shown in Figure 7. When this line is activated, the tri-state device functions the same way as ordinary logic devices. When the third line is disabled, the logic device goes into the high impedance state-as if it were disconnected from the system. Ordinarily, current is required to drive a device in logic 0 and logic 1 states. In the high impedance state, practically no current is drawn from the system. Figure 7(a) shows a tri-state inverter. When the Enable is high, the circuit functions as an ordinary inverter; when the Enable line is low, the inverter stays in the high impedance state. Figure 7(b) also shows a tri-state inverter with active low Enable line-notice the bubble. When the Enable line is high, the inverter stays in the high impedance state.

In microcomputer systems, peripherals are connected in parallel between the address bus and the data bus. However, because of the tri-state interfacing devices, peripherals do not load the system buses. The microprocessor communicates with one device at a time by enabling the tri-state line of the interfacing device. Tri-state logic is critical to proper functioning of the microcomputer.

ii Buffer

The buffer is a logic circuit that amplifies the current or power. It has one input line and one output line (a simple buffer is shown in Figure 8a). The logic level of the output is the same as that of the input; logic 1 input provides logic 1 output (the opposite of an inverter). The buffer is used primarily to increase the driving capability of a logic circuit. It is also known as a driver.

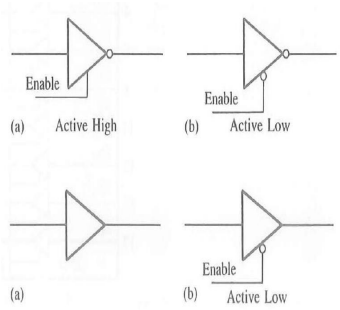

Figure 6**:** Tri-State Inverters with Active High and Active Low Enable Lines

Figure 7 shows a tri-state buffer. When the Enable line is low, the circuit functions as a buffer; otherwise it stays in the high impedance state. The buffer is commonly used to increase the driving capability of the data bus and the address bus.

iii Examples of Tri-State Buffers and Bidirectional Buffers

Tri-state buffers play a crucial role in microcomputer systems, particularly in bus-oriented architectures, where multiple devices share the same bus. These buffers allow controlled access to the data and address buses, ensuring that only one device drives the bus at a time, while others remain in a high-impedance state to avoid conflicts. One common example of a tri-state buffer is the **74LS244** octal buffer, also known as a line driver or line receiver. This device is frequently used as an address bus driver in systems that require multiple components to share the address bus. The 74LS244 contains two groups of four buffers, each controlled by an active-low enable line ($1\overline{G}$ and $2\overline{G}$). When these enable lines are low, the corresponding buffers are activated, allowing data to pass through. When the enable lines are high, the outputs enter a high-impedance state, effectively disconnecting from the bus. The 74LS244 is designed to sink up to 24 mA and source up to -15 mA, making it suitable for driving address lines over long distances with minimal signal degradation.

iv Bidirectional Buffer: 74LS245 Octal Bus Transceiver

Unlike the address bus, which is unidirectional, the data bus in a microcomputer system is bidirectional, requiring specialized buffers that allow data to flow in both directions. The 74LS245 octal bus transceiver is a widely used bidirectional buffer designed to manage data transmission between two buses, commonly labeled as A bus and B bus.

The 74LS245 contains 16 bus drivers, divided equally for two-way communication, with eight drivers dedicated to each direction. The data flow direction is controlled by the DIR pin. When DIR is high, data flows from the A bus to the B bus, whereas when DIR is low, data is transferred from the B bus to the A bus.

Additionally, the device includes an active-low Enable signal ($\overline{G}$), which functions as a master control switch. The Enable ($\overline{G}$) and Direction (DIR) signals are logically ANDed to determine when and in which direction the bus lines will be activated. This mechanism ensures controlled bidirectional data transfer, preventing bus conflicts and signal contention. Like the 74LS244, the 74LS245 is designed to handle a sink current of 24 mA and a source current of -15 mA, making it efficient for driving buses over extended circuit layouts.

v Importance of Tri-State and Bidirectional Buffers in Microprocessors

Tri-state and bidirectional buffers are essential in microprocessor-based systems because they enable efficient bus sharing, signal integrity, and reduced power consumption. In microprocessors such as the 8085, where the data bus (D7-D0) is multiplexed with the lower 8 bits of the address bus (AD7-AD0), tri-state buffers and latches (such as the **74LS373**) are used to separate the address and data lines effectively. The use of octal buffers like **74LS244** and **74LS245** ensures that only the required device actively communicates on the bus while keeping all other devices in a high-impedance state, thereby minimizing bus contention and enhancing system reliability. These buffers are commonly found in memory interfacing, peripheral communication, and data bus management applications, making them fundamental components in microprocessor architecture and digital systems.

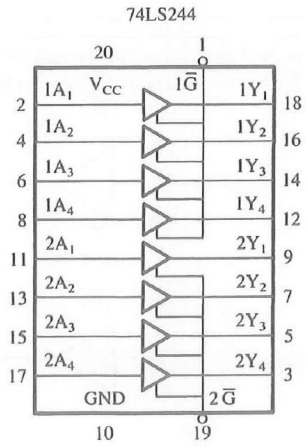

Figure 7: Logic Diagram of the 74LS244 Octal Buffer

Chapter 3. The 8085 Instruction Set

The 8085 instructions can be classified into the following five functional categories: data transfer (copy) operations, arithmetic operations, logical operations, branching operations, and machine-control operations. The 8085 instruction set is structured to perform a wide range of operations essential for efficient computation and system control. **Data transfer, arithmetic, logical, branching, and machine control instructions** collectively enable the microprocessor to execute complex tasks and interact with memory and peripheral devices effectively. Understanding these instruction groups is fundamental for developing efficient assembly language programs and optimizing system performance in embedded and general-purpose applications.

1. Addressing Modes Available in 8085

The 8085 microprocessor uses different addressing modes to access data efficiently depending on the instruction type. These modes enhance program flexibility and efficiency by allowing operations on immediate values, registers, and memory locations. Addressing modes define how the operand (data or memory address) is specified in an instruction. The 8085 microprocessor supports the following five types of addressing modes:

1. Immediate Addressing Mode

In this mode, the operand (data) is explicitly specified in the instruction. The data is provided immediately after the opcode in memory.

```
MVI A, 32H  ; Load the immediate value 32H into the accumulator
```

2. Register Addressing Mode

The operand is stored in a register, and the instruction specifies the register name. The operation is performed using register contents.

```
MOV A, B  ; Move the contents of register B to register A
```

3. Direct Addressing Mode

The instruction provides a 16-bit memory address where the operand is stored. The μp fetches the data from that memory location.

```
LDA 2500H  ; Load the contents of memory location 2500H into A
```

4. Indirect Addressing Mode

The memory address where the operand is stored is specified in a register pair (HL, BC, or DE). The microprocessor accesses the memory using the address stored in the register pair.

```
MOV A, M  ; Load the contents of memory location pointed by HL into A
```

5. Implicit (Implied) Addressing Mode

The operand is implied by the instruction itself; it does not require additional data or an address. The operation is performed on a predefined register (usually accumulator A).

```
CMA   ; Complement (invert) the contents of the accumulator
```

2. Classification of 8085 Instructions

The instructions in the 8085 microprocessor can be categorized into five primary functional groups: data transfer operations, arithmetic operations, logical operations, branching operations, and machine control operations. Each category serves a specific role in processing and executing instructions efficiently within the microprocessor.

i Data Transfer (Copy) Operations

Data transfer operations involve copying data from one location, referred to as the source, to another location, known as the destination, while preserving the original contents of the source. Although the term "transfer" is commonly used, it does not imply that the original data is removed; instead, it remains intact at its source location. These operations facilitate communication between different components within the microprocessor, ensuring smooth data flow.

Data transfer can occur in multiple ways. It includes copying data between registers, where one register's contents are duplicated in another. The microprocessor can also load a specific data byte into a register or a memory location, allowing immediate data usage. Another common operation involves transferring data between memory and a register, where the contents of a memory location are loaded into a register or stored back into memory. Additionally, data transfer between an I/O device and the accumulator is an essential function, enabling interaction with external peripherals such as keyboards, displays, or sensors.

Examples of these operations include copying the contents of register B into register D, loading register B with the hexadecimal value 32H, moving data from memory location 2000H to register B, or transferring input from a keyboard into the accumulator.

ii Arithmetic Operations

Arithmetic instructions are responsible for executing fundamental mathematical operations such as addition, subtraction, increment, and decrement. These operations primarily involve the accumulator, which serves as the central register for arithmetic computations. In addition operations, an 8-bit number, the contents of a register, or data stored in a memory location can be added to the contents of the accumulator. The sum is stored in the accumulator, ensuring the processor maintains the most recent computed result. However, direct addition between two general-purpose registers, such as adding register B to register C, is not possible. The DAD instruction, which handles 16-bit data, is an exception, as it allows direct addition of values stored in register pairs.

Subtraction follows a similar approach, where an 8-bit value, register contents, or memory data can be subtracted from the accumulator. The operation is carried out using two's complement arithmetic, ensuring accurate representation of negative results when necessary. If the result is negative, it is stored in two's complement form within the accumulator. Like addition, subtraction cannot be directly performed between two general-purpose registers. Increment and decrement operations modify data by increasing or decreasing the value of a register or memory location by one. These operations differ

from standard addition and subtraction because they can be executed on any register or memory location without requiring the use of the accumulator. Additionally, the 16-bit contents of register pairs, such as BC, can also be incremented or decremented directly.

iii Logical Operations

Logical instructions execute bitwise operations, enabling data manipulation at the binary level. These operations include AND, OR, Exclusive-OR (XOR), rotate, compare, and complement functions, all of which interact with the contents of the accumulator.

Bitwise AND, OR, and XOR operations allow data stored in the accumulator to be logically combined with an 8-bit number, a register, or a memory location. The result is stored in the accumulator, enabling logical decision-making and bitwise data processing. Rotation operations shift the bits within the accumulator to the left or right, moving each bit to its adjacent position. This operation is particularly useful in applications such as bitwise multiplication, division, and data encryption.

The compare operation checks whether an 8-bit number, register contents, or memory value is equal to, greater than, or less than the contents of the accumulator. The results of this operation influence the status flags, which can be used in conditional branching decisions. Complementing the accumulator reverses its binary content, replacing all 0s with 1s and 1s with 0s. This is commonly used in negative number representation and bitwise inversion tasks. Logical instructions in the 8085 microprocessor are used to perform bit-wise operations, such as AND, OR, XOR, and Complement. The instructions below do not affect carry flags but modify zero (Z), sign (S), and parity (P) flags. Here are four important logical instructions:

1. CMA (Complement Accumulator)

Operation: Inverts all bits of the accumulator (A). This instruction is useful in bitwise inversion operations and logical manipulations. Effect: Converts each 0 to 1 and each 1 to 0 in A. Example:

```
MVI A, 55H     ; Load A with 55H (01010101 in binary)
CMA            ; Complement A → A becomes AAH (10101010)
```

2. ANI (AND Immediate with Accumulator)

Operation: Performs bitwise AND between the accumulator (A) and an immediate 8-bit value. This instruction is used for bit masking and clearing unwanted bits. Instruction Format: ANI 8-bit data

Effect: A = A & Immediate Value, Example:

```
MVI A, F0H     ; Load A with F0H (11110000 in binary)
ANI 0FH        ; AND A with 0FH (00001111)
```

Before ANI: A = F0H (11110000)

Masking with 0FH: F0H AND 0FH = 11110000 AND 00001111 = 00000000

After ANI: A = 00H

3. XRI (Exclusive-OR Immediate with Accumulator)

Operation: Performs bitwise XOR between the accumulator (A) and an immediate 8-bit value.

Instruction Format: XRI data, Effect: A = A ⊕ Immediate Value, Example:

MVI A, 55H ; Load A with 55H (01010101 in binary)

XRI AAH ; XOR A with AAH (10101010)

Binary XOR: 01010101 ⊕ 10101010 = 11111111 (FFH)

After XRI: A = FFH

This instruction is used for toggling specific bits and encryption algorithms.

