PERIYAR UNIVERSITY

NAAC 'A++' Grade – State University – NIRF Rank 56- State Public University Rank 25
SALEM - 636 011, Tamil Nadu, India

CENTRE FOR DISTANCE AND ONLINE EDUCATION

(CDOE)

B.SC COMPUTER SCIENCE

SEMESTER - II

CORE COURSE: MICROPROCESSOR AND MICROCONTROLLER

(Candidates admitted from 2024 onwards)

PERIYAR UNIVERSITY

CENTRE FOR DISTANCE AND ONLINE EDUCATION (CDOE)

B.Sc COMPUTER SCIENCE 2024 admission onwards

CORE COURSE
Microprocessor and Microcontroller

Prepared by:

Centre for Distance and Online Education (CDOE)
Periyar University, Salem – 11.

LIST OF CONTENTS

UNIT I

Digital Computers - Microcomputer Organization-Computer languages –Microprocessor Architecture and its operations – Microprocessor initiated operations and 8085 Bus organization – Internal Data operations and 8085 registers - Peripheral or External initiated operations.

Unit Objectives:

The objective of this unit is to provide a foundational understanding of digital computers, focusing on microcomputer organization and computer languages. Students will learn about microprocessor architecture, its operations, and the 8085 bus organization. The course covers internal data operations, 8085 registers, and both microprocessor-initiated and peripheral/external-initiated operations, equipping students with essential knowledge for microprocessor-based systems.

1.1 DIGITAL COMPUTERS

A digital computer is an electronic device that processes data using binary digits (0s and 1s). It performs a wide range of tasks by executing instructions from software programs, making it capable of handling complex calculations, data storage, and information retrieval.

1.1.1 KEY CHARACTERISTICS OF DIGITAL COMPUTERS:

Binary System - Digital computers use the binary number system (0s and 1s) for data representation, simplifying circuit design and ensuring reliable data processing.

High Speed and Accuracy - They perform billions of operations per second with high precision, limited only by the hardware and software's inherent limitations.

Programmability - These computers can execute a wide range of software applications, allowing them to perform diverse tasks and making them highly versatile.

Large Storage Capacity - Digital computers offer significant storage capabilities, enabling the handling and processing of large volumes of data in both primary and secondary storage.

Automation - Once programmed, digital computers can perform tasks automatically without human intervention, executing complex instruction sequences to achieve specific objectives.

1.1.2 FUNCTIONAL COMPONENTS OF A DIGITAL COMPUTER:

1.1.2.1 Central Processing Unit (CPU)

- The brain of the computer, responsible for executing instructions and processing data through its Arithmetic Logic Unit (ALU) and Control Unit (CU).

1.1.2.2 Memory and Storage

- **Memory (RAM and ROM):** RAM provides temporary storage for active data and instructions, while ROM stores permanent firmware essential for booting and basic operations.
- **Storage Devices:** Hard drives and SSDs provide long-term data storage, retaining information even when the computer is powered off.

1.1.2.3 Input and Output Devices

- **Input Devices:** Tools like keyboards, mice, and scanners that allow users to enter data and commands into the computer.
- **Output Devices:** Devices such as monitors, printers, and speakers that present processed data to the user in visual, printed, or audio formats.

1.1.2.4 Motherboard and Bus System

- **Motherboard:** The main circuit board that houses the CPU, memory, and other components, providing connectivity and communication between them.
- **Bus System:** A communication system that transfers data between components inside the computer, including data, address, and control buses.

1.1.2.5 Power Supply and Network Interface

- **Power Supply Unit (PSU):** Converts electrical power from an outlet into usable power for the computer’s internal components, ensuring stable and reliable operation.
- **Network Interface Card (NIC):** Allows the computer to connect to a network, enabling data exchange with other computers and internet connectivity.

1.1.3 BASIC OPERATION OF A DIGITAL COMPUTER:

Input: Receiving data and instructions from input devices like keyboards, mice, and scanners for processing.

Processing: Executing instructions and performing calculations using the CPU to transform input data into meaningful output.

Storage: Saving data and instructions in memory (RAM) for quick access or in storage devices (HDD, SSD) for long-term retention.

Output: Presenting processed data to the user through output devices like monitors, printers, and speakers in visual, printed, or audio formats.

Control: Managing and coordinating the operations of all computer components through the Control Unit (CU) to ensure efficient processing and communication.

Let us sum up:

Digital Computers

1. **Definition**: Digital computers are electronic devices that process data in binary form (0s and 1s) and perform computations based on predefined instructions.
2. **Applications**: They are used in various fields, including business, science, engineering, healthcare, and entertainment, for tasks such as data processing, calculations, and automation.

Key Characteristics of Digital Computers

1. **Accuracy**: Digital computers perform calculations with high precision, minimizing errors compared to analog systems.
2. **Speed**: They can execute millions of instructions per second, enabling rapid processing of data.
3. **Storage Capacity**: Digital computers can store vast amounts of data in various formats, including text, images, and multimedia.

4. **Versatility**: They can perform a wide range of tasks, from simple arithmetic operations to complex simulations and data analysis.
5. **Automation**: Digital computers can automate repetitive tasks and processes, improving efficiency and reducing human intervention.

Functional Components of a Digital Computer

1. **Central Processing Unit (CPU)**: The CPU is the brain of the computer, responsible for executing instructions and managing operations.
2. **Memory**: This includes both primary memory (RAM) for temporary data storage and secondary storage (e.g., hard drives) for long-term data retention.
3. **Input Devices**: Devices like keyboards, mice, and scanners that allow users to enter data into the computer.
4. **Output Devices**: Monitors, printers, and speakers that convey processed information to the user.
5. **Bus System**: A communication system that transfers data between the CPU, memory, and other components.

Basic Operation of a Digital Computer

1. **Input**: Data is entered into the computer through input devices.
2. **Processing**: The CPU processes the input data according to the instructions provided by software applications.
3. **Storage**: Processed data can be temporarily stored in RAM or permanently saved on storage devices.
4. **Output**: The results of the processing are displayed or printed using output devices, making them available for user interpretation or further action.

Check Your Progress

1. What system do digital computers use for data representation?

A) Decimal B) Binary

C) Hexadecimal D) Octal

Answer: B) Binary

2. Which unit processes instructions in a computer?

A) GPU	B) CPU
C) RAM	D) ROM

Answer: B) CPU

3. What is the primary long-term storage device in a digital computer?

A) RAM	B) ROM
C) HDD	D) GPU

Answer: C) HDD

4. Which component allows a computer to connect to a network?

A) PSU	B) RAM
C) NIC	D) HDD

Answer: C) NIC

5. Which bus type is used to transfer memory addresses?

A) Address	B) Data
C) Control	D) Power

Answer: A) Address

1.2 MICROCOMPUTER ORGANIZATION

A microcomputer is a small, relatively inexpensive computer with a microprocessor as its central processing unit (CPU).

Examples: Personal computers (PCs), laptops, tablets, and embedded systems.

1.2.1 INTRODUCTION AND ARCHITECTURE OF MICROCOMPUTERS

Microcomputers, also known as personal computers (PCs), are compact, relatively inexpensive digital computers that use a microprocessor as their central processing unit (CPU). These machines are ubiquitous in modern society, found in homes, offices, and embedded in various devices. They have evolved significantly from early systems, becoming more powerful and capable over time.

Historical Background

The concept of microcomputers emerged in the 1970s with the development of the microprocessor, which integrated the CPU on a single chip. This innovation led to the creation of the first personal computers like the Altair 8800 and the Apple I. Since then, microcomputers have undergone rapid advancement, with improvements in processing power, memory capacity, and connectivity.

Examples of Microcomputers

Microcomputers come in various forms, including desktop PCs, laptops, tablets, and embedded systems in appliances and vehicles. They are distinguished by their ability to perform tasks ranging from basic computing operations to complex calculations and multimedia processing. Today's microcomputers are essential tools in education, business, scientific research, entertainment, and everyday life.

1.2.2 MICROCOMPUTER ARCHITECTURE

Microcomputer architecture consists of several key components:

1. **Central Processing Unit (CPU):**
   - **Arithmetic and Logic Unit (ALU):** Performs arithmetic and logical operations.
   - **Control Unit:** Directs the operation of the processor.
   - **Registers:** Small, fast storage locations for data and instructions.
2. **Memory:**
   - **RAM (Random Access Memory):** Temporary storage for data and instructions.
   - **ROM (Read-Only Memory):** Permanent storage for essential programs.
3. **Input/Output (I/O) Devices:**
   - Interfaces for external devices like keyboards, monitors, and printers.

4. **Bus System:**
   - **Data Bus:** Transfers data.
   - **Address Bus:** Transfers memory addresses.
   - **Control Bus:** Transfers control signals.

These components ensure efficient data processing and communication within a microcomputer.

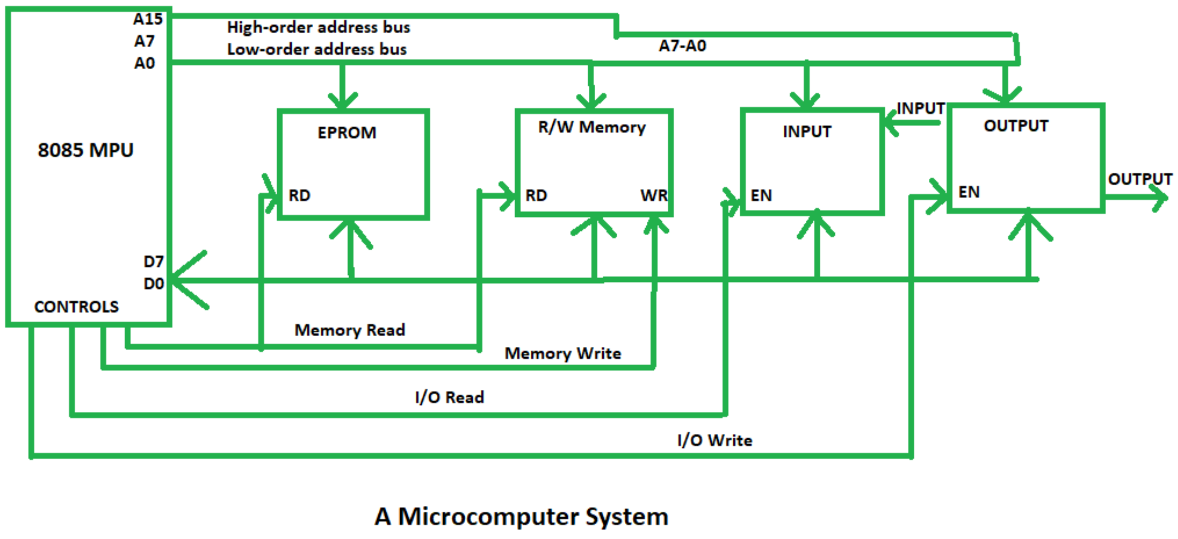


Fig 1. Microcomputer Architecture

Let us sum up:

Microcomputer Organization

1. **Definition**: Microcomputers are small computing devices that consist of a microprocessor, memory, and input/output interfaces, designed for individual use or specific tasks.
2. **Applications**: Commonly used in personal computers, embedded systems, and various consumer electronics for tasks such as word processing, gaming, and data analysis.

Introduction and Architecture of Microcomputers

1. **Architecture Overview**: Microcomputers typically follow a von Neumann architecture, where the CPU, memory, and I/O devices share a common bus for communication.
2. **Key Components**:
   - **Microprocessor**: The central processing unit (CPU) that executes instructions and performs calculations.
   - **Memory**: Includes both volatile (RAM) and non-volatile (ROM) storage for program instructions and data.
   - **Input/Output Interfaces**: Components that allow the microcomputer to interact with external devices, such as keyboards, mice, displays, and printers.

Microcomputer Architecture

1. **Central Processing Unit (CPU)**: Comprises the arithmetic logic unit (ALU), control unit (CU), and registers that handle data processing and instruction execution.
2. **Memory Organization**:
   - **Primary Memory**: RAM for temporary data storage and fast access.
   - **Secondary Memory**: Hard drives, SSDs, and other storage devices for long-term data retention.
3. **Bus Structure**:
   - **Data Bus**: Transfers data between the CPU, memory, and I/O devices.
   - **Address Bus**: Carries the address of the data to be accessed in memory.
   - **Control Bus**: Transmits control signals to coordinate the operations of the microcomputer components.
4. **I/O System**: Manages communication between the microcomputer and peripheral devices, facilitating data input and output operations.

Check Your Progress

1. Which bus transfers data between components in a microcomputer?

A) Address Bus B) Data Bus C) Control Bus D) Power Bus

Answer: B) Data Bus

2. Which type of memory provides temporary storage for active data?

A) RAM B) ROM C) SSD D) HDD

Answer: A) RAM

3. What is the CPU of a microcomputer based on?

A) Mainframe B) Supercomputer

C) Microprocessor D) Minicomputer

Answer: C) Microprocessor

4. Which bus transfers memory addresses?

A) Data Bus B) Control Bus C) Power Bus D) Address Bus

Answer: D) Address Bus

5. Which part of the architecture interfaces with external devices like printers?

A) Bus System B) I/O Devices C) Control Unit D) Registers

Answer: B) I/O Devices

1.3 COMPUTER LANGUAGES

Computer languages are the means by which we communicate instructions to a computer. They are categorized into various levels based on their closeness to machine code and their abstraction from the hardware. The primary types of computer languages include machine language, assembly language, and high-level languages.

1.3.1 Machine Language

Machine language is the most fundamental level of programming language. It consists of binary code (0s and 1s) that the computer's central processing unit (CPU) directly executes.

- **Binary Code**: Instructions are in the form of binary digits.
- **Direct Execution**: Machine language instructions are executed directly by the hardware.
- **Hardware Specific**: Each type of CPU has its own machine language.

1.3.2. Assembly Language

Assembly language is a low-level programming language that uses symbolic names to represent machine-level instructions. It is one step above machine language and provides a more readable way to program.

- **Mnemonic Codes**: Uses mnemonics (short codes) for instructions, such as ADD, SUB, MOV.
- **Assembler Required**: An assembler converts assembly language code into machine language.
- **Hardware Specific**: Assembly language is specific to a particular CPU architecture.

Example:

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

```
ADD A, C  ; Add the contents of register C to register A
```

1.3.3. High-Level Languages

High-level languages provide a greater level of abstraction from the hardware, making programming easier and more efficient. They are designed to be easy to read and write, resembling natural languages or mathematical notation.

- **Easy to Read**: Uses English-like syntax, making the code more understandable.
- **Compiler or Interpreter Required**: A compiler or interpreter translates high-level language code into machine language.
- **Portable**: High-level language programs can run on different types of hardware with minimal modification.

Common high-level languages include:

- **C**: Known for its performance and low-level access to memory.
- **Java**: Object-oriented, platform-independent, widely used in web and enterprise applications.
- **Python**: Known for its simplicity and readability, widely used in web development, data science, and automation.

Example in Python:

```
def add(a, b):

return a + b

result = add(5, 3)

print(result)  # Output: 8
```

Let us sum up:

Computer Languages

1. **Definition**: Computer languages are formal languages comprising a set of instructions used to communicate with computers for programming and performing tasks.
2. **Types**: Computer languages can be broadly categorized into machine language, assembly language, and high-level language, each serving different levels of abstraction and usability.

Machine Language

1. **Definition**: Machine language is the lowest-level programming language, consisting of binary code (0s and 1s) that the computer's CPU directly understands and executes.
2. **Characteristics**:
    - **Hardware-Specific**: Each type of CPU has its own machine language, making it non-portable across different systems.
    - **Efficiency**: Programs written in machine language execute very quickly as they require no translation.
    - **Complexity**: Programming in machine language is error-prone and difficult for humans to read or write due to its binary nature.

Assembly Language

1. **Definition**: Assembly language is a low-level programming language that uses symbolic representations of machine instructions, making it more human-readable than machine language.
2. **Characteristics**:
    - **Mnemonics**: Uses mnemonic codes (e.g., MOV, ADD, SUB) to represent machine instructions, allowing for easier coding and debugging.

- **Hardware-Specific**: Like machine language, assembly language is also specific to a particular CPU architecture.
- **Assembler Required**: Programs written in assembly language must be translated into machine language by an assembler before execution.

High-Level Language

1. **Definition**: High-level languages are programming languages that provide a high level of abstraction from the hardware, allowing programmers to write instructions in a more human-friendly syntax.
2. **Characteristics**:
    - **Portability**: High-level languages are generally portable across different systems and require a compiler or interpreter to translate the code into machine language.
    - **Ease of Use**: They include features like data types, control structures (loops, conditionals), and libraries, making them easier to learn and use for complex applications.
    - **Examples**: Common high-level languages include Python, Java, C++, and JavaScript, which are widely used for application development and software engineering.

Check Your Progress

1. What is used to convert assembly language into machine code?

a) Compiler b) Assembler c) Interpreter d) Emulator

Answer: b) Assembler

2. What language uses mnemonic codes like ADD and MOV?

a) High-level b) Python c) Assembly d) C

Answer: c) Assembly

3. What does an interpreter translate?

a) Machine code	b) High-level language
c) Assembly	d) Binary code

Answer: b) High-level language

4. Which language is known for its simplicity and readability?

a) C b) Assembly c) Java d) Python

Answer: d) Python

5. Which high-level language is known for performance and memory access?

a) Java b) Python c) C d) Ruby

Answer: c) C

1.4 MICROPROCESSOR ARCHITECTURE AND ITS OPERATIONS

The 8085 microprocessor, introduced by Intel in 1976, is an 8-bit microprocessor widely used in the early stages of microprocessor development. It is a part of the MCS-85 family and is designed to address various computing tasks. Understanding its architecture is fundamental for grasping how microprocessors function.

1.4.1 KEY COMPONENTS OF 8085 ARCHITECTURE:

1. **Arithmetic and Logic Unit (ALU):** The ALU performs arithmetic operations (addition, subtraction) and logical operations (AND, OR, NOT, XOR). It works on 8-bit data and sets the flags according to the result of operations.
2. **Registers:** The 8085 has several registers, including:
    - **Accumulator (A):** An 8-bit register used in arithmetic and logical operations.
    - **General Purpose Registers (B, C, D, E, H, L):** These six 8-bit registers can be used individually or in pairs (BC, DE, HL) to hold 16-bit data.

- **Program Counter (PC):** A 16-bit register that holds the address of the next instruction to be executed.
- **Stack Pointer (SP):** A 16-bit register that points to the top of the stack.
- **Temporary Register:** Used internally by the ALU for intermediate results.
- **Instruction Register and Decoder:** Holds the current instruction and decodes it to generate the necessary control signals.
- **Flag Register:** An 8-bit register with five flags (Sign, Zero, Auxiliary Carry, Parity, and Carry) that indicate the status of the ALU operations.

3. **Control Unit:** The control unit coordinates all operations by generating timing and control signals. It controls the data flow between the CPU and peripherals, memory, and input/output devices. It includes the following:
   - **Timing and Control Circuitry:** Generates the necessary control signals.
   - **Instruction Decoder:** Interprets the instructions fetched from memory.
4. **System Bus:** The system bus is used to transfer data, addresses, and control signals between the microprocessor and peripherals. It includes:
   - **Data Bus:** An 8-bit bidirectional bus used for data transfer.
   - **Address Bus:** A 16-bit unidirectional bus used to address memory locations.
   - **Control Bus:** Carries control signals like RD (Read), WR (Write), ALE (Address Latch Enable), etc.
5. **Clock Generator:** The 8085 uses a single-phase clock generator to synchronize its operations. The typical clock speed is 3 MHz.
6. **Interrupts:** The 8085 microprocessor has five interrupt lines (TRAP, RST7.5, RST6.5, RST5.5, INTR) which allow external devices to halt the normal execution of instructions and execute an interrupt service routine.
7. **Serial I/O Control:** The microprocessor includes a serial input (SID) and a serial output (SOD) line for simple serial communication.

The 8085 microprocessor's architecture consists of an ALU, a set of registers, a control unit, a system bus, a clock generator, interrupt control, and serial I/O control. This architecture forms the backbone of the microprocessor, enabling it to perform complex computations and control operations effectively. Understanding these components and their interactions is crucial for working with or designing systems based on the 8085 microprocessor.

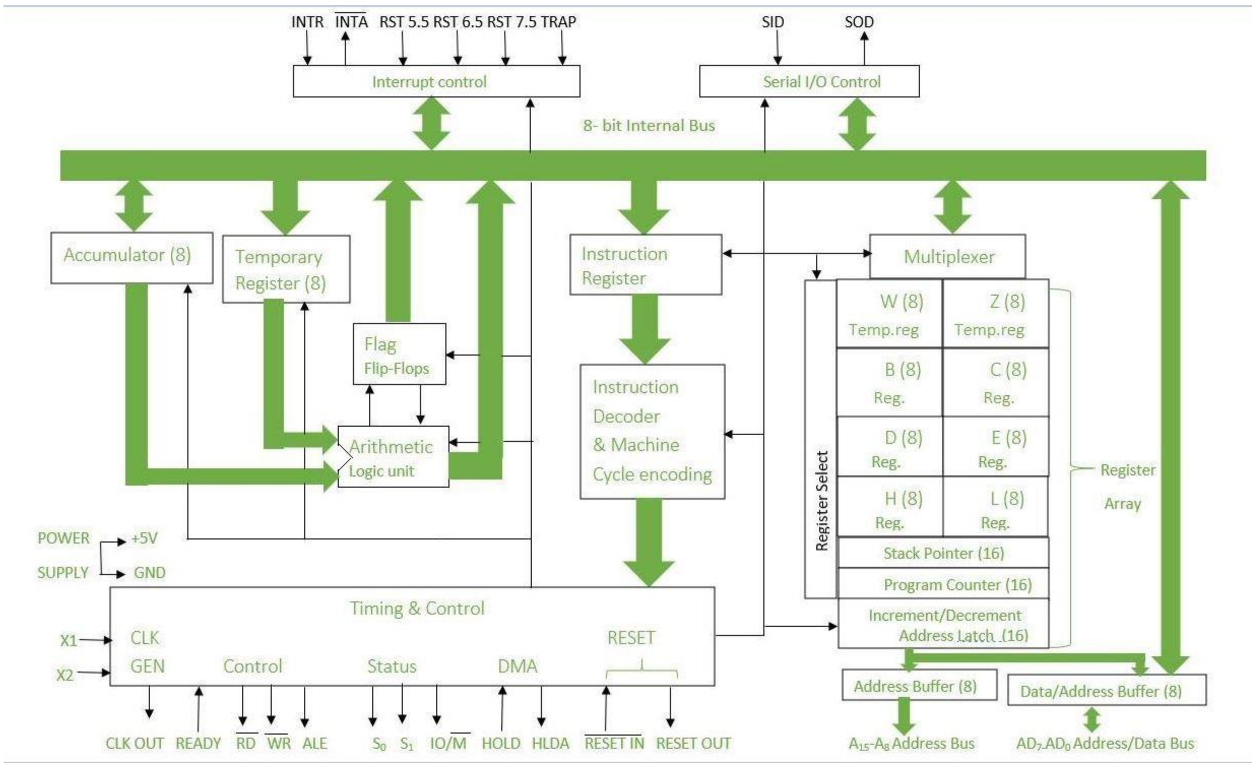


Fig 2. 8085 Architecture

Let us sum up:

Microprocessor Architecture and Its Operations

1. **Definition**: Microprocessor architecture refers to the internal structure and operational organization of a microprocessor, defining how it processes instructions and manages data.
2. **Functions**: It encompasses the design of the CPU, memory interface, input/output systems, and control mechanisms that enable the microprocessor to execute programs effectively.

Key Components of 8085 Architecture

1. **Central Processing Unit (CPU)**:
    - **Arithmetic Logic Unit (ALU)**: Performs arithmetic and logical operations, including addition, subtraction, and comparisons.
    - **Control Unit (CU)**: Directs the operation of the microprocessor, fetching instructions from memory and decoding them for execution.
2. **Registers**:
    - **General-Purpose Registers**: Includes registers A, B, C, D, E, H, and L, which store data and addresses temporarily during processing.
    - **Special Purpose Registers**:
        - **Accumulator (A)**: Used for arithmetic and logic operations; it holds the results of these operations.
        - **Program Counter (PC)**: Holds the address of the next instruction to be executed.
        - **Stack Pointer (SP)**: Points to the top of the stack in memory, used for storing return addresses and temporary data.
3. **Memory Interface**:
    - **Address Bus**: A 16-bit bus used to address memory locations (up to 64KB of memory).
    - **Data Bus**: An 8-bit bus used to transfer data between the microprocessor and memory or I/O devices.
4. **Control Signals**:
    - **Read and Write Signals**: Control the flow of data to and from memory, indicating when data should be read from or written to a specific address.
    - **Interrupt Signals**: Allow the microprocessor to respond to external events or interrupts, enabling multitasking and efficient handling of tasks.
5. **Input/Output Ports**:
    - **I/O Interface**: Facilitates communication between the microprocessor and peripheral devices (such as keyboards, displays, and printers) using various I/O ports.

  - **Memory-Mapped I/O**: The same address space is used for both memory and I/O devices, allowing for flexible addressing and control.

6. **Clock Generator**:
   - Provides the timing signals necessary for synchronizing the operations of the microprocessor and coordinating data transfers.

Check Your Progress

1. **Which unit in the 8085 microprocessor performs arithmetic and logical operations?**

a) Control Unit b) Program Counter c) ALU d) Stack Pointer

Answer: c) ALU

2. **Which register is used to hold the address of the next instruction to be executed?**

a) Accumulator b) Stack Pointer

c) Instruction Register d) Program Counter

Answer: d) Program Counter

3. **Which of the following is a flag in the 8085 microprocessor?**

a) Overflow b) Zero c) Negative d) Auxiliary

Answer: b) Zero

4. What is the size of the address bus in the 8085 microprocessor?

a) 8-bit b) 32-bit c) 16-bit d) 64-bit

Answer: c) 16-bit

5. Which of the following interrupts has the highest priority in the 8085 microprocessor?

a) RST5.5 b) INTR c) RST7.5 d) TRAP

Answer: d) TRAP

1.5 MICROPROCESSOR INITIATED OPERATIONS AND 8085 BUS ORGANIZATION

1.5.1 8085 MICROPROCESSOR INITIATED OPERATIONS

The 8085 microprocessor initiates several operations to manage and control its internal processes. These operations include control signals, status signals, and machine cycles. Understanding these operations is essential for effective programming and hardware interfacing.

1. Control Signals

Control signals are used by the microprocessor to manage data flow between itself and external devices. The primary control signals in the 8085 microprocessor are:

- **RD (Read)**: This signal indicates that the microprocessor is reading data from memory or an I/O device. It is active low.
- **WR (Write)**: This signal indicates that the microprocessor is writing data to memory or an I/O device. It is active low.
- **ALE (Address Latch Enable)**: This signal is used to latch the lower byte of the address into an external latch during the first clock cycle of a machine cycle. It is active high.
- **IO/M**: This signal distinguishes between memory operations (when low) and I/O operations (when high).

2. Status Signals

Status signals provide information about the current state of the microprocessor. They help in coordinating operations with external devices. The primary status signals are:

- **S0 and S1**: These signals indicate the status of the microprocessor and are used to differentiate between various machine cycles.

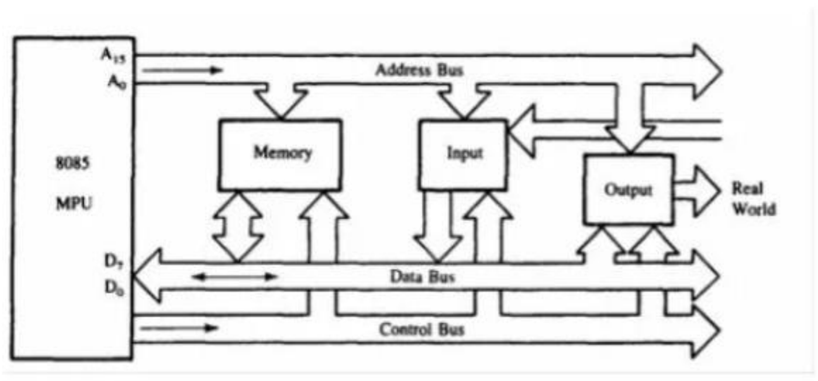
- **READY**: This signal is used to synchronize slower peripheral devices with the microprocessor. When READY is low, the microprocessor enters a wait state until it goes high.

3. Machine Cycles

The 8085 microprocessor executes instructions through a series of machine cycles, each comprising several clock cycles. The main types of machine cycles are:

- **Opcode Fetch Cycle**: This cycle is used to fetch the opcode of an instruction from memory.
- **Memory Read/Write Cycle**: These cycles are used to read data from or write data to memory.
- **I/O Read/Write Cycle**: These cycles are used to read data from or write data to an I/O device.
- **Interrupt Acknowledge Cycle**: This cycle is used to acknowledge an interrupt request and transfer control to the interrupt service routine.

1.5.2 8085 BUS ORGANIZATION



- Address Bus
- Data Bus
- Control Bus

Fig 3. Bus Organization

The bus organization of the 8085 microprocessor is a critical aspect of its architecture, enabling communication between the microprocessor and peripheral devices, memory, and I/O ports. The 8085 microprocessor has three main types of buses: Address Bus, Data Bus, and Control Bus.

1. Address Bus

The address bus is used to specify the address of the memory location or I/O port that the microprocessor intends to read from or write to. Key characteristics include:

- **Width**: The address bus is 16 bits wide, allowing the 8085 to address up to 64KB (2^16) of memory.
- **Unidirectional**: The address bus is unidirectional, meaning that the address flows only from the microprocessor to the memory or I/O device.

The address bus consists of two parts:

- **Higher Order Address Bus (A8-A15)**: These are the higher 8 bits of the address.
- **Lower Order Address/Data Bus (AD0-AD7)**: These multiplexed lines serve as the lower 8 bits of the address and data bus.

2. Data Bus

The data bus is used for transferring data between the microprocessor, memory, and I/O devices. Key characteristics include:

- **Width**: The data bus is 8 bits wide, allowing the transfer of 8-bit data.
- **Bidirectional**: The data bus is bidirectional, meaning data can flow both to and from the microprocessor.