4. ORA (Logical OR with Accumulator)

Operation: Performs bitwise OR between the accumulator (A) and another register or memory location. This instruction is used for setting specific bits in a register.

Instruction Format: ORA R (for register) / ORA M (for memory)

Effect: A = A | R (or Memory), Example:

MVI A, 0CH ; Load A with 0CH (00001100 in binary)

MVI B, 03H ; Load B with 03H (00000011 in binary)

ORA B ; A ← A | B

Before ORA: A = 0CH (00001100), B = 03H (00000011)

OR operation: 00001100 OR 00000011 = 00001111 (0FH)

After ORA: A = 0FH

iv Rotate Instructions in 8085

Rotate Instructions in 8085 Microprocessor: Rotate instructions in the 8085 microprocessor are used to shift or rotate the bits in the accumulator (A) either left or right. These instructions help in bitwise operations.

Table 6 Types of Rotate Instructions in 8085

Instruction	Mnemonic	Operation	Effect on Carry Flag (CY)
Rotate Left	RLC	Shifts all bits left; MSB → CY, CY → LSB	CY = MSB
Rotate Right	RRC	Shifts all bits right; LSB → CY, CY → MSB	CY = LSB
Rotate Left through Carry	RAL	Shifts all bits left; MSB → CY, CY → LSB	CY is used as extra bit

Instruction	Mnemonic	Operation	Effect on Carry Flag (CY)
Rotate Right through Carry	RAR	Shifts all bits right; LSB → CY, CY → MSB	CY is used as extra bit

1. RLC (Rotate Left)

Mnemonic: RLC

Operation: Shifts all bits left by one position. The MSB (D7) is copied into Carry (CY) and also placed in LSB (D0).

Before: A = 10101100 (AC)

After : A = 01011001 (59)

Carry = 1

```
MVI A, 0ACH    ; Load A with AC (10101100)
RLC            ; Rotate left, A = 59 (01011001), CY = 1
```

2. RRC (Rotate Right)

Mnemonic: RRC

Operation: Shifts all bits right by one position. The LSB (D0) is copied into Carry (CY) and also placed in MSB (D7).

Before: A = 10101100 (AC)

After : A = 01010110 (56)

Carry = 0

```
MVI A, 0ACH    ; Load A with AC (10101100)
RRC            ; Rotate right, A = 56 (01010110), CY = 0
```

3. RAL (Rotate Left through Carry)

Mnemonic: RAL

Operation: Shifts all bits left, but the previous Carry flag (CY) is shifted into LSB (D0). The MSB (D7) is moved to Carry.

Before: A = 10101100 (AC), CY = 1

After : A = 01011001 (59)

Carry = 1

```
MVI A, 0ACH    ; Load A with AC (10101100)
STC            ; Set Carry Flag (CY = 1)
RAL            ; Rotate left through carry
```

4. RAR (Rotate Right through Carry)

Mnemonic: RAR

Operation: Shifts all bits right, but the previous Carry flag (CY) is shifted into MSB (D7).
The LSB (D0) is moved to Carry.

Before: A = 10101100 (AC), CY = 1

After : A = 11010110 (D6)

Carry = 0

```
MVI A, 0ACH    ; Load A with AC (10101100)
STC            ; Set Carry Flag (CY = 1)
RAR            ; Rotate right through carry
```

v Branching Operations

Branching instructions alter the normal sequence of program execution by redirecting control flow based on specific conditions. These operations allow programs to implement loops, conditional execution, and subroutine calls, making them fundamental for structured programming.

Jump instructions provide an essential mechanism for conditional execution. Conditional jumps evaluate a specific condition, such as checking if the zero flag or carry flag is set, before altering the program sequence. If the condition is met, execution jumps to a designated memory address, otherwise, the program continues sequentially. In contrast, unconditional jumps redirect execution to a different memory location without evaluating any conditions, ensuring direct program flow redirection.

vi Machine Control Operations

Machine control instructions manage essential processor functions, such as halting execution, handling interrupts, and performing no-operation (NOP) tasks. These instructions provide direct control over the microprocessor's behavior and are particularly useful in power management, debugging, and interrupt servicing. The HLT instruction stops the execution of the program indefinitely, causing the processor to remain idle until an external reset or interrupt resumes operation. Interrupt handling is another critical function, allowing the processor to respond to external events and execute designated interrupt service routines. Additionally, the NOP (No Operation) instruction performs no computational task but occupies one machine cycle, making it useful for timing adjustments and debugging purposes.

Machine control instructions in the 8085 microprocessor are used to control processor operations, such as halting execution, enabling/disabling interrupts, and controlling the system bus. These instructions do not affect the flags and are primarily used for system control and efficiency.

1. HLT (Halt the Processor)

Operation: Stops execution and places the microprocessor in a halt state. The system clock continues, but instruction execution stops until a reset occurs. Used at the end of a program to stop execution.

```
MVI A, 32H     ; Load 32H into accumulator
HLT            ; Halt the processor
```

2. NOP (No Operation)

Operation: Executes no operation and simply increments the Program Counter (PC). No changes to registers or memory. Used for delays, timing adjustments, and debugging programs.

NOP ; No operation, simply advances to the next instruction

3. DI (Disable Interrupts)

Operation: Disables all maskable interrupts (except TRAP). Prevents any interrupts (INTR, RST 7.5, RST 6.5, RST 5.5) from occurring. Used in critical sections of code where interrupts should not interfere.

4. EI (Enable Interrupts)

Operation: Enables all maskable interrupts (INTR, RST 7.5, RST 6.5, RST 5.5). Allows the microprocessor to recognize and process interrupts. Used after DI to re-enable interrupts in a program.

Example programs in 8085 microprocessor add two 16-bit numbers

```
MVI C, 2EH      ; Move immediate byte 2EH to register C.
MVI E, DBH      ; Move immediate byte DBH to register E.
MOV A, E        ; Move the content of the E register to register A.
ADD C           ; A ← A+C, Add the content of C register to Accumulator.
MOV L, A        ; Copy the content of Accumulator(A) to L register
MVI B, 2FH      ; Move immediate byte 2FH into the B register.
MVI D, 3BH      ; Move immediate byte 3BH into the D register.
MOV A, D        ; Move the content of the D register to the “A” register.
ADC B           ; A ← A + B+ [CY], ADD the content of B register to A with carry flag.
MOV H, A        ; Copies the content of the Accumulator to the H register.
HLT             ; Terminating the program.
```

Chapter 4. Memory and I/O Interfacing

Memory can be classified into two groups: prime (system or main) memory and storage memory. The R/WM and ROM are examples of prime memory; this is the memory the microprocessor uses in executing and storing programs. This memory should be able to respond fast enough to keep up with the execution speed of the microprocessor. Therefore, it should be random access memory, meaning that the microprocessor should be able to access information from any register with the same speed (independent of its place in the chip). The size of a memory chip is specified in terms of bits. For example, a 1K memory chip means it can store 1K (1024) bits (not bytes). On the other hand, memory in a system such as a PC is specified in bytes. For example, 4M memory in a PC means it has 4 megabytes of memory. Basic Definitions in Microprocessors and Computing are given below,

Bit (Binary Digit): A bit is the smallest unit of data in a computer and can have only two possible values: 0 or 1. It represents the binary number system, which is the foundation of all digital computing.

Byte: A byte consists of eight bits and is the standard unit for data storage and memory addressing.

Example: 11010101 (8-bit binary number) represents one byte.

Word: A word is a fixed-sized group of bits that a processor handles as a unit.

The size of a word depends on the processor architecture: 8-bit processors (like 8085) use an 8-bit word. 16-bit processors use a 16-bit word. 32-bit processors use a 32-bit word.

Double Word: A double word consists of two words (twice the word size of the processor).

In a 16-bit processor, a double word is 32 bits (2 × 16 bits). In a 32-bit processor, a double word is 64 bits (2 × 32 bits).

Quad Word: A quad word is four times the size of a word. For a 16-bit processor, a quad word is 64 bits. For a 32-bit processor, a quad word is 128 bits. These are commonly used in advanced computing for data processing and floating-point operations.

Instruction: An instruction is a command given to the microprocessor to perform a specific operation. It consists of an opcode (operation code) and operands (data or addresses involved in the operation). Instructions are classified into data transfer, arithmetic, logical, control, and branching instructions.

1. Memory Classification

The other group is the storage memory, such as magnetic disks and tapes. This memory is used to store programs and results after the completion of

i RAM

As the name suggests, the microprocessor can write into or read from this memory; it is popularly known as Random Access memory (RAM). It is used primarily for information that is likely to be altered, such as writing programs or receiving data. This memory is volatile, meaning that when the power is turned off, all the contents are destroyed. Two

types of R/W memories - static and dynamic-are available; they are described in the following paragraphs. A key characteristic of R/W memory is that it is volatile, meaning that all stored information is lost when the power supply is turned off. Due to this property, RAM is often supplemented by non-volatile storage devices such as hard drives or flash memory to retain important data.

ii Static RAM (SRAM)

Static Random-Access Memory (SRAM) is a high-speed memory type built using flip-flops, where each memory cell requires six transistors to store a single bit. Unlike Dynamic RAM (DRAM), SRAM retains its data as long as power is supplied, without requiring periodic refreshing.

Advantages: Faster access time compared to DRAM. More reliable since it does not require refreshing. Used as cache memory in high-speed processors (e.g., Intel 486 and Pentium).

Disadvantages: Lower density due to its complex circuitry. Higher power consumption compared to DRAM. More expensive than DRAM, making it impractical for large memory sizes. For performance enhancement, SRAM is often integrated within the processor as cache memory or used externally to improve system speed.

iii Dynamic RAM (DRAM)

Dynamic Random-Access Memory (DRAM) consists of MOS transistor gates that store each bit as an electrical charge. Unlike SRAM, the charge leaks over time, requiring periodic refreshing to maintain data integrity.

Advantages: Higher density than SRAM, allowing more storage per chip. Lower power consumption compared to SRAM. Cheaper to manufacture, making it suitable for large memory requirements.

Disadvantages: Slower than SRAM due to refresh cycles. Requires additional circuitry for memory refreshing, adding to system complexity. DRAM is commonly used in main memory (RAM) of computers, particularly when memory requirements exceed 8 KB, as it is more cost-effective than SRAM for larger systems.

iv Masked ROM (Read-Only Memory)

Masked ROM is a permanently programmed memory where bit patterns are recorded using a masking and metallization process during manufacturing.

Advantages: Ideal for mass production since it offers low per-unit cost at high volumes. Provides fast and stable storage for firmware and essential system code.

Disadvantages: Non-modifiable once programmed, making updates impossible. High initial production cost, making it economical only for large-scale manufacturing. Masked ROM is commonly used in consumer electronics, embedded systems, and industrial controllers, where the stored program does not change.

v Programmable Read-Only Memory (PROM)

PROM is a type of ROM that allows the user to program it once using a special PROM programmer. The memory consists of nichrome or polysilicon wires arranged in a matrix, which are selectively burned during programming to store a permanent bit pattern.

Advantages: Allows customization of memory content after manufacturing. More flexible than Masked ROM.

Disadvantages: Once programmed, it cannot be erased or reprogrammed. Any errors in programming make the chip unusable. PROM is useful in applications where firmware needs to be customized post-manufacturing but does not require future modifications.

vi Erasable Programmable Read-Only Memory (EPROM)

EPROM stores data by charging the floating gate of a Field-Effect Transistor (FET). It can be erased by exposing the chip to ultraviolet (UV) light through a quartz window on the chip and then reprogrammed using an EPROM programmer.

Advantages: Can be erased and reused multiple times, making it ideal for development and testing. Provides non-volatile storage for firmware and system software.

Disadvantages: Erasure requires specialized UV light exposure, which is time-consuming. Limited reprogramming cycles compared to modern alternatives like EEPROM and Flash memory. EPROM is commonly used in prototyping, product development, and experimental projects where reprogramming may be necessary.

vii Electrically Erasable Programmable Read-Only Memory (EEPROM)

EEPROM is similar to EPROM but allows erasure and reprogramming using electrical signals, eliminating the need for UV light exposure.