3. Control Bus

The control bus carries control signals from the microprocessor to coordinate and manage the operations of the system. Key control signals include:

- **RD (Read)**: Indicates that the microprocessor is reading data from memory or an I/O port (active low).
- **WR (Write)**: Indicates that the microprocessor is writing data to memory or an I/O port (active low).
- **ALE (Address Latch Enable)**: Used to latch the lower 8 bits of the address during the first clock cycle of a machine cycle (active high).
- **IO/M**: Distinguishes between memory operations (low) and I/O operations (high).

Other control signals include **READY**, **S0/S1** (status signals), and **INTA** (interrupt acknowledge).

Let us sum up:

Microprocessor Initiated Operations

1. **Instruction Fetch**:
   - The microprocessor fetches the instruction from memory using the Program Counter (PC) to identify the memory address.
   - The instruction is loaded into the Instruction Register (IR) for decoding and execution.
2. **Instruction Decode and Execute**:
   - After fetching, the instruction is decoded to determine the required operation and the operands involved.
   - The Control Unit (CU) generates the necessary control signals to execute the instruction, which may involve ALU operations, data transfer, or I/O operations.
3. **Data Transfer**:
   - Data can be moved between registers, memory, and I/O devices based on the instructions.
   - The microprocessor initiates read or write operations, controlling data flow through the data bus.
4. **Arithmetic and Logic Operations**:

  - The Arithmetic Logic Unit (ALU) performs calculations or logical operations as dictated by the instruction.
  - Results are stored back in the accumulator or designated registers for further processing or output.

5. **Interrupt Handling**:
   - The microprocessor can respond to external interrupt signals, pausing its current operations to handle high-priority tasks.
   - This process involves saving the current state and jumping to the appropriate interrupt service routine.

8085 Bus Organization

1. **Bus Structure**:
   - The 8085 microprocessor utilizes a tri-state bus system, consisting of an 8-bit data bus, a 16-bit address bus, and control signals.
   - The address bus is used to specify memory locations, while the data bus transfers actual data between the microprocessor and memory or I/O devices.
2. **Address Bus**:
   - The 16-bit address bus can access 2^16 = 64KB of memory locations.
   - Each address corresponds to a unique memory location, allowing the microprocessor to read or write data as needed.
3. **Data Bus**:
   - The 8-bit data bus allows the transfer of 8 bits of data at a time between the CPU, memory, and I/O devices.
   - This bus facilitates the flow of information during read and write operations.
4. **Control Signals**:
   - The control bus transmits control signals (such as READ, WRITE, and INTERRUPT) that manage the timing and direction of data flow.
   - Control signals coordinate communication between the microprocessor and other components.

5. **I/O Addressing**:
    - The 8085 microprocessor supports two I/O addressing modes: Memory-Mapped I/O and I/O-Mapped I/O.
    - In memory-mapped I/O, all devices share the same address space as memory, while I/O-mapped I/O uses a separate address space for I/O devices.

Check Your Progress

1. **What is the function of the ALE signal in the 8085 microprocessor?**

a) Read data from memory b) Write data to I/O port
c) Latch the lower byte of the address d) Acknowledge an interrupt

Answer: c) Latch the lower byte of the address

2. Which signal is used to distinguish between memory and I/O operations?

a) RD b) IO/M c) WR d) S0/S1

Answer: b) IO/M

3. How many bits wide is the address bus of the 8085 microprocessor?

a) 16 bits b) 8 bits c) 32 bits d) 64 bits

Answer: a) 16 bits

4. The address bus in the 8085 microprocessor is:

a) Bidirectional b) Unidirectional c) Half-duplex d) Full-duplex

Answer: b) Unidirectional

5. Which lines in the 8085 microprocessor represent the higher 8 bits of the address bus?

a) AD0-AD7 b) IO/M c) S0-S1 d) A8-A15

Answer: d) A8-A15

1.6 INTERNAL DATA OPERATIONS AND 8085 REGISTERS

1.6.1 INTERNAL DATA OPERATIONS IN 8085 MICROPROCESSOR

The 8085 microprocessor, an 8-bit microprocessor introduced by Intel, performs a variety of internal data operations crucial for executing instructions and processing data. These operations can be broadly categorized into arithmetic, logical, data transfer, and branching operations. Understanding these operations is fundamental for programming and utilizing the 8085 microprocessor effectively.

1. Arithmetic Operations

Arithmetic operations in the 8085 microprocessor include addition, subtraction, increment, and decrement. These operations are primarily performed using the accumulator and involve the following instructions:

- **Addition (ADD)**: This instruction adds the contents of a specified register or memory location to the contents of the accumulator. For example, ADD B adds the contents of register B to the accumulator.
- **Addition with Carry (ADC)**: Similar to ADD, but it also adds the carry flag to the result. For example, ADC C adds the contents of register C and the carry flag to the accumulator.
- **Subtraction (SUB)**: This instruction subtracts the contents of a specified register or memory location from the contents of the accumulator. For example, SUB D subtracts the contents of register D from the accumulator.
- **Subtraction with Borrow (SBB)**: Similar to SUB, but it also subtracts the borrow (carry) flag from the result. For example, SBB E subtracts the contents of register E and the carry flag from the accumulator.
- **Increment (INR)**: Increments the contents of a specified register or memory location by one. For example, INR H increments the contents of register H by one.

- **Decrement (DCR)**: Decrements the contents of a specified register or memory location by one. For example, DCR L decrements the contents of register L by one.

2. Logical Operations

Logical operations involve manipulating the bits of data within the registers. These include AND, OR, XOR, compare, and rotate operations:

- **AND (ANA)**: This instruction performs a bitwise AND operation between the contents of the accumulator and a specified register or memory location. For example, ANA B performs a bitwise AND between the accumulator and register B.
- **OR (ORA)**: This instruction performs a bitwise OR operation between the contents of the accumulator and a specified register or memory location. For example, ORA C performs a bitwise OR between the accumulator and register C.
- **XOR (XRA)**: This instruction performs a bitwise XOR operation between the contents of the accumulator and a specified register or memory location. For example, XRA D performs a bitwise XOR between the accumulator and register D.
- **Compare (CMP)**: Compares the contents of the accumulator with a specified register or memory location. For example, CMP E compares the accumulator with register E.
- **Complement (CMA)**: This instruction complements (inverts) the bits of the accumulator.
- **Rotate**: These instructions rotate the bits in the accumulator left or right. RLC rotates bits to the left, and RRC rotates bits to the right.

3. Data Transfer Operations

Data transfer operations move data between registers, between registers and memory, or between the accumulator and memory. Key instructions include:

- **Move (MOV)**: Transfers data from one register to another. For example, MOV A, B copies the contents of register B to the accumulator.
- **Move Immediate (MVI)**: Transfers an immediate value to a register. For example, MVI C, 32H moves the immediate value 32H to register C.
- **Load and Store (LDA, STA)**: LDA loads data from a specified memory location to the accumulator. STA stores data from the accumulator to a specified memory location.
- **Exchange (XCHG)**: Exchanges the contents of register pairs. For example, XCHG exchanges the contents of register pairs DE and HL.

4. Branching Operations

Branching operations alter the sequence of instruction execution based on certain conditions. These include jumps, calls, and returns:

- **Jump (JMP)**: Transfers control to a specified memory address. For example, JMP 2000H jumps to address 2000H.
- **Conditional Jump**: Transfers control based on specific conditions (e.g., zero, carry, parity). For example, JZ 3000H jumps to address 3000H if the zero flag is set.
- **Call (CALL)**: Transfers control to a subroutine at a specified address. For example, CALL 4000H calls the subroutine at address 4000H.
- **Return (RET)**: Returns control from a subroutine to the main program.

These internal data operations enable the 8085 microprocessor to perform complex computational tasks and process data efficiently. Understanding these operations is essential for programming and optimizing microprocessor-based systems.

1.6.2 8085 MICROPROCESSOR REGISTERS

The 8085 microprocessor, designed by Intel, contains several registers that are used to store data temporarily during the execution of instructions. Registers are faster to access than memory and play a crucial role in the microprocessor's operation. The 8085 microprocessor includes the following types of registers: Accumulator, General

Purpose Registers, Special Purpose Registers (Program Counter, Stack Pointer), and Status Flags.

1. Accumulator (A)

The Accumulator is an 8-bit register used in arithmetic, logical, and data transfer operations. It is one of the most important registers in the 8085 microprocessor because:

- It holds one of the operands in arithmetic and logical operations.
- It stores the result of these operations.
- It is often used in data transfer instructions to transfer data to and from memory.

2. General Purpose Registers

The 8085 has six general-purpose registers, each of 8 bits, which can be used individually or as pairs for 16-bit operations. These registers are named B, C, D, E, H, and L.

- **B and C**: Used individually for 8-bit data or combined as a 16-bit register pair BC.
- **D and E**: Similarly used for 8-bit data or combined as the 16-bit register pair DE.
- **H and L**: Also used individually for 8-bit data or combined as the 16-bit register pair HL.

The HL pair is particularly important as it is often used to store memory addresses that point to data in memory.

3. Special Purpose Registers

Program Counter (PC)

- The Program Counter is a 16-bit register that holds the address of the next instruction to be executed.

- It is automatically incremented after fetching an instruction.
- It is used to sequence the execution of the instructions in a program.
- If the current instruction is a branching instruction (e.g., JMP, CALL), the Program Counter is updated to the new address.

Stack Pointer (SP)

- The Stack Pointer is a 16-bit register that points to the top of the stack.
- The stack is a special area of memory used for temporary storage of data and addresses during program execution, especially during function calls and interrupts.
- The Stack Pointer is decremented or incremented by 2 as data is pushed onto or popped off the stack.

4. Temporary Registers

The 8085 microprocessor uses several temporary registers that are not accessible to the programmer but are crucial for its internal operations. These registers are used during the execution of instructions to hold intermediate results.

5. Status Flags

The 8085 microprocessor has five status flags that indicate the results of arithmetic and logical operations. These flags are part of the 8-bit flag register, where only five bits are used:

- **Sign Flag (S)**: Set if the most significant bit (MSB) of the result is 1, indicating a negative number in signed operations.
- **Zero Flag (Z)**: Set if the result of the operation is zero.
- **Auxiliary Carry Flag (AC)**: Set if there is a carry out of the lower nibble (4 bits) during an arithmetic operation.
- **Parity Flag (P)**: Set if the number of 1 bits in the result is even.
- **Carry Flag (CY)**: Set if there is a carry out of the most significant bit during an arithmetic operation or if the result is too large for the register.

6. Instruction Register and Decoder

- The Instruction Register holds the opcode of the current instruction being executed.
- The Decoder interprets the opcode and generates the necessary control signals to execute the instruction.

Addition Example:

```
MVI A, 5H   ; Load 5H into the accumulator

MVI B, 3H   ; Load 3H into register B

ADD B       ; Add the contents of B to the accumulator
```

In this example, the Accumulator (A) is used to store the operand and the result of the addition operation. Register B holds another operand.

Data Transfer Example:

```
MVI H, 20H  ; Load 20H into register H

MVI L, 30H  ; Load 30H into register L

MOV A, M    ; Move the data from memory location pointed by HL pair to accumulator
```

Here, the HL pair points to a memory location, and the data from that location is moved to the Accumulator.

Let us sum up:

Internal Data Operations in 8085 Microprocessor

1. **Data Fetching**:
   - The microprocessor retrieves data from memory or I/O devices based on the instructions fetched from memory.

   - Data is loaded into the registers for processing, enabling efficient manipulation during operations.
2. **Data Processing**:
   - The Arithmetic Logic Unit (ALU) performs various operations such as addition, subtraction, logical AND, OR, and NOT on the data stored in registers.
   - Intermediate results are temporarily stored in the accumulator or other registers as the operations are executed.
3. **Data Storage**:
   - After processing, the final results are written back to memory or stored in registers for future use.
   - This involves sending the data to the specified memory address or register as dictated by the instruction set.
4. **Status Flags**:
   - The 8085 microprocessor utilizes status flags in the flag register to indicate the outcome of arithmetic and logical operations (e.g., Zero flag, Carry flag, Sign flag).
   - These flags influence conditional branching and decision-making in program execution.
5. **Interrupt Handling**:
   - The microprocessor can temporarily halt its current operations to address high-priority tasks initiated by external devices.
   - The state of ongoing operations is saved, and upon completion of the interrupt service routine, the microprocessor resumes normal operation.

8085 Microprocessor Registers

1. **General-Purpose Registers**:
   - The 8085 microprocessor contains six 8-bit general-purpose registers: B, C, D, E, H, and L.
   - These registers can be used to hold data, addresses, or intermediate results during computations.

2. **Accumulator (A)**:
    - The accumulator is a special 8-bit register used for arithmetic and logical operations.
    - It is the primary register for storing results from the ALU and is involved in most instructions.
3. **Program Counter (PC)**:
    - The Program Counter is a 16-bit register that holds the address of the next instruction to be executed.
    - It increments automatically after each instruction fetch, ensuring sequential execution of programs.
4. **Stack Pointer (SP)**:
    - The Stack Pointer is a 16-bit register that points to the top of the stack in memory, facilitating temporary storage of data, return addresses, and function parameters.
    - It is essential for managing function calls and returns during program execution.
5. **Instruction Register (IR)**:
    - The Instruction Register holds the current instruction being executed after it has been fetched from memory.
    - It plays a crucial role in instruction decoding, enabling the control unit to generate the necessary control signals for execution.
6. **Flag Register**:
    - The Flag Register is an 8-bit register that contains status flags (Zero, Carry, Sign, Parity, and Auxiliary Carry) that reflect the outcome of arithmetic and logical operations.
    - These flags are used for decision-making in conditional instructions.

Check Your Progress

1. What does the INR instruction do in the 8085 microprocessor?

a) Decrements the contents of a register

b) Subtracts the contents of a register

c) Increments the contents of a register

d) Compares the contents of a register

Answer: c) Increments the contents of a register

2. Which operation does the XRA instruction perform in the 8085 microprocessor?

a) Bitwise XOR b) Bitwise AND c) Bitwise OR d) Bitwise shift

Answer: a) Bitwise XOR

3. Which instruction is used to transfer control to a subroutine in the 8085 microprocessor?

a) JMP b) CALL c) RET d) MOV

Answer: b) CALL

4. What does the Carry Flag (CY) indicate in the 8085 microprocessor?

a) Zero result in an operation

b) End of an instruction cycle

c) Even parity

d) Overflow of the most significant bit

Answer: d) Overflow of the most significant bit

5. The MOV A, M instruction in the 8085 microprocessor performs which of the following operations?

a) Move data from accumulator to a memory location

b) Move data between registers

c) Move data from memory to accumulator

d) Move immediate data to a register

Answer: c) Move data from memory to accumulator

1.7 PERIPHERAL OR EXTERNAL INITIATED OPERATIONS

The 8085 microprocessor can interact with peripheral devices and respond to external events through a variety of operations. These interactions are crucial for implementing real-world applications where the microprocessor must communicate with external hardware like keyboards, displays, memory, and other I/O devices. Peripheral or external initiated operations in the 8085 microprocessor include interrupt handling, direct memory access (DMA), and input/output operations.

1.7.1. Interrupt Handling

Interrupts are signals that temporarily halt the current execution of the microprocessor to attend to a high-priority task. The 8085 microprocessor supports several types of interrupts, categorized as hardware and software interrupts.

Hardware Interrupts:

- **TRAP**: A non-maskable interrupt with the highest priority. It cannot be disabled and is typically used for critical events like power failure.
- **RST7.5, RST6.5, RST5.5**: Maskable vectored interrupts with decreasing priority from RST7.5 to RST5.5. They can be enabled or disabled using software instructions.
- **INTR**: A non-vectored interrupt with the lowest priority. It requires an external device to provide the interrupt vector (address of the interrupt service routine).

Software Interrupts:

- These are instructions like RST0 to RST7, which cause the microprocessor to jump to a predefined memory location to execute a subroutine.

Interrupt Process: When an interrupt occurs, the following steps are performed:

1. The microprocessor completes the current instruction.
2. The Program Counter (PC) and the status of the microprocessor are saved on the stack.
3. The microprocessor jumps to a fixed memory location (for vectored interrupts) or an address provided by the external device (for non-vectored interrupts).
4. The interrupt service routine (ISR) is executed.
5. The microprocessor restores the PC and status from the stack and resumes the execution of the interrupted program.

1.7.2. Direct Memory Access (DMA)

DMA allows peripheral devices to transfer data directly to or from memory without involving the microprocessor, thereby freeing the microprocessor to perform other tasks. The 8085 supports DMA through the HOLD and HLDA signals.

- **HOLD**: An external device requests the use of the system buses by asserting the HOLD signal.
- **HLDA**: The microprocessor acknowledges the HOLD request by asserting the HLDA signal, indicating that it has relinquished control of the buses.

In DMA mode, the external device directly controls the memory address and data buses to transfer data between memory and the peripheral device.

1.7.3. Input/Output Operations

The 8085 microprocessor communicates with peripheral devices through I/O ports using two primary methods: I/O-mapped I/O and memory-mapped I/O.

I/O-mapped I/O:

- Uses special instructions IN and OUT to transfer data between the accumulator and I/O ports.
- The I/O ports have an 8-bit address space, allowing for 256 distinct I/O ports.

Example:

```
IN 80H   ; Read data from I/O port with address 80H into the accumulator
OUT 81H  ; Write data from the accumulator to I/O port with address 81H
```

Memory-mapped I/O:

- Treats I/O devices as memory locations, using standard data transfer instructions.
- The I/O devices share the same address space as memory, allowing for more complex data handling.

Example:

```
MVI A, 30H   ; Load 30H into the accumulator
```

STA 5000H ; Store the accumulator content at memory location 5000H (which is an I/O device in memory-mapped I/O)

1.7.4. Serial Communication

The 8085 microprocessor also supports serial communication through the SID (Serial Input Data) and SOD (Serial Output Data) lines, allowing for bit-by-bit data transfer with external devices.

SID: Used for receiving serial data. **SOD**: Used for transmitting serial data.

Example:

```
SIM   ; Set Serial Output Data (SOD) based on accumulator content
```

RIM ; Read Serial Input Data (SID) into the accumulator

Peripheral or external initiated operations in the 8085 microprocessor are essential for effective communication with external devices and handling high-priority tasks. Understanding interrupt handling, DMA, I/O operations, and serial communication enables the implementation of responsive and efficient microprocessor-based systems. These capabilities allow the 8085 to interact seamlessly with a wide range of peripherals, making it suitable for various applications in embedded systems and real-time processing.

Let us sum up:

Peripheral or External Initiated Operations

1. **Interrupt Handling**:
    - **Definition**: Interrupts are signals from peripheral devices indicating that they require attention from the microprocessor, allowing it to pause its current operations to respond to the event.
    - **Types of Interrupts**: Includes hardware interrupts (triggered by external devices) and software interrupts (initiated by programs). The 8085 microprocessor supports several interrupt types, such as RST (Restart) and INTR (Interrupt Request).
    - **Process**: Upon receiving an interrupt, the microprocessor saves the current execution context, determines the type of interrupt, and executes the corresponding interrupt service routine (ISR) to address the request before resuming normal operations.
2. **Direct Memory Access (DMA)**:
    - **Definition**: DMA is a method that allows peripheral devices to transfer data to and from memory independently of the CPU, improving system performance by freeing up the processor for other tasks.
    - **Operation**: A DMA controller takes control of the address and data buses, allowing it to read from or write to memory directly. The microprocessor is notified once the data transfer is complete, minimizing the need for constant CPU intervention.

  - **Advantages**: Reduces CPU overhead and increases data transfer efficiency, making it suitable for high-speed data transfer applications like disk drives and video processing.

3. **Input/Output Operations**:
   - **I/O Interfaces**: The 8085 microprocessor interfaces with various peripheral devices through memory-mapped I/O and I/O-mapped I/O addressing modes, allowing it to read from and write to devices such as keyboards, displays, and sensors.
   - **Control Signals**: The microprocessor generates control signals (READ, WRITE) to manage data flow between itself and I/O devices, coordinating communication effectively.
   - **Polling vs. Interrupts**: In polling, the CPU regularly checks the status of I/O devices, which can be inefficient. In contrast, interrupts allow the CPU to react only when an I/O device requires attention, improving efficiency.
4. **Serial Communication**:
   - **Definition**: Serial communication refers to the process of transmitting data one bit at a time over a single communication channel or wire, making it suitable for long-distance communication and reducing wiring complexity.
   - **Protocols**: Common serial communication protocols include UART (Universal Asynchronous Receiver-Transmitter) and SPI (Serial Peripheral Interface), which define the methods for data transmission, error checking, and synchronization.
   - **Application**: Used in various applications, including connecting microprocessors to sensors, data acquisition systems, and communication between devices in embedded systems.

Check Your Progress

1. Which interrupt in the 8085 microprocessor has the highest priority and cannot be disabled?

a) RST7.5 b) INTR c) TRAP d) RST5.5

Answer: c) TRAP

2. What is the purpose of the HLDA signal in DMA operations?

a) To request use of the system buses

b) To acknowledge the HOLD request

c) To initiate a data transfer

d) To terminate the DMA operation

Answer: b) To acknowledge the HOLD request

3. How many distinct I/O ports can be addressed using I/O-mapped I/O in the 8085?

a) 128 b) 256 c) 512 d) 1024

Answer: b) 256

4. What is the purpose of the SID line in the 8085 microprocessor?

a) For transmitting serial data

b) For receiving serial data

c) For handling interrupts

d) For DMA operations

Answer: b) For receiving serial data

5. Which instruction is used to set the Serial Output Data (SOD) based on accumulator content?

a) SIM b) RIM c) SOD d) OUT

Answer: a) SIM

Summary:

The unit begins with an exploration of digital computers, covering their basic structure and functionality. It then moves on to microcomputer organization, detailing the integration of the CPU, memory, and input/output devices. Students will study computer languages, including machine, assembly, and high-level languages, and learn about microprocessor architecture, understanding its design principles and operational modes. The course also covers microprocessor initiated operations, discussing control flow and execution mechanisms. Additionally, students will gain insights into the 8085 bus organization, focusing on data, address, and control buses, as well as internal data operations and 8085 registers, which involve register functions and data handling techniques. Finally, the unit explores peripheral or external initiated operations, emphasizing the microprocessor's interaction with external devices.

Activities

Activity 1: Microprocessor Architecture and Operations

Objective: To understand the architecture of microprocessors, their operations, and bus organization.

Activity 2: Internal Data Operations and 8085 Registers

Objective: To understand internal data operations and the role of registers in the 8085 microprocessor.

Check Your Progress

1. Explain the architecture of a microprocessor and its operational functions

   ----

   ----

2. Discuss the organization and functions of buses in the 8085 microprocessor.

   ----

3. Explain internal data operations and the role of registers in the 8085 microprocessor.

   ----

4. Describe how external devices initiate operations in a microprocessor system.

   ----

5. Compare and contrast machine, assembly, and high-level languages used in microprocessor programming.

Self-Assessment Questions

1. What is the primary number system used by digital computers for data representation?
2. Which component of a digital computer is responsible for executing instructions and processing data?
3. What is the width of the data bus in the 8085 microprocessor?
4. How many general-purpose registers does the 8085 microprocessor have?

5. Which register in the 8085 microprocessor is used to store the address of the next instruction to be executed?
6. What is the function of the ALE signal in the 8085 microprocessor?
7. Which type of interrupt in the 8085 has the highest priority and cannot be disabled?
8. What is the purpose of the HLDA signal in DMA operations?
9. In the 8085, what is the difference between I/O-mapped I/O and memory-mapped I/O?
10. What are the two lines used for serial communication in the 8085 microprocessor?
11. Which instruction is used to read data from an I/O port into the accumulator in the 8085?
12. What is the function of the Stack Pointer (SP) register in the 8085?
13. How many bits wide is the address bus of the 8085 microprocessor?
14. What does the CMP instruction do in the 8085 microprocessor?
15. Which flag in the 8085 is set if the result of an operation is zero?

Further Reading and References

Textbooks

1. **“Microprocessor Architecture, Programming, and Applications with the 8085" by Ramesh S. Gaonkar**
   - Detailed explanation of the 8085's internal structure, including its registers, ALU, and bus organization; in-depth coverage of 8085 assembly language programming, including instruction set, addressing modes, and programming techniques.
   - ISBN: 978-8189866051
2. **“The 8085 Microprocessor: Architecture, Programming, and Interfacing" by K. Udaya Kumar and B.S. Umashankar**
   - Coverage of 8085 assembly language programming, including instruction set and programming techniques.
   - ISBN: 978-8177584554

Online Resources

1. https://www.tutorialspoint.com/microprocessor/microprocessor_8085_architecture.htm
2. https://www.electronicshub.org/8085-microprocessor-architecture/
3. https://www.geeksforgeeks.org/introduction-of-8085-microprocessor/

UNIT II: MICROPROCESSOR

8085 Microprocessor – Pinout and Signals – Functional block diagram - 8085 Instruction Set and Classifications

Unit Objectives:

This unit aims to equip students with a detailed understanding of the 8085 microprocessor's internal structure and operation. By the end of this unit, students should be able to analyze the pin configuration, describe signal interactions, interpret the functional block diagram, and classify and utilize the 8085 instruction set effectively in programming scenarios.

2.1 8085 MICROPROCESSOR

Definition and Importance:

The 8085 microprocessor, developed by Intel in 1976, is an 8-bit microprocessor designed for simple embedded systems and educational purposes. It was widely used in early microcomputer systems due to its simplicity and ease of use. Understanding its architecture is fundamental for grasping how microprocessors function.

2.1.1. Introduction to Microprocessors

A microprocessor is an integrated circuit (IC) that acts as the central processing unit (CPU) of a computer. It processes data according to instructions stored in its memory. The 8085 microprocessor is an 8-bit processor, which means it processes data in 8-bit chunks. It has a 16-bit address bus, capable of addressing up to 64 KB of memory, and an 8-bit data bus.

2.1.2. Pinout and Signals

The 8085 microprocessor has a 40-pin dual in-line package (DIP). Each pin serves a specific function, including addressing, data transfer, control, and power supply.

- **Pin Configuration:**
  - **Address Bus (A15-A8, AD7-AD0):** These pins are used to transmit the address and data signals between the microprocessor and memory or peripherals.
  - **Data Bus (D7-D0):** These are bidirectional pins used for transferring data between the microprocessor and external devices.
  - **Control and Status Signals (RD, WR, IO/M, etc.):** These pins control the timing and operation of data transfer and indicate the status of the microprocessor.
  - **Power Supply and Clock Signals (Vcc, Vss, CLK):** These pins provide power and clock signals to the microprocessor.
- **Signal Description:**
  - **Address Bus (A15-A8, AD7-AD0):** The address bus carries the addresses of data or instructions between the microprocessor and memory or I/O devices.
  - **Data Bus (D7-D0):** The data bus carries data between the microprocessor and memory or I/O devices.
  - **Control Signals (RD, WR, IO/M, etc.):** These signals control the data flow between the microprocessor and memory or I/O devices, specifying whether a read or write operation is being performed.
  - **Clock Signal (CLK):** The clock signal synchronizes the internal operations of the microprocessor.

2.1.3. Functional Block Diagram

The functional block diagram of the 8085 microprocessor provides an overview of its internal architecture and how different components interact with each other.

- **Functional Units:**
  - **Arithmetic and Logic Unit (ALU):** Performs arithmetic and logical operations on data.
  - **Registers:** Holds data and addresses temporarily during processing.
  - **Control Unit:** Directs the operations of the microprocessor by generating control signals.
  - **Instruction Register (IR):** Holds the current instruction being executed.

- **Timing and Control Unit:** Generates timing and control signals for coordinating operations.
- **Interrupt Control:** Manages interrupt signals from external devices.

➢ **Block Diagram Components:**

- **ALU:** Performs arithmetic (addition, subtraction) and logical (AND, OR, NOT, XOR) operations.
- **Registers (Accumulator, BC, DE, HL, SP, PC):** Temporarily stores data and addresses.
- **Instruction Decoder:** Decodes instructions fetched from memory to produce control signals.
- **Timing and Control Unit:** Generates timing signals necessary for the execution of instructions.
- **Interrupt Control:** Manages interrupts from external devices.

2.1.4. Internal Architecture

The internal architecture of the 8085 microprocessor consists of various registers, a control unit, and an arithmetic and logic unit (ALU).

➢ **Registers:**

- **Accumulator (A):** Stores the result of most arithmetic and logical operations.
- **General Purpose Registers (B, C, D, E, H, L):** Used for various purposes, such as storing operands and addresses.
- **Special Purpose Registers:**
- **Program Counter (PC):** Holds the memory address of the next instruction to be fetched.
- **Stack Pointer (SP):** Points to the top of the stack.
- **Flag Register:** Contains flags that indicate the result of an operation (zero, carry, etc.).

➢ **Control Unit:**

- Decodes instructions fetched from memory.
- Generates control signals to coordinate the activities of the other units.

➢ **Arithmetic and Logic Unit (ALU):**

- Performs arithmetic and logical operations on data.
- Uses the data from the registers and stores the result back into the registers.

➢ **Timing and Control Signals:**

- **Machine Cycle:** The execution of one instruction or operation.
- **Instruction Cycle:** The time taken to complete an instruction.

2.1.5. Instruction Set and Classifications

The instruction set of the 8085 microprocessor consists of 74 instructions, which are classified based on their functions and addressing modes.

➢ **Instruction Categories:**

- **Data Transfer Instructions:** MOV, MVI, LXI, LDA, STA, SHLD, LHLD.
- **Arithmetic Instructions:** ADD, SUB, INR, DCR, ADC, SBB.
- **Logical Instructions:** ANA, ORA, XRA, CMP, CMA, CMC.
- **Branching Instructions:** JMP, JC, JNC, JZ, JNZ, CALL, RET.
- **Stack Operations:** PUSH, POP.
- **Addressing Modes:**
- **Immediate Addressing:** Data is specified directly in the instruction.
- **Direct Addressing:** Data is stored in a memory location specified in the instruction.
- **Register Addressing:** Data is stored in a register specified in the instruction.
- **Indirect Addressing:** Data is stored in a memory location whose address is specified in a register pair.

➢ **Instruction Execution:**

- Fetch: The microprocessor fetches the next instruction from memory.
- Decode: The instruction is decoded to determine what operation to perform.
- Execute: The operation specified by the instruction is performed.

2.1.6. Programming Model

The programming model of the 8085 microprocessor describes how the registers are used in programming.

- **Register Usage:**
  - **Accumulator (A):** Primary register for arithmetic and logic operations.
  - **General Purpose Registers (B, C, D, E, H, L):** Used for various purposes, including arithmetic, data transfer, and address manipulation.
  - **Program Counter (PC):** Holds the address of the next instruction.
  - **Stack Pointer (SP):** Points to the top of the stack.
  - **Flag Register:** Contains flags that indicate the status of the ALU operations.
- **Programming Examples:**
  - Simple arithmetic operations.
  - Data transfer between registers and memory.
  - Branching and looping using conditional instructions.

2.1.7. Interrupts and I/O

The 8085 microprocessor supports interrupt-driven I/O operations.

- **Interrupts:**
  - **Types:** Maskable and non-maskable interrupts.
  - **Handling:** Interrupt service routine (ISR) and interrupt vector table.
  - **I/O Operations:**
  - **Instructions:** IN, OUT.
  - **Addressing:** I/O ports are addressed separately from memory.

2.1.8. Applications and Advantages

The 8085 microprocessor was used in a wide range of applications due to its simplicity and versatility.

- **Applications:**
  - Embedded systems.
  - Industrial control systems.
  - Educational purposes.
- **Advantages:**
  - Simple and easy to understand.
  - Low cost and power consumption.

- Suitable for basic control applications.

Let Us Sum Up:

1. Introduction to Microprocessors

- A microprocessor is an integrated circuit (IC) that contains the functions of a central processing unit (CPU) of a computer.
- Performs arithmetic, logic, control, and input/output (I/O) operations.
- Acts as the brain of a computer or embedded system.
- Key components: Arithmetic Logic Unit (ALU), Control Unit (CU), and Registers.
- Examples: Intel 8085, Intel 8086, ARM processors.

2. Pinout and Signals

- Pinout refers to the arrangement of pins on the microprocessor, each with a specific function.
- Common signals include:
- Power and Ground Pins: For powering the microprocessor (e.g., Vcc, GND).
- Data Bus: Carries data between the microprocessor and memory or peripherals.
- Address Bus: Sends the address of memory locations to be accessed.
- Control Pins: Controls operations like reading, writing, and interrupt handling (e.g., RD, WR, RESET, INTR).
- Clock Signal: Synchronizes the microprocessor’s operations (CLK).

3. Functional Block Diagram

- A visual representation of the internal structure of the microprocessor.
- Main blocks:
- ALU: Performs arithmetic and logical operations.
- Control Unit: Manages the execution of instructions.
- Registers: Temporary storage for data and instructions.
- Bus Interface Unit (BIU): Manages data transfer between the processor and memory/peripherals.
- Memory: Stores instructions and data.

4. Internal Architecture

- Describes the internal working of the microprocessor.
- Key components:
- ALU: Performs operations like addition, subtraction, AND, OR.
- Control Unit: Directs the operation of the processor by interpreting instructions.
- Registers: Small, fast storage locations for immediate data access.
- Instruction Decoder: Interprets the instructions fetched from memory.
- Stack Pointer and Program Counter: Manage execution flow and memory addressing.

5. Instruction Set and Classifications

- Instruction Set: A group of commands the microprocessor can execute.
- Classifications:
- Data Transfer Instructions: MOV, IN, OUT, etc.
- Arithmetic Instructions: ADD, SUB, MUL, DIV, etc.
- Logical Instructions: AND, OR, XOR, NOT, etc.
- Control Instructions: JMP, CALL, RET, etc.
- Branching Instructions: Conditional (JZ, JNZ) and unconditional (JMP).
- CISC (Complex Instruction Set Computing) vs. RISC (Reduced Instruction Set Computing).

6. Programming Model

- Describes how the processor executes instructions and manipulates data.
- Components include:
- Registers: Program Counter (PC), Stack Pointer (SP), Accumulator (A), Flag Register.
- Memory: Program memory and data memory for instruction and data storage.
- Flags: Status indicators such as Zero Flag (Z), Carry Flag (CY), etc.
- Provides an abstraction for low-level programming.

7. Interrupts and I/O

- Interrupts: Signals that temporarily halt the microprocessor’s current task to attend to high-priority tasks.
- Types of Interrupts: Maskable, Non-maskable, Software, and Hardware interrupts.
- Examples: INTR, TRAP, RST in 8085 microprocessors.
- I/O (Input/Output): Interface for communication between the microprocessor and external devices.
- Memory-mapped I/O: I/O devices treated as memory locations.
- Port-mapped I/O: Special instructions for I/O operations using ports.

8. Applications and Advantages

- Applications:
  - Embedded systems: Automotive systems, consumer electronics.
  - Computing systems: Laptops, desktops, servers.
  - Communication systems: Routers, modems.
  - Control systems: Industrial automation, robotics.
- Advantages:
  - Compact size and low power consumption.
  - High processing speed and efficiency.
  - Versatility: Used in diverse fields from consumer electronics to industrial systems.
  - Programmable for different tasks and applications.

Check your Progress:

1. **The 8085 microprocessor has how many address lines?**

   a) 8

   b) 16

   c) 32

   d) 64

   Answer: b) 16

2. **Which pin is used for providing the clock signal to the 8085 microprocessor?**

   a) X1, X2

   b) CLK OUT

   c) ALE

   d) IO/M

   Answer: a) X1, X2

3. **What is the size of the data bus in the 8085 microprocessor?**

   a) 8-bit

   b) 16-bit

   c) 32-bit

   d) 64-bit

   Answer: a) 8-bit

4. **What is the maximum memory that can be addressed by the 8085 microprocessor?**

   a) 1 KB

   b) 32 KB

   c) 64 KB

   d) 128 KB

   Answer:c) 64 KB

5. **The 8085 microprocessor operates at a clock frequency of:**

   a) 1 MHz

   b) 3 MHz

   c) 10 MHz

   d) 5 MHz

   Answer: b) 3 MHz

2.2 PINOUT AND SIGNALS

Here's a breakdown of the pinout signals for the 8085 microprocessor:

2.2.1. Address Bus (AD0-AD15):

- The Address Bus consists of 16 lines (AD0-AD15).
- These lines carry the memory address or I/O port address during memory read/write or I/O operations.
- The 8085 microprocessor can address up to 64KB of memory and 256 I/O ports.

2.2.2. Data Bus (D0-D7):

- The Data Bus comprises 8 lines (D0-D7).
- These lines transfer data between the microprocessor and external devices such as memory and I/O ports.
- During memory read/write or I/O operations, the data to be read from or written to external devices is transmitted through these lines.

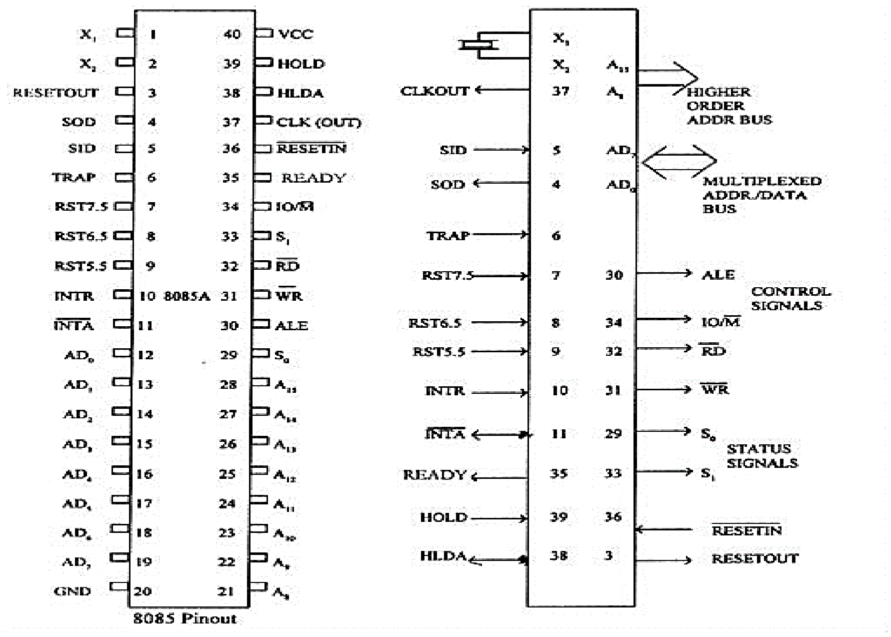

X1 1, X2 2, RESETOUT 3, SOD 4, SID 5, TRAP 6, RST7.5 7, RST6.5 8, RST5.5 9, INTR 10, INTA 11, AD0 12, AD1 13, AD2 14, AD3 15, AD4 16, AD5 17, AD6 18, AD7 19, GND 20
8085A
40 VCC, 39 HOLD, 38 HLDA, 37 CLK (OUT), 36 RESETIN, 35 READY, 34 IO/M, 33 S1, 32 RD, 31 WR, 30 ALE, 29 S0, 28 A15, 27 A14, 26 A13, 25 A12, 24 A11, 23 A10, 22 A9, 21 A8
8085 Pinout
CLKOUT
SID
SOD
TRAP
RST7.5
RST6.5
RST5.5
INTR
INTA
READY
HOLD
HLDA
HIGHER ORDER ADDR BUS
MULTIPLEXED ADDR./DATA BUS
ALE
CONTROL SIGNALS
IO/M
RD
WR
S0
S1
STATUS SIGNALS
RESETIN
RESETOUT

Fig. 1 Pin diagram of 8085

2.2.3. Control Signals:

➢ **ALE (Address Latch Enable):**

- ALE is a pulse generated by the microprocessor during the first clock cycle of each machine cycle.
- It is used to latch the address from the address bus onto the external latch ICs.

➢ **RD (Read) and WR (Write):**

- RD and WR signals indicate whether the microprocessor is performing a read or write operation.
- RD is active low during a memory read operation, while WR is active low during a memory write operation.

➢ **IO/M (Input/Output and Memory):**

- IO/M indicates whether the microprocessor is performing an I/O operation or a memory operation.
- When IO/M is high, the microprocessor performs an I/O operation; when it's low, it performs a memory operation.

➢ **S0, S1, S2 (Status Signals):**

- These signals provide status information about the current operation of the microprocessor.
- Different combinations of S0, S1, and S2 indicate different machine cycles.

➢ **INTA (Interrupt Acknowledge):**

- INTA is generated by the microprocessor to acknowledge interrupt requests from external devices.
- It indicates that the microprocessor has accepted the interrupt request and is ready to proceed with the interrupt service routine.

2.2.4. Interrupt Signals:

➢ **INTR (Interrupt Request):**

- INTR is a hardware interrupt request signal generated by external devices.
- When an external device requests an interrupt, it asserts the INTR signal to the microprocessor.

- **INT (Interrupt):**
  - INT is a maskable interrupt request input.
  - It can be enabled or disabled by the microprocessor using software instructions.

2.2.5. Power Supply and Ground Pins:

- **Vcc (Power Supply):**
  - Vcc is the supply voltage pin for the microprocessor.
  - It provides the necessary operating voltage for the microprocessor.
- **GND (Ground):**
  - GND is the ground or reference voltage pin for the microprocessor.
  - It provides the reference point for the electrical circuit and completes the circuit path.
  - Understanding these pinout signals is essential for interfacing the 8085 microprocessor with external devices and designing systems around it.

Let us Sum Up

Address Bus (AD0-AD15)

- Carries memory addresses: Used to specify the location in memory or I/O where data will be read or written.
- 16-bit width: In systems like the 8085/8086, it allows addressing up to 64 KB (2^16) of memory space.

Data Bus (D0-D7)

- Transfers data: An 8-bit bus used to send and receive data between the microprocessor, memory, and peripherals.
- Bidirectional: Data can flow both ways—into the processor during read operations and out during write operations.

Control Signals

- Manage read/write operations: Signals like RD and WR control whether the microprocessor reads data from or writes data to memory/I/O.
- Synchronize devices: ALE (Address Latch Enable) and READY signals help coordinate operations with slower peripherals.

Interrupt Signals

- Handle external events: Signals like INTR (Interrupt Request) and TRAP notify the processor of high-priority tasks or emergencies.
- Acknowledge interrupts: INTA (Interrupt Acknowledge) confirms that the microprocessor is responding to an interrupt.

Power Supply and Ground Pins

- Vcc: Provides the operating voltage necessary for the microprocessor to function (e.g., 5V for 8085).
- GND: Establishes the reference ground for the circuit, ensuring proper power flow and operation stability.

Check your Progress:

1. **Which of the following is a control signal of the 8085 microprocessor?**

   a) S0, S1

   b) ALE

   c) HOLD

   d) All of the above

   Answer:d) All of the above

2. **The 8085 microprocessor has how many registers?**

   a) 6

   b) 8

   c) 10

   d) 12

   Answer: a) 6 (B, C, D, E, H, L)

3. **What is the purpose of the ALE (Address Latch Enable) pin in the 8085?**

   a) To latch the higher-order address lines

   b) To latch the lower-order address lines

c) To provide clock input

d) To reset the microprocessor

Answer:b) To latch the lower-order address lines

4. **Which register pair is used as the Program Counter in 8085?**

a) BC

b) DE

c) SP

d) None of the above

Answer: d) None of the above

5. **What is the function of the HOLD pin in the 8085?**

a) Hold operation for DMA

b) Provides interrupt handling

c) Generates clock pulses

d) Restarts the microprocessor

Answer: a) Hold operation for DMA

2.3 FUNCTIONAL BLOCK DIAGRAM

The functional block diagram of the 8085 microprocessor illustrates its internal architecture and how different functional units interact with each other. Here's a breakdown of the major components typically depicted in the block diagram:

2.3.1. Arithmetic and Logic Unit (ALU):

- The ALU performs arithmetic and logic operations on data received from the registers and memory.
- It can perform operations such as addition, subtraction, logical AND, logical OR, etc.

2.3.2. Registers:

- **Accumulator (A):** The primary register for arithmetic and logic operations. Results of ALU operations are often stored here.
- **General-Purpose Registers (B, C, D, E, H, L):** These registers are used for various data manipulation tasks.
- **Stack Pointer (SP):** Used to manage the stack for subroutine calls and returns.
- **Program Counter (PC):** Keeps track of the memory address of the next instruction to be fetched and executed.
- **Instruction Register (IR):** Holds the current instruction being executed.
- **Temporary Register (Temp):** Provides temporary storage for data during certain operations.

2.3.3. Control Unit:

- The Control Unit coordinates the execution of instructions by controlling the flow of data between different functional units.
- It generates control signals to regulate the operation of the ALU, registers, and other components.

2.3.4. Instruction Decoder:

- The Instruction Decoder decodes the instructions fetched from memory and generates the necessary control signals to execute them.
- It interprets the opcode of each instruction and directs the microprocessor to perform the corresponding operation.

2.3.5. Address and Data Bus Interface:

- The Address Bus carries the memory address or I/O port address during memory read/write or I/O operations.
- The Data Bus transfers data between the microprocessor and external devices such as memory and I/O ports.

2.3.6. Clock Generator:

- The Clock Generator generates the clock signals required for synchronizing the operation of different components within the microprocessor.
- It provides timing signals to control the execution of instructions and coordinate data transfers.

2.3.7. Interrupt Control Unit:

- The Interrupt Control Unit manages interrupt requests from external devices.
- It prioritizes and handles interrupt requests, suspending the execution of the current program to service the interrupt.

2.3.8. Memory Interface:

- The Memory Interface manages communication between the microprocessor and external memory devices.
- It includes address buffers, data buffers, and control logic to facilitate memory read/write operations.

➢ **Processing Unit:**

- **Arithmetic and Logic Unit (ALU):** Performs arithmetic and logical operations on data fetched from memory or input/output devices. It operates on data provided by the registers and produces results back to the registers.

➢ **Instruction Unit:**

- **Instruction Register (IR):** Holds the current instruction being executed.
- **Instruction Decoder:** Decodes the instructions fetched from memory to produce control signals for the ALU, registers, and other units.

➢ **Storage Interface Unit:**

- **Registers:** Includes the Accumulator (A), General Purpose Registers (B, C, D, E, H, L), Program Counter (PC), Stack Pointer (SP), and Flag Register. These registers store data and addresses temporarily during processing.

By understanding the functional block diagram of the 8085 microprocessor, one can gain insights into its internal structure and operation, facilitating the design and development of software and hardware systems based on this architecture.

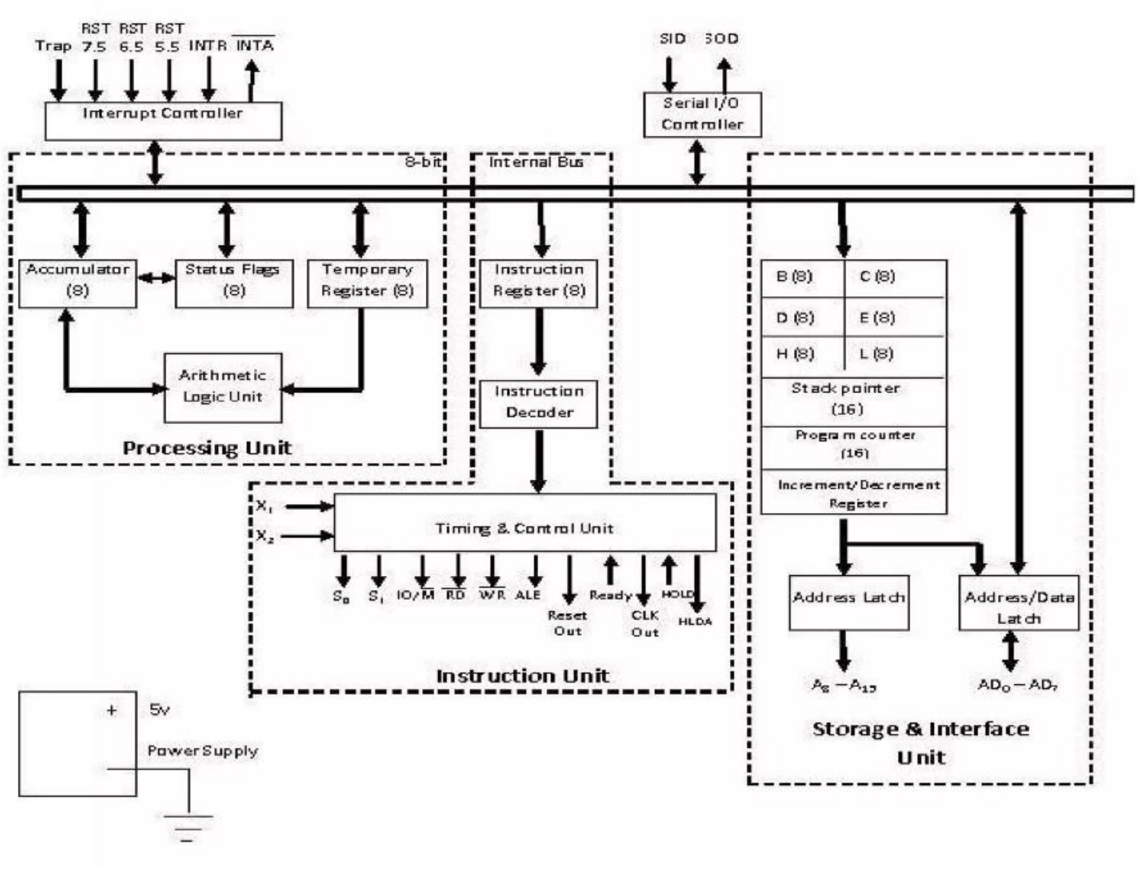


Fig 2. Functional Block Diagram

Let Us Sum Up

Arithmetic and Logic Unit (ALU)

- Performs arithmetic operations: Handles basic arithmetic like addition, subtraction, multiplication, and division.
- Logical operations: Executes logical operations such as AND, OR, NOT, and XOR, and sets flags based on the results.

Registers

- Temporary data storage: Small, fast storage locations inside the CPU to hold data and instructions during processing.

- Types of registers: Includes accumulator, general-purpose registers, program counter (PC), and stack pointer (SP).

Control Unit

- Manages execution: Directs the flow of data between the ALU, memory, and I/O devices, ensuring correct execution of instructions.
- Generates control signals: Issues control signals to synchronize and manage the microprocessor's operations.

Instruction Decoder

- Interprets instructions: Converts machine language instructions fetched from memory into control signals for execution.
- Decodes opcodes: Reads the operation codes (opcodes) to determine the appropriate operation for the microprocessor.

Address and Data Bus Interface

- Handles data transfer: Manages communication between the processor, memory, and peripheral devices via the data and address buses.
- Multiplexing: In some microprocessors, it multiplexes address and data lines to reduce pin count, separating address and data during different clock cycles.

Clock Generator

- Synchronizes operations: Generates a clock signal that governs the timing of all operations inside the microprocessor.
- Controls execution speed: Defines the pace at which instructions are executed by the processor.

Interrupt Control Unit

- Manages interrupts: Prioritizes and handles external interrupt signals, temporarily halting current tasks to respond to urgent tasks.
- Interrupt vectors: Determines the appropriate interrupt service routine (ISR) to handle the interrupt request.

Memory Interface

- Connects to memory: Manages reading from and writing to memory, controlling address decoding and data flow.
- Supports different memory types: Interfaces with both RAM (for volatile data storage) and ROM (for storing fixed instructions).

Check your Progress:

1. **Which of the following is a machine cycle in the 8085 microprocessor?**

   a) Fetch

   b) Execute

   c) Write

   d) All of the above

   Answer: d) All of the above

2. **Which interrupt has the highest priority in the 8085 microprocessor?**

   a) RST 7.5

   b) INTR

   c) TRAP

   d) RST 5.5

Answer: c) TRAP

3. **What is the function of the SID (Serial Input Data) pin?**

   a) Provides serial input data to the microprocessor

   b) Provides serial output data from the microprocessor

   c) Used to reset the microprocessor

   d) Used for direct memory access

   Answer: a) Provides serial input data to the microprocessor

4. **The instruction "MOV A, B" does what in the 8085?**

   a) Moves data from register B to accumulator A

   b) Moves data from accumulator A to register B

   c) Moves data from memory to register A

   d) Moves data from register A to memory

   Answer: a) Moves data from register B to accumulator A

5. How many machine cycles are required for the "STA" instruction?

a) 2

b) 3

c) 4

d) 5

Answer: d) 5

2.4 8085 INSTRUCTION SET ARCHITECTURE AND CLASSIFICATIONS

The 8085 microprocessor instruction set is a collection of instructions that the microprocessor can execute. These instructions are classified based on their functions and addressing modes. Understanding the instruction set architecture (ISA) of the 8085 microprocessor is essential for programming and effectively utilizing its capabilities.

2.4.1. Overview of Instruction Set

- The 8085 microprocessor has a total of 74 instructions. Each instruction performs a specific operation, such as data transfer, arithmetic and logical operations, branching, and control operations. These instructions are encoded as binary codes and are stored in memory for the microprocessor to fetch and execute.

2.4.2. Instruction Format

- The instructions in the 8085 microprocessor have varying formats depending on their types and addressing modes. The basic instruction format includes an opcode and operands, as described below:
- **Opcode:** Specifies the operation to be performed. **Operands:** Data or addresses on which the operation is to be performed.

2.4.3. Instruction Types and Examples

- The instructions in the 8085 microprocessor can be categorized into several types based on their functions. Here are the main types of instructions along with examples:

➢ **Data Transfer Instructions:**

- **MOV Rd, Rs:** Moves data from register Rs to register Rd or memory.
- **MVI Rd, data:** Moves immediate data into register Rd or memory.
- **LDA address:** Loads accumulator with data from memory location specified by address.
- **STA address:** Stores accumulator contents into memory location specified by address.

➢ **Arithmetic and Logical Instructions:**

- **ADD Rd:** Adds contents of register Rd to accumulator.
- **ADI data:** Adds immediate data to accumulator.
- **SUB Rd:** Subtracts contents of register Rd from accumulator.
- **INR Rd:** Increments contents of register Rd.
- **DCR Rd:** Decrements contents of register Rd.
- **ANA Rd:** Performs bitwise AND operation between accumulator and register Rd.

➢ **Branching Instructions:**

- **JMP address:** Jumps to the memory location specified by address.
- **JC address:** Jumps to the specified address if carry flag is set.
- **JZ address:** Jumps to the specified address if zero flag is set.
- **Control Transfer Instructions:**
- **CALL address:** Calls a subroutine at the specified address.
- **RET:** Returns from subroutine.

- **HLT:** Halts the microprocessor.

➢ **Stack Operations:**

- **PUSH Rp:** Pushes contents of register pair Rp onto the stack.
- **POP Rp:** Pops contents from the stack into register pair Rp.

2.4.4. Addressing Modes

- Addressing modes specify how operands are specified in the instructions. The 8085 microprocessor supports various addressing modes:

➢ **Immediate Addressing:**

- **MVI Rd, data:** Loads immediate data into register Rd.

➢ **Direct Addressing:**

- **LDA address:** Loads accumulator with data from memory location specified by address.

➢ **Register Addressing:**

- **MOV Rd, Rs:** Moves data from register Rs to register Rd.

➢ **Indirect Addressing:**

- **LHLD address:** Loads H and L registers with data from the memory location specified by address.

2.4.5. Instruction Classification

- Instructions in the 8085 microprocessor are classified into groups based on their functions and operational characteristics. The classifications include:

➢ **Data Transfer Group:**

- MOV, MVI, LXI, LDA, STA, SHLD, LHLD.

➢ **Arithmetic Group:**

- ADD, ADC, SUB, SBB, INR, DCR, CMP.

➢ **Logical Group:**

- ANA, XRA, ORA, CMP.

- **Branching Group:**
  - JMP, JC, JNC, JZ, JNZ, CALL, RET.
- **Control Group:**
  - NOP, HLT, RIM, SIM.
- **Stack Operations:**
  - PUSH, POP.

2.4.6. Instruction Execution Cycle

- The execution cycle of an instruction in the 8085 microprocessor typically involves three stages:
- **Fetch:** The instruction is fetched from memory into the instruction register (IR). **Decode:** The opcode of the instruction is decoded. **Execute:** The instruction is executed, and the result is stored back in the appropriate register or memory location.

Let Us Sum Up:

Instruction Format

- Structure of instructions: Typically includes the opcode (operation code) and operands (data or memory addresses involved).
- Varying lengths: Instructions can be one-byte, two-byte, or three-byte depending on the processor architecture and the number of operands.

Instruction Types and Examples

- Data transfer instructions: Move data between registers, memory, or I/O devices (e.g., MOV, LDA).
- Arithmetic and logical instructions: Perform operations like addition, subtraction, AND, OR, and XOR (e.g., ADD, SUB, AND).

Addressing Modes

- Immediate addressing: The operand is provided directly within the instruction (e.g., MVI A, 05H).

- Direct, indirect, and register addressing: Specifies the location of operands either directly, via registers, or through memory addresses (e.g., LDA 2500H, MOV A, B).

Instruction Classification

- RISC vs. CISC:
  - RISC: Simple instructions that execute in a single clock cycle.
  - CISC: Complex instructions that may take multiple clock cycles.
- Classes of instructions: Include data transfer, arithmetic/logical, branching, and control instructions.

Instruction Execution Cycle

- Fetch: The processor retrieves the instruction from memory.
- Decode and execute: The instruction is interpreted and carried out by the control unit and ALU.

Check your Progress:

1. The "IN" instruction in the 8085 is used to:

a) Input data from a port to the accumulator

b) Output data from the accumulator to a port

c) Load data from memory to the accumulator

d) Store data from the accumulator to memory

Answer: a) Input data from a port to the accumulator

2. What does the instruction "NOP" do?

a) Clears the accumulator

b) No operation

c) Increments the program counter by 2

d) None of the above

Answer: b) No operation

3. The flag register of the 8085 has how many flags?

a) 4

b) 5

c) 6

d) 8

Answer:b) 5 (Sign, Zero, Auxiliary Carry, Parity, Carry)

4. The stack in the 8085 microprocessor grows:

a) Upward

b) Downward

c) Both directions

d) Randomly

Answer:b) Downward

5. Which instruction is used to load the stack pointer with a 16-bit address in 8085?

a) LXI SP

b) MOV SP

c) MVI SP

d) STA SP

Answer: a) LXI SP

Summary:

To sum up, the 8085 microcontroller is characterized by its architecture and pinout signals, each serving specific roles in data communication, control, and interfacing with external devices. By comprehensively understanding these aspects, one can harness the full potential of the 8085 microcontroller in various applications, from simple automation tasks to complex embedded systems. This unit covers the 8085 microprocessor's pinout and signals, explaining the functional block diagram, and details its instruction set and classifications. It provides a comprehensive understanding of the microprocessor's architecture, instruction types, addressing modes, and execution cycle for effective programming and operation.

Activities

- Design a simple traffic light control system using the 8085 microcontroller. Use LEDs to represent the traffic lights and program the microcontroller to regulate their operation based on predefined timing sequences.
- Develop a temperature monitoring system utilizing a temperature sensor interfaced with the 8085 microcontroller. Program the microcontroller to read sensor data and display temperature readings on an output device.
- Implement a basic security system using the 8085 microcontroller and motion sensors. Program the microcontroller to detect motion and trigger an alarm or notification.

Check Your Progress

1. Identify and explain the functions of at least five pinout signals of the 8085 microcontroller?

----

2. Describe the architecture of the 8085 microcontroller, highlighting its key components and their roles?

----

3. Discuss the significance of the 8085 microcontroller in the context of embedded systems and microcontroller-based applications?