Advantages: Selective erasure and reprogramming at the byte level, unlike EPROM which requires full erasure. Useful for remote software updates without requiring physical access to the chip.

Disadvantages: Slower write operations compared to modern Flash memory. Limited write endurance, typically allowing 10,000 to 100,000 rewrite cycles. EEPROM is used in applications where frequent updates to firmware or configuration data are required, such as BIOS chips, smart cards, and industrial automation systems.

viii Flash Memory

Flash memory is an advanced version of EEPROM, designed for faster erasure and programming. It is widely used for storage devices, embedded systems, and modern microcontrollers.

Advantages: Higher write endurance than EEPROM (up to 1 million cycles). Lower power consumption, making it suitable for battery-powered devices. Can be programmed using low voltage levels (as low as 1.8V).

Disadvantages: Unlike EEPROM, it cannot be erased at the byte level. Instead, erasure occurs at sector (block) level or entire memory level. Flash memory is extensively used in USB drives, memory cards, SSDs (Solid-State Drives), and embedded system firmware storage.

Comparison of Different Memory Types

Memory Type	Volatility	Rewrite Capability	Speed	Use Case
SRAM	Volatile	Yes	Very Fast	Cache memory, high-speed computing
DRAM	Volatile	Yes	Moderate	Main system memory (RAM)
Masked ROM	Non-volatile	No	Fast	Mass-produced firmware storage
PROM	Non-volatile	One-time programmable	Fast	Custom firmware (fixed programs)
EPROM	Non-volatile	Erasable (UV light)	Moderate	Prototyping, development
EEPROM	Non-volatile	Electrically erasable	Moderate	Firmware updates, configuration data
Flash Memory	Non-volatile	Block-wise rewrite	Fast	USB drives, SSDs, embedded systems

2. Memory Read Machine Cycle

Sequence of Events When 8085 Reads from Memory

When the 8085 microprocessor performs a memory read operation, it follows a specific sequence of events to fetch data from memory. The steps involved in this process are as follows:

Step-by-Step Memory Read Cycle

Place Address on Address Bus: The 8085 places the 16-bit memory address (from the Program Counter or other register) on the Address Bus (A0–A15). The lower eight bits (A0–A7) are multiplexed with the data bus (AD0–AD7).

Activate the Control Signals: The Memory Read ($\overline{RD}$) signal is activated (low) to indicate a read operation. The IO/$\overline{M}$ signal is set to 0, indicating that a memory operation (not an I/O operation) is in progress.

Enable Address Latch (ALE Signal): The Address Latch Enable (ALE) signal is generated to separate the lower address bits from the multiplexed address/data bus. This allows external latch circuits to hold the lower address until the read cycle is completed.

Memory Device Responds: The memory chip, selected by the address bus, places the requested data byte on the data bus (D0–D7).

Microprocessor Reads Data: 8085 reads the data from the data bus and stores it in the accumulator or a register.

Deactivate Control Signals: The $\overline{RD}$ (Read) signal is deactivated (set to high). The Address and Data Bus are released for the next operation.

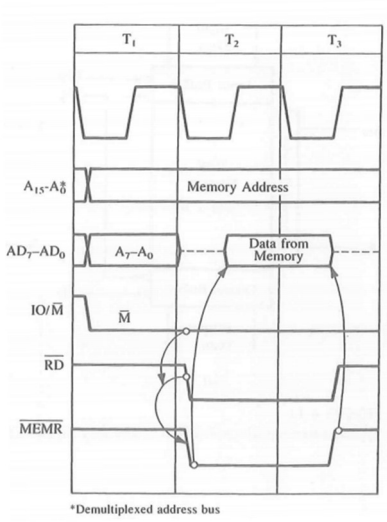

Figure 8. Timing diagram of 8085 Memory Read machine cycle.

3. Memory-Write Operation

During a memory-write operation, the 8085 microprocessor stores data from a register usually the accumulator) into a specific memory location. The process involves generating appropriate control signals and managing data flow. The microprocessor generates the following control signals to coordinate the writing process:

IO/$\overline{M}$ (Input/Output or Memory Select Signal)

Value: 0 (Low), Function: Specifies that a memory operation (not an I/O operation) is in progress.

$\overline{WR}$ (Write Control Signal)

Value: 0 (Active Low), Function: Enables the memory device to store data from the microprocessor.

ALE (Address Latch Enable)

Value: Pulsed High, Function: Latches the lower byte of the address onto an external latch since the address/data bus (AD_0–AD_7) is multiplexed.

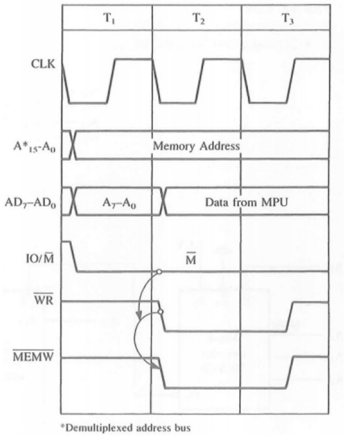

Figure 9: Timing diagram of memory write machine cycle.

Direction of Data Flow on the Data Bus

- The 8085 places the memory address on the Address Bus (A15–A0).

- The ALE signal is activated to latch the lower address byte.
- The data to be written is placed on the Data Bus (D7–D0).
- The $\overline{WR}$ signal goes LOW, instructing memory to store the data.
- Once the memory acknowledges the write operation, the $\overline{WR}$ signal returns HIGH.

4. Requirements of a memory chip and microprocessor bus

To interface a memory chip with a microprocessor, certain requirements must be met to ensure proper addressing, selection, and data transfer.

1. Address Lines and Memory Register Identification: A memory chip consists of multiple memory registers, each requiring a unique address. The number of address lines needed for a memory chip is determined by the total number of registers (memory locations) it contains. In 8085 microprocessor, which has 16 address lines, it can theoretically address up to: 65,536 (64 KB). However, not all 16 address lines may be required by every memory chip. The necessary address lines for the memory chip must be directly connected to the microprocessor's address bus to enable memory addressing.

Chip Select ($C\bar{S}$) Signal for Memory Enable: A memory chip must be activated before it can be accessed by the microprocessor. This activation is controlled by the Chip Select ($C\bar{S}$) signal.

Chip Selection: The remaining unused address lines from the microprocessor can be connected to the ($C\bar{S}$) pin of the memory chip through interfacing logic (such as a decoder or address latch).

Enabling the Memory Chip: When the correct logic combination of address lines activates the $C\bar{S}$ signal, the corresponding memory chip is enabled, allowing data transfer.

3. Addressing a Specific Memory Register: Once the memory chip is selected, the microprocessor must further identify the specific memory register (location) within the chip where data needs to be read or written. The $C\bar{S}$ signal selects the memory chip. The address lines of the microprocessor are then used to specify the exact memory location (register) within the chip. The logic states (0s and 1s) of all connected address lines determine the exact memory address being accessed.

4. Control Signals for Read and Write Operations: The microprocessor uses control signals to manage data transfer between memory and itself via the data bus.

Memory Read Operation: The Read ($R\bar{D}$) control signal is activated by the microprocessor. This enables the output buffer of the memory chip. The contents of the selected memory register are placed on the data bus and sent to the microprocessor.

Memory Write Operation: The Write ($W\bar{R}$) control signal is activated by the microprocessor. This enables the input buffer of the memory chip. Data from the microprocessor is transferred via the data bus and stored in the selected memory register.

A model of typical memory chip representing the above requirements is shown in figure below.

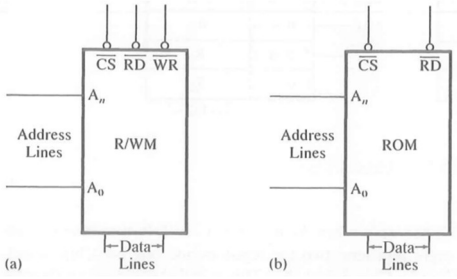

Figure 10: Model of typical memory chip

5. Interfacing of 1K (1024×8) memory

The memory chip has 1024 registers; therefore ten address lines $(A_9 - A_0)$ are required to identify the registers. The remaining six address lines $(A_{15} - A_{10})$ of the microprocessor are used for the Chip Select $(\overline{CS})$ signal. In figure 11 the memory chip is enabled when the address lines $A_{15} - A_{10}$ are at logic 0. The address lines $A_9 - A_0$ can assume any address of the 1024 registers, starting from all zeros to all 1s. The memory addresses range from 0000H to 03FFH. By combining the high-order and low-order address lines we can specify the complete address range of a given chip.

The memory address can be changed to any other location by changing the hardware of the $\overline{CS}$ line. For example, if A_{15} is connected to the NAND gate without an inverter, the memory addresses will range from 8000 H to 83FFH.

6. Timing Diagram for Opcode Fetch Operation

The Opcode Fetch Cycle is the first step in executing an instruction. During this cycle, the 8085 microprocessor fetches the opcode (operation code) of the instruction from memory. Steps in the Opcode Fetch Cycle—

T1 (First Clock Cycle) Program Counter (PC) places the memory address on the Address Bus (A15–A0). The Address Latch Enable (ALE) signal is HIGH, indicating that the lower byte of the address is being latched. The IO/$\overline{M}$ signal is LOW (0), indicating a memory operation. $\overline{RD}$ (Read) is HIGH (inactive).

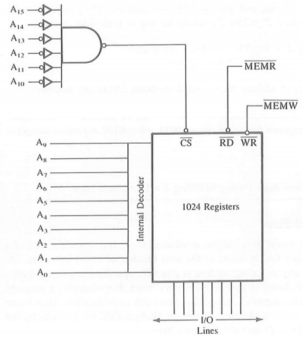

Figure 11: Interfacing memory with microprocessor

T2 (Second Clock Cycle): The memory responds by placing the opcode on the Data Bus (D7–D0).

The $\overline{RD}$ signal is activated (LOW), allowing data transfer from memory to the microprocessor.

The ALE signal goes LOW after address latching.

T3 (Third Clock Cycle): The opcode is read into the instruction register of the microprocessor.

The $\overline{RD}$ signal is deactivated (HIGH), completing the memory read cycle.

The Data Bus is now free for the next operation.

T4 (Fourth Clock Cycle): The fetched opcode is decoded, and execution of the instruction begins.

The processor prepares for the next operation.

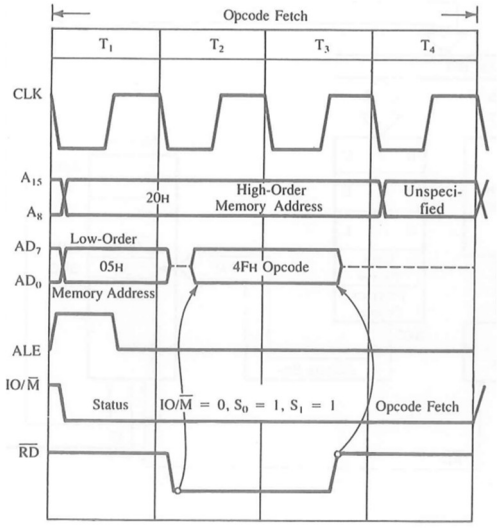

Figure 12. Opcode fetch machine cycle timing diagram

7. Memory-Mapped I/O and I/O-Mapped I/O Schemes

In 8085 microprocessor, peripheral devices (I/O devices) can be interfaced using two addressing schemes: Memory-Mapped I/O and I/O-Mapped I/O. These schemes determine how the processor accesses I/O devices.

1. Memory-Mapped I/O

In this scheme, I/O devices are treated as memory locations. The 16-bit address bus is used to address both memory and I/O devices. Standard memory access instructions (LDA, STA, MOV M, A) are used to transfer data to and from the I/O device. No separate I/O instructions are required.

LDA 3000H ; Load data from I/O device at address 3000H into A

STA 3001H ; Store data from A to I/O device at address 3001H

2. I/O-Mapped I/O

I/O devices are assigned separate address space (using an 8-bit address). Special I/O instructions (IN and OUT) are used for data transfer. The 8085 uses the IO/$\overline{M}$ control signal to differentiate between memory and I/O access. A maximum of 256 I/O ports (00H to FFH) can be interfaced.