Self-Assessment Questions

1. How many pins does the 8085 microprocessor have, and what are the functions of its major pins (e.g., ALE, RD, WR)?
2. Describe the role of the control signals in the 8085 microprocessor.
3. What is the function of the following pins: HOLD, HLDA, and READY?
4. How do the higher-order and lower-order address buses differ in 8085?
5. Explain the architecture of the 8085 microprocessor with a diagram. What are the key components, and how do they interact?
6. What is the role of the accumulator in the 8085 microprocessor?
7. Discuss the functions of the Program Counter (PC) and Stack Pointer (SP) in 8085.
8. Describe the role of the ALU (Arithmetic and Logic Unit) and how it works with the flags register.
9. Differentiate between data transfer, arithmetic, logical, and control instructions in the 8085.
10. What is the difference between immediate, direct, and register addressing modes? Give examples of each.
11. Explain the function of the following instructions:

    MOV A, B

    MVI B, 05H

    ADD B

    RST 7.5

12. How does the branching instruction "JMP" differ from "CALL" and "RET"?
13. What happens when the "HLT" instruction is executed in the 8085 microprocessor?
14. Describe the difference between "IN" and "OUT" instructions.
15. How does the 8085 microprocessor differentiate between memory-mapped I/O and I/O-mapped I/O?
16. Explain how the address bus and data bus are used to interface with external memory.
17. Describe the memory addressing capacity of the 8085 microprocessor and how it accesses up to 64 KB of memory.
18. How are interrupts handled in the 8085? Name the various types of interrupts and their priorities.
19. Explain the function of the RST (Restart) instructions in the 8085.
20. What is the purpose of the "EI" and "DI" instructions in handling interrupts?
21. What are machine cycles, and how do they relate to instruction execution time in the 8085?
22. Write a small program to add two 8-bit numbers and store the result in memory using 8085 instructions.
23. Develop a program using the 8085 instruction set that reads input from an I/O port and displays it on another port.
24. How would you implement a delay in a program using the 8085 microprocessor?

Further Reading and References

Textbooks

1. **“Microprocessor Architecture, Programming, and Applications with the 8085" by Ramesh S. Gaonkar**
    - One of the most popular and widely-used textbooks on the 8085 microprocessor, it covers the architecture, programming, and practical

applications. It provides a strong foundation in microprocessor concepts with a focus on the 8085.

- ISBN-10: 8187972881
- ISBN-13: 978-8187972884

2. **"The 8085 Microprocessor: Architecture, Programming, and Interfacing" by K. Udaya Kumar and B.S. Umashankar**
    - This book focuses on the architecture, programming, and interfacing aspects of the 8085 microprocessor. It provides detailed examples of how to use the 8085 in real-world applications, along with interfacing techniques.
    - ISBN-10: 8131732460
    - ISBN-13: 978-8131732466
3. **"Microprocessors and Microcontrollers: Architecture, Programming and System Design 8085, 8086, 8051, 8096" by Krishna Kant**
    - This book not only covers the 8085 but also delves into other microprocessors and microcontrollers like the 8086, 8051, and 8096, providing a broader perspective on microprocessor and microcontroller design.
    - ISBN-10: 8120331915
    - ISBN-13: 978-8120331914

UNIT III

BCD to Binary and Binary to BCD conversions - ASCII to BCD and BCD to ASCII conversions - Binary to ASCII and ASCII to Binary conversions. BCD Arithmetic - BCD addition and Subtraction - Multibyte Addition and Subtraction - Multiplication and Division.

Unit Objectives:

This unit aims to provide students with a thorough understanding of essential digital conversion techniques and arithmetic operations involving binary-coded decimal (BCD), binary, and ASCII formats. It is designed to build foundational knowledge and practical skills for various applications in computer science, digital electronics, and information technology.

At the end of this unit, students will be able to:

1. Understand Number Systems and Conversions:
    - Grasp the fundamental concepts of binary, BCD, and ASCII number systems.
    - Convert between BCD and binary formats efficiently and accurately.
    - Convert between ASCII and BCD, and between ASCII and binary formats seamlessly.
2. Perform BCD Arithmetic Operations:
    - Execute BCD addition and subtraction operations with confidence.
    - Handle multibyte addition and subtraction, understanding the complexities involved.
    - Perform multiplication and division using BCD, recognizing the practical applications and challenges.
3. Implement Practical Applications:
    - Apply conversion techniques and arithmetic operations in real-world scenarios.
    - Develop algorithms and programs that incorporate these conversions and operations.
    - Troubleshoot and solve problems related to digital conversions and arithmetic in computing and electronic systems.
4. Develop Analytical and Problem-Solving Skills:
    - Analyze problems and devise effective strategies for digital conversions and arithmetic operations.

- Enhance critical thinking skills by working through practical examples and exercises.
- Build a solid foundation for advanced studies in digital systems, computer architecture, and related fields.

3.1 Introduction to Number

Number systems are fundamental to the fields of mathematics and computer science, providing a standardized way to represent and work with numbers. Different number systems are used depending on the context and the type of computation or representation required. This module explores number systems such as BCD, Binary, and ASCII.

3.1.1 Binary Number System

The binary number system is a base-2 numeral system that uses only two symbols: 0 and 1. Each digit in a binary number is called a bit. The binary numbers here are expressed in the base-2 numeral system. For example, (101)2 is a binary number.

Some binary notation lists of decimal numbers from 1 to 30 are below.

Number	Binary Number	Number	Binary Number	Number	Binary Number
1	1	11	1011	21	10101
2	10	12	1100	22	10110
3	11	13	1101	23	10111
4	100	14	1110	24	11000
5	101	15	1111	25	11001
6	110	16	10000	26	11010
7	111	17	10001	27	11011
8	1000	18	10010	28	11100
9	1001	19	10011	29	11101
10	1010	20	10100	30	11110

Binary is the fundamental language of computers and digital systems. All operations in a computer's hardware are performed using binary arithmetic.

3.1.2 Binary-Coded Decimal Number System

Binary-coded decimal (BCD) is a form of digital encoding where each decimal digit (0-9) is represented by a fixed binary number. In binary-coded decimal, each individual digit of a number is converted into a binary number, and then the BCD code is generated by combining them all. But always remember that a binary-coded decimal is not a binary representation of a decimal number. BCD notation for the digits 0-9 is listed below.

Number	BCD Number
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

3.1.3 ASCII (American Standard Code for Information Interchange)

ASCII (American Standard Code for Information Interchange) is a character encoding standard that uses 7-bit binary numbers (extended ASCII uses 8 bits) to represent text characters, control codes, and other symbols used in computers and communication equipment. ASCII was developed in the 1960s to standardize text representation for compatibility between devices and systems.

ASCII assigns a unique 7-bit binary code to each character. This includes:

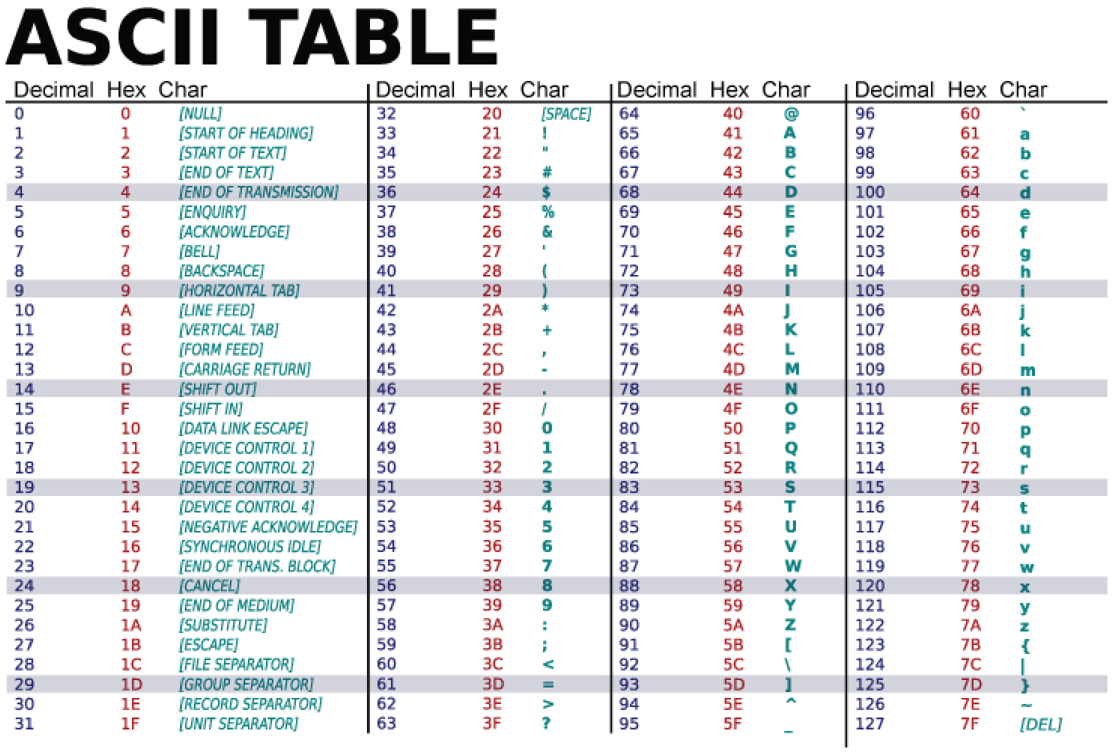
- 128 characters in total (2^7), ranging from 0 to 127.

- Control characters (0-31 and 127) for control and formatting purposes (e.g., line feed, carriage return).
- Printable characters (32-126) including letters, digits, punctuation marks, and special symbols.

Example

Here are a few ASCII characters and their corresponding binary representations:

- Decimal 65 (uppercase 'A') is represented in binary as 01000001.
- Decimal 97 (lowercase 'a') is represented in binary as 01100001.
- Decimal 48 (digit '0') is represented in binary as 00110000.
- Decimal 33 (exclamation mark '!') is represented in binary as 00100001.



ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[END OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Fig 1. ASCII Table

Let us sum up:

Binary Number System

- Base-2 system: Uses only two digits, 0 and 1, to represent numbers, where each digit is a bit (binary digit).
- Place value: Each bit represents a power of 2, starting from 202^020 on the right. Example: Binary 1011 is equal to 1×23+0×22+1×21+1×20=111 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 111×23+0×22+1×21+1×20=11 in decimal.

Binary-Coded Decimal (BCD) Number System

- Representation of decimal digits: Each decimal digit (0-9) is represented by its corresponding 4-bit binary equivalent. For example, decimal 7 is 0111 in BCD.
- Used in digital systems: Commonly used in calculators and digital clocks where exact decimal representation is needed.

ASCII (American Standard Code for Information Interchange)

- 7-bit encoding: ASCII represents characters (letters, digits, symbols) using 7 bits, allowing for 128 unique characters. Example: The letter 'A' is represented by 65 in decimal, or 1000001 in binary.
- Standard for text: Widely used for encoding text in computers and communication systems, making it possible to interchange data between systems.

Check your Progress:

1. Which number system uses only two digits, 0 and 1, to represent all numbers?

A) Decimal
B) Binary

C) Hexadecimal

D) Octal

Answer: B) Binary

2. How many bits are used to represent each decimal digit in Binary-Coded Decimal (BCD)?

A) 2 bits

B) 3 bits

C) 4 bits

D) 8 bits

Answer: C) 4 bits

3. In the binary number system, what is the decimal equivalent of the binary number 1011?

A) 9

B) 10

C) 11

D) 13

Answer: C) 11

4. What does the acronym ASCII stand for?

A) American Standard Code for Instruction Interchange

B) Advanced System for Code Information

C) American Standard Code for Information Interchange

D) Automated Standard Coding for Information Interaction

Answer: C) American Standard Code for Information Interchange

5. How many unique characters can be represented using a 7-bit ASCII code?

A) 64

B) 128

C) 256

D) 512

Answer: B) 128

3.2 BCD to Binary and Binary to BCD Conversion

3.2.1 BCD TO BINARY CONVERSION

In this section, we are converting a BCD number to its binary equivalent. First, we need to cut each nibble of the input. We can simply mask the number by adding 0FH and F0H to the input. When the higher-order nibble is cut, rotate it to the right four times to transfer it to the lower nibble. Then multiply it by 10 (0Ah).

Add the lower digit:

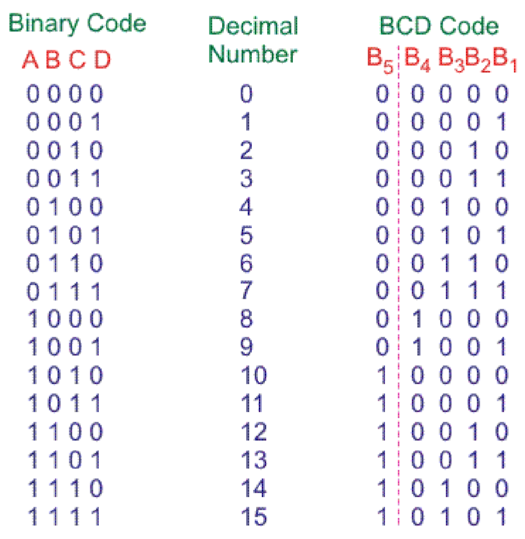
- Move the original BCD value to the accumulator.
- Mask off the higher nibble, keeping only the lower digit.
- Add this to the result.

The following code converts the BCD number to a Binary number with an example code to convert BCD 25 (stored in register B) to binary.

```
MOV A, B     ; Move BCD value (0010 0101) to accumulator
ANI 0F0H     ; Mask lower nibble (A = 0010 0000)
RRC          ; Rotate right (A = 0001 0000)
RRC          ; Rotate right (A = 0000 1000)
RRC          ; Rotate right (A = 0000 0100)
RRC          ; Rotate right (A = 0000 0010) - higher digit (2) in lower nibble
MOV C, A     ; Store 2 in register C
MVI D, 0AH   ; Move 10 (0Ah) to register D
MOV A, C     ; Move 2 to accumulator
MUL D        ; Multiply (A = 0001 0100) binary 20
```

```
MOV E, A       ; Store result in E
MOV A, B       ; Move original BCD (0010 0101) to A
ANI 0FH        ; Mask higher nibble (A = 0000 0101) - lower digit (5)
ADD E          ; Add to previous result (A = 0001 1001) binary 25
MOV BCD_BIN A ; Store the result in BCD_BIN
```

This code's result is stored in BCD_BIN. The table for BCD code equivalent to Binary code is shown below.



Binary Code	Decimal Number	BCD Code	
A B C D		B_5	B_4 $B_3$$B_2$$B_1$
0 0 0 0	0	0	0 0 0 0
0 0 0 1	1	0	0 0 0 1
0 0 1 0	2	0	0 0 1 0
0 0 1 1	3	0	0 0 1 1
0 1 0 0	4	0	0 1 0 0
0 1 0 1	5	0	0 1 0 1
0 1 1 0	6	0	0 1 1 0
0 1 1 1	7	0	0 1 1 1
1 0 0 0	8	0	1 0 0 0
1 0 0 1	9	0	1 0 0 1
1 0 1 0	10	1	0 0 0 0
1 0 1 1	11	1	0 0 0 1
1 1 0 0	12	1	0 0 1 0
1 1 0 1	13	1	0 0 1 1
1 1 1 0	14	1	0 1 0 0
1 1 1 1	15	1	0 1 0 1

Fig 2. BCD Code

3.2.2 BINARY TO BCD CONVERSION

The general algorithm for binary to BCD conversion is:

1. Divide the binary number by 10.
2. The remainder is the least significant BCD digit.
3. Repeat the division with the quotient until the quotient becomes 0.
4. Collect the remainders in reverse order to get the BCD representation.

In 8085, there was no direct division instruction. Instead, we use a subroutine that performs division by repeated subtraction.

Example code:

- Convert binary 52 (110100) to BCD
- Assume binary 52 is in register pair HL (H=00, L=34H)

```
    MVI H, 00H        ; Higher byte of binary number (00)
    MVI L, 34H        ; Lower byte of binary number (34H = 110100 = 52 in decimal)
    LXI B, 000AH      ; BC = 000A (divisor 10, 0AH in hex)
    PUSH B            ; Save divisor on stack

    CALL BINDIV       ; Call binary division subroutine

    ; After BINDIV, L contains remainder (2), H contains quotient (5)
    MOV A, L          ; A = 02 (BCD for 2)
    RLC               ; Rotate left (A = 0010 0000)
    RLC               ; Rotate left (A = 0100 0000)
    RLC               ; Rotate left (A = 1000 0000)
    RLC               ; Rotate left (A = 0000 0000, carry flag set)
    MOV A, H          ; A = 05 (BCD for 5)
    RAL               ; Rotate left through carry (A = 0101 0000)
    ORI 02H           ; OR with 02 (A = 0101 0010) - BCD 52
                    ; Result: Accumulator (A) now contains 0101 0010 (BCD 52)

       HLT          ; Halt program
```

Binary Division Subroutine (BINDIV)

```
BINDIV:
    MVI D, 00H        ; D will count number of subtractions (quotient)
DIVLP:
    MOV A, L          ; Move lower byte of dividend to A
    SUB C             ; Subtract lower byte of divisor
    MOV L, A          ; Store result back in L
    MOV A, H          ; Move higher byte of dividend to A
    SBB B           ; Subtract higher byte of divisor with borrow
    MOV H, A        ; Store result back in H
    JC DIVDN        ; If borrow (carry), division is done
    INR D           ; Increment quotient
    JMP DIVLP       ; Continue division
DIVDN:
    DAD B           ; Add divisor back (undo last subtraction)
    MOV A, D        ; Move quotient to A
    MOV H, A        ; Store quotient in H

    RET             ; Return from subroutine
```

Let us sum up:

BCD to Binary Conversion

1. **Convert each BCD digit to its binary equivalent**: Each 4-bit BCD digit (0-9) is converted to its corresponding binary value. For example, BCD 0101 0011 (which represents decimal 53) is converted to binary 110101.
2. **Combine binary digits**: After converting each BCD digit to binary, combine the binary results to form the final binary number.

Binary to BCD Conversion

1. **Divide the binary number**: Separate the binary number into its decimal equivalent, and then convert each decimal digit to a 4-bit BCD representation. For example, binary 101101 (which equals 45 in decimal) is converted to BCD 0100 0101.
2. **Pack into 4-bit groups**: Each decimal digit is represented by a 4-bit BCD code, which is packed together to represent the entire number.

Check your Progress:

1. In BCD (Binary-Coded Decimal), how many bits are used to represent each decimal digit?

A) 2 bits

B) 3 bits

C) 4 bits

D) 8 bits

Answer: C) 4 bits

2. What is the BCD representation of the decimal number 25?

A) 0010 0101

B) 1010 0101

C) 1100 0011

D) 0011 0100

Answer: A) 0010 0101

3. Which of the following binary numbers represents the decimal value 13?

A) 1010

B) 1110

C) 1101

D) 1001

Answer: C) 1101

4. When converting the BCD 0110 1001 to binary, what is the resulting binary number?

A) 101110

B) 110101

C) 100001

D) 1000101

Answer: A) 101110 (BCD 0110 1001 = 69 in decimal = 101110 in binary)

5. What is the BCD equivalent of the binary number 110010 (which equals 50 in decimal)?

A) 0101 0000
B) 0011 0100
C) 0100 0011
D) 0110 1010

Answer: A) 0101 0000 (BCD for 50 is 0101 0000)

6. How is the binary number 101101 (which equals 45 in decimal) represented in BCD?

A) 0101 1010
B) 1001 1100
C) 0100 0101
D) 1101 0110

Answer: C) 0100 0101 (BCD for 45 is 0100 0101)

3.3 BCD to ASCII and ASCII to BCD Conversion

3.3.1 BCD TO ASCII CONVERSION

Converting BCD (Binary Coded Decimal) to ASCII in 8085 assembly involves manipulating 4-bit nibbles representing decimal digits. The process uses bitwise operations (ANI) to isolate high and low nibbles, rotations (RRC) to shift the high nibble if needed, and addition (ADD 30H) to convert digits to their ASCII equivalents. ASCII values for digits start at 30H ('0'). The program typically uses memory (DB) to store BCD input and ASCII output, moves data with MOV, and employs subroutines (CALL, RET) for clarity. This conversion is crucial for displaying numeric data on ASCII-based devices in microprocessor systems.

```
ORG 0000H                              ; Starting address


MOV A, BCD_NUM                         ; Load the BCD number into the accumulator

CALL CONVERT_HIGH_NIBBLE               ; Convert high nibble to ASCII

MOV ASCII_HIGH, A                      ; Store the high ASCII digit


MOV A, BCD_NUM                         ; Load the BCD number into the accumulator
again

CALL CONVERT_LOW_NIBBLE                ; Convert low nibble to ASCII

MOV ASCII_LOW, A                       ; Store the low ASCII digit


HLT                                    ; Halt the program


BCD_NUM     DB 25H                     ; Example BCD number (25 in BCD)

ASCII_HIGH  DB ?                       ; Resulting high ASCII digit

ASCII_LOW   DB ?                       ; Resulting low ASCII digit


CONVERT_HIGH_NIBBLE:                   ; Convert high nibble to ASCII

   MOV B, A                            ; Copy the BCD number to register B

   ANI 0F0H                            ; Mask the lower nibble

   RRC                        ; Rotate right to bring the high nibble to lower nibble

   RRC

   RRC

   RRC
   MOV A, B                    ; Move the original BCD number back to accumulator
   ANI 0FH                     ; Mask the upper nibble
   ADD B                       ; Add the high nibble to the lower nibble
   ADD 30H                     ; Convert to ASCII by adding ASCII value of '0'
   RET


CONVERT_LOW_NIBBLE: ; Convert low nibble to ASCII
   ANI 0FH                     ; Mask the upper nibble
   ADD 30H                     ; Convert to ASCII by adding ASCII value of '0'
   RET
END                            ; End of program
```

3.3.2 ASCII TO BCD CONVERSION

This 8085 assembly code converts a single ASCII digit ('7') to two-digit BCD (Binary Coded Decimal). It subtracts ASCII '0' (30H) to get the binary value, then uses ANI and DAA for BCD adjustment. The high BCD digit is set directly (0 for single digits). For the low digit, it divides 10 by the input using DIV, placing the input in the ones place. The code assumes single-digit input, using SUB, ANI, DAA, and DIV instructions. It's useful for systems receiving ASCII input but requiring BCD for processing or display, though limited to single digits without modification.

```
ORG 0000H                   ; Starting address


MOV A, ASCII_DATA           ; Load the ASCII character into the accumulator
SUB '0'                     ; Convert ASCII to binary
MOV BCD_DATA, A             ; Store the binary equivalent
```

```
MVI A, 00H              ; Clear the accumulator
MOV B, A                ; Clear the B register

MOV A, BCD_DATA         ; Move the binary number to accumulator
ANI 0FH                 ; Mask higher nibble
DAA                     ; Decimal adjust for BCD

MOV BCD_HIGH, A         ; Store the high BCD digit

MOV A, BCD_DATA         ; Move the binary number to accumulator
MOV B, A                ; Move it to B register for the division

MVI A, 0AH              ; Move decimal 10 to accumulator for division
DIV B                   ; Divide by 10

MOV BCD_LOW, A          ; Store the low BCD digit

HLT                     ; Halt

ASCII_DATA DB '7'       ; Example ASCII character representing a digit
BCD_HIGH DB ?           ; Resulting high BCD digit
BCD_LOW DB ?            ; Resulting low BCD digit
BCD_DATA DB ?           ; Temporary storage for the binary equivalent
```

```
END ; End of program
```

Let us sum up:

BCD to ASCII Conversion

1. **Convert each BCD digit to its ASCII equivalent**: Each BCD digit (4-bit binary) is converted to its 8-bit ASCII code by adding a fixed value of 0011 0000 (or 30H in hexadecimal). For example, BCD 0101 (which represents the digit 5) becomes ASCII 0011 0101 (or 35H in hexadecimal).
2. **Store as ASCII characters**: After conversion, each BCD digit is stored as a corresponding ASCII character, which is typically used for displaying numbers in readable text format.

ASCII to BCD Conversion

1. **Remove the ASCII prefix**: Convert each ASCII character (which is in the range of 0011 0000 to 0011 1001 for digits 0-9) to BCD by subtracting 0011 0000 (30H). For example, ASCII 0011 0101 (which represents the character '5') becomes BCD 0101.
2. **Pack into 4-bit BCD**: After removing the ASCII prefix, the resulting binary is stored as a 4-bit BCD value representing the original decimal digit.

Check your Progress:

1. How is each BCD digit converted to its ASCII equivalent?

A) By adding 0000 0000
B) By adding 0011 0000
C) By subtracting 0011 0000
D) By multiplying by 2

Answer: B) By adding 0011 0000

2. What is the ASCII representation of the BCD value 0011 (which represents the digit 3)?

A) 0011 0011

B) 0011 0001

C) 0010 0011

D) 0011 0100

Answer: A) 0011 0011 (ASCII for digit 3 is 0011 0011)

3. When converting the ASCII character 0011 0101 (which represents the character '5') to BCD, what is the resulting BCD value?

A) 0101

B) 0011

C) 0000

D) 0010

Answer: A) 0101 (ASCII 0011 0101 - 0011 0000 = BCD 0101)

4. Which of the following is true regarding ASCII values for digits 0-9?

A) They range from 0011 0000 to 0011 1000

B) They range from 0000 0000 to 0000 1001

C) They range from 0010 0000 to 0010 1001

D) They range from 0011 0000 to 0011 1001

Answer: D) They range from 0011 0000 to 0011 1001

5. What is the BCD equivalent of the ASCII character 0011 1000 (which represents the character '8')?

A) 0111

B) 1000

C) 0011

D) 1001

Answer: A) 1000 (ASCII 0011 1000 - 0011 0000 = BCD 1000)

3.4 Binary to ASCII and ASCII to Binary Conversion

3.4.1 BINARY TO ASCII CONVERSION

To convert a binary value to its ASCII representation in 8085 assembly, you need to add the ASCII offset for '0' (30H). This process can be summarized in the following steps:

1. **Input:** Obtain the binary value (0 to 9) to be converted to its ASCII representation.
2. **Conversion:** Add 30H to the binary value to get the corresponding ASCII code.
3. **Output:** Store or display the ASCII result.

```
ORG 0000H              ; Starting address
MOV A, BINARY_VAL      ; Load the binary value into the accumulator
ADD A, 30H             ; Add 30H to convert binary value to ASCII
MOV ASCII_VAL, A       ; Store the ASCII result

HLT                    ; Halt the program
```

3.4.2 ASCII to Binary Conversion

To convert ASCII digits to binary in 8085 assembly, load the ASCII character (values 30H to 39H) into the accumulator. Subtract 30H (the ASCII value for '0') to obtain the binary equivalent, as digits '0' to '9' map to values 0-9.

```
ORG 0000H               ; Starting address
MOV A, ASCII_DATA       ; Load the ASCII character into the accumulator
SUB 30H                 ; Convert ASCII to binary
MOV BINARY_DATA, A      ; Store the binary equivalent


HLT                     ; Halt the program
```

Let us sum up:

Binary to ASCII Conversion

1. **Group binary bits**: Divide the binary input into 8-bit segments, as ASCII characters are represented by 8 bits. For example, the binary 01000001 corresponds to the ASCII character 'A'.
2. **Convert to ASCII character**: Each 8-bit segment is converted to its corresponding ASCII character based on the ASCII table. For instance, the binary 01101000 translates to the ASCII character 'h'.

ASCII to Binary Conversion

1. **Convert each ASCII character to binary**: Each character from the ASCII input is represented by its corresponding 8-bit binary value. For example, the ASCII character 'A' is represented as 01000001.
2. **Concatenate binary values**: The binary values for each ASCII character are concatenated together to form a continuous binary string. For example, the ASCII string "Hi" is converted to binary as 01001000 01101001.

Check your Progress:

1. How many bits are used to represent each ASCII character?

A) 4 bits

B) 6 bits

C) 8 bits
D) 16 bits

Answer: C) 8 bits

2. What is the ASCII representation of the binary value 01000010?

A) 'A'
B) 'B'
C) 'C'
D) 'D'

Answer: B) 'B' (The binary 01000010 corresponds to the ASCII character 'B')

3. When converting the ASCII character 'C' (which has a decimal value of 67) to binary, what is the resulting binary value?

A) 01000010
B) 01000011
C) 01000100
D) 01000101

Answer: B) 01000011 (The ASCII character 'C' is represented in binary as 01000011)

4. What is the binary equivalent of the ASCII string "Hi"?

A) 01001000 01101001
B) 01001000 01101010
C) 01000111 01101001
D) 01001001 01101000

Answer: A) 01001000 01101001 (The ASCII string "Hi" corresponds to the binary values for 'H' and 'i')

5. If the binary sequence 01101000 is converted to ASCII, which character is represented?

A) 'g'
B) 'h'
C) 'i'
D) 'j'

Answer: B) 'h' (The binary 01101000 corresponds to the ASCII character 'h')

3.5 BCD Addition and Subtraction

3.5.1 BCD ADDITION

This 8085 assembly program adds two BCD numbers and stores the result in BCD format. It begins by loading the first BCD number (25H) into the accumulator and then adds the second BCD number (37H). The DAA instruction is used to adjust the result to a proper BCD format.

```
ORG 0000H  ; Starting address


; Assume the BCD numbers to be added are in BCD1 and BCD2
; The result will be stored in RESULT


MOV A, BCD1         ; Load the first BCD number into the accumulator
ADD BCD2            ; Add the second BCD number
DAA                 ; Decimal adjust the result to convert it back to BCD
MOV RESULT, A       ; Store the result


HLT ; Halt the program
```

```
BCD1    DB 25H      ; Example BCD number 1 (25)

BCD2    DB 37H      ; Example BCD number 2 (37)

RESULT  DB ?        ; Result of BCD addition

END                 ; End of program
```

3.5.2 BCD SUBTRACTION

Steps for BCD Subtraction

1. **Load the BCD Numbers**: Load the BCD numbers to be subtracted into registers.
2. **Perform Subtraction**: Subtract the second BCD number from the first.
3. **Adjust Result**: Use the DAA (Decimal Adjust Accumulator) instruction to adjust the result to a valid BCD format.
4. **Store the Result**: Store the adjusted result in the desired memory location.

```
ORG 0000H           ; Starting address


; Assume the BCD numbers to be subtracted are in BCD1 and BCD2

; The result will be stored in RESULT


MOV A, BCD1                 ; Load the first BCD number into the accumulator

SUB BCD2                    ; Subtract the second BCD number

CPI 0AH                     ; Check if the result needs decimal adjustment

JC NO_ADJUST                ; Jump if no adjustment is needed

ADI 90H                     ; Adjust the result for decimal subtraction
```

```
SUI 6                    ; Subtract 6 to correct the subtraction result


NO_ADJUST:

DAA                      ; Decimal adjust the result to convert it back to BCD

MOV RESULT, A            ; Store the result


HLT ; Halt the program


BCD1    DB 55H           ; Example BCD number 1 (55)

BCD2    DB 27H           ; Example BCD number 2 (27)

RESULT  DB ?             ; Result of BCD subtraction


END                      ; End of program
```

Let us sum up:

BCD Addition

1. **Add BCD digits**: Each pair of BCD digits is added using regular binary addition. The sum may exceed the maximum BCD digit (9), which is represented as 1001 in binary.
2. **Adjust for carry**: If the sum of any BCD digit exceeds 9 (or 1001), a correction is needed. A constant of 0000 1010 (which is 10 in decimal) is added to the result, and the carry is accounted for in the next digit.

BCD Subtraction

1. **Subtract BCD digits**: BCD subtraction involves subtracting the BCD digits using binary subtraction rules. If the result of a subtraction operation is negative, it requires adjustment.

2. **Borrowing and adjustment**: If a BCD digit is subtracted from a smaller digit, a borrowing process occurs, and a correction factor of 0000 1010 (10) is added to the result to ensure it remains a valid BCD representation.

Check your Progress:

1. What is the maximum BCD digit?

A) 7

B) 8

C) 9

D) 10

Answer: C) 9

2. In BCD addition, what is the correction factor added if the sum exceeds 9?

A) 0000 0110

B) 0000 1000

C) 0000 1010

D) 0000 1100

Answer: C) 0000 1010 (which is 10 in decimal)

3. When performing BCD subtraction, what occurs if the minuend is less than the subtrahend?

A) Nothing; the result is valid

B) A borrow is required

C) The subtraction is invalid

D) The result is automatically adjusted to 0

Answer: B) A borrow is required

4. In BCD addition, what would the result be for the operation 0110 (6) + 0101 (5)?

A) 0001 0011

B) 0000 1011

C) 0001 0001

D) 0000 1100

Answer: A) 0001 0011 (The sum is 11 in decimal; since it's a BCD result, a correction factor is applied)

5. What is the BCD result of subtracting 0100 (4) from 0011 (3)?

A) 0000

B) 0001 0110

C) 0001 0010

D) 0000 1000

Answer: D) 0000 1000 (Since you need to borrow, you effectively get 10 - 4 = 6)

6. In BCD subtraction, after borrowing, which adjustment factor is typically added to maintain the BCD representation?

A) 0000 0001

B) 0000 0100

C) 0000 1000

D) 0000 1010

Answer: D) 0000 1010 (This adjustment is made when the result is negative)

3.6 Multibyte BCD Addition and Subtraction

3.6.1 MULTIBYTE BCD ADDITION

Steps for Multibyte BCD Addition

1. **Initialize Carry:** Set the carry flag (CY) to 0 before starting the addition.
2. **Load the BCD Numbers:** Load each BCD number into separate registers or memory locations.
3. **Add the Least Significant Digits (LSDs):** Add the LSDs of the BCD numbers, including the previous carry if any.
4. **Adjust for BCD:** Use the DAA (Decimal Adjust Accumulator) instruction to adjust the result.
5. **Store the LSD Result:** Store the LSD result in memory or a register.
6. **Add the Most Significant Digits (MSDs):** Add the MSDs of the BCD numbers, including the carry from the LSD addition.
7. **Adjust for BCD:** Use DAA to adjust the MSD result.
8. **Store the MSD Result:** Store the MSD result in memory or a register.
9. **Repeat for Remaining Bytes:** If there are more bytes to add, repeat steps 3-8 for each byte.
10. **Handle Overflow:** Check for overflow after adding the last byte. If overflow occurs, handle it appropriately (e.g., flag an error or adjust the result).

```
ORG 0000H                  ; Starting address

; Assuming the numbers are 3 bytes long
LXI H, BCD1                       ; Point to the start of the first BCD
number
```

```
LXI D, BCD2                        ; Point to the start of the second
BCD number
LXI B, RESULT                      ; Point to the start of the result
MVI C, 03                    ; Set byte count to 3

; Clear carry
MOV A, 00
MOV E, A
ADD_LOOP:
   LDAX D                          ; Load byte from BCD2
   ADD M                           ; Add byte from BCD1
   DAA                             ; Decimal adjust accumulator
   MOV M, A                        ; Store result byte in RESULT
   INX H                            ; Increment pointer for BCD1
   INX D                     ; Increment pointer for BCD2
   INX B                     ; Increment pointer for RESULT
   DCR C                      ; Decrement byte count
   JZ DONE                         ; If byte count is zero, end
   MOV A, E                        ; Move carry to accumulator
   JNC NO_CARRY
   MVI A, 01                       ; Set A to 1 if carry
NO_CARRY:
   MOV E, A                        ; Store carry
   JMP ADD_LOOP
DONE:
   HLT                             ; Halt the program
BCD1    DB 25H, 34H, 12H           ; Example BCD number 1
BCD2    DB 37H, 45H, 23H           ; Example BCD number 2
RESULT  DS 3                       ; Space for the result (3 bytes)

END                                ; End of program
```

3.6.2 MULTIBYTE BCD SUBTRACTION

Steps for Multibyte BCD Subtraction

1. **Initialize Borrow**: Set the borrow flag (CY) to 0 before starting the subtraction.
2. **Load the BCD Numbers**: Load each BCD number into separate registers or memory locations.
3. **Subtract the Least Significant Digits (LSDs)**: Subtract the LSDs of the BCD numbers, including the previous borrow if any.
4. **Adjust for BCD**: Use the DAA (Decimal Adjust Accumulator) instruction to adjust the result.
5. **Store the LSD Result**: Store the LSD result in memory or a register.
6. **Subtract the Most Significant Digits (MSDs)**: Subtract the MSDs of the BCD numbers, including the borrow from the LSD subtraction.
7. **Adjust for BCD**: Use DAA to adjust the MSD result.
8. **Store the MSD Result**: Store the MSD result in memory or a register.
9. **Repeat for Remaining Bytes**: If there are more bytes to subtract, repeat steps 3-8 for each byte.
10. **Handle Borrow**: Check for borrow after subtracting the last byte. If borrow occurs, handle it appropriately (e.g., flag an error or adjust the result).

```
ORG 0000H ; Starting address

; Assuming the numbers are 3 bytes long
LXI H, BCD1                  ; Point to the start of the first BCD number
LXI D, BCD2                  ; Point to the start of the second BCD number
LXI B, RESULT                ; Point to the start of the result
MVI C, 03              ; Set byte count to 3

; Clear carry
MOV A, 00
MOV E, A
```

```
ADD_LOOP:
   LDAX D                        ; Load byte from BCD2
   ADD M                         ; Add byte from BCD1
   DAA                           ; Decimal adjust accumulator
   MOV M, A                      ; Store result byte in RESULT
   INX H                         ; Increment pointer for BCD1
   INX D                         ; Increment pointer for BCD2
   INX B                         ; Increment pointer for RESULT
   DCR C                         ; Decrement byte count
   JZ DONE                       ; If byte count is zero, end
   MOV A, E                      ; Move carry to accumulator
   JNC NO_CARRY
   MVI A, 01                     ; Set A to 1 if carry
NO_CARRY:
   MOV E, A                      ; Store carry
   JMP ADD_LOOP

DONE:
   HLT ; Halt the program

BCD1    DB 25H, 34H, 12H         ; Example BCD number 1
BCD2    DB 37H, 45H, 23H         ; Example BCD number 2
RESULT  DS 3                     ; Space for the result (3 bytes)

END                              ; End of program
```

Let us sum up:

Multibyte BCD Addition

1. **Segment the BCD values**: Multibyte BCD numbers are divided into individual bytes, with each byte representing a single BCD digit (4 bits each). For

example, the BCD number 0011 0001 0100 represents the decimal number 314.

2. **Perform byte-wise addition**: Add each BCD byte starting from the least significant byte (LSB) to the most significant byte (MSB), carrying any overflow to the next byte. If the sum of any byte exceeds 1001, add the correction factor 0000 1010 and carry over the adjustment to the next byte.

Multibyte BCD Subtraction

1. **Segment the BCD values**: Similar to addition, multibyte BCD numbers are divided into individual bytes for subtraction. Each byte is processed separately.
2. **Perform byte-wise subtraction**: Subtract each BCD byte starting from the LSB to the MSB. If a subtraction results in a negative value, borrow from the next higher byte, and add the correction factor 0000 1010 to maintain a valid BCD representation.

Check your Progress:

1. When performing multibyte BCD addition, what should you do if the sum of a byte exceeds 1001?

A) Ignore the overflow
B) Subtract 0000 0001
C) Add the correction factor 0000 1010
D) Convert the result to decimal

Answer: C) Add the correction factor 0000 1010

2. In multibyte BCD subtraction, if a BCD byte results in a negative value, what is the necessary adjustment?

A) Add 0000 0001 to the result
B) Borrow from the next higher byte and add 0000 1010
C) Set the result to zero
D) Ignore the borrow

Answer: B) Borrow from the next higher byte and add 0000 1010

3. What is the first step in performing multibyte BCD addition?

A) Convert BCD to decimal
B) Segment the BCD values into individual bytes
C) Multiply the BCD numbers
D) Perform a binary addition

Answer: B) Segment the BCD values into individual bytes

4. In a multibyte BCD number 0011 0001 0100 (representing 314), how would you add 0000 0101 (representing 5) to it?

A) Add each byte directly without adjustments
B) Add the LSB first, check for overflow, and adjust as necessary
C) Convert both to decimal, add, and convert back
D) Both A and B are correct

Answer: D) Both A and B are correct

3.7 Multibyte BCD Multiplication and Division

3.7.1 MULTIBYTE BCD MULTIPLICATION

BCD multiplication can be approached by performing repeated addition. Here, we'll multiply two 2-byte BCD numbers.

Data:

- BCD1: First BCD number
- BCD2: Second BCD number
- RESULT: Result of the multiplication (up to 4 bytes)

```
ORG 0000H            ; Starting address

LXI H, BCD1                  ; Point to the start of the first BCD number

LXI D, BCD2                  ; Point to the start of the second BCD number

LXI B, RESULT                ; Point to the start of the result

MVI C, 04                    ; Set byte count for the result (4 bytes)


 ; Clear the RESULT array

MVI A, 00

MOV M, A

INX B

MOV M, A

INX B

MOV M, A

INX B

MOV M, A


LXI H, BCD2                  ; Point to the start of the second BCD number

LXI D, RESULT                ; Point to the start of the result


MUL_LOOP:

   MOV A, M                  ; Load byte from BCD2

   CALL MULTIPLY_BYTE ; Multiply each byte

   INX H                      ; Increment pointer for BCD2
   DCR C                    ; Decrement byte count
   JNZ MUL_LOOP             ; Repeat for all bytes


HLT ; Halt the program


MULTIPLY_BYTE:
   LXI H, BCD1              ; Point to the start of the first BCD number
   MOV E, A                 ; Store multiplier in E
   MVI A, 00
   MOV F, A                 ; Clear F (upper byte of product)


MUL_BYTE_LOOP:
   LDAX H                   ; Load byte from BCD1
   CALL ADD_BCD_BYTE ; Add the BCD number to the result
   INX H               ; Increment pointer for BCD1
   DCR E               ; Decrement multiplier
   JNZ MUL_BYTE_LOOP ; Repeat for the number of times specified by E


   RET


ADD_BCD_BYTE:               ; Add the bytes with decimal adjustment
   MOV A, M
   ADD D
```

```
    DAA

    MOV M, A

    INX D

    RET


BCD1   DB 25H, 34H          ; Example BCD number 1 (3425)

BCD2   DB 12H, 34H          ; Example BCD number 2 (3412)

RESULT DS 4         ; Space for the result (4 bytes)


END                         ; End of program
```

3.7.2 MULTIBYTE BCD DIVISION

BCD division can be approached by repeated subtraction. Here, we'll divide two 2-byte BCD numbers.

Data:

- BCD1: Dividend (number to be divided)
- BCD2: Divisor (number to divide by)
- RESULT: Quotient of the division (up to 2 bytes)
- REMAINDER: Remainder of the division (up to 2 bytes)

```
ORG 0000H                   ; Starting address



LXI H, BCD1                 ; Point to the start of the dividend
LXI D, BCD2                ; Point to the start of the divisor

LXI B, RESULT              ; Point to the start of the quotient

LXI H, REMAINDER           ; Point to the start of the remainder



; Clear the RESULT and REMAINDER arrays

MVI A, 00

MOV M, A

INX B

MOV M, A

INX B

MOV M, A

INX B

MOV M, A

DIV_LOOP:

   LXI H, BCD1                          ; Point to the start of the dividend

   CALL SUBTRACT_BCD_BYTE          ; Subtract the divisor

   JC DIV_DONE                    ; If carry (dividend < divisor), division done

   INX B

   JMP DIV_LOOP
```

```
DIV_DONE:

   HLT ; Halt the program

SUBTRACT_BCD_BYTE:          ; Subtract the bytes with decimal adjustment

   MOV A, M

   SUB D

   CPI 0AH                  ; Check if adjustment needed

   JC NO_ADJUST

   SUI 06H                  ; Subtract 6 for decimal correction

NO_ADJUST:

   DAA

   MOV M, A

   INX D

   RET

BCD1     DB 56H, 78H        ; Example BCD dividend (7856)

BCD2     DB 12H, 34H        ; Example BCD divisor (3412)

RESULT    DS 2              ; Space for the quotient (2 bytes)

REMAINDER DS 2              ; Space for the remainder (2 bytes)

END ; End of program
```

Let us sum up:

Multibyte BCD Multiplication

1. **Segment the BCD values**: Similar to addition and subtraction, multibyte BCD numbers are divided into individual bytes, where each byte represents a single BCD digit (4 bits each).
2. **Perform multiplication**: Each BCD digit is multiplied using binary multiplication rules. The products may be larger than a single BCD digit (up to 16 in decimal), so each product is checked. If it exceeds 1001, a correction of 0000 1010 must be added, and any overflow is carried to the next byte.

Multibyte BCD Division

1. **Segment the BCD values**: The multibyte BCD numbers are divided into individual bytes for division, just like in addition, subtraction, and multiplication.
2. **Perform division**: Each BCD digit is divided using binary division rules. If a byte results in a quotient or remainder that is not valid in BCD (greater than 9), adjustments may be required. The quotient is checked, and if necessary, the correction factor 0000 1010 is used to adjust the result to maintain valid BCD representation.

Check your Progress:

1. In multibyte BCD multiplication, what should be done if a product exceeds 1001?

A) Ignore the overflow
B) Set the product to zero
C) Add the correction factor 0000 1010
D) Divide the product by 2

Answer: C) Add the correction factor 0000 1010

2. When performing multibyte BCD division, what is done if the quotient exceeds 9?

A) Ignore the result
B) Adjust using the correction factor 0000 1010
C) Convert to binary
D) Subtract 1 from the quotient

Answer: B) Adjust using the correction factor 0000 1010

3. What is the first step in multibyte BCD multiplication?

A) Convert BCD to decimal
B) Segment the BCD values into individual bytes
C) Add the BCD numbers
D) Perform binary multiplication

Answer: B) Segment the BCD values into individual bytes

4. In multibyte BCD division, which of the following is a necessary condition for the quotient?

A) It must be a decimal number
B) It must be a valid BCD representation
C) It must be an even number
D) It must be less than 10

Answer: B) It must be a valid BCD representation

5. During multibyte BCD multiplication, what is the maximum value a single BCD product can represent?

A) 5
B) 9

C) 10

D) 15

Answer: D) 15 (as the maximum value for two BCD digits, 9 * 9)

Conclusion

This unit comprehensively studies digital conversion techniques and arithmetic operations involving binary-coded decimal (BCD), binary, and ASCII formats. Students learn the fundamental concepts of these number systems and how to convert between them accurately. They also gain proficiency in performing BCD arithmetic operations, including addition, subtraction, multiplication, and division, with a focus on multibyte operations. Practical applications in real-world scenarios are emphasized, helping students develop algorithms and troubleshoot issues related to digital conversions and arithmetic. The unit also enhances analytical and problem-solving skills, preparing students for advanced studies in digital systems and computer architecture.

Summary:

In this unit, students will gain a comprehensive understanding of digital conversion techniques and arithmetic operations involving binary-coded decimal (BCD), binary, and ASCII formats. They will learn the fundamental concepts of these number systems and how to convert between them accurately.

Students will develop skills in performing BCD arithmetic operations, including addition, subtraction, multiplication, and division, with a focus on multibyte operations. They will also learn to implement these concepts in practical applications, develop algorithms, and troubleshoot issues related to digital conversions and arithmetic.

The unit aims to enhance students' analytical and problem-solving skills, preparing them for advanced studies in digital systems and computer architecture.

Activities

Activity 1: Convert the BCD number 0101 1001 to its binary equivalent.

- **Objective:** To understand number system differences and conversion techniques for microprocessor operations.

Activity 2: What is the ASCII equivalent of the character 'A'?

- **Objective:** To understand how ASCII values represent characters in digital systems.

Check Your Progress

1. Explain the process of converting a Binary-Coded Decimal (BCD) number to a binary number. Provide an example to illustrate the conversion.

----

2. Describe the steps involved in performing BCD addition and BCD subtraction. What rules must be followed to ensure the correct results in BCD arithmetic?

----

3. How do you perform ASCII to BCD and BCD to ASCII conversions? Provide a detailed explanation and an example for each conversion

----

Self-Assessment Questions

1. Explain the difference between BCD and binary representations of numbers. Why might one be preferred over the other in certain situations?
2. Convert the following decimal numbers to binary and BCD. Show your work. a) 42 b) 79 c) 105
3. What is the purpose of the ASCII encoding system? How does it relate to binary and BCD?
4. Describe the process of converting a BCD number to binary. What are the key steps involved?
5. How would you convert an ASCII character representing a digit to its binary equivalent? Provide an example.
6. Explain the importance of the DAA (Decimal Adjust Accumulator) instruction in BCD arithmetic. When and why is it used?
7. Write a pseudo-code algorithm for performing multibyte BCD addition. What challenges might you encounter, and how would you address them?
8. Compare and contrast the processes of BCD addition and BCD subtraction. What are the key similarities and differences?
9. Describe the approach for performing BCD multiplication. Why might this be more complex than binary multiplication?
10. How would you handle overflow in BCD arithmetic operations? Provide an example scenario.
11. Explain the process of converting a binary number to its ASCII representation. What considerations must be made for multi-digit numbers?
12. What are the potential applications of BCD arithmetic in real-world computing systems? Provide at least two examples.
13. Describe the steps involved in performing a multibyte BCD division. What challenges might arise, and how can they be addressed?
14. How does the binary representation of a number differ from its BCD representation? Provide an example to illustrate the difference.

Further Reading and References

Textbooks

1. **"Digital Design and Computer Architecture" by David Money Harris and Sarah L. Harris**
    - Covers Boolean algebra and logic gates, Sequential circuits and finite state machines, the design of microprocessors, including instruction sets and pipelining.
    - ISBN: 978-0128200643
2. **"Microprocessor Architecture, Programming, and Applications with the 8085" by Ramesh S. Gaonkar**
    - Includes in-depth study of the 8085 microprocessor, focusing on its architecture, programming, and applications.
    - ISBN: 978-8187972884
3. **"Digital Computer Electronics" by Albert P. Malvino and Jerald A. Brown**
    - A foundational understanding of digital electronics with a focus on the practical implementation of digital computer circuits.
    - ISBN: 978-0028028337

Online Resources

1. https://www.tutorialspoint.com/bcd-to-binary-conversion
2. https://www.geeksforgeeks.org/bcd-or-binary-coded-decimal-addition/
3. https://www.electronicshub.org/ascii-to-binary-and-binary-to-ascii-conversion/

UNIT IV

The 8085 Interrupts – RIM AND SIM instructions - 8259 Programmable Interrupt Controller - Direct Memory Access (DMA) and 8257 DMA controller.

Unit Objectives:

In this unit, students will have a comprehensive understanding of microcontrollers, with a focus on the 8085 Interrupts. Learners will be able to analyze different types of 8085 Interrupts, RIM AND SIM instructions, 8259 Programmable Interrupt Controller, Direct Memory Access (DMA), and 8257 DMA controller to understand the workings of simple embedded systems. By the end of this unit, students will be able to

- Understand the interrupt system of the 8085 microprocessor, including the classification and priority of interrupts.
- Explain the functionality and usage of interrupt management instructions (EI, DI, SIM, RIM).
- Describe the role and features of the 8259 Programmable Interrupt Controller.
- Understand the concept and working of the Direct Memory Access (DMA) controller and its impact on system performance.
- Analyze the advantages and challenges associated with using interrupts and DMA in microprocessor systems.

4.1 The 8085 Interrupts

In the 8085 microprocessor, an interrupt is a signal that temporarily suspends the normal execution of a program and redirects the control to a specific interrupt service routine (ISR). Interrupts allow the microprocessor to respond to external events, such as user input, system events, or hardware signals, without the need for constant polling.

4.1.1 There are five interrupt signals in the 8085 microprocessor:

TRAP: The TRAP interrupt is a non-maskable interrupt that is generated by an external device, such as a power failure or a hardware malfunction. The TRAP interrupt has the highest priority and cannot be disabled.

RST 7.5: The RST 7.5 interrupt is a maskable interrupt that is generated by a software instruction. It has the second highest priority.

RST 6.5: The RST 6.5 interrupt is a maskable interrupt that is generated by a software instruction. It has the third highest priority.

RST 5.5: The RST 5.5 interrupt is a maskable interrupt that is generated by a software instruction. It has the fourth highest priority.

INTR: The INTR interrupt is a maskable interrupt that is generated by an external device, such as a keyboard or a mouse. It has the lowest priority and can be disabled.

When microprocessor receives any interrupt signal from peripheral(s) which are requesting its services, it stops its current execution and program control is transferred to a sub-routine by generating CALL signal and after executing sub-routine by generating RET signal again program control is transferred to main program from where it had stopped. When microprocessor receives interrupt signals, it sends an acknowledgement (INTA) to the peripheral which is requesting for its service. Interrupts can be classified into various categories based on different parameters:

Hardware and Software Interrupts – When microprocessors receive interrupt signals through pins (hardware) of microprocessor, they are known as Hardware Interrupts. There are 5 Hardware Interrupts in 8085 microprocessor. They are – INTR, RST 7.5, RST 6.5, RST 5.5, TRAP Software Interrupts are those which are inserted in between the program which means these are mnemonics of microprocessor. There are 8 software interrupts in 8085 microprocessor. They are – RST 0, RST 1, RST 2, RST 3, RST 4, RST 5, RST 6, RST 7.

Vectored and Non-Vectored Interrupts – Vectored Interrupts are those which have fixed vector address (starting address of sub-routine) and after executing these, program control is transferred to that address. Vector Addresses are calculated by the formula 8 * TYPE

INTERRUPT	VECTOR ADDRESS
TRAP (RST 4.5)	24 H
RST 5.5	2C H
RST 6.5	34 H
RST 7.5	3C H

1. For Software interrupts vector addresses are given by:

INTERRUPT	VECTOR ADDRESS
RST 0	00 H
RST 1	08 H
RST 2	10 H
RST 3	18 H
RST 4	20 H
RST 5	28 H
RST 6	30 H
RST 7	38 H

Non-Vectored Interrupts are those in which vector address is not predefined. The interrupting device gives the address of sub-routine for these interrupts. INTR is the only non-vectored interrupt in 8085 microprocessor.

Maskable and Non-Maskable Interrupts – Maskable Interrupts are those which can be disabled or ignored by the microprocessor. These interrupts are either edge-triggered or level-triggered, so they can be disabled. INTR, RST 7.5, RST 6.5, RST 5.5 are maskable interrupts in 8085 microprocessor. Non-Maskable Interrupts are those which cannot be disabled or ignored by microprocessor. TRAP is a non-maskable interrupt. It consists of both level as well as edge triggering and is used in critical power failure conditions.

Priority of Interrupts – When microprocessor receives multiple interrupt requests simultaneously, it will execute the interrupt service request (ISR) according to the priority of the interrupts.

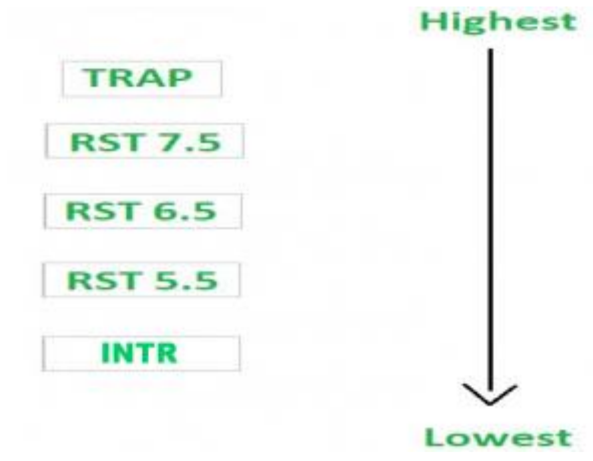


Fig 1. Interrupts

4.1.2 Instruction for Interrupts –

1. **Enable Interrupt (EI) –** The interrupt enable flip-flop is set and all interrupts are enabled following the execution of next instruction followed by EI. No flags are affected. After a system reset, the interrupt enable flip-flop is reset, thus disabling the interrupts. This instruction is necessary to enable the interrupts again (except TRAP).
2. **Disable Interrupt (DI) –** This instruction is used to reset the value of enable flip-flop hence disabling all the interrupts. No flags are affected by this instruction.
3. **Set Interrupt Mask (SIM) –** It is used to implement the hardware interrupts (RST 7.5, RST 6.5, RST 5.5) by setting various bits to form masks or generate output data via the Serial Output Data (SOD) line. First the

required value is loaded in accumulator then SIM will take the bit pattern from it.

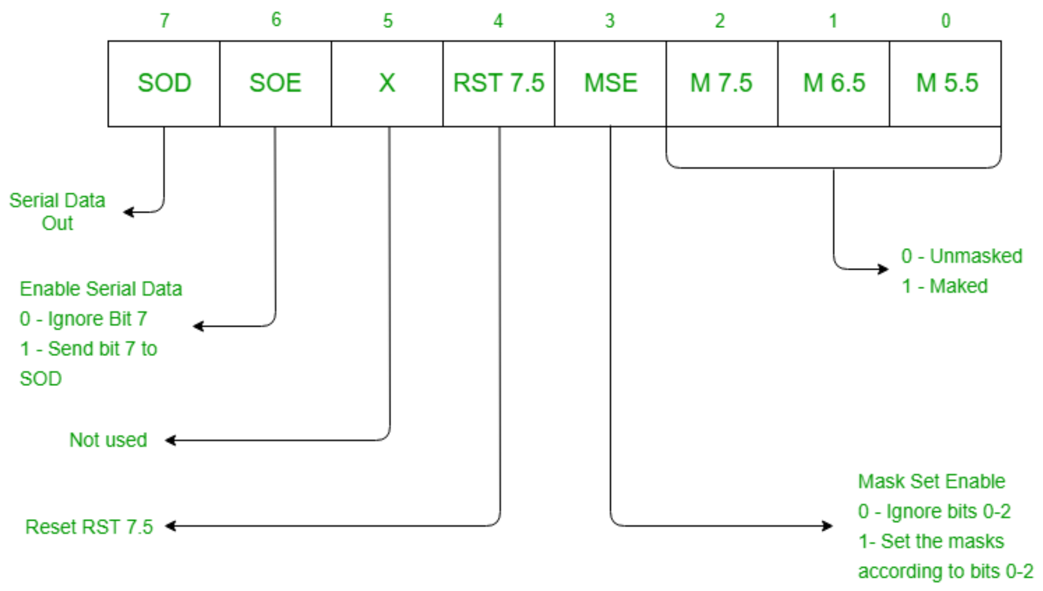


Fig 2.Set Interrupt Mask

4. Read Interrupt Mask (RIM) – This instruction is used to read the status of the hardware interrupts (RST 7.5, RST 6.5, RST 5.5) by loading into the A register a byte which defines the condition of the mask bits for the interrupts. It also reads the condition of SID (Serial Input Data) bit on the microprocessor.

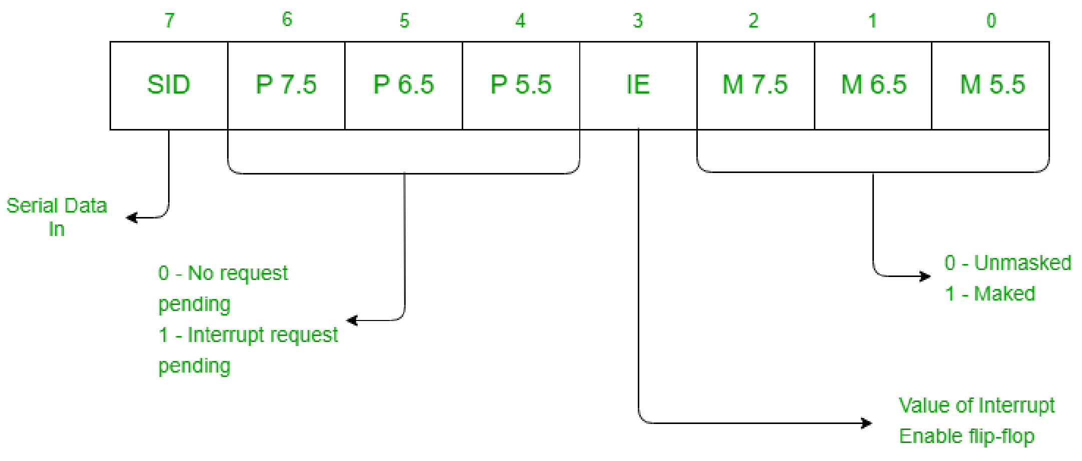


Fig 3. Read Interrupt Mask

4.1.3 Uses of Interrupts in 8085 microprocessor:

Interrupts in the 8085 microprocessor are used for various purposes, including:

Real-time processing: Interrupts allow the microprocessor to respond quickly to external events in real-time, such as user input or hardware signals. This is particularly useful in applications such as control systems and data acquisition systems where time-critical operations are required.

Multi-tasking: Interrupts enable the microprocessor to perform multiple tasks simultaneously by temporarily suspending the current task and executing the ISR for the interrupting event. This allows the microprocessor to switch between different tasks and maximize the utilization of system resources.

Input/output operations: Interrupts can be used for handling input/output operations, such as data transfer between the microprocessor and external devices. This allows the microprocessor to perform other tasks while waiting for input/output operations to complete.