IN 20H ; Read data from I/O port 20H into A

OUT 30H ; Send data from A to I/O port 30H

Table 7 Difference between the two techniques

Feature	**Memory-Mapped I/O**	**I/O-Mapped I/O**
Addressing Space	Uses 16-bit memory addresses	Uses 8-bit I/O addresses
Number of Devices	64 KB (65,536 I/O locations)	256 Input and 256 Output devices (00H–FFH)
Instructions Used	Uses memory instructions (LDA, STA, MOV M, A)	Uses I/O instructions (IN, OUT)
Operations on Data	Supports arithmetic and logical operations directly	Requires data to be transferred to registers before processing
Speed	Slower (because of memory decoding)	Faster (direct I/O access)
Control Signal Used	Uses $\overline{\text{MEMR}}$ and $\overline{\text{MEMW}}$	Uses $\overline{\text{IOR}}$ and $\overline{\text{IOW}}$
Application	Used in systems with fewer I/O devices	Used when many I/O devices are needed

8. Interface a 2732 EPROM (4K bytes) memory chip

The interfacing circuit for 2732 chip is shown below with steps for understanding.

Interfacing Circuit

Figure 13 below shows an interfacing circuit using a 3-to-8 decoder to interface the 2732 EPROM memory chip. It is assumed here that the chip has already been programmed, and we will analyze the interfacing circuit in terms of the same three steps outlined previously: Step 1: The 8085 address lines $A_{11} - A_0$ are connected to pins $A_{11} - A_0$ of the memory chip to address 4096 registers.

Step 2: The decoder is used to decode four address lines $\mathrm{A_{15} - A_{12}}$. The output $\mathrm{O_0}$ of the decoder is connected to Chip Enable ($\overline{\text{CE}}$). The $\overline{\text{CE}}$ is asserted only when the address on $\mathrm{A_{15} - A_{12}}$ is 0000; $\mathrm{A_{15}}$ (low) enables the decoder and the input 000 asserts the output $\mathrm{O_0}$.

Step 3: For this EPROM, we need one control signal: Memory Read ($\overline{\text{MEMR}}$), active low. The $\overline{\text{MEMR}}$ is connected to $\overline{\text{OE}}$ to enable the output buffer; $\overline{\text{OE}}$ is the same as $\overline{\text{RD}}$ signal.

Address Decoding and Memory Addresses

We can obtain the address range of this memory chip by analyzing the possible logic levels on the 16 address lines. The logic levels on the address lines $\mathrm{A_{15} - A_{12}}$ must be 0000 to assert the Chip Enable, and the address lines $\mathrm{A_{11} - A_0}$ can assume any combinations from

all 0s to all 1s. Therefore, the memory address of this chip ranges from 0000H to 0FFFH, as shown below.

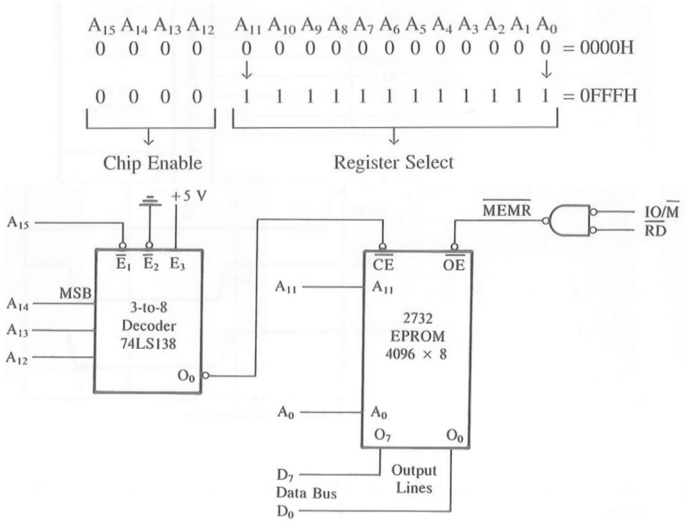

Figure 13: Interfacing the 2732 EPROM chip

Interfacing I/O Devices

The I/O devices, such as keyboards and displays, are the ears and eyes of the MPUs; they are the communication channels to the "outside world." Data can enter (or exit) in groups of eight bits using the entire data bus; this is called the parallel I/O mode. The other method is the serial I/O, whereby one bit is transferred using one data line; typical examples include peripherals such as the CRT terminal and cassette tape.

i Interfacing I/O Devices in 8085 Microprocessor

Input/Output (I/O) devices serve as the primary communication channels between a microprocessor and the external world. These devices include keyboards, displays, printers, and storage peripherals, which facilitate data exchange between the user and the microprocessor unit (MPU). Data can be transferred using two fundamental methods: parallel I/O and serial I/O.

In parallel I/O mode, data is transferred in groups of eight bits (one byte) simultaneously using the entire data bus. This method is efficient for high-speed communication, commonly used in keyboards, printers, and memory-mapped devices. In contrast, serial I/O mode transfers data one bit at a time over a single data line. This approach is employed in peripherals such as CRT terminals, serial communication interfaces, and cassette tape storage systems, where bitwise data transmission is necessary due to hardware constraints or communication protocols.

ii Peripheral I/O Instructions in 8085

The 8085 microprocessor provides two specialized instructions for data transfer between the processor and I/O devices: IN and OUT. These instructions allow the microprocessor to communicate with external input and output devices by reading data from or writing data to an assigned I/O port address. It is also called I/O-mapped I/O.

The IN instruction (opcode DB) is used to receive data from an input device, such as a keyboard. The data is loaded into the accumulator (A) for further processing. The OUT instruction (opcode D3) sends the contents of the accumulator to an output device, such as an LED display, which then displays the data. Both IN and OUT are two-byte instructions, where the first byte contains the opcode, and the second byte specifies the port address of the I/O device.

For example, the OUT instruction follows this format:

Opcode Operand → OUT 8-bit Port Address

This instruction copies the contents of the accumulator (A) and sends them to the output device assigned to the specified port address. If an LED display is mapped to port address 01H, the instruction stored in memory would be:

```
MVI A, 55H      ; Load accumulator with data (e.g., 55H)
OUT 01H         ; Send the data to the output port (LED display)
```

In this case, if port 01H is assigned to an LED display, the binary equivalent of 55H (01010101) would be displayed.

iii I/O Addressing and Device Assignment

The 8085 microprocessor can interface with 256 different I/O ports, each identified by an 8-bit address ranging from 00H to FFH. This means that up to 256 distinct input and output devices can be assigned unique addresses for communication.

The assignment of an I/O port address to a specific device is not predetermined by the microprocessor. Instead, it is arbitrarily decided based on system design constraints and available logic circuits. The hardware designer selects the appropriate I/O addresses for devices based on chip select logic, decoder circuits, and available ports in the system.

To fully understand how the 8085 executes IN/OUT instructions, it is essential to analyze the bus operations and signal control mechanisms that facilitate communication between the processor and peripheral devices. Proper design of I/O interfacing circuits ensures efficient data transfer, minimal bus conflicts, and optimal performance in microprocessor-based systems.

Absolute vs. Partial Decoding in Microprocessor-Based Systems

Address decoding is an essential process in microprocessor-based systems, ensuring that memory and I/O devices are accessed correctly. Decoding techniques determine how uniquely an address is mapped to a specific peripheral or memory location. The two primary approaches are absolute decoding and partial decoding, each with its own advantages and trade-offs.

iv Absolute Decoding

In absolute decoding, all relevant address lines are fully decoded to generate a unique selection signal for a specific memory or I/O device. This method ensures that the device

is activated only at a single, predetermined address. For example, in a system where an output port is mapped to address 01H, absolute decoding ensures that the device responds exclusively to this address and is not mistakenly accessed by any other address.

This approach is considered a good design practice, particularly in large systems with multiple I/O devices and memory modules, because it eliminates the risk of overlapping addresses and prevents unintentional device activation. However, absolute decoding requires more hardware components, such as address decoders or logic gates, making it a costlier solution in terms of circuit complexity and space.

v Partial Decoding

To reduce hardware complexity and minimize costs, some systems use partial decoding, where only a subset of address lines is used to select a device. Instead of decoding all address bits, some address lines are left unconnected or replaced with control signals such as IO/$\bar{M}$ and W$\bar{R}$. This means that multiple addresses can select the same device, leading to address redundancy, similar to foldback memory addressing.

For example, if address lines A1 and A0 are not used for decoding, a device originally assigned to address 01H can now be accessed by multiple addresses, such as 00H, 01H, 02H, and 03H. In this case, the latch or output port will respond to any of these four addresses, effectively increasing the number of addresses that can activate the same device.

Partial decoding is commonly used in small systems where the number of peripherals is limited and overlapping addresses do not cause conflicts. As long as these extra addresses are not assigned to other devices, the system can function correctly. However, if multiple devices share the same address unintentionally, it can lead to erroneous data transfer or system malfunction.

vi Choosing Between Absolute and Partial Decoding

Absolute decoding provides precise address selection and prevents conflicts, making it ideal for large-scale systems with multiple devices. However, it requires additional decoding hardware, increasing circuit complexity and cost. Partial decoding is a cost-effective solution, reducing hardware requirements by allowing fewer address lines to control device selection. This approach is suitable for simpler systems with fewer peripherals, provided that overlapping addresses do not interfere with other devices.

9. Comparison of Memory-Mapped I/O and Peripheral I/O

Table 8 Comparison of Memory-Mapped I/O and Peripheral I/O

Characteristics	Memory-Mapped I/O	Peripheral I/O
1. Device address	16-bit	8-bit
2. Control signals for Input/Output	$\overline{\text{MEMR}}$ / $\overline{\text{MEMW}}$	$\overline{\text{IOR/IOW}}$

3. Instructions available	Memory-related instructions such as STA; LDA; LDAX; STAX; MOV M,R: ADD M; SUB M; ANA M: etc.	IN and OUT
4. Data transfer	Between any register and I/O	Only between I/O and the accumulator
5. Maximum number of I/Os possible	The memory map (64 K) is shared between I/Os and system memory	The I/O map is independent of the memory map; 256 input devices and 256 output devices can be connected
6. Execution speed	13 T-states (STA,LDA) 7 T-states (MOV M,R)	10 T -states
7. Hardware requirements	More hardware is needed to decode 16-bit address	Less hardware is needed to decode 8-bit address
8. Other features	Arithmetic or logical operations can be directly performed with I/O data	Not available

Chapter 5. 8085 Programming Model

The programming model consists of some segments of the ALU and the registers. This model does not reflect the physical structure of 8085 but includes the information that is critical in writing assembly language programs. The architecture of 8085 includes an arithmetic and logic unit (ALU), a register array, and a control unit. It supports a variety of instruction types, including arithmetic, logical, data transfer, and control instructions. The microprocessor communicates with external memory and peripherals using a system bus, which consists of an address bus, data bus, and control bus. The instruction set of the 8085 microprocessor is designed to support efficient execution of operations. It includes single-byte, two-byte, and three-byte instructions, allowing for flexible programming. The microprocessor can execute instructions in a few clock cycles, making it suitable for applications requiring real-time processing.

Machine language is a low-level programming language that consists of binary code (0s and 1s) that the microprocessor can directly execute. It is specific to a particular processor and difficult for humans to read or write. Assembly language is a human-readable version of machine language, using mnemonics (such as MOV, ADD, and SUB) instead of binary code. It requires an assembler to convert the assembly code into machine code. Assembly language provides better readability and ease of programming compared to machine language.

1. Programming Model

The model consists of six registers, accumulators, and flag registers. In addition, it has two 16-bit registers: the stack pointer and the program counter. They are described briefly as follows.

i REGISTERS

The 8085 has six general-purpose registers to store 8-bit data, identified as B, C, D, E, H, and L. They can be combined as register pairs BC, DE, and HL to perform 16-bit operations. The programmer can use these registers to store or copy data into the registers using data copy instructions.

ii ACCUMULATOR

The accumulator is an 8-bit register part of the arithmetic/logic unit (ALU). This register is used to store 8-bit data and to perform arithmetic and logical operations. The result of an operation is stored in the accumulator: The accumulator is also identified as register A.

iii FLAGS

The ALU includes five flip-flops, which are set or reset after an operation according to the data conditions of the result in the accumulator and other registers. They are called Zero (Z), Carry (CY), Sign (S), Parity (P), and Auxiliary Carry (AC) flags. The most commonly used flags are Zero, Carry, and Sign. The microprocessor uses these flags to test data conditions. These flags have critical importance in the decision-making process of the microprocessor. The flags' conditions (set or reset) are tested through software instructions. For example,

the instruction JC (Jump On Carry) is implemented to change the sequence of a program when the CY flag is set. A thorough understanding of flags is essential in writing assembly language programs.