Error handling: Interrupts can be used for error handling and exception handling, such as detecting and recovering from hardware or software errors.

Power management: Interrupts can be used for power management, such as putting the microprocessor into a low-power mode when it is not needed and waking it up when an interrupt occurs.

4.1.4 Issues of Interrupts in 8085 microprocessor:

There are several issues that need to be considered when using interrupts in the 8085 microprocessor:

1. **Priority conflicts:** The 8085 microprocessor supports multiple interrupt signals with different priorities. If multiple interrupts occur simultaneously, it can lead to priority conflicts and result in incorrect operation or system failure. Therefore, the priority levels of each interrupt signal need to be carefully designed and tested to avoid conflicts.
2. **Race conditions:** Race conditions can occur when multiple processes try to access the same resources, such as registers or memory locations,

simultaneously. This can lead to incorrect results or system failure. Therefore, interrupt handlers need to be carefully designed to avoid race conditions, such as by disabling interrupts during critical operations.

3. **Interrupt latency:** Interrupt latency is the time delay between when an interrupt occurs and when the corresponding ISR starts executing. Interrupt latency can affect the system's responsiveness and real-time performance. Therefore, interrupt handlers need to be designed to minimize interrupt latency, such as by using fast interrupt service routines and optimizing the interrupt handling process.
4. **Interrupt nesting:** Interrupt nesting occurs when an interrupt occurs while the microprocessor is executing an ISR for another interrupt. Interrupt nesting can lead to complex interrupt handling and priority conflicts. Therefore, interrupt handlers need to be carefully designed to avoid interrupt nesting, such as by disabling lower-priority interrupts during critical operations.
5. **Interrupt overhead:** Interrupt overhead is the additional processing time and resources required to handle interrupts. Interrupt overhead can affect the system's performance and efficiency, especially if the system experiences a high volume of interrupts. Therefore, interrupt handlers need to be designed to minimize interrupt overhead, such as by optimizing the interrupt handling process and reducing unnecessary operations.

Let us sum up

The 8085 Interrupts

1. **TRAP Interrupt**:
   - **Description**: TRAP is a non-maskable interrupt that is the highest priority among the 8085 interrupts, allowing it to interrupt the microprocessor immediately.
   - **Functionality**: It is typically used for critical tasks such as emergency stop functions or system failures, ensuring that important events are handled promptly.

  - **Vector Address**: The TRAP interrupt is assigned a vector address of 0024H, where the corresponding interrupt service routine (ISR) begins execution.

2. **RST 7.5 Interrupt**:
    - **Description**: RST 7.5 is a maskable interrupt with a medium priority level, allowing it to be enabled or disabled through the interrupt enable (EI) and disable (DI) instructions.
    - **Functionality**: It is used for handling specific events that are not as critical as TRAP but still require attention from the microprocessor.
    - **Vector Address**: The vector address for RST 7.5 is 003CH, where the ISR for this interrupt is executed.
3. **RST 6.5 Interrupt**:
    - **Description**: RST 6.5 is also a maskable interrupt with a lower priority than RST 7.5, allowing the microprocessor to manage multiple interrupt requests effectively.
    - **Functionality**: This interrupt can be used for less critical peripheral devices or specific tasks that can tolerate some delay in processing.
    - **Vector Address**: The vector address for RST 6.5 is 0034H, directing the microprocessor to the appropriate ISR.
4. **RST 5.5 Interrupt**:
    - **Description**: RST 5.5 is another maskable interrupt with the lowest priority among the RST interrupts, providing a mechanism for managing lower-priority tasks.
    - **Functionality**: It is typically used for routine tasks that do not require immediate action, ensuring that higher-priority interrupts are processed first.
    - **Vector Address**: The vector address for RST 5.5 is 002CH, where the ISR for this interrupt begins execution.
5. **INTR Interrupt**:
    - **Description**: INTR (Interrupt Request) is a general-purpose maskable interrupt that allows external devices to interrupt the microprocessor, providing flexibility in handling peripheral events.

- **Functionality**: Unlike RST interrupts, INTR requires an external device to provide the vector address, making it versatile for various applications.
- **Vector Address**: The vector address for INTR is determined by the value placed on the data bus by the interrupting device, allowing it to specify which ISR to execute.

Check your Progress

1. **What is the role of an interrupt in the 8085 microprocessor?**
   a) To execute normal programs faster
   b) To temporarily halt the execution of the current program and redirect control to an interrupt service routine (ISR)
   c) To store data in memory
   d) To reset the microprocessor

   Answer: b) To temporarily halt the execution of the current program and redirect control to an interrupt service routine (ISR)

2. **Which of the following interrupts has the highest priority in the 8085 microprocessor?**
   a) RST 7.5
   b) RST 6.5
   c) INTR
   d) TRAP

   Answer: d) TRAP

3. **Which interrupt in the 8085 microprocessor is non-maskable?**
   a) INTR
   b) RST 7.5
   c) TRAP
   d) RST 5.5

   Answer: c) TRAP

4. **Which interrupt is non-vectored in the 8085 microprocessor?**
   a) TRAP
   b) RST 6.5
   c) RST 7.5
   d) INTR

   Answer: d) INTR

5. **How many hardware interrupts are available in the 8085 microprocessor?**
   a) 3
   b) 5
   c) 8
   d) 10

   Answer: b) 5

6. **Which instruction is used to enable interrupts in the 8085 microprocessor?**
   a) DI
   b) RIM
   c) SIM
   d) EI

   Answer: d) EI

7. **What is the vector address for the RST 5.5 interrupt in the 8085 microprocessor?**
   a) 00H
   b) 2CH
   c) 3CH
   d) 28H

   Answer: b) 2CH

4.2 RIM AND SIM instructions

The SIM (Set Interrupt Mask) and RIM (Read Interrupt Mask) instructions are two instructions in the 8085 microprocessor that are used to control interrupts.

The SIM instruction allows the microprocessor to selectively enable or disable interrupts by setting the appropriate bits in the Interrupt Mask Register (IMR). The IMR is a register that controls which interrupts are allowed to be processed by the microprocessor. The SIM instruction takes one byte as an operand and sets the corresponding bits in the IMR based on the value of the operand.

The RIM instruction, on the other hand, reads the current value of the IMR and copies it to the accumulator. This allows the microprocessor to check which interrupts are currently enabled or disabled. The RIM instruction takes no operands and simply reads the value of the IMR.

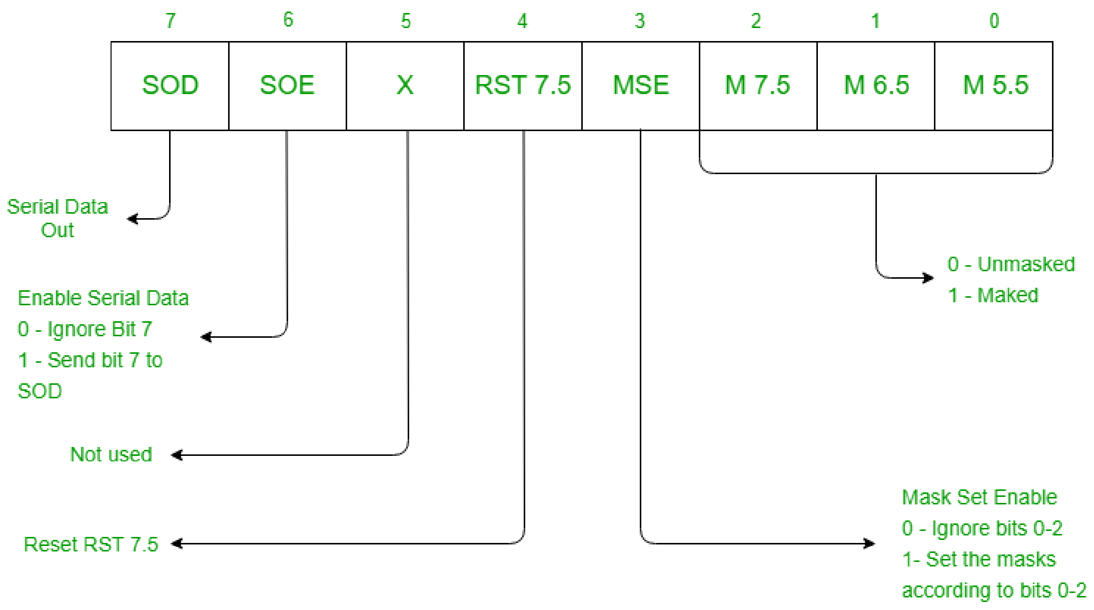


Fig 4. Set Interrupt Mask

These instructions are commonly used in embedded systems and other applications that require interrupt-driven processing. By selectively enabling or disabling interrupts, the microprocessor can control the flow of data and ensure that critical tasks are executed in a timely manner. The SIM and RIM instructions are important tools for managing interrupts and optimizing the performance of the microprocessor in various applications.

Set Interrupt Mask (SIM) : In 8085 Instruction set, SIM stands for “Set Interrupt Mask”. It is 1-Byte instruction and it is a multi-purpose instruction.

The main uses of SIM instruction are –

- Masking/unmasking of RST7.5, RST6.5, and RST5.5
- Reset to 0 RST7.5 flip-flop.
- Perform serial output of data.

Read Interrupt Mask (RIM) : In 8085 Instruction set, RIM stands for “Read Interrupt Mask”. It is a 1-Byte multi-purpose instruction.

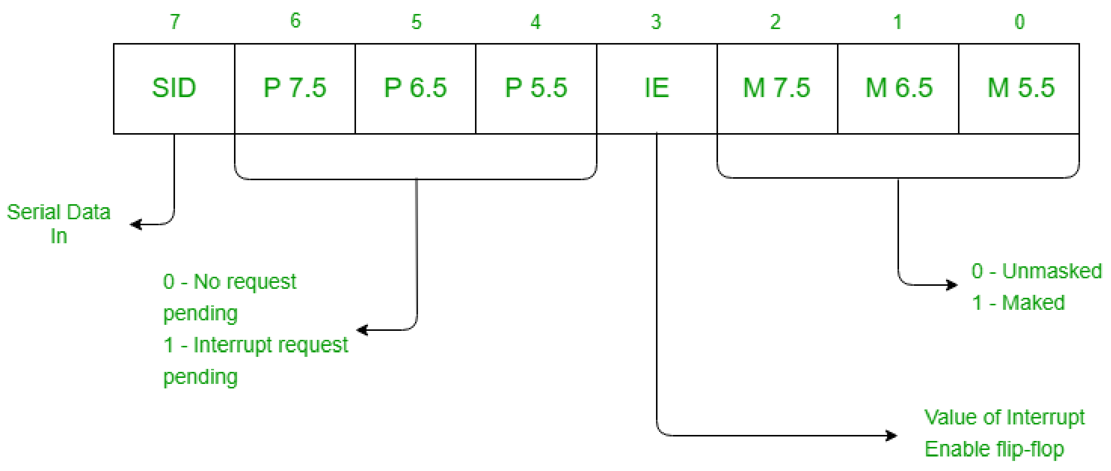


Fig 5. 8085 Read Interrupt Mask

It is used for the following purposes –

- To check whether RST7.5, RST6.5, and RST5.5 are masked or not.
- To check whether interrupts are enabled or not.
- To check whether RST7.5, RST6.5, or RST5.5 interrupts are pending or not.
- To perform serial input of data.

4.2.1 Difference between SIM and RIM instructions in 8085 microprocessor:

Sr. No.	Sim Instruction	Rim Instruction
1	SIM stands for Set Interrupt Mask.	RIM stands for Read Interrupt Mask.
2	It is responsible for masking/unmasking of RST 7.5, RST 6.5 and RST 5.5.	It checks whether RST 7.5, RST 6.5, RST 5.5 are masked or not.
3	It resets to 0 RST 7.5 flip flop.	It checks whether interrupts are enabled or not and to check whether RST 7.5, RST 6.5 or RST 5.5 interrupts are pending or not.
4	The content of the Accumulator decides the action to be taken. So before executing the SIM instruction, it is mandatory to initialize Accumulator with the required value.	The contents of the Accumulator after the execution of the RIM instruction provide this information.Thus, it is essential to look into the Accumulator contents after the RIM instruction is executed.
5	SIM instruction can be used for serial output of data.	RIM instruction can be used for serial input of data.

Sr. No.	Sim Instruction	Rim Instruction
6	Its opcode(in Hex) is 30.	Its opcode(in Hex) is 20.
7	Takes a byte operand	Takes no operand
8	Sets the corresponding bits in the Interrupt Mask Register (IMR) based on the operand	Copies the current value of the IMR to the accumulator
9	Enables or disables interrupts selectively	Reads the current interrupt statu
10	SIM instruction changes the contents of the IMR	RIM instruction does not modify any registers or flags
11	SIM instruction is used for setting the mask register before an interrupt is enabled	RIM instruction is used for reading the mask register during interrupt service routine
12	SIM instruction affects the maskability of the interrupts	RIM instruction does not affect the maskability of the interrupts

Let us sum up

RIM (Read Interrupt Mask) Instruction

1. **Purpose**: The RIM instruction is used to read the status of interrupt signals and the interrupt mask from the 8085 microprocessor, allowing the program to check which interrupts are currently enabled or pending.
2. **Functionality**: Upon execution, RIM places the status of the interrupt system in the accumulator. This includes:
    - The status of the interrupt lines (TRAP, RST 7.5, RST 6.5, RST 5.5, and INTR).
    - The state of the interrupt mask (which interrupts are enabled or disabled).
3. **Register Status**: The flags in the accumulator after executing RIM indicate which interrupts are currently active, enabling the program to make informed decisions based on the interrupt status.

SIM (Set Interrupt Mask) Instruction

1. **Purpose**: The SIM instruction is used to configure the interrupt system by setting or clearing the interrupt mask, thereby enabling or disabling specific interrupts for the 8085 microprocessor.
2. **Functionality**: Upon execution, SIM allows the program to:
    - Specify which maskable interrupts (RST 7.5, RST 6.5, and RST 5.5) are to be enabled or disabled.
    - Control the status of the interrupt system, influencing how the microprocessor responds to incoming interrupt requests.
3. **Mask Control**: The bits in the data byte provided to SIM determine the state of each interrupt line, allowing fine-grained control over the interrupt handling mechanism.

Difference between SIM and RIM Instructions

1. **Functionality**:

- **RIM**: Reads and reports the status of the interrupt system and identifies which interrupts are active.
- **SIM**: Configures the interrupt system by enabling or disabling specific interrupts based on program requirements.

2. **Operation**:
   - **RIM**: Outputs the interrupt status to the accumulator, providing feedback to the program.
   - **SIM**: Takes input from the accumulator or a data byte to control the interrupt mask, modifying the behavior of the interrupt system.
3. **Usage Context**:
   - **RIM**: Typically used when the program needs to check which interrupts are active before making decisions on handling them.
   - **SIM**: Used when setting up or modifying the interrupt configuration to control how the microprocessor interacts with peripheral devices.

Check your Progress

1. **What does the SIM instruction in the 8085 microprocessor stand for?**
   a) Set Interrupt Mask
   b) Serial Interrupt Mask
   c) Signal Interrupt Mask
   d) Select Interrupt Mode

   Answer: a) Set Interrupt Mask

2. **What is the primary purpose of the RIM instruction in the 8085 microprocessor?**
   a) To mask interrupts
   b) To read the current status of interrupts
   c) To reset interrupts
   d) To generate interrupts

   Answer: b) To read the current status of interrupts

3. **Which interrupt lines are controlled by the SIM instruction in the 8085 microprocessor?**
   a) RST 5.5, RST 6.5, and RST 7.5
   b) TRAP, INTR, and RST 7.5
   c) RST 4.5, RST 5.5, and RST 6.5
   d) RST 0, RST 1, and RST 2

   Answer: a) RST 5.5, RST 6.5, and RST 7.5

4. **Before executing the SIM instruction, what must be initialized in the 8085 microprocessor?**
   a) Interrupt Enable Flip-Flop
   b) Accumulator with the required value
   c) Stack Pointer
   d) Program Counter

   Answer: b) Accumulator with the required value

5. **What does the RIM instruction return after execution?**
   a) The status of serial output
   b) The value of the Interrupt Mask Register (IMR)
   c) The opcode of the instruction
   d) The program counter value

   Answer: b) The value of the Interrupt Mask Register (IMR)

4.3 8259 Programmable Interrupt Controller

Intel 8259 is a Programmable Interrupt Controller (PIC). There are 5 hardware interrupts and 2 hardware interrupts in Intel 8085 and Intel 8086 microprocessors respectively. But by connecting Intel 8259 with these microprocessors, we can increase their interrupt handling capability. Intel 8259 combines the multi-interrupt input sources into a single interrupt output. Interfacing of single PIC provides 8 interrupts inputs from IR0-IR7. For example, Interfacing of 8085 and 8259 increases the interrupt handling capability of 8085 microprocessor from 5 to 8 interrupt levels.

Features of Intel 8259 PIC are as follows:

- Intel 8259 is designed for Intel 8085 and Intel 8086 microprocessor.
- It can be programmed either in level triggered or in edge triggered interrupt level.
- We can mask individual bits of interrupt request register.
- We can increase interrupt handling capability upto 64 interrupt level by cascading further 8259 PICs.
- Clock cycle is not required.

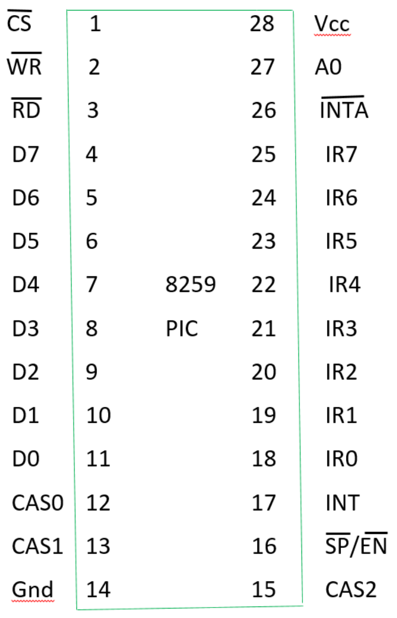


Fig 6. Pin Diagram

Pin Diagram of 8259 – We can see through above diagram that there are total 28 pins in Intel 8259 PIC where Vcc : 5V Power supply and Gnd : ground. Other pins use are explained below. **Block Diagram of 8259 PIC microprocessor –**

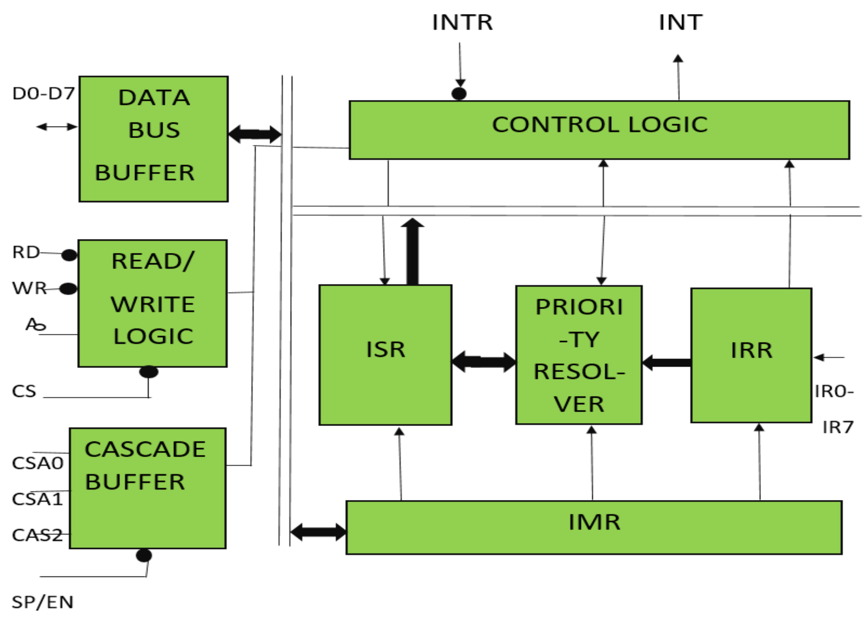


Fig 7. Block Diagram

The Block Diagram consists of 8 blocks which are – Data Bus Buffer, Read/Write Logic, Cascade Buffer Comparator, Control Logic, Priority Resolver and 3 registers-ISR, IRR, IMR.

1. **Data bus buffer –** This Block is used as a mediator between 8259 and 8085/8086 microprocessor by acting as a buffer. It takes the control word from the 8085 (let say) microprocessor and transfer it to the control logic of 8259 microprocessor. After selection of Interrupt by 8259 microprocessor (based on priority of the interrupt), it transfer the opcode of the selected Interrupt and address of the Interrupt service sub routine to the other connected microprocessor. The data bus buffer consists of 8 bits represented as D0-D7 in the block diagram. Thus, shows that a maximum of 8 bits data can be transferred at a time.
2. **Read/Write logic –** This block works only when the value of pin CS is low (as this pin is active low). This block is responsible for the flow of data depending upon the inputs of RD and WR. These two pins are active low pins used for read and write operations.
3. **Control logic –** It is the center of the PIC and controls the functioning of every block. It has pin INTR which is connected with other microprocessor

for taking interrupt request and pin INT for giving the output. If 8259 is enabled, and the other microprocessor Interrupt flag is high then this causes the value of the output INT pin high and in this way 8259 responds to the request made by other microprocessor.

4. **Interrupt request register (IRR) –** It stores all the interrupt level which are requesting for Interrupt services.
5. **Interrupt service register (ISR) –** It stores the interrupt level which are currently being executed.
6. **Interrupt mask register (IMR) –** It stores the interrupt level which have to be masked by storing the masking bits of the interrupt level.
7. **Priority resolver –** It examines all the three registers and set the priority of interrupts and according to the priority of the interrupts, interrupt with highest priority is set in ISR register. Also, it reset the interrupt level which is already been serviced in IRR.
8. **Cascade buffer –** To increase the Interrupt handling capability, we can further cascade more number of pins by using cascade buffer. So, during increment of interrupt capability, CSA lines are used to control multiple interrupt structure.

SP/EN (Slave program/Enable buffer) pin is when set to high, works in master mode else in slave mode. In Non Buffered mode, SP/EN pin is used to specify whether 8259 work as master or slave and in Buffered mode, SP/EN pin is used as an output to enable data bus.

Advantages:

Interrupt Management: The 8259 PIC is designed to handle interrupts efficiently and effectively, allowing for faster and more reliable processing of interrupts in a system.
Flexibility: The 8259 PIC is programmable, meaning that it can be customized to suit the specific needs of a given system, including the number and type of interrupts that need to be managed.

Compatibility: The 8259 PIC is compatible with a wide range of microprocessors, making it a popular choice for managing interrupts in many different systems.
Multiple Interrupt Inputs: The 8259 PIC can manage up to 8 interrupt inputs, allowing for the management of complex systems with multiple devices.
Ease of Use: The 8259 PIC includes simple interface pins and registers, making it relatively easy to use and program.

Disadvantages:

Cost: While the 8259 PIC is relatively affordable, it does add cost to a system, particularly if multiple PICs are required.
Limited Number of Interrupts: The 8259 PIC can manage up to 8 interrupt inputs, which may be insufficient for some applications.
Complex Programming: Although the interface pins and registers of the 8259 PIC are relatively simple, programming the 8259 can be complex, requiring careful attention to interrupt prioritization and other parameters.
Limited Functionality: While the 8259 PIC is a useful peripheral for interrupt management, it does not include more advanced features, such as DMA (direct memory access) or advanced error correction.

Let us sum up

- The **8259 Programmable Interrupt Controller (PIC)** manages multiple interrupt requests from up to eight devices (IRQ0 to IRQ7), allowing efficient handling of interrupts in a microprocessor system.
- It prioritizes interrupts so that urgent requests can preempt less critical ones.
- The PIC is programmable, enabling users to configure which interrupts are enabled and their priority levels.
- It includes key components like IRQ lines for receiving requests, control logic for managing priorities, and a data bus interface for communication with the CPU.

- The 8259 also allows monitoring of pending interrupts, ensuring timely responses to external device signals.

Check your Progress

1. **What is the primary function of the Intel 8259 PIC?**
   a) To provide direct memory access
   b) To combine multiple interrupt sources into a single interrupt output
   c) To manage data transmission between microprocessors
   d) To handle memory management

   Answer: b) To combine multiple interrupt sources into a single interrupt output

2. **How many interrupt inputs can a single 8259 PIC manage?**
   a) 5
   b) 2
   c) 8
   d) 16

   Answer: c) 8

3. **Which of the following is NOT a feature of the Intel 8259 PIC?**
   a) Programmable interrupt levels
   b) Direct Memory Access (DMA)
   c) Can operate in level or edge-triggered modes
   d) Can cascade with other 8259 chips

   Answer: b) Direct Memory Access (DMA)

4. **What is the function of the Interrupt Request Register (IRR) in the 8259 PIC?**
   a) To store the interrupt levels currently being serviced
   b) To store the interrupt levels requesting service

c) To mask interrupt levels

d) To reset the microprocessor

Answer: b) To store the interrupt levels requesting service

5. **What is the purpose of the Interrupt Mask Register (IMR) in the 8259 PIC?**

a) To store interrupt levels that have to be masked

b) To store interrupt levels currently being executed

c) To prioritize interrupt levels

d) To reset all interrupt levels

Answer: a) To store interrupt levels that have to be masked

4.4 8259 Programmable Interrupt Controller

DMA Controller is a hardware device that allows I/O devices to directly access memory with less participation of the processor. DMA controller needs the same old circuits of an interface to communicate with the CPU and Input/Output devices.

What is a DMA Controller?

Direct Memory Access uses hardware for accessing the memory, that hardware is called a DMA Controller. It has the work of transferring the data between Input Output devices and main memory with very less interaction with the processor. The direct Memory Access Controller is a control unit, which has the work of transferring data.

DMA Controller Diagram in Computer Architecture

DMA Controller is a type of control unit that works as an interface for the data bus and the I/O Devices. As mentioned, DMA Controller has the work of transferring the data without the intervention of the processors, processors can control the data transfer. DMA Controller also contains an address unit, which generates the address and selects an I/O device for the transfer of data. Here we are showing the block diagram of the DMA Controller.

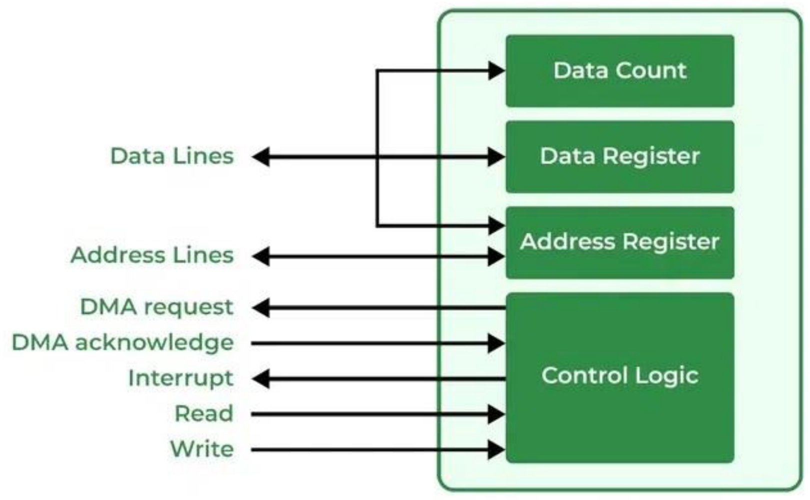


Fig 8. DMA Controller

4.4.1 Types of Direct Memory Access (DMA)

- Single-Ended DMA
- Dual-Ended DMA
- Arbitrated-Ended DMA
- Interleaved DMA

Single-Ended DMA: Single-Ended DMA Controllers operate by reading and writing from a single memory address. They are the simplest DMA.

Dual-Ended DMA: Dual-Ended DMA controllers can read and write from two memory addresses. Dual-ended DMA is more advanced than single-ended DMA.

Arbitrated-Ended DMA: Arbitrated-Ended DMA works by reading and writing to several memory addresses. It is more advanced than Dual-Ended DMA.

Interleaved DMA: Interleaved DMA are those DMA that read from one memory address and write from another memory address.

Working of DMA Controller

The DMA controller registers have three registers as follows.

- **Address register –** It contains the address to specify the desired location in memory.
- **Word count register –** It contains the number of words to be transferred.
- **Control register –** It specifies the transfer mode.

Note: All registers in the DMA appear to the CPU as I/O interface registers. Therefore, the CPU can both read and write into the DMA registers under program control via the data bus.

The figure below shows the block diagram of the DMA controller. The unit communicates with the CPU through the data bus and control lines. Through the use of the address bus and allowing the DMA and RS register to select inputs, the register within the DMA is chosen by the CPU. RD and WR are two-way inputs. When BG (bus grant) input is 0, the CPU can communicate with DMA registers. When BG (bus grant) input is 1, the CPU has relinquished the buses and DMA can communicate directly with the memory.

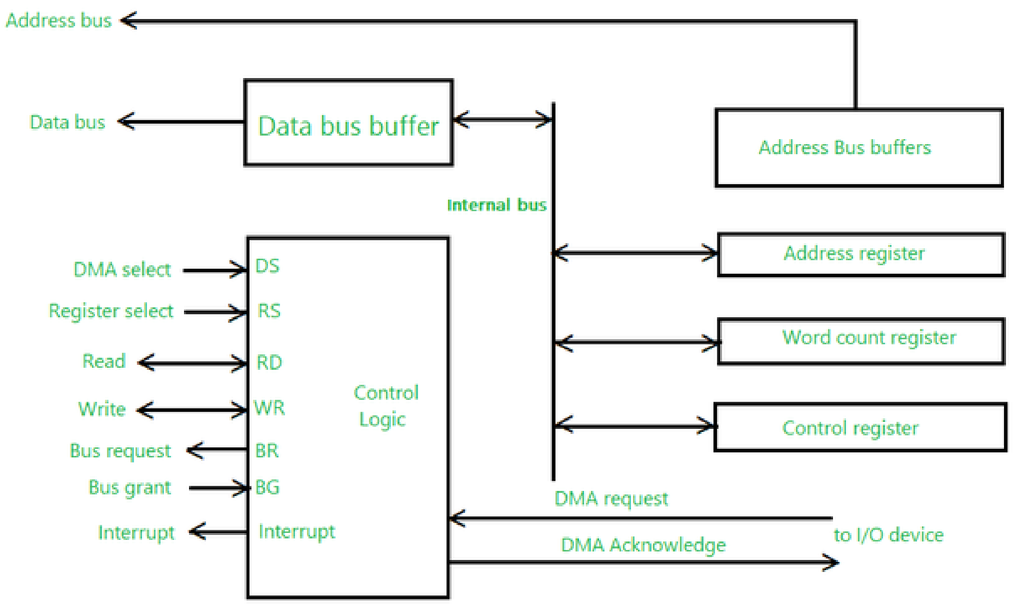


Fig 9. DMA Register

Explanation: The CPU initializes the DMA by sending the given information through the data bus.