The following flags are set or reset after executing an arithmetic or logic operation; data copy instructions do not affect any flags. See the instruction set (Appendix F) to find how flags are affected by an instruction.

Z-Zero: The Zero flag is set to 1 when the result is zero; otherwise, it is reset. **CY-Carry**: If an arithmetic operation results in a carry, the CY flag is set; otherwise, it is reset. **S-Sign**: The Sign flag is set if bit D_7 of the result $= 1$; otherwise it is reset. **P-Parity**: If the result has an even number of 1s, the flag is set; for an odd number of 1s, the flag is reset. **AC-Auxiliary Carry**: In an arithmetic operation, when a carry is generated by a digit D_3 and passed to the digit D_4, the AC flag is set. This flag is used internally for BCD (binary-coded decimal) operations; there is no Jump instruction associated with this flag.

iv Program Counter (PC) and Stack Pointer (SP)

These are two 16-bit registers used to hold memory addresses. These registers are 16 bits because the memory addresses are 16 bits. The microprocessor uses the PC register to sequence the execution of the instructions. The function of the program counter is to point to the memory address from which the next byte is to be fetched. When a byte (machine code) is being fetched, the program counter is incremented by one to point to the next memory location.

The stack pointer is also a 16-bit register used as a memory pointer. It points to a memory location in R/W memory called the stack. The beginning of the stack is defined by loading a 16-bit address in the stack pointer.

2. Instruction Classification

An instruction is a binary pattern designed inside a microprocessor to perform a specific function. The entire group of instructions, called the instruction set, determines what functions the microprocessor can perform. The 8085 microprocessor includes the instruction set of its predecessor, the 8080A, plus two additional instructions.

The 8085 microprocessor supports the following addressing modes:
1. Immediate Addressing – The operand is specified in the instruction (e.g., MVI A, 05H).
2. Register Addressing – The operand is in a register (e.g., MOV A, B).
3. Direct Addressing – The address of the operand is specified (e.g., LDA 2500H).
4. Indirect Addressing – The address is stored in a register pair (e.g., MOV A, M).
5. Implied Addressing – The operand is implied (e.g., CMA).

i Immediate Addressing Mode

In immediate addressing mode, the operand is specified directly in the instruction. The data is provided immediately after the opcode in memory. This mode is used when the value to be operated on is known at the time of programming.

Example:

MVI A, 32H ; Load the immediate value 32H into the accumulator

ii Direct Addressing Mode

In direct addressing mode, the memory address from which data is to be fetched or stored is specified in the instruction. This mode allows direct access to a specific memory location. Example:

LDA 2500H ; Load the contents of memory location 2500H into the accumulator

iii Register Addressing Mode

In register addressing mode, the operand is stored in one of the registers of the microprocessor. This mode allows faster execution as data is accessed directly from registers rather than memory. Example:

MOV A, B ; Copy the contents of register B into register A

iv Indirect Addressing Mode

In indirect addressing mode, the memory address of the operand is stored in a register pair. The microprocessor first retrieves the memory address from the register pair and then accesses data from that memory location. Example:

MOV A, M ; Load the contents of the memory location pointed to by HL into A

v Implied Addressing Mode

In implied addressing mode, the operand is not explicitly mentioned in the instruction. Instead, it is understood by the operation itself. This mode is often used in accumulator-based operations. Example:

CMA ; Complement the contents of the accumulator

3. Instruction Format and Examples

An instruction in the 8085 microprocessor consists of an opcode (operation code) and an operand (data, memory location, or register). Based on their structure, instructions are classified into three formats.

i One-Byte Instructions

These instructions consist of a single byte, including both opcode and operand (if any). They are simple instructions that require minimal memory. Example:

CMA ; Complement the accumulator

ii Two-Byte Instructions

These instructions consist of an opcode followed by an 8-bit operand (data or address). They require two memory locations for execution. Example:

MVI A, 32H ; Move immediate value 32H to accumulator

iii Three-Byte Instructions

These instructions consist of an opcode followed by a 16-bit operand. They require three memory locations for execution. Example:

LDA 2500H ; Load the contents of memory location 2500H into A

Example program using DAA instruction

The DAA (Decimal Adjust Accumulator) instruction is used in the 8085 microprocessor to adjust the result of BCD (Binary-Coded Decimal) addition. The program below adds two BCD numbers and ensures a correct BCD result using the DAA instruction.

```
MVI A, 27H      ; Load first BCD number (27 in BCD)
MVI B, 59H      ; Load second BCD number (59 in BCD)
ADD B           ; A ← A + B (27H + 59H = 80H)
DAA             ; Adjust the result to BCD format
STA 3000H       ; Store the final BCD result at memory location 3000H
HLT             ; Halt the program
```

The DAA instruction works in two steps:

If the lower nibble (D3-D0) of the accumulator is greater than 9 or the Auxiliary Carry (AC) flag is set, DAA adds 06H to the accumulator. If the upper nibble (D7-D4) is greater than 9 or the Carry (CY) flag is set, DAA adds 60H to the accumulator.

This program adds two BCD numbers in 8085 using the DAA instruction to ensure a valid BCD result. DAA instruction is useful when working with BCD arithmetic, as it automatically corrects any invalid BCD results.

Jump Instructions in 8085 Microprocessor

Jump instructions in the 8085 microprocessor are used to alter the normal sequence of program execution by transferring control to a specified memory location. These instructions are classified into two types: *Conditional Jump* and *Unconditional Jump*.

Conditional jump instructions transfer control only if a condition is met, making them useful in decision-making operations. Unconditional jump instructions always transfer control, ensuring a direct program flow change. These instructions help in implementing loops, conditional execution, and function calls, making programs more structured and efficient.

1. Conditional Jump Instructions

Conditional jump instructions transfer control to a specified address only if a specific condition is met. These conditions depend on the status of the flag register after an arithmetic or logical operation.

Table 9 Summary of various jump instructions

Instruction	Mnemonic	Condition	Example
Jump If Zero	JZ addr	Z = 1 (Zero flag set)	JZ 2000H (Jump to 2000H if the result is zero)
Jump If Not Zero	JNZ addr	Z = 0 (Zero flag reset)	JNZ 3000H (Jump to 3000H if the result is non-zero)
Jump If Carry	JC addr	CY = 1 (Carry flag set)	JC 2500H (Jump to 2500H if carry is generated)
Jump If No Carry	JNC addr	CY = 0 (Carry flag reset)	JNC 4000H (Jump to 4000H if no carry)
Jump If Positive	JP addr	S = 0 (Sign flag reset)	JP 5000H (Jump to 5000H if the result is positive)
Jump If Minus	JM addr	S = 1 (Sign flag set)	JM 4500H (Jump to 4500H if the result is negative)
Jump If Parity Even	JPE addr	P = 1 (Parity flag set)	JPE 5500H (Jump to 5500H if parity is even)
Jump If Parity Odd	JPO addr	P = 0 (Parity flag reset)	JPO 6000H (Jump to 6000H if parity is odd)

Program Using Conditional Jump:

```
        MVI A, 05H    ; Load A with 05H
        CPI 05H       ; Compare A with 05H
        JZ MATCH      ; Jump to MATCH if A = 05H
        HLT           ; Halt if not matched
MATCH:  MVI B, 10H    ; Load B with 10H
        HLT           ; Halt the program
```

If A equals 05H, control jumps to the MATCH label; otherwise, the program halts.

2. Unconditional Jump Instructions

Unconditional jumps always transfer control to a specified address, regardless of any conditions.

Instruction	Mnemonic	Operation	Example
Jump	JMP addr	Unconditionally jumps to specified address	JMP 2000H (Jump to 2000H)

Program Using Unconditional Jump:

```
LXI H, 2500H  ; Load HL with 2500H
JMP 3000H     ; Unconditionally jump to 3000H
HLT           ; This instruction is skipped
```

4. Arithmetic Instructions in 8085

Arithmetic instructions in the 8085 microprocessor perform basic mathematical operations such as addition, subtraction, increment, and decrement. These operations primarily involve the accumulator (A) since the 8085 is an accumulator-based microprocessor. The results of these operations affect the flag register, updating flags such as Zero (Z), Sign (S), Carry (CY), Auxiliary Carry (AC), and Parity (P) to indicate the status of computations.

Below are the different arithmetic instructions used in the 8085 microprocessor, along with their examples and descriptions.

1. Addition Instructions

(i) ADD R (Add Register to Accumulator)

This instruction adds the contents of a register (B, C, D, E, H, or L) to the accumulator (A) and stores the result in the accumulator.

Opcode: ADD R

Example: ADD B (Adds the contents of register B to A and stores the result in A)

(ii) ADD M (Add Memory to Accumulator)

This instruction adds the contents of the memory location pointed to by the HL register pair to the accumulator.

Opcode: ADD M

Example: If HL = 2500H and Memory[2500H] = 10H, executing ADD M will add 10H to A.

(iii) ADI Data (Add Immediate Data to Accumulator)

This instruction adds an 8-bit immediate value to the accumulator.

Opcode: ADI 8-bit data

Example: ADI 05H (Adds 05H to A and stores the result in A)

(iv) ADC R (Add Register to Accumulator with Carry)

This instruction performs addition along with the carry (CY) flag. It adds the contents of a register and the carry flag to the accumulator.

Opcode: ADC R

Example: If A = 08H, B = 02H, and CY = 1, executing ADC B results in A = 0BH (08H + 02H + 1).

(v) ADC M (Add Memory to Accumulator with Carry)

This instruction adds the memory contents (pointed by HL) and the carry flag to the accumulator.

Opcode: ADC M

Example: If HL = 2500H, Memory[2500H] = 15H, and CY = 1, executing ADC M will add 15H + Carry to A.

(vi) ACI Data (Add Immediate Data with Carry to Accumulator)

This instruction adds an 8-bit immediate value and the carry flag to the accumulator.

Opcode: ACI 8-bit data

Example: ACI 10H (Adds 10H and Carry to A and stores the result in A)

2. Subtraction Instructions

(i) SUB R (Subtract Register from Accumulator)

This instruction subtracts the contents of a register from the accumulator (A) and stores the result in A.

Opcode: SUB R

Example: SUB C (Subtracts the contents of register C from A and stores the result in A)

(ii) SUB M (Subtract Memory from Accumulator)

This instruction subtracts the contents of a memory location (pointed by HL) from the accumulator.

Opcode: SUB M

Example: If Memory[2500H] = 05H, executing SUB M will subtract 05H from A.

(iii) SUI Data (Subtract Immediate Data from Accumulator)

This instruction subtracts an 8-bit immediate value from the accumulator.

Opcode: SUI 8-bit data

Example: SUI 0AH (Subtracts 0AH from A and stores the result in A)

(iv) SBB R (Subtract Register from Accumulator with Borrow)

This instruction subtracts the contents of a register and the carry (borrow) flag from the accumulator.

Opcode: SBB R

Example: If A = 10H, D = 05H, and CY = 1, executing SBB D results in A = 0AH (10H - 05H - 1).

(v) SBB M (Subtract Memory from Accumulator with Borrow)

This instruction subtracts the memory contents (pointed by HL) and the borrow flag from the accumulator.

Opcode: SBB M

Example: If Memory[2500H] = 08H, CY = 1, executing SBB M subtracts 08H and CY from A.

(vi) SBI Data (Subtract Immediate Data with Borrow from Accumulator)

This instruction subtracts an 8-bit immediate value and the borrow flag from the accumulator.

Opcode: SBI 8-bit data

Example: SBI 05H (Subtracts 05H and Carry from A and stores the result in A)

(iv) DCR M (Decrement Memory by 1)

This instruction decrements the contents of the memory location pointed by HL.