- The starting address of the memory block where the data is available (to read) or where data are to be stored (to write).
- It also sends word count which is the number of words in the memory block to be read or written.
- Control to define the mode of transfer such as read or write.

- A control to begin the DMA transfer

Modes of Data Transfer in DMA

There are 3 modes of data transfer in DMA that are described below.

- **Burst Mode:** In Burst Mode, buses are handed over to the CPU by the DMA if the whole data is completely transferred, not before that.
- **Cycle Stealing Mode:** In Cycle Stealing Mode, buses are handed over to the CPU by the DMA after the transfer of each byte. Continuous request for bus control is generated by this Data Transfer Mode. It works more easily for higher-priority tasks.
- **Transparent Mode:** Transparent Mode in DMA does not require any bus in the transfer of the data as it works when the CPU is executing the transaction.

8237 DMA Controller

8237 DMA Controller is a type of DMA Controller which has a flexible number of channels but generally works on 4 Input-Output channels. In these present channels, the channel has to be given the highest priority to be decided by the Priority Encoder. Each channel in the 8237 DMA Controller has to be programmed separately.

8257 DMA Controller

8257 DMA Controller is a type of DMA Controller, that when a single Intel 8212 I/O device is paired with it, becomes 4 channel DMA Controller. In 8257 DMA Controller, the highest priority channel is acknowledged. It contains two 16-bit registers, one is DMA Address Register and the other one is Terminal Count Register.

Advantages of DMA Controller

- Data Memory Access speeds up memory operations and data transfer.
- CPU is not involved while transferring data.
- DMA requires very few clock cycles while transferring data.
- DMA distributes workload very appropriately.
- DMA helps the CPU in decreasing its load.

Disadvantages of DMA Controller

- Direct Memory Access is a costly operation because of additional operations.
- DMA suffers from Cache-Coherence Problems.
- DMA Controller increases the overall cost of the system.
- DMA Controller increases the complexity of the software.

Let us sum up

Direct Memory Access (DMA)

1. **Definition**:
   DMA is a method that allows certain hardware components to access system memory independently of the CPU, enabling high-speed data transfer.
2. **Functionality**:
   DMA transfers data between I/O devices and memory without CPU intervention, freeing the CPU to perform other tasks.

8257 DMA Controller

1. **Purpose**:
   The 8257 is a dedicated DMA controller that manages DMA operations, coordinating data transfers between peripherals and memory.
2. **Channels**:
   The 8257 can support up to **four DMA channels**, allowing multiple devices to perform DMA transfers simultaneously.

Types of DMA Controllers

1. **Single-Channel DMA**:
   A single DMA controller that handles one device at a time, leading to simpler designs but limited efficiency.

2. **Multi-Channel DMA**:
   Controllers like the 8257 that support multiple channels, enabling concurrent data transfers and improving overall system performance.

Advantages of DMA Controller

1. **Increased Throughput**:
   By allowing devices to transfer data without CPU intervention, DMA increases the overall data transfer rates and system efficiency.
2. **Reduced CPU Overhead**:
   The CPU is freed from managing data transfers, allowing it to perform other computations and improving multitasking capabilities.

Disadvantages of DMA Controller

1. **Complexity**:
   Implementing DMA adds complexity to the system design, requiring additional hardware and control logic.
2. **Potential for Conflicts**:
   Since DMA allows devices direct access to memory, it may lead to conflicts or data corruption if not properly managed, especially in systems with multiple active devices.

Check your Progress

1. **What is the primary function of the Intel 8259 PIC?**
   a) To provide direct memory access
   b) To combine multiple interrupt sources into a single interrupt output
   c) To manage data transmission between microprocessors
   d) To handle memory management

   Answer: b) To combine multiple interrupt sources into a single interrupt output

2. **How many interrupt inputs can a single 8259 PIC manage?**
   a) 5
   b) 2
   c) 8
   d) 16

   Answer: c) 8

3. **Which of the following is NOT a feature of the Intel 8259 PIC?**
   a) Programmable interrupt levels
   b) Direct Memory Access (DMA)
   c) Can operate in level or edge-triggered modes
   d) Can cascade with other 8259 chips

   Answer: b) Direct Memory Access (DMA)

4. **What is the function of the Interrupt Request Register (IRR) in the 8259 PIC?**
   a) To store the interrupt levels currently being serviced
   b) To store the interrupt levels requesting service
   c) To mask interrupt levels
   d) To reset the microprocessor

   Answer: b) To store the interrupt levels requesting service

5. **What is the purpose of the Interrupt Mask Register (IMR) in the 8259 PIC?**
   a) To store interrupt levels that have to be masked
   b) To store interrupt levels currently being executed
   c) To prioritize interrupt levels
   d) To reset all interrupt levels

   Answer: a) To store interrupt levels that have to be masked

Conclusion

The 8085 microprocessor's interrupt handling capabilities are vital for efficient system performance, enabling real-time processing, multitasking, and effective I/O management. With five primary interrupt signals (TRAP, RST 7.5, RST 6.5, RST 5.5, and INTR), it showcases a versatile approach to handling external events. Instructions like EI, DI, SIM, and RIM provide granular control over interrupts, essential for optimizing various applications.

Furthermore, the 8259 Programmable Interrupt Controller (PIC) significantly enhances the 8085's interrupt handling capacity, while the DMA controller streamlines data transfer between memory and I/O devices, reducing CPU load and improving overall system efficiency. Despite their complexity and additional cost, these features are instrumental in enhancing the microprocessor's functionality and performance.

Let Us Sum Up

In this unit module, we covered:

- The interrupt system of the 8085 microprocessor, including the five primary interrupt signals, their priority levels, and classification.
- We also discussed instructions for managing interrupts, such as EI, DI, SIM, and RIM.
- Additionally, we examined the functionality and benefits of the 8259 Programmable Interrupt Controller (PIC) and the Direct Memory Access (DMA) controller, highlighting their roles in enhancing system performance.

Activities

Activity 1: Identify and Classify Interrupts: Create a table to classify the interrupts of the 8085 microprocessor based on priority, type (maskable/non-maskable), and whether they are vectored or non-vectored.

Activity 2: Simulate Interrupt Handling: Write a program to simulate how the 8085 microprocessor handles multiple simultaneous interrupt requests, including priority resolution.

Activity 2: Implement DMA: Develop a simple code to illustrate the working of DMA in transferring data between memory and an I/O device.

Check Your Progress

1. What are the five interrupt signals in the 8085 microprocessor? Describe their priority levels.

----

2. Explain the difference between vectored and non-vectored interrupts with examples.

----

3. Describe the role of the 8259 PIC and how it expands the interrupt handling capability of the 8085 microprocessor.

----

4. What are the advantages and disadvantages of using a DMA controller in a microprocessor system?

Summary:

The 8085 microprocessor's interrupt handling capabilities are a crucial aspect of its architecture, enabling efficient real-time processing, multitasking, and effective management of input/output operations. With five primary interrupt signals (TRAP, RST 7.5, RST 6.5, RST 5.5, and INTR) and the ability to classify interrupts into various categories such as hardware/software, vectored/non-vectored, and maskable/non-maskable, the 8085 demonstrates a versatile approach to managing external events and ensuring system responsiveness.

The instructions for enabling and managing interrupts, such as EI, DI, SIM, and RIM, provide granular control over the interrupt system, allowing for precise configuration and optimization based on application needs. The distinctions between SIM and RIM further highlight their roles in setting and reading the interrupt mask register, essential for tailored interrupt handling.

To expand the interrupt handling capabilities, the integration of the 8259 Programmable Interrupt Controller (PIC) enhances the 8085's capacity from 5 to 8 interrupt levels, and potentially up to 64 through cascading. The 8259 PIC manages multiple interrupt sources efficiently, ensuring prioritized and orderly processing of interrupt requests. While it brings advantages such as increased flexibility and compatibility, it also introduces complexity in programming and additional system cost.

Lastly, the Direct Memory Access (DMA) controller presents a sophisticated mechanism for data transfer between memory and I/O devices with minimal CPU intervention. This enhances system performance by offloading the CPU and enabling faster data throughput. Despite its benefits, DMA comes with challenges, including higher system costs and complexity.

Overall, the interrupt and DMA features of the 8085 microprocessor and its auxiliary controllers like the 8259 PIC and 8237/8257 DMA controllers, significantly enhance system performance, enabling robust and efficient handling of various tasks and external events in a streamlined manner.

Self-Assessment Questions

1. What is an interrupt in the 8085 microprocessor, and why is it important?

2. Describe the difference between maskable and non-maskable interrupts in the 8085 microprocessor.

3. Explain the role of the TRAP interrupt in the 8085 microprocessor. Why is it considered non-maskable?

4. What are the differences between hardware and software interrupts in the 8085 microprocessor?

5.How does the 8085 microprocessor handle multiple simultaneous interrupt requests?

6Explain the functionality of the RIM and SIM instructions in the 8085 microprocessor.

7.What is the purpose of the Interrupt Mask Register (IMR) in the 8259 Programmable Interrupt Controller?

8.Describe how the 8259 PIC increases the interrupt handling capacity of the 8085 microprocessor.

9. What are the main advantages and disadvantages of using the 8259 PIC in microprocessor systems?

10.How does the Priority Resolver in the 8259 PIC handle interrupt requests, and why is it critical to interrupt management?

11.Describe how the Direct Memory Access (DMA) controller minimizes CPU involvement in data transfers.

12.What are the three types of DMA modes, and how do they differ in terms of data transfer efficiency?

13.Explain the role of the 8237/8257 DMA controllers in managing data transfers.

14.What are the key advantages and disadvantages of using DMA in a microprocessor system?

Further Reading and References

Books

1. "Microprocessor and Interfacing" by A. P. Godse & D. A. Godse

- Covers detailed explanations of interrupts, 8259 PIC, and DMA controller usage in 8085 microprocessors.
- ISBN: 978-8184314625

2. "8085 Microprocessor: Programming and Interfacing" by N. K. Srinath

- Provides comprehensive coverage of the 8085 microprocessor's interrupt system and interfacing with external controllers.
- ISBN: 978-8120327857

3. "Microprocessor Architecture, Programming, and Applications with the 8085" by Ramesh Gaonkar

- A classic textbook that explores the complete architecture of the 8085, including interrupts and peripherals like the 8259 PIC.
- ISBN: 978-8131509646

4. "Advanced Microprocessors and Peripherals" by K. M. Bhurchandi & A. K. Ray

- Includes in-depth analysis of the 8259 PIC, DMA controllers, and their applications in real-world scenarios.
- ISBN: 978-0070140690

Online Resources

1. TutorialsPoint - 8085 Microprocessor Interrupts

- Detailed tutorials on the interrupt system of the 8085 microprocessor, including RIM and SIM instructions and the 8259 PIC.
- [TutorialsPoint Microprocessor](https://www.tutorialspoint.com/microprocessor/microprocessor_8085_addressing_modes_and_interrupts.htm)

2. GeeksforGeeks - 8259 Programmable Interrupt Controller

- Offers clear explanations on how the 8259 PIC works with the 8085 and 8086 microprocessors to enhance interrupt management.
- [GeeksforGeeks 8259 PIC](https://www.geeksforgeeks.org/8259-programmable-interrupt-controller/)

3. Electronics Hub - DMA Controllers in Microprocessors

- Provides easy-to-understand articles on Direct Memory Access (DMA) and its significance in microprocessor systems.
- [Electronics Hub DMA](https://www.electronicshub.org/dma-controller/)

4. Khan Academy - Microprocessor Architecture

- Interactive lessons on microprocessor architecture, including interrupt systems, RIM and SIM instructions, and the 8259 PIC.
- [Khan Academy Microprocessor](https://www.khanacademy.org/)

Video Resources

1. YouTube - 8085 Microprocessor Playlist by Neso Academy

- A detailed video series explaining interrupts, RIM and SIM instructions, and the 8259 PIC.
- [Neso Academy YouTube Playlist](https://www.youtube.com/playlist?list=PLBlnK6fEyqRjoG6aJ4FvFU1tlX5W3GO9r)

2. MIT OpenCourseWare - Digital Systems and Microcontrollers

- Video lectures on microprocessors, with a focus on interrupt handling and peripherals like the 8259 PIC and DMA.
- [MIT OpenCourseWare Microprocessors](https://ocw.mit.edu/)

3. Udemy - Microprocessor and Microcontroller Courses

- A paid course offering in-depth coverage of 8085 microprocessor programming and interfacing with external devices like the 8259 PIC.
- [Udemy Microprocessor Course](https://www.udemy.com/course/microprocessor/)

UNIT V

Introduction to Microcontroller - Microcontroller Vs Microprocessor - 8051 Microcontroller architecture - 8051 pin description. Timers and Counters – Operating Modes- Control Registers. Interrupts – Interrupts in 8051 - Interrupts Control Register – Execution of interrupt.

Unit Objectives:

In this unit students will have a comprehensive understanding of microcontrollers, with a focus on the 8051 microcontrollers. Learners will be able to analyze different types of microcontrollers, interrupts, and control register to understand the working of simple embedded system. By the end of this unit, students will be able to

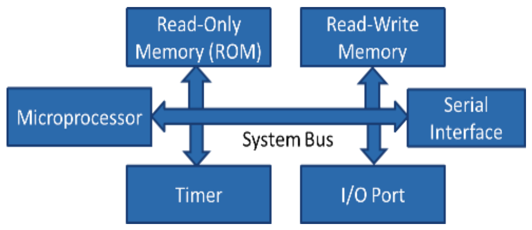
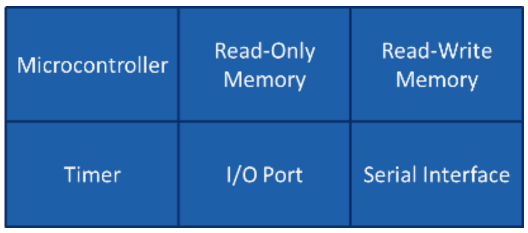
- Define and Understand Microcontrollers and Microprocessors:
  - Comprehend the differences between microcontrollers and microprocessors, their roles in electronic systems, and the contexts in which each is used.
- Describe and Implement 8051 Microcontroller Architecture:
  - Understand the architecture of the 8051 microcontrollers, including its CPU, memory, I/O ports, timers, counters, and serial communication features
- Understand 8051 Pin Configuration:
  - Know the functions of each pin and their connections
- Implement Timers and Counters:
  - Configure and use timers/counters for timing and event counting.
- Handle Interrupts in 8051:
  - Configure and manage interrupts for responsive system handling.

5.1 INTRODUCTION TO MICROCONTROLLER

This module introduces learners to microcontrollers, with a focus on the 8051 microcontrollers. It aims to provide a foundational understanding of microcontroller technology, covering both theoretical concepts and practical applications. By the module's end, learners will comprehend the 8051's architecture, including its CPU, memory, I/O ports, timers, and interrupts. They'll also grasp essential skills like pin configuration and the implementation of timers, counters, and interrupts. With this knowledge, learners will be equipped to design and implement systems utilizing the 8051 microcontrollers for a wide range of embedded applications.

5.1.1 Microcontroller Vs Microprocessor:

Difference between Microprocessor and microcontroller

Microprocessor	Micro Controller
 Read-Only Memory (ROM) Read-Write Memory Microprocessor Serial Interface System Bus Timer I/O Port	 Microcontroller Read-Only Memory Read-Write Memory Timer I/O Port Serial Interface
Microprocessor is heart of Computer system.	Micro Controller is a heart of embedded system.
It is just a processor. Memory and I/Ocomponents have to be connected externally	Micro controller has processor along withinternal memory and i/O components
Since memory and I/O has to be connectedexternally, the circuit becomes large.	Since memory and I/O are present internally, thecircuit is small.
Cannot be used in compact systems and henceinefficient	Can be used in compact systems and hence it isan efficient technique
Cost of the entire system increases	Cost of the entire system is low
Due to external components, the entire power consumption is high. Hence it is not suitable to used with devices running on stored power like batteries.	Since external components are low, total power consumption is less and can be used with devices running on stored power like batteries.
Most of the microprocessors do not have powersaving features.	Micro controllers have power saving modes like idle mode and power saving mode. This helps to reduce power consumption evenfurther.

Since memory and I/O components are all external, each instruction will need external operation, hence it is relatively slower.	Since components are internal, most of the operations are internal instruction, hence speed is fast.
Microprocessors are based on von Neumann model/architecture where program and data are stored in same memory module	Micro controllers are based on Harvard architecture where program memory and Data memory are separate

Let us sum up:

- **Microprocessor**:
  - Requires external memory and I/O components. Based on the von Neumann architecture, where program and data share the same memory space, resulting in relatively slower performance.
  - Due to external memory and I/O components, it occupies more space and is less efficient for compact systems.
- **Microcontroller:**
  - Follows the Harvard architecture, where program and data memory are separate, allowing faster execution.
  - Integrated components reduce cost and power consumption, and it often includes power-saving modes (e.g., idle and sleep modes).

Check your Progress:

1. **Which of the following is a major advantage of a microcontroller over a microprocessor?**

   A) Higher processing speed

   B) Power saving modes

   C) Larger memory capacity

   D) External memory requirement

Answer: B) Power saving modes

2. **What architecture do most microcontrollers use?**

   A) von Neumann architecture

   B) Harvard architecture

   C) Dual-core architecture

   D) Modular architecture

   Answer: B) Harvard architecture

3. **Why is a microcontroller considered more efficient for compact systems?**

   A) Because it has higher clock speeds

   B) Due to internal memory and I/O components

   C) It can connect to multiple external devices

   D) It consumes more power

   Answer: B) Due to internal memory and I/O components

4. **What is a typical feature of most microprocessors?**

   A) Power saving mode

   B) External memory and I/O components

   C) Integrated system design

   D) Idle mode

   Answer: B) External memory and I/O components

5. **Which architecture stores program and data in the same memory module?**

   A) Harvard architecture

   B) ARM architecture

   C) von Neumann architecture

   D) RISC architecture

   Answer: C) von Neumann architecture

5.2 8051 MICROCONTROLLER ARCHITECTURE

5.2.1 Definition:

8051 microcontroller is designed by Intel in 1981. It is an 8-bit microcontroller. It is built with 40 pins DIP (dual inline package), 4kb of ROM storage and 128 bytes of RAM storage, 2 16-bit timers. It consists of are four parallel 8-bit ports, which are programmable as well as addressable as per the requirement. An on-chip crystal oscillator is integrated in the microcontroller having crystal frequency of 12 MHz.

In the following diagram, the system bus connects all the support devices to the CPU. The system bus consists of an 8-bit data bus, a 16-bit address bus and bus control signals.

8051 Microcontroller Architecture

The 8051 Microcontroller Architecture consists of the following major components:

- Interrupt control: It supports five interrupt sources, which can interrupt the normal program execution to handle specific events.
- Central Processing Unit (CPU): This is the brain of the microcontroller, which is responsible for executing instructions and performing arithmetic and logical operations.
- Oscillator: It requires an external oscillator to provide a clock signal to the microcontroller
- Bus control: The 8051 microcontroller includes a bus controller that manages data transfer between the CPU and peripheral devices, such as memory or input/output devices.
- 4k byte ROM: The 8051-microcontroller architecture includes a 4 kilobyte (4k) read-only memory (ROM) for storing the program instructions that are executed by the CPU.
- 128-byte RAM**:** The 8051 microcontrollers also has a 128-byte random-access memory (RAM) for storing data that is used by the program instructions during runtime.

- Input/Output Ports**:** They have four 8-bit input/output (I/O) ports that can be configured as either input or output.
- Timers and Counters: They have two 16-bit timers/counters that can be used for a variety of tasks such as measuring time intervals, generating PWM signals, and counting external events.
- Serial Communication Interface (SCI): It has a built-in serial port that can be used for asynchronous serial communication.

In this section, the 8051-microcontroller architecture block diagram is depicted as:

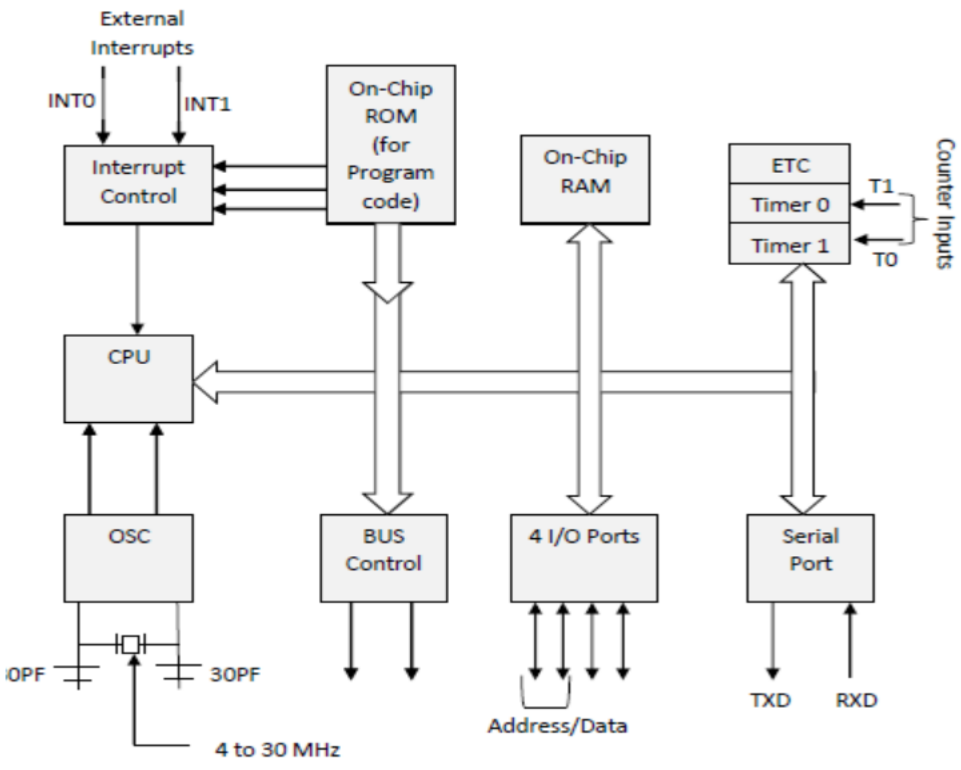


Fig 1. 8051 Microcontroller Architecture

5.2.2 Applications

The microcontroller 8051 applications include a large number of machines, principally because it is simple to incorporate in a project or to assemble a machine around it. The following are the key spots of the spotlight:

1. **Energy Management:** Competent measuring device systems aid in calculating energy consumption in domestic and industrialized applications. These meter systems are prepared competent by integrating microcontrollers.
2. **Touch screens:** A high degree of microcontroller suppliers integrate touch sensing abilities in their designs. Transportable devices such as media players, gaming devices & cell phones are some illustrations of micro-controller integrated with touch sensing screens.
3. **Automobiles:** The microcontroller 8051 discovers broad recognition in supplying automobile solutions. They are extensively utilized in hybrid motor vehicles to control engine variations
4. **Medical Devices:** Handy medicinal gadgets such as glucose & blood pressure monitors bring into play micro-controllers, to put on view the measurements, as a results.

5.2.3 Advantages:

- It is a cheaper option, which makes it a popular choice for low-cost applications.
- Designed to consume very low power for applications.
- 8051 Microcontroller Architecture is known for its high reliability.

5.2.4 Disadvantages:

- It has limited processing power compared to microcontrollers
- Memory is limited for applications it requires a lot of data storage.
- It highly requires advanced peripherals such as USB or Ethernet connectivity

Let us sum up:

➢ **Definition**:

- The 8051 microcontroller, designed by Intel in 1981, is an 8-bit microcontroller with 40 pins, 4KB ROM, 128 bytes of RAM, and two 16-bit timers.
- It includes four 8-bit programmable I/O ports and a built-in 12 MHz crystal oscillator

- **Applications:**
    - The 8051 microcontroller is used in energy management systems to calculate consumption, integrated into touch screens for devices like phones and gaming consoles, controls hybrid vehicles in the automotive industry, and powers medical devices like glucose and blood pressure monitors.

Check your Progress:

1. **What is the size of the 8051 microcontroller's ROM?**

    A) 2KB

    B) 4KB

    C) 8KB

    D) 128KB

    Answer: B) 4KB

2. **Which of the following is a key characteristic of a List ADT?**

    A) 16-bit timers

    B) Serial Communication Interface
    C) On-chip oscillator

    D) 64KB RAM

    Answer: D) 64KB RAM

3. **What kind of architecture does the 8051 microcontroller follow?**

    A) Harvard

    B) von Neumann

    C) ARM

    D) MIPS

    Answer: A) Harvard

4. **Where are 8051 microcontrollers commonly used?**

    A) Spacecraft

B) Automobiles

C) Nuclear Plants

D) Oil Refineries

Answer: B) Automobiles

5. **Which is an advantage of the 8051 microcontroller?**

A) High processing power

B) Low power consumption

C) Large memory space

D) USB connectivity

Answer: B) Low power consumption

5.3 8051 PIN DESCRIPTION

5.3.1 8051 Input Output Ports:

8051 microcontrollers have 4 I/O ports each of 8-bit, which can be configured as input or output. Hence, total 32 input/output pins allow the microcontroller to be connected with the peripheral devices.

- Pin configuration, i.e. the pin can be configured as 1 for input and 0 for output as per the logic state.
  - Input/Output (I/O) pin − All the circuits within the microcontroller must be connected to one of its pins except P0 port because it does not have pull-up resistors built-in.
  - Input pin − Logic 1 is applied to a bit of the P register. The output FE transistor is turned off and the other pin remains connected to the power supply voltage over a pull-up resistor of high resistance.
- Port 0 − The P0 (zero) port is characterized by two functions −
  - When the external memory is used then the lower address byte (addresses A0A7) is applied on it, else all bits of this port are configured as input/output.

- When P0 port is configured as an output then other ports consisting of pins with built-in pull-up resistor connected by its end to 5V power supply, the pins of this port have this resistor left out.

5.3.2 Input and Output Configuration

If any pin of this port is configured as an input, then it acts as if it "floats", i.e. the input has unlimited input resistance and in-determined potential.

When the pin is configured as an output, then it acts as an "open drain". By applying logic 0 to a port bit, the appropriate pin will be connected to ground (0V), and applying logic 1, the external output will keep on "floating". In order to apply logic 1 (5V) on this output pin, it is necessary to build an external pullup resistor.

Port 1

P1 is a true I/O port as it doesn't have any alternative functions as in P0, but this port can be configured as general I/O only. It has a built-in pull-up resistor and is completely compatible with TTL circuits.

Port 2

P2 is similar to P0 when the external memory is used. Pins of this port occupy addresses intended for the external memory chip. This port can be used for higher address byte with addresses A8-A15. When no memory is added then this port can be used as a general input/output port similar to Port 1.

Port 3

In this port, functions are similar to other ports except that the logic 1 must be applied to appropriate bit of the P3 register.

Pins Current Limitations

When pins are configured as an output (i.e. logic 0), then the single port pins can receive a current of 10mA.

When these pins are configured as inputs (i.e. logic 1), then built-in pull-up resistors provide very weak current, but can activate up to 4 TTL inputs of LS series.

If all 8 bits of a port are active, then the total current must be limited to 15mA (port P0: 26mA).

If all ports (32 bits) are active, then the total maximum current must be limited to 71mA.

Pins 1 to 8 − These pins are known as Port 1. This port doesn’t serve any other functions. It is internally pulled up, bi-directional I/O port.

Pin 9 − It is a RESET pin, which is used to reset the microcontroller to its initial values.

Pins 10 to 17 − These pins are known as Port 3. This port serves some functions like interrupts, timer input, control signals, serial communication signals RxD and TxD, etc.

Pins 18 & 19 − These pins are used for interfacing an external crystal to get the system clock.

Pin 20 − This pin provides the power supply to the circuit.

Pins 21 to 28 − These pins are known as Port 2. It serves as I/O port. Higher order address bus signals are also multiplexed using this port.

Pin 29 − This is PSEN pin which stands for Program Store Enable. It is used to read a signal from the external program memory.

Pin 30 − This is EA pin which stands for External Access input. It is used to enable/disable the external memory interfacing.

Pin 31 − This is ALE pin which stands for Address Latch Enable. It is used to demultiplex the address-data signal of port.

Pins 32 to 39 − These pins are known as Port 0. It serves as I/O port. Lower order address and data bus signals are multiplexed using this port.

Pin 40 − This pin is used to provide power supply to the circuit.