Opcode: DCR M

Example: If Memory[3000H] = 25H, executing DCR M results in Memory[3000H] = 24H.

(v) INX RP (Increment Register Pair by 1)

This instruction increments a 16-bit register pair (BC, DE, or HL) by 1.

Opcode: INX RP

Example: INX H (Increments HL register pair by 1)

(vi) DCX RP (Decrement Register Pair by 1)

This instruction decrements a 16-bit register pair by 1.

Opcode: DCX RP

Example: DCX B (Decrements BC register pair by 1)

Arithmetic instructions in the 8085 microprocessor allow performing addition, subtraction, increment, and decrement operations efficiently. Since 8085 is an accumulator-based microprocessor, most arithmetic operations involve register A (accumulator). These instructions play a crucial role in data manipulation, counters, and mathematical computations, making them essential for programming microprocessor-based systems.

5. Increment Instructions In 8085

The 8085 microprocessor provides different types of increment instructions to increase the value of registers, memory locations, and register pairs. These instructions modify the value stored in the specified location.

1. *INR (Increment Register or Memory)*

Mnemonic: INR R (for register)/INR M (for memory)

Operation: Increments the 8-bit contents of a register or a memory location by 1. Affects Zero (Z), Sign (S), Parity (P), and Auxiliary Carry (AC) flags but does not affect the Carry (CY) flag.

Incrementing Register (INR B)

```
MVI B, 05H      ; Load B with 05H
INR B       ; Increment B → B = 06H
```

Incrementing Memory Location (INR M)

```
LXI H, 2000H    ; Load HL with address 2000H
```

```
INR M          ; Increment contents at memory location 2000H
```

2. *INX (Increment Register Pair)*

Mnemonic: INX RP

Operation: Increments the 16-bit contents of a register pair (BC, DE, HL, SP) by 1. No flags are affected.

Incrementing HL Register Pair (INX H)

```
LXI H, 2500H   ; Load HL with 2500H
INX H          ; HL = HL + 1 → HL = 2501H
```

Incrementing Stack Pointer (INX SP)

```
LXI SP, 3000H  ; Load SP with 3000H
INX SP         ; SP = 3001H
```

6. Simple Assembly Language Programs for 8085

The following 8085 assembly language programs demonstrate basic operations such as data transfer, arithmetic operations, looping, and I/O handling.

i Move Data from One Register to Another

This program moves data from register **B** to register **C**.

```
        MVI B, 55H      ; Load register B with 55H
        MOV C, B        ; Copy contents of B to C
        HLT             ; Halt the program
```

ii Addition of Two 8-bit Numbers

This program adds two 8-bit numbers and stores the result in the accumulator.

```
        MVI A, 25H      ; Load first number (25H) into accumulator
        MVI B, 15H      ; Load second number (15H) into register B
        ADD B           ; Add B to A
        HLT             ; Halt the program
```

iii Increment a Number 10 Times Using a Loop

This program increments a number 10 times using a loop.

```
        MVI C, 0AH      ; Load counter with 10
        MVI A, 00H      ; Initialize accumulator with 0
LOOP:   INX A           ; Increment A
        DCR C           ; Decrement counter
        JNZ LOOP        ; Jump to LOOP if C ≠ 0
        HLT     ; Halt the program
```

Chapter 6. 8051 Microcontroller

A microcontroller is a compact, self-contained computing device designed to perform specific control-oriented tasks. Unlike a microprocessor, which requires external memory and peripherals, a microcontroller integrates a CPU, memory (RAM and ROM), input/output (I/O) ports, timers, and serial communication interfaces on a single chip. Microcontrollers are commonly used in embedded systems, where they control devices such as home appliances, automotive systems, medical instruments, and industrial automation equipment.

1. Introduction to the 8051 Microcontroller

The 8051 microcontroller, developed by Intel in 1980, is one of the most widely used 8-bit microcontrollers. It is based on the Harvard architecture, meaning it has separate memory spaces for program instructions and data. The 8051 family includes several variants, with different manufacturers producing compatible versions that extend its functionality.

i Difference Between Microprocessor and Microcontroller

Although microprocessors and microcontrollers share similarities in architecture and function, they are distinct in their design and application. A microprocessor is a general-purpose processing unit that requires external components such as memory, input/output interfaces, and peripheral devices to function. It is primarily used in systems where flexibility and computational power are required, such as personal computers and high-performance computing devices.

A microcontroller, on the other hand, is a compact integrated circuit that combines a processor, memory, and input/output peripherals on a single chip. It is designed for specific control-oriented applications, such as embedded systems in automobiles, industrial automation, consumer electronics, and medical devices. Due to its integrated design, a microcontroller typically operates with lower power consumption and requires minimal external hardware compared to a microprocessor-based system.

The key differences between a microprocessor and a microcontroller can be summarized as follows. A microprocessor is suited for high-speed data processing applications that demand complex computations and external memory management, whereas a microcontroller is optimized for real-time control applications where efficiency, low power consumption, and compact design are priorities. Microcontrollers often include additional features such as analog-to-digital converters (ADC), timers, and communication protocols, making them highly suitable for embedded applications.

The 8085 microprocessor and 8051 microcontroller differ in terms of architecture, instruction set, and intended applications.

ii Instruction Set Differences

8085 has an instruction set designed mainly for arithmetic and logical operations, control instructions, and memory interfacing. 8051 includes additional instructions for bitwise

operations, I/O operations, and timer/counter handling, making it suitable for embedded applications.

iii Architecture Differences

Table 10 Differences between microprocessor and microcontroller

Feature	8085 Microprocessor	8051 Microcontroller
Type	General-purpose microprocessor	Embedded system microcontroller
Data Bus	8-bit	8-bit
Address Bus	16-bit (can address 64 KB memory)	16-bit (can address 64 KB external memory)
Internal Memory	No built-in RAM or ROM (requires external memory)	It has 4 KB ROM and 128 bytes RAM
I/O Ports	Needs external interfacing	Has four built-in 8-bit I/O ports
Timers/Counters	Not available	Two built-in 16-bit timers

2. Architecture of the 8051 Microcontroller

The 8051 microcontroller is based on the Harvard architecture, where program memory and data memory are separate, allowing simultaneous instruction execution and data access. It is an 8-bit microcontroller that includes an on-chip CPU, memory, input/output ports, timers, interrupts, and a serial communication interface.

Main Components of the 8051 Architecture

i Central Processing Unit (CPU)

The CPU is responsible for fetching, decoding, and executing instructions stored in program memory. It controls all internal operations and coordinates data transfer between registers, memory, and peripherals. The 8051 follows a single-cycle instruction execution process for most instructions, ensuring efficient operation.

ii Memory Organization

The 8051 microcontroller has two types of memory: Program Memory (ROM): The 8051 includes 4 KB of on-chip ROM for storing program instructions. However, external program memory up to 64 KB can be interfaced for larger applications. Data Memory (RAM): The microcontroller contains 128 bytes of on-chip RAM, which is divided into:

32 General-Purpose Registers (R0–R7) organized into four banks.

16-bit Addressable Registers (Bit-addressable space: 20H to 2FH).

A general-purpose memory area (30H to 7FH) for temporary data storage.

Special Function Registers (SFRs) used for controlling I/O operations, timers, interrupts, and serial communication.

iii Special Function Registers (SFRs)

SFRs are dedicated registers that control various functionalities of the microcontroller. Some important SFRs include:

Accumulator (A) – Used for arithmetic and logical operations.

B Register – Used for multiplication and division instructions.

Program Status Word (PSW) – Contains status flags (carry, auxiliary carry, overflow, parity, etc.).

Stack Pointer (SP) – Points to the top of the stack in RAM.

Data Pointer (DPTR) – A 16-bit register used for external memory addressing.

Timer Registers (TCON, TMOD, TH0, TL0, TH1, TL1) – Control timer/counter operations.

iv Input/Output Ports

The 8051 has four 8-bit bidirectional I/O ports:

Port 0 (P0) – Acts as a multiplexed address/data bus when interfacing with external memory.

Port 1 (P1) – A simple I/O port with no additional functionality.

Port 2 (P2) – Used for higher-order address bytes in external memory operations.

Port 3 (P3) – Contains special function pins for serial communication, interrupts, and timers.

v Timers and Counters

The 8051 microcontroller has two 16-bit timers/counters (Timer 0 and Timer 1) used for time delays, event counting, and waveform generation. These timers operate in different modes (Mode 0 to Mode 3) to handle various timing and counting functions.

vi Interrupt System

The 8051 supports five interrupt sources, which enable efficient handling of external and internal events. These include:

External Interrupts (INT0 and INT1).

Timer Interrupts (TF0 and TF1).

Serial Communication Interrupt (RI/TI).

The microcontroller prioritizes interrupts using an interrupt enable register (IE) and interrupt priority register (IP).

vii Serial Communication Interface

The 8051 features a full-duplex UART (Universal Asynchronous Receiver-Transmitter) for serial communication. The SBUF register is used for transmitting and receiving data, while

the SCON register configures the serial communication mode. The microcontroller can communicate with external devices via RS-232, SPI, or I2C protocols.

viii Clock and Oscillator Circuit

The 8051 requires an external crystal oscillator (typically 12 MHz) to generate the clock signal, which synchronizes all operations. The machine cycle in the standard 8051 takes 12 clock cycles, making the execution speed 1 MIPS (Million Instructions Per Second) at 12 MHz.

The 8051 microcontroller architecture is optimized for control applications with its efficient CPU, memory management, I/O capabilities, timer/counter operations, interrupt handling, and serial communication support. Its compact design and built-in peripherals make it widely used in embedded systems, industrial automation, consumer electronics, and automotive applications.

3. I/O Ports in 8051 Microcontroller

The 8051 microcontroller has four 8-bit parallel I/O ports (P0, P1, P2, and P3), each consisting of 8 bidirectional lines. These ports allow the microcontroller to interface with external devices such as sensors, displays, and communication peripherals.

Port 0 (P0) – Serves as a multiplexed address and data bus when interfacing with external memory. It requires external pull-up resistors when used as an I/O port.

Port 1 (P1) – A simple bidirectional I/O port that does not have any alternate functions. It is used for general-purpose input/output operations.

Port 2 (P2) – Acts as an I/O port or higher-order address lines (A8–A15) when accessing external memory.

Port 3 (P3) – Functions as an I/O port, but also has alternate functions, such as:

P3.0 (RXD) & P3.1 (TXD): Serial communication pins.

P3.2 (INT0) & P3.3 (INT1): External interrupts.

P3.4 (T0) & P3.5 (T1): Timer inputs.

P3.6 (WR) & P3.7 (RD): External memory read/write control.

Each I/O port in 8051 can be programmed as input or output by configuring the corresponding bits in the Special Function Registers (SFRs).

4. Basic Concept of Memory in 8051

The 8051 microcontroller follows Harvard architecture, meaning it has separate memory spaces for program and data storage. It includes both internal and external memory options to store instructions and data.

Program Memory (ROM): The 8051 has 4 KB of internal ROM, used to store the program code. If additional space is needed, external ROM (up to 64 KB) can be interfaced using the PSEN̄ (Program Store Enable) signal.

Data Memory (RAM): The microcontroller includes 128 bytes of internal RAM, divided into:

General-purpose RAM (30H–7FH): Used for temporary data storage.

Register Banks (00H–1FH): Four banks of registers (R0–R7) for quick data access.

Bit-Addressable Memory (20H–2FH): Allows bitwise operations on 16 bytes.

Stack Memory: Located within internal RAM and managed using the Stack Pointer (SP).

External Data Memory (up to 64 KB) can be connected via Port 0 and Port 2 for applications requiring larger data storage.

i Basic Idea of Addressing Modes in 8051

Addressing modes define how an instruction specifies the operand (data or memory location) to be processed. The 8051 microcontroller supports five addressing modes:

Immediate Addressing Mode: The operand is directly specified in the instruction.

Example: MOV A, #25H (Move 25H into the accumulator)

Register Addressing Mode: The operand is stored in a register inside the CPU.