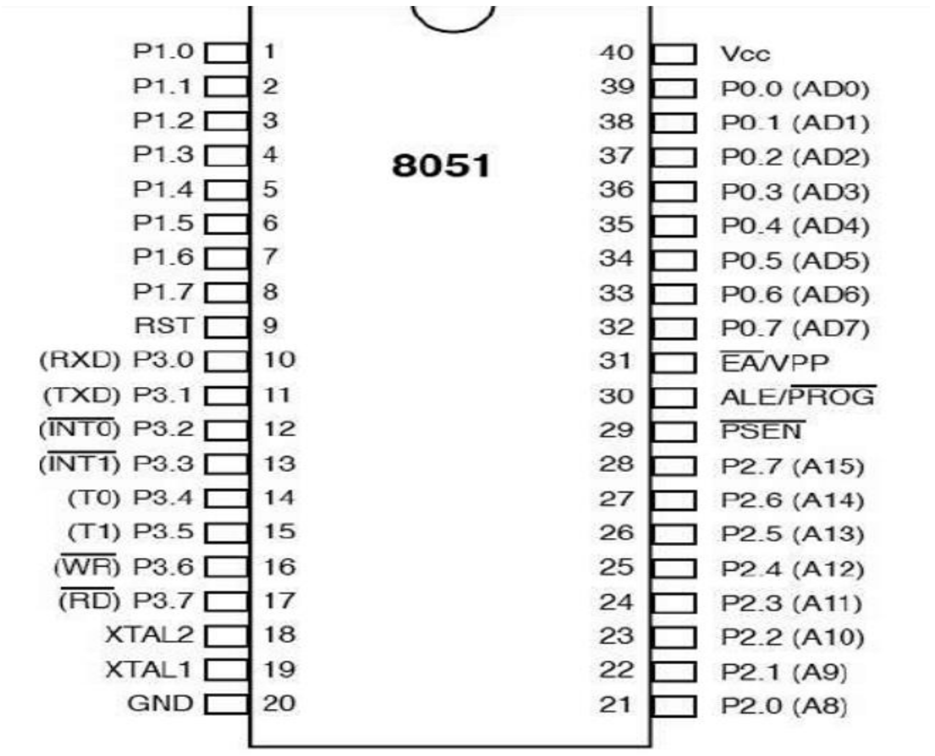


Fig 2. 8051 Pin Configuration

Many of the microcontroller applications require counting of external events such as frequency of the pulse trains and generation of precise internal time delays between computer actions. Both these tasks can be implemented by software techniques, but software loops for counting, and timing will not give the exact result rather more important functions are not done. To avoid these problems, timers and counters in the micro-controllers are better options for simple and low-cost applications. These timers and counters are used as interrupts in 8051 microcontroller.

Let us sum up:

- **Pin Current Limitations**:
  - Output pins can drive a current of up to 10mA per pin.
  - If all pins in a port are active, the total current for the port must not exceed 15mA (26mA for Port 0).
  - The total maximum current across all ports must be limited to 71mA

➢ **I/O Ports**:

- The 8051 microcontroller has 4 input/output (I/O) ports, each 8-bit wide, offering a total of 32 I/O pins.
- These can be configured as input or output as needed. Port 0 can be used for I/O and lower address byte transmission when external memory is used

Check your Progress:

1. **How many I/O ports does the 8051 microcontroller have?**

   A) 2

   B) 3

   C) 4

   D) 5

   Answer: C) 4

2. **Which port in the 8051 microcontroller is used for both I/O and carrying the lower address byte when external memory is used?**

   A) Port 1

   B) Port 2

   C) Port 0

   D) Port 3

   Answer: C) Port 0

3. **What is the function of Pin 9 in the 8051 microcontroller?**

   A) Reset the microcontroller

   B) Provide system clock

   C) Read external memory

   D) Enable I/O ports

   Answer: A) Reset the microcontroller

4. **Which pin is used for interfacing with an external crystal to get the system clock?**

   A) Pin 20

   B) Pins 18 & 19

   C) Pin 30

   D) Pin 29

   Answer: B) Pins 18 & 19

5. **What does the PSEN pin stand for in the 8051 microcontroller?**

   A) Program Store Enable

   B) Program Save Enable

   C) Power Supply Enable

   D) Port System Enable

   Answer: A) Program Store Enable

6. **What is the current limit for a single port pin configured as output in the 8051 microcontroller?**

   A) 5mA

   B) 10mA

   C) 15mA

   D) 20mA

   Answer: B) 10mA

5.4 TIMERS AND COUNTERS

There are two 16-bit timers and counters in 8051 microcontroller: timer 0 and timer 1. Both timers consist of 16-bit register in which the lower byte is stored in TL and the higher byte is stored in TH. Timer can be used as a counter as well as for timing operation that depends on the source of clock pulses to counters.

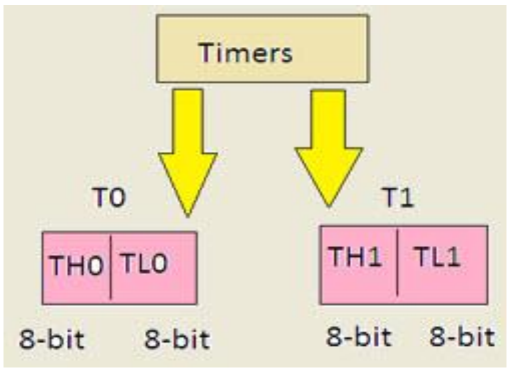


Fig 3. Timers and counters

Counters and Timers in 8051 microcontroller contain two special function registers: TMOD (Timer Mode Register) and TCON (Timer Control Register), which are used for activating and configuring timers and counters. There are four different modes of the Timer or Counter. The Mode 0 to Mode 2 are for both of the Timer/Counter. Mode 3 has a different meaning for each timer register. There is a register called TMOD. This register can be programmed to configure these timers or counters. The Serial port is used for serial communication in mode 1 and 3. Timer1 is used for generating the baud rate. So only Timer0 is available for timer or counter operations.

5.4.1 Operating Modes:

TMOD Register

TMOD (Timer Mode) is an SFR. The address of this register is 89H. This is not bit-addressable.

Timer	**Timer1 Mode**				**Timer0 operating Mode**			
Bit Details	Gate (G)	C/T	M1	M0	Gate (G)	C/T	M1	M0

Now, let us see the circuit that controls the running of the timers.

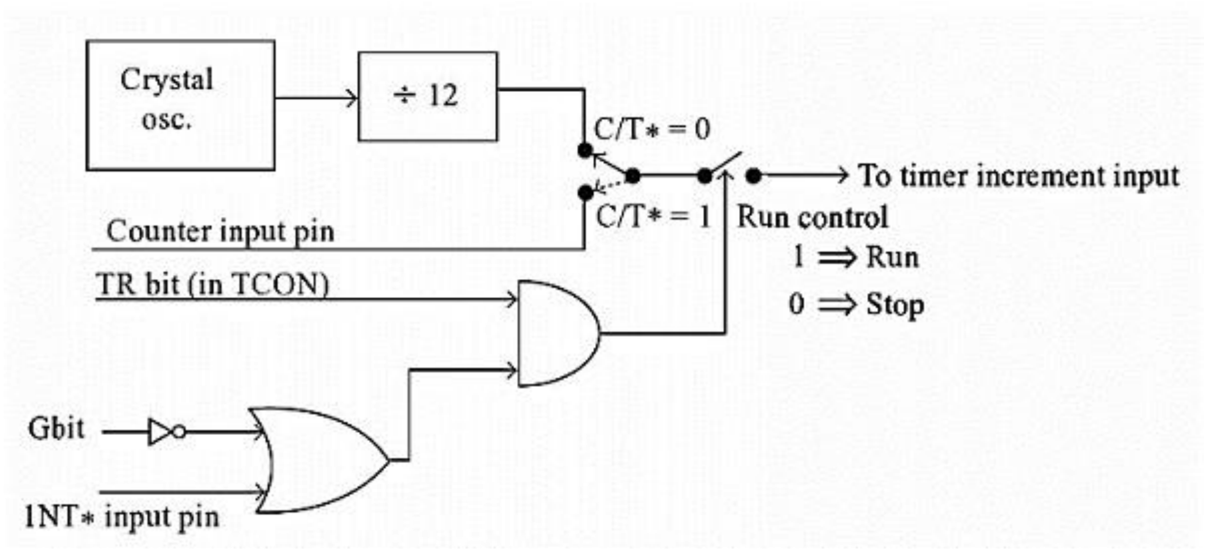


Fig 3. Operating Modes Circuit Control

In the following table, we will see the bit details and their different operations for high or low value.

Bit Details	**High Value (1)**	**Low Value (0)**
C/**T**	Configure for the Counter operations	Configure for the Timer operations
Gate (G)	Timer0 or Timer1 will be in RunMode when TRX bit of TCON register is high.	Timer0 or Timer1 will be in RunMode when TRX bit of TCON register is high and INT0 or INT1 is high.

Bit Details	**00**	**01**	**10**	**11**
M1 M0	This is for Mode 0. (8-bit timer/counter, with 5-bit pre-scaler)	This is Mode 1. (16-bit timer/counter)	This is Mode 3 (8-bit auto reload-timer/counter)	This is Mode 3 (The function depends on Timer0 or Timer1)

The Gate bit will be high when the timer or counter is in mode 0 to 2.

Operating Mode 0 of Timer/Counter

The Mode 0 operation is the 8-bit timer or counter with a 5-bit pre-scaler. So it is a 13-bit timer/counter. It uses 5 bits of TL0 or TL1 and all of the 8-bits of TH0 or TH1.

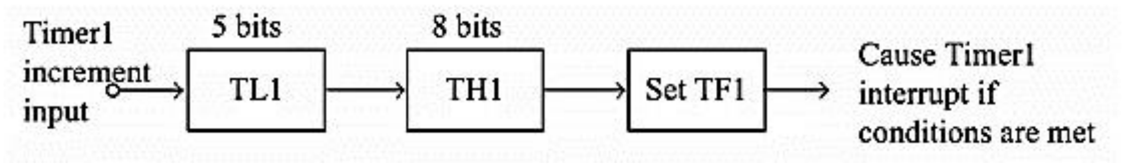


Fig 4. Operating Mode 0 of Timer/Counter

In this example the Timer1is selected, in this case, every 32 (25)event for counter operations or 32 machine cycles for timer operation, the TH1 register will be incremented by 1. When the TH1overflows from FFH to 00H, then the TF1 of TCON register will be high, and it stops the timer/counter. So for an example, we can say that if the TH1 is holding F0H, and it is in timer mode, then TF1will be high after 10H * 32 = 512 machine cycles.

MOVTMOD, #00H

MOVTH1, #0F0H

```
MOVIE, #88H

SETB TR1
```

In the above program, the Timer1 is configured as timer mode 0. In this case Gate = 0. Then the TH1 will be loaded with F0H, then enable the Timer1 interrupt. At last set the TR1 of TCON register, and start the timer.

Operating Mode 1 of Timer/Counter

The Mode 1 operation is the 16-bit timer or counter. In the following diagram, we are using Mode 1 for Timer0.

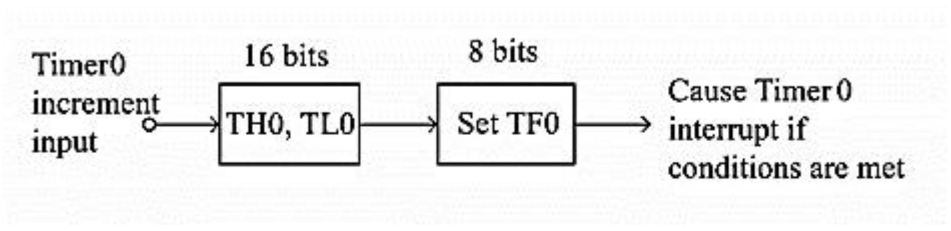


Fig 5. Operating Mode 1 of Timer/Counter

In this case every event for counter operations or machine cycles for timer operation, the TH0– TL0 register-pair will be incremented by 1. When the register pair overflows from FFFFH to 0000H, then the TF0 of TCON register will be high, and it stops the timer/counter. So for an example, we can say that if the TH0 – TL0 register pair is holding FFF0H, and it is in timer mode, then TF0 will be high after 10H = 16 machine cycles. When the clock frequency is 12MHz, then the following instructions generate an interrupt 16 µs after Timer0 starts running.

```
MOVTMOD, #01H

MOVTL0, #0F0H

MOVTH0, #0FFH

MOVIE, #82H

SETB TR0
```

In the above program, the Timer0 is configured as timer mode 1. In this case Gate = 0. Then the TL0 will be loaded with F0H and TH0 is loaded with FFH, then enable the Timer0 interrupt. At last set the TR0 of TCON register, and start the timer.

Operating Mode 2 of Timer/Counter

The Mode 2 operation is the 8-bit auto reload timer or counter. In the following diagram, we are using Mode 2 for Timer1.

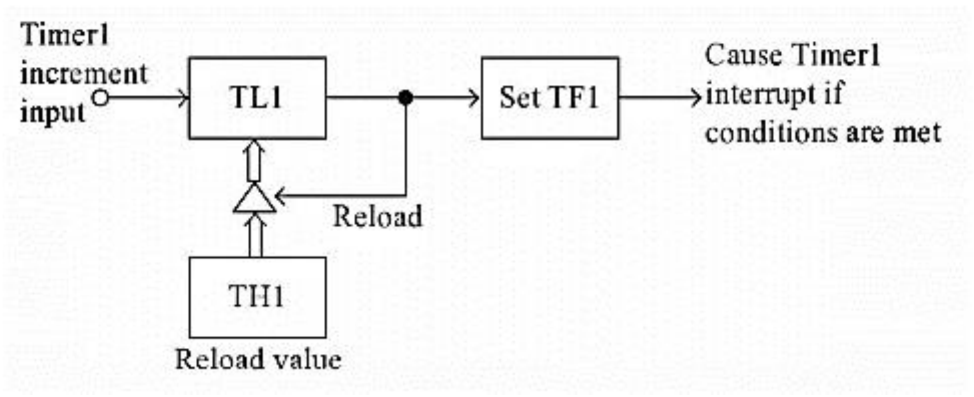


Fig 6. Operating Mode 2 of Timer/Counter

In this case every event for counter operations or machine cycles for timer operation, the TL1register will be incremented by 1. When the register pair overflows from FFH to 00H, then the TF1 of TCON register will be high, also theTL1 will be reloaded with the content of TH1 and starts the operation again. So for an example, we can say that if the TH1 and TL1 register both are holding F0H and it is in timer mode, then TF1 will be high after 10H= 16 machine cycles. When the clock frequency is 12MHz this happens after 16 µs, then the following instructions generate an interrupt once every 16 µs after Timer1 starts running.

```
MOVTMOD, #20H

MOVTL1, #0F0H

MOVTH1, #0F0H

MOVIE, #88H
```

SETBTR1

In the above program, the Timer1 is configured as timer mode 2. In this case Gate = 0. Then the TL1 and TH1 are loaded with F0H. then enable the Timer1 interrupt. At last set the TR1 of TCON register, and start the timer. Timer1 in mode 2 generates the desired baud rate when the serial port is working on Mode 1 or 3.

Operating Mode 3 of Timer/Counter

Mode 3 is different for Timer0 and Timer1. When the Timer0 is working in mode 3, the TL0 will be used as an 8-bit timer/counter. It will be controlled by the standard Timer0 control bits, T0 and INT0 inputs. The TH0 is used as an 8-bit timer but not the counter. This is controlled by Timer1 Control bit TR1. When the TH0 overflows from FFH to 00H, then TF1 is set to 1. In the following diagram, we can Timer0 in Mode 3.

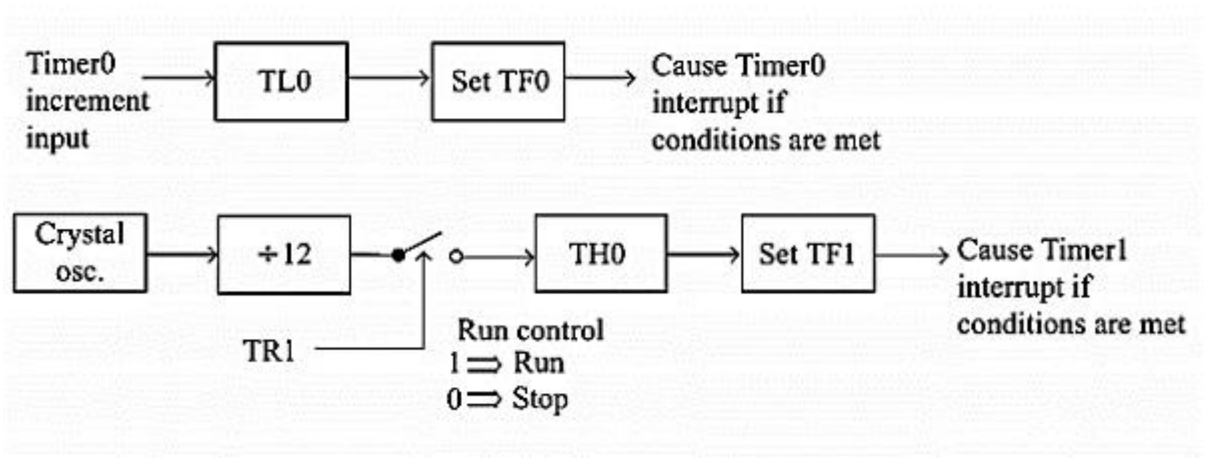
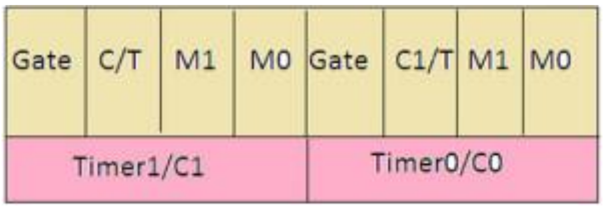

Timer0 increment input
TL0
Set TF0
Cause Timer0 interrupt if conditions are met
Crystal osc.
÷12
TH0
Set TF1
Cause Timer1 interrupt if conditions are met
TR1
Run control
1 ⇒ Run
0 ⇒ Stop

Fig 7. Operating Mode 0 of Timer/Counter

When the Timer1 is working in Mode 3, it simply holds the count but does not run. When Timer0 is in mode 3, the Timer1 is configured in one of the mode 0, 1 and 2. In this case, the Timer1 cannot interrupt the microcontroller. When the TF1 is used by TH0 timer, the Timer1 is used as Baud Rate Generator. The meaning of gate bit in Timer0 and Timer1 for mode 3 is as follows: It controls the running of 8-bit timer/counter TL0 as like Mode 0, 1, or 2. The running of TH0 is controlled by TR1 bit only. So the gate bit in this mode for Timer0 has no specific role. The mode 3 is present for applications requiring an extra 8-bit timer/counter. In Mode 3 of Timer0, the 8051 has three timers. One 8-bit timer by TH0, another8-bit timer/counter by TL0, and one 16-bit timer/counter by Timer1.

5.4.2 Control Register:

Timer Mode Control (TMOD): TMOD is an 8-bit register used for selecting timer or counter and mode of timers. Lower 4-bits are used for control operation of timer 0 or counter0, and remaining 4-bits are used for control operation of timer1 or counter1.This register is present in SFR register, the address for SFR register is 89th.



Gate	C/T	M1	M0	Gate	C1/T	M1	M0
Timer1/C1				Timer0/C0			

Fig 8. Timer Mode Control (TMOD)

Gate: If the gate bit is set to ‘0’, then we can start and stop the “software” timer in the same way. If the gate is set to ‘1’, then we can perform hardware timer.

C/T: If the C/T bit is ‘1’, then it is acting as a counter mode, and similarly when set C+ =/T bit is ‘0’; it is acting as a timer mode.

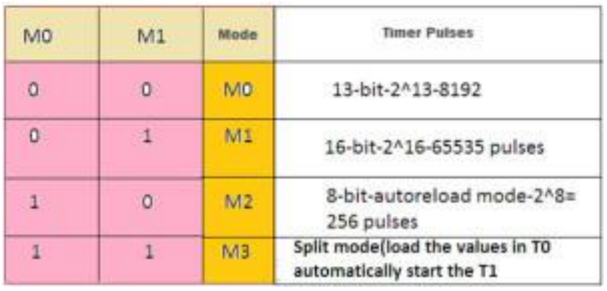
Mode select bits: The M1 and M0 are mode select bits, which are used to select the timer operations. There are four modes to operate the timers.

Mode 0: This is a 13-bit mode that means the timer operation completes with “8192” pulses.

Mode 1: This is a16-bit mode, which means the timer operation completes with maximum clock pulses that “65535”.

Mode 2: This mode is an 8-bit auto reload mode, which means the timer operation completes with only “256” clock pulses.

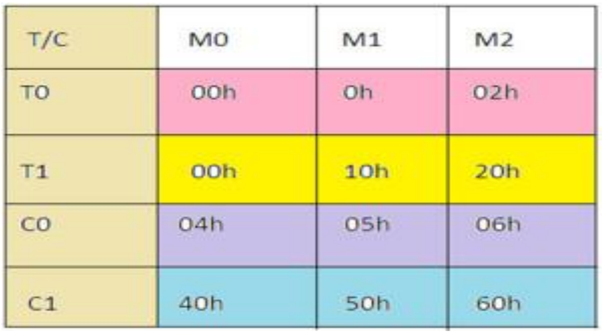
Mode 3: This mode is a split-timer mode, which means the loading values in T0 and automatically starts the T1.



M0	M1	Mode	Timer Pulses
0	0	M0	13-bit-2^13-8192
0	1	M1	16-bit-2^16-65535 pulses
1	0	M2	8-bit-autoreload mode-2^8= 256 pulses
1	1	M3	Split mode(load the values in T0 automatically start the T1

Fig 9. Mode Selection Bits

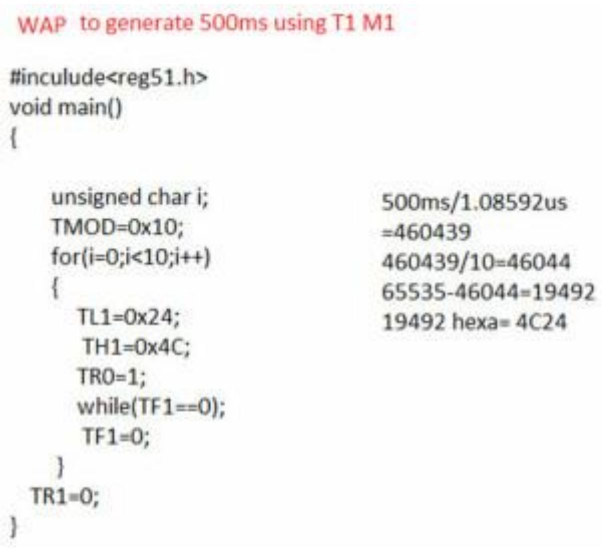
Mode selection Values of timers and counter in 8051



T/C	M0	M1	M2
T0	00h	0h	02h
T1	00h	10h	20h
C0	04h	05h	06h
C1	40h	50h	60h

Fig 10. Mode selection values of timers and counters

Sample Program:



WAP to generate 500ms using T1 M1

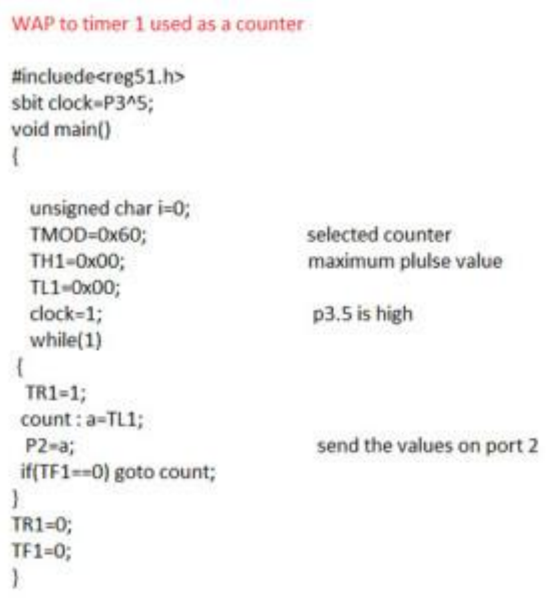
```
#inculude<reg51.h>
void main()
{

    unsigned char i;
    TMOD=0x10;
    for(i=0;i<10;i++)
    {
       TL1=0x24;
        TH1=0x4C;
       TR0=1;
       while(TF1==0);
        TF1=0;
     }
  TR1=0;
}
```

```
500ms/1.08592us
=460439
460439/10=46044
65535-46044=19492
19492 hexa= 4C24
```

Fig 11. Sample Program

Counters in 8051

Keep a counter by keeping C/T bit high, i.e., logic ‘1’ in the TMOD register. For better understanding, we have given one program which uses timer 1 as a counter. Here the LEDs are connected to 8051 Port 2, and the switch to the timer1 pin P3.5; and therefore, if the switch is pressed, the value will be counted. Otherwise, an externally connected sensor to this counter pin as input does this counting operation.



WAP to timer 1 used as a counter

```
#includede<reg51.h>
sbit clock=P3^5;
void main()
{

  unsigned char i=0;
  TMOD=0x60;                    selected counter
  TH1=0x00;                     maximum plulse value
  TL1=0x00;
  clock=1;                      p3.5 is high
  while(1)
 {
  TR1=1;
 count : a=TL1;
  P2=a;                          send the values on port 2
 if(TF1==0) goto count;
}
TR1=0;
TF1=0;
}
```

Fig 12. Sample Counter Program

Applications of Timers and Counters in 8051- Digital Counter with 8051

The Digital counter with 8051 is achieved by programming the microcontroller as discussed above and by attaching a sensor system to it. This object counter uses IR sensor that detects the obstacle near to it and also enables the pin of the microcontroller 06. When an object passes through the sensors, then the microcontroller gets an interrupt signal from the IR sensors and increment the count which is displayed in the 7-segment display.

Let us sum up:

- **Timers and Counters**: The 8051 microcontroller contains two 16-bit timers/counters, Timer 0 and Timer 1.

- These timers consist of a 16-bit register, split into lower and higher bytes (TL and TH). Both can be used either as timers (to generate time delays) or as counters (to count external events).
- **Operating Modes**:
  - **Mode 0:** Functions as a 13-bit timer/counter where 5 bits are from TL and 8 bits are from TH. The timer increments every 32 machine cycles**.**
  - **Mode 1:** A full 16-bit timer/counter that increments every machine cycle. When it overflows, the overflow flag (TF) is set**.**
  - **Mode 2:** An 8-bit auto-reload mode where the TL register increments. On overflow, TL is reloaded from TH, and the operation continues**.**

Check your Progress

1. **How many timers/counters are available in the 8051 microcontroller?**

   a) 1

   b) 2

   c) 3

   d) 4

   Answer: b) 2

2. **Which register is used to configure the mode of timers in the 8051?**

   a) TCON

   b) TMOD

   c) TH

   d) TL

   Answer: b) TMOD

3. **What is the size of Timer 0 in Mode 1?**

   a) 8-bit

   b) 16-bit

c) 13-bit

d) 32-bit

Answer: b) 16-bit

4. **In which mode does Timer 0 act as two separate 8-bit timers?**

   a) Mode 0

   b) Mode 1

   c) Mode 2

   d) Mode 3

   Answer: d) Mode 3

5. **What does the C/T bit in the TMOD register determine?**

   a) Timer mode

   b) Counter or Timer mode

   c) Interrupt enable

   d) Baud rate selection

   Answer: b) Counter or Timer mode

5.5 INTERRUPTS

IMPORTANCE OF INTERRUPTS.

During program execution if peripheral devices need service from microcontroller, device will generate interrupt and gets the service from microcontroller. When peripheral device activates the interrupt signal, the processor branches to a program called interrupt service routine. After executing the interrupt service routine the processor returns to the main program. Steps taken by processor while processing an interrupt:

1. It completes the execution of the current instruction.

2. PSW is pushed to stack.

3. PC content is pushed to stack.

4. Interrupt flag is reset.

5. PC is loaded with ISR address.

ISR will always ends with RETI instruction. The execution of RETI instruction results in the following.

1. POP the current stack top to the PC.

2. POP the current stack top to PSW.

Classification of interrupts.

1. External and internal interrupts.

External interrupts are those initiated by peripheral devices through the external pins of the microcontroller. Internal interrupts are those activated by the internal peripherals of the microcontroller like timers, serial controller etc.)

2. Maskable and non-maskable interrupts.

The category of interrupts which can be disabled by the processor using program is called maskable interrupts. Non-maskable interrupts are those categories by which the programmer cannot disable it using program.

3. Vectored and non-vectored interrupt.

Starting address of the ISR is called interrupt vector. In vectored interrupts the starting address is predefined. In non-vectored interrupts, the starting address is provided by the peripheral as follows.

- Microcontroller receives an interrupt request from external device.
- Controller sends an acknowledgement (INTA) after completing the execution of current instruction.

The peripheral device sends the interrupt vector to the microcontroller.

5.5.1 Interrupts in 8051:

Interrupts are the events that temporarily suspend the main program, pass the control to the external sources and execute their task. It then passes the control to the main program where it had left off.8051 has 5 interrupt signals, i.e. INT0, TFO, INT1, TF1, RI/TI. Each interrupt can be enabled or disabled by setting bits of the IE register and the whole interrupt system can be disabled by clearing the EA bit of the same register.

- **IE (Interrupt Enable) Register**

This register is responsible for enabling and disabling the interrupt. EA register is set to one for enabling interrupts and set to 0 for disabling the interrupts. Its bit sequence and their meanings are shown in the following figure.

EA	-	-	ES	ET1	EX1	ET0	EX0

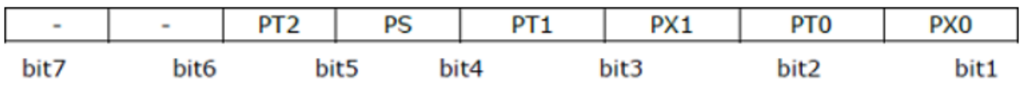
EA	IE.7	It disables all interrupts. When EA = 0 no interrupt will be acknowledged and EA = 1 enables the interrupt individually.
-	IE.6	Reserved for future use.
-	IE.5	Reserved for future use.
ES	IE.4	Enables/disables serial port interrupt.
ET1	IE.3	Enables/disables timer1 overflow interrupt.
EX1	IE.2	Enables/disables external interrupt1.
ET0	IE.1	Enables/disables timer0 overflow interrupt.

EX0	IE.0	Enables/disables external interrupt0.

- **IP (Interrupt Priority) Register**

The priority levels of the interrupts can be altered by changing the corresponding bit in the Interrupt Priority (IP) register as shown in the following figure.

- A low priority interrupt can only be interrupted by the high priority interrupt, but not interrupted by another low priority interrupt.
- If two interrupts of different priority levels are received simultaneously, the request of higher priority level is served.
- If the requests of the same priority levels are received simultaneously, then the internal polling sequence determines which request is to be serviced.



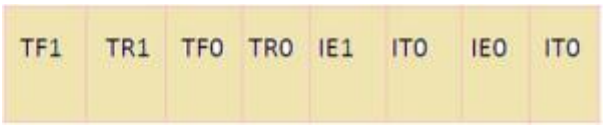
-	-	PT2	PS	PT1	PX1	PT0	PX0
bit7	bit6	bit5	bit4	bit3	bit