Example: MOV A, R2 (Move contents of register R2 into the accumulator)

Direct Addressing Mode: The operand is located in internal RAM or SFRs, and the instruction provides the exact address.

Example: MOV A, 40H (Move data from RAM address 40H into A)

Indirect Addressing Mode: The address of the operand is stored in a register (R0 or R1), which acts as a pointer.

Example: MOV A, @R0 (Move data from memory location pointed by R0 into A)

Indexed Addressing Mode: Used to access program memory (ROM), commonly for lookup tables.

Example: MOVC A, @A+DPTR (Move contents of address (A + DPTR) into A)

These addressing modes provide flexibility and efficiency in handling data manipulation, memory access, and instruction execution in the 8051 microcontroller.

5. Basic Instructions in 8051 Microcontroller

The 8051 microcontroller supports a variety of instructions for data transfer, arithmetic operations, logical operations, and program control. These instructions help in executing tasks such as moving data, performing calculations, making decisions, and controlling program flow. Below are some commonly used 8051 instructions, categorized based on their functionality, along with examples.

i Data Transfer Instructions

MOV (Move Data)

This instruction copies data from one location to another without altering the source.

Example 1: MOV A, #30H → Loads 30H into the accumulator.

Example 2: MOV R0, A → Copies the accumulator's contents to register R0.

MOVX (Move External Data)

Transfers data between the accumulator and external RAM (used in systems with external memory).

Example: MOVX A, @DPTR → Moves data from the external memory location (pointed by DPTR) into A.

MOVC (Move Code Memory)

Used to fetch data from program memory (ROM).

Example: MOVC A, @A+DPTR → Moves data from (A + DPTR) in ROM to A (used in lookup tables).

ii Arithmetic Instructions

ADD (Addition)

Performs 8-bit addition between the accumulator and another register or memory location.

Example 1: ADD A, R3 → Adds R3 to A, storing the result in A.

Example 2: ADD A, #25H → Adds 25H to A.

SUBB (Subtraction with Borrow)

Subtracts a value from A, including the carry (borrow) flag.

Example: SUBB A, R5 → Subtracts R5 and carry flag from A.

INC (Increment)

Increases the value of a register or memory location by 1.

Example 1: INC A → Increments the accumulator by 1.

Example 2: INC DPTR → Increments the data pointer (DPTR).

DEC (Decrement)

Decreases the value of a register or memory location by 1.

Example: DEC R2 → Decreases the value in R2 by 1.

iii Logical Instructions

ANL (Logical AND)

Performs a bitwise AND between the accumulator and another operand.

Example: ANL A, #0FH → Performs A = A AND 0FH.

ORL (Logical OR)

Performs a bitwise OR operation.

Example: ORL A, R1 → Performs A = A OR R1.

XRL (Exclusive-OR)

Performs a bitwise XOR operation.

Example: XRL A, #AAH → Performs A = A XOR AAH.

CPL (Complement)

Inverts all bits in the accumulator.

Example: CPL A → Converts all 1s to 0s and 0s to 1s in A.

iv Branching (Jump) Instructions

SJMP (Short Jump)

Performs a relative jump within a 256-byte range from the current instruction address.

Example: SJMP LABEL → Jumps to LABEL if within -128 to +127 bytes.

LJMP (Long Jump)

Performs a direct jump anywhere in the 64 KB program memory.

Example: LJMP 2000H → Jumps to address 2000H.

JZ (Jump if Zero)

Jumps to a specified address if A = 0.

Example:

```
MOV A, #00H
JZ TARGET  ; Jumps to TARGET if A is zero.
```

JNZ (Jump if Not Zero)

Jumps to a specified address if A ≠ 0.

Example:

```
MOV A, #10H
JNZ TARGET  ; Jumps to TARGET if A is non-zero.
```

v Special Instructions

NOP (No Operation)

Performs no operation but consumes one machine cycle, useful for delays and debugging.

Example: NOP

RET (Return from Subroutine)

Returns control from a subroutine to the main program.

Example:

```
CALL SUBROUTINE
...
SUBROUTINE:
RET
```

CLR (Clear Register)

Clears the contents of a register, setting it to 0.

Example: CLR A → Sets A = 00H.

The 8051 microcontroller provides a rich set of instructions for data transfer, arithmetic, logical operations, branching, and control flow. These instructions enable efficient data processing, decision-making, and program execution in embedded systems. Understanding and using these instructions correctly is essential for effective assembly language programming on the 8051 microcontroller.

vi Applications of the 8051 Microcontroller

The 8051 microcontroller is widely used in various embedded systems and automation applications due to its low power consumption, compact size, built-in peripherals, and cost-effectiveness. Below are some key applications where the 8051 microcontroller is commonly used:

vii Consumer Electronics

Used in home appliances such as microwave ovens, washing machines, air conditioners, and refrigerators for automation and control. Found in TV remote controls, DVD players, and digital cameras for user interface and system control.

viii Automotive Systems

Controls engine management systems (ECUs) for fuel injection, ignition timing, and emissions control. Used in antilock braking systems (ABS), airbags, and power steering control units. Plays a role in speedometers, odometers, and other dashboard displays.

ix Industrial Automation

Employed in automated manufacturing and process control systems for monitoring and controlling industrial equipment. Used in robotics applications to manage actuators, sensors, and motion control systems. Plays a role in programmable logic controllers (PLCs) for industrial automation.

x Medical Devices

Integrated into heart rate monitors, blood pressure measuring devices, and glucose meters. Used in electronic stethoscopes, pacemakers, and infusion pumps for precise control.

xi Security Systems

Used in biometric authentication systems such as fingerprint scanners and facial recognition devices. Found in home security alarms, surveillance cameras, and access control systems.

xii Embedded Systems and IoT

Integrated into Internet of Things (IoT) applications for smart automation, remote monitoring, and sensor-based control. Used in smart traffic control systems for real-time vehicle management.

The 8051 microcontroller continues to be widely used in modern embedded systems due to its simplicity, flexibility, and ease of integration with external devices.

6. Simple Programs for the 8051 Microcontroller

Below are four simple assembly language programs for the 8051 microcontroller, demonstrating basic operations such as data transfer, arithmetic operations, and looping.

Move Data from One Register to Another

This program moves data from register R1 to register R2.

```
MOV R1, #55H   ; Load R1 with 55H
```

```
MOV R2, R1    ; Copy the contents of R1 to R2
END           ; End of the program
```

Explanation:

MOV R1, #55H → Loads the immediate value 55H into register R1.

MOV R2, R1 → Copies the contents of R1 into R2.

Addition of Two Numbers

This program adds two 8-bit numbers and stores the result in the accumulator.

```
MOV A, #25H   ; Load first number (25H) into accumulator
ADD A, #15H   ; Add second number (15H) to accumulator
END           ; End of the program
```

Explanation:

MOV A, #25H → Loads 25H into the accumulator (A).

ADD A, #15H → Adds 15H to the accumulator. The result is stored in A.

Loop to Increment a Number 10 Times

This program increments a number in register R0 ten times using a loop.

```
              MOV R0, #00H   ; Initialize R0 with 00H
              MOV R1, #0AH   ; Load counter with 10 (0AH)

LOOP:         INC R0         ; Increment R0 by 1
              DJNZ R1, LOOP  ; Decrement R1 and repeat if not zero

              END            ; End of the program
```

Explanation:

MOV R0, #00H → Initializes R0 with 0H.

MOV R1, #0AH → Loads 10 (0AH) into R1 (loop counter).

INC R0 → Increments R0.

DJNZ R1, LOOP → Decrements R1 and jumps back to LOOP until R1 = 0.

Sending Data to an Output Port (LED Blinking Example)

This program sends data to Port 1 (P1) to control an LED or output device.

```
              MOV P1, #0FFH  ; Turn ON all LEDs connected to Port 1
              CALL DELAY     ; Call delay subroutine
              MOV P1, #00H   ; Turn OFF all LEDs
              CALL DELAY     ; Call delay subroutine
              SJMP HERE      ; Repeat indefinitely

DELAY:        MOV R7, #0FFH  ; Outer loop
              MOV R6, #0FFH  ; Inner loop
D1:           DJNZ R6, D1    ; Decrement R6 and repeat
              DJNZ R7, DELAY ; Decrement R7 and repeat
              RET            ; Return from subroutine
```

Explanation:

MOV P1, #0FFH → Sends FFH (all 1s) to Port 1, turning ON all LEDs.

CALL DELAY → Calls the delay subroutine to create a time delay.

MOV P1, #00H → Sends 00H (all 0s) to turn OFF the LEDs.

SJMP HERE → Jumps back to repeat the process.

DELAY subroutine creates a delay using nested loops.

These basic 8051 programs demonstrate essential data movement, arithmetic, loops, and I/O control. They form the foundation for more advanced embedded system applications using the 8051 microcontroller.

Microprocessor MCQ (Multiple Choice Questions)

1. How many memory locations are available in 8085 µp ___
   a) 32K
   b) 64K
   c) 16K
   d) 128K
2. Interfacing of memory chip 6116 (2K bytes) with 8085 µp requires how many address lines
   a) 10
   b) 11
   c) 12
   d) 13
3. A hypothetical microprocessor has 10 bits of address buses, then total memory locations available is __
   a) 1K
   b) 2K
   c) 4K
   d) 8K
4. What are the two 16-bit registers available in 8085 µp ?
   a) SP, PC
   b) A, B
   c) A, Flag register
   d) A, increment/decrement latch
5. The microprocessor of a computer can operate on any information if it is present in only.
   a) Program Counter
   b) Flag
   c) Memory
   d) None of these
6. Which of the following technologies was used by Intel to design its first 8-bit microprocessor?
   a) NMOS
   b) HMOS
   c) PMOS
   d) TTL
7. A 'DMA' transfer implies
   a) direct transfer of data between memory and accumulator.
   b) Direct transfer of data between memory and I/O devices without use of µP

c) Transfer of data exclusively within µP registers

d) A fast transfer of data between µP and I/O devices

8. In 8-bit microprocessor, how many opcodes are theoretically possible?

a) 246

b) 278

c) 250

d) 256

9. Which of the following is not true about the address bus?

a) It consists of control PIN 21 to 28

b) It is a bidirectional bus

c) It is 16 bits in length

d) Lower address bus lines (AD0 – AD7) are called "Line number"

10. Which of the following is true about 8085 microprocessors?

a) It has an internal memory

b) It has interfacing circuits

c) It contains ALU, CU, and registers

d) None of these

11. Which of the following is the possible sequence of operations in a microprocessor?

a) Opcode fetch, memory read, memory write,

b) Opcode fetch, memory write, memory read,

c) I/O read, opcode fetch, memory read

d) I/O read, opcode fetch, memory write

12. Which of the following is not a property of TRAP interrupt in 8085 a microprocessor?

a) It is a non-maskable interrupt

b) It is of the highest priority

c) It uses an edge-triggered signal

d) It is a vectored interrupt

13. Which of the following is a property of RST 7.5 interrupt?

a) It is a non-maskable interrupt

b) It has the third highest priority

c) It uses a level-triggered signal

d) Its vectored address is 0034H

14. Which of the following is a special-purpose register of microprocessor?

a) Program counter

b) Instruction register

c) Accumulator

d) All of these

15. Which of the following circuits is used as a special signal to demultiplex the address bus and data bus?
    a) Priority Encoder
    b) Decoder
    c) Address Latch Enable
    d) Demultiplexer
16. How many flip-flops are there in a flag register of 8085 microprocessor?
    a) 4
    b) 5
    c) 7
    d) 10
17. Which of the following flags are used for BCD arithmetic operations in microprocessor?
    a) Sign flag
    b) Auxiliary Carry flag
    c) Parity flag
    d) Zero flag
18. What does a microprocessor understand after decoding opcode?
    a) Perform ALU operation
    b) Go to memory
    c) Length of the instruction and number of operations
    d) Go to the output device
19. How many address lines are present in 8086 microprocessor?
    a) 16
    b) 20
    c) 32
    d) 40
20. Which of the following is a status flag in 8085 μp?
    a) Parity flag
    b) Direction flag
    c) Interrupt flag
    d) Index flag
21. Which of the following is not a status flag in 8085?
    a) Trap flag
    b) Auxiliary Carry flag
    c) Parity flag
    d) Zero flag
22. Which of the following register is not used in opcode fetch operations?

a) Program counter

b) Memory address register

c) Memory data register

d) Flag register

23. A memory connected to a microprocessor has 20 address lines and 16 data lines. What will the memory capacity be?

a) 8 KB

b) 2 MB

c) 16 MB

d) 64 KB

24. Which of the following is not true about 8085 microprocessor?

a) It is an 8-bit microprocessor

b) It is a 40 pin DIP chip

c) It is manufactured using PMOS technology

d) It has 16 address lines

25. Which of the following is a non-vectored input?

a) TRAP

b) RST-7.5

c) RST-6.5

d) INTR

26. Which of the following is true for 8085 μp?

a) Every instruction has two parts i.e., opcode and operands

b) MOV B, C is a two-byte instruction

c) MVI A, 90H is a three-byte instruction

d) Maximum number of T-states possible for the execution of an instruction is 16

27. What can be stored in the HL general-purpose register pair?

a) Opcode

b) Address of memory

c) Address of next instruction

d) Temporary data

28. If an 8KB memory has to be connected to an 8085 μp, how many address lines are required?

a) 11

b) 12

c) 13

d) None of these

29. Which of the following is a software interrupt?

a) TRAP

b) INTR

c) RST-6.5

d) RST-5

30. What is the vectored address of RST-5?

a) 0010 H

b) 0032 H

c) 0028 H

d) 0030 H

31. Which of the following is true about stack pointer?

a) Stack pointer contains the address of the top of the stack memory

b) Stack pointer is an 8-bit register

c) Stack pointer stores data permanently

d) Stack pointer is initialized after stack operation

32. If a peripheral is interfaced with 8085 μp in memory mapped I/O mode then it has 16-bit address.

a) True

b) False

c) It depends on the peripheral used

33. In 8085, CALL instruction requires how many machine cycles?

a) 2

b) 3

c) 4

d) 5

34. Which of the following is true about MOV A, B instruction?

a) It means moving the content of register A to register B

b) It uses immediate addressing mode

c) It does not affect the flag register

d) It is a 2-byte instruction

35. Which of the following is false about LDA instruction?

a) It is a 3-byte instruction

b) It uses indirect addressing mode

c) It has 13 T-states

d) It does not affect any flags

36. Which is of the following is true about STA instruction?

a) It uses immediate addressing mode

b) It is a 3-byte instruction

c) It requires three machine cycles

d) The Accumulator is loaded with the content of memory

37. Which pin is used to interface the slow peripheral with the 8085 μp?

a) RESET

b) Ready

c) TRAP

d) INTR

38. What are two 16-bit registers available in 8051μc?

a) DPTR

b) PC

c) Both a and b

d) None of these

39. Which of the following is true for the DAA instruction in 8085?

a) It is used after hexadecimal addition

b) It is used after BCD addition

c) It is used after BCD subtraction

d) It is used after hexadecimal subtraction

40. Which of the following flag does not have a jump instruction associate with it?

a) CY flag

b) Parity flag

c) Sign flag

d) AC flag

41. Suppose registers 'A' and 'B' contain 50H and 40H, respectively. After instruction MOV A, B, what will be the contents of registers A and B?

a) 40H, 40H

b) 50H, 40H

c) 50H, 50H

d) 60H, 40H

42. Which microprocessor pins are used in the direct memory access operation?

a) RESET

b) Ready

c) SID

d) None of these

43. Which of the following flag does not have a call instruction associate with it in 8085 μp ?

a) Z

b) AC

c) CY

d) P

44. Which of the following is not correct about HLT instruction?

a) It is a machine control instruction

b) It is used to start the execution of the program

c) PC is disconnected from the address bus

d) A reset interrupt is required to come out of halt state

45. When data required for instruction is present inside the register of a microprocessor then which of the following addressing mode is used?

a) Indexed

b) Register

c) Relative

d) Direct

46. Which of the following interfacing IC is a DMA controller?

a) 8257/37

b) 8155

c) 8253/54

d) 8279

47. Which of the following is a 3-byte instruction(s)?

a) LDA 2500H

b) IN 01H

c) Both a and b

d) None of these

48. Which of the following is a register-indirect addressing mode instruction?

a) LDA 2700H

b) ADI 36H

c) DAA

d) LDAX B

49. How many machine cycles are required by the LDA instruction?

a) 2

b) 3

c) 4

d) 5

50. The address range of bit addressable area in 8051 microcontrollers is__

a) 20H–21H

b) 30H–31H

c) 20H–2FH

d) None of these

51. In an 8085 microprocessor, which one of the following instructions changes the content

of the accumulator?

a) MOV B, M

b) PCHL

c) RNZ

d) SBI BEH

52. Write an instruction which is equivalent to 1-byte CALL instruction in 8085.

a) CALL 2050H

b) DAA

c) PUSH PSW

d) RST 1

53. Write an instruction to clear the accumulator.

a) ADD A

b) XRA B

c) XRA A

d) SUB B

54. What is/are the machine cycle(s) used by DCR M instructions in 8085?

a) 1

b) 2

c) 3

d) 4

55. Scratch-pad memory available in 8051μc ranges from __

a) 20H-21H

b) 21H-2FH

c) 30H-3FH

d) None of these

56. The memory address of the last location of an 8K byte memory chip is FFFFH. Find the starting address.

a) 1000H

b) 2000H

c) 1FFFH

d) None of these

57. What are the machine cycles IN 23H instruction have?

a) Opcode fetch, memory read, memory write

b) Opcode fetch, memory read, I/O read

c) Opcode fetch, memory read, memory read

d) Opcode fetch, memory read, I/O write

58. What will be the value of register A after executing the following code?

MVI A, 0FH

CMA

a) F0H

b) 00H

c) F1H

d) F2H

59. Which of the following instructions is used to disable interrupts in the 8085 microprocessor?

a) DI

b) EI

c) HLT

d) NOP

60. Which is of the following is true about SPHL instruction?

a) It uses indexed addressing mode

b) It is a 3-byte instruction

c) It requires three T-states

d) Contents of HL pair are moved to SP

Answers to multiple choice questions

Q no	Answer	Q no	Answer	Q no	Answer	Q no	Answer	Q no	Answer	Q no	Answer
1	b	11	a	21	a	31	a	41	a	51	d
2	b	12	c	22	d	32	a	42	d	52	d
3	a	13	a	23	c	33	d	43	b	53	c
4	a	14	d	24	c	34	c	44	b	54	c
5	c	15	c	25	d	35	b	45	b	55	d
6	a	16	b	26	d	36	b	46	a	56	d
7	b	17	b	27	b	37	b	47	a	57	b
8	d	18	c	28	c	38	c	48	d	58	a
9	b	19	b	29	d	39	b	49	c	59	a
10	c	20	a	30	c	40	d	50	c	60	d

Short Answer Questions

1. What is 8085 microprocessor?

2. What are the basic units or building blocks of an 8085 microprocessor?

3. What is Software and Hardware?

4. What is the purpose of the stack in the 8085 microprocessor?

5. What is assembly language?

6. What are machine language and assembly language programs?

7. Write an 8085-assembly language program to add two numbers stored in registers A and B, and store the result in register C.

8. What are the different types of addressing modes in 8085?

9. What is the drawback in machine language and assembly language programs?

10. Define bit, byte, and word.

11. What is a bus? Why do data bus is bi-directional?

12. Why is address bus unidirectional?

13. What is the function of microprocessor in a system?

14. What are the key features of the 8051 microcontroller?

15. Write an 8051-assembly program to load the value 55H into register A and then complement it.

16. Write and explain any two logical instructions of 8085 microprocessor.

17. What is the function of the Program Counter (PC) in the 8085 microprocessor?

18. What is the function of the TMOD register in the 8051 microcontroller?

19. What is the purpose of the DPTR register in the 8051 microcontroller?

20. How many machine cycles constitute one instruction cycle in 8085? Define opcode and operand.

21. What is an opcode fetch cycle?

22. What operation is performed during the first T -state of every machine cycle in 8085?

23. Why are status signals provided in microprocessor?

24. Can an input port and an output port have the same port address?

25. How does the 8085 processor differentiate a memory (read/write) and I/O access (read/write)?

26. How will the port number be affected if we decode the high-order address lines A_{15} – A_8 rather than A_7 – A_0?

27. If high-order lines are partially decoded, how can one determine whether it is peripheral I/O or memory-mapped I/O?

28. In a memory mapped I/O, how does the microprocessor differentiate between an I/O and memory? Can an I/O have the same address as a memory register?

29. Why is a 16-bit address (data) stored in memory in the reversed order-the low-order byte first, followed by the high-order byte?

30. When does the 8085-processor check for an interrupt? What is an interrupt acknowledge cycle?

31. How are the interrupts affected by system reset?

32. What are Software interrupts?

33. What are Hardware interrupts?

34. What is the difference between Hardware and Software interrupt?

35. What is vectored and non-Vectored interrupt?

36. Whether HOLD has higher priority than TRAP or not?

37. What is masking and why is it required?

38. When does the 8085 processor accept hardware interrupt?

39. When will the 8085 processor disable the interrupt system?

40. What is the function performed by EI and Dl instruction?

41. How is the vector address generated for the INTR interrupt of 8085?

42. How clock signals are generated in 8085 and what is the frequency of the internal clock?

43. Explain the memory structure of the 8051 microcontroller.

44. How does the 8051 handle input and output operations?

45. What happens in a single-board microcomputer when the power is turned on and the Reset key is pushed?

46. How does the microprocessor know how and when to start?

47. What is a monitor program?

48. What is an assembler?

49. How does the microprocessor differentiate among a positive number, a negative number, and a bit pattern?

50. If flags are individual flip-flops, can they be observed on an oscilloscope?

51. If the program counter is always one count ahead of the memory location from which the machine code is being fetched, how does the microprocessor change the sequence of program execution with a Jump instruction?

Descriptive Type Questions

1. Differentiate between 8085 microprocessor and 8051 microcontroller with respect to their architecture and instructions.

2. Describe any four logical instructions and explain them in detail.

3. What are the different pins in the timing and control unit of the 8085 microprocessor? Explain these pins with their functionality.

4. What are the different types of register available in 8085? Classify these registers based on their size, operations performed, user accessibility and other criteria if any.

5. Write a program in 8085 to add two BCD number in 8085 with the use of DAA instruction. Explain the instructions used in the program.

6. Explain conditional and unconditional jump instructions available in the 8085 microprocessor.

7. Define bit, byte, word, double word, quad word, and instruction.

8. List all the interrupt signals available in 8085 microprocessor.

9. List the sequence of events that occur when the 8085 MPU reads from memory.

10. Explain the memory address range of **1K** (**1024 × 8**) memory chip using 3-to-8 decoder and explain the changes in the addresses if the hardware of the $\overline{\mathbf{CS}}$ line is modified.

11. Summarize the requirements of a memory chip and match the requirements with the microprocessor bus concepts.

12. Draw and explain the timing diagram for opcode fetch operation.

13. What are the distinct types of addressing modes available in 8085? Discuss them with example.

14. Specify the control signal and the direction of the data flow on the data bus in a memory-write operation.

15. What is an assembler? What are low- and high-level languages?

16. Write and explain four machine control instructions in 8085.

17. Write an assembly language program in 8085 to add two 16-bit numbers.

18. Write and explain about the rotate instructions in 8085 microprocessor. Give suitable examples.

19. What is the function of program counter and stack pointer registers?

20. What are the various types of increment instructions in 8085, like INR B, INR M, INX H.

21. Differentiate between memory-mapped I/O and I/O-mapped I/O schemes of interfacing the peripherals with a microprocessor.

22. What are the steps in writing and executing an assembly language program?

23. Discuss the various data formats like ASCII code, extended ASCII, BCD code, signed integer, and unsigned integers.

24. Summarize the requirements of the memory chip and match the requirements with the microprocessor bus concepts.

25. Interface a 2732 EPROM (4K bytes) memory chip with 8085μp using a 3-to-8 decoder (74LS138 chip) and external NAND gates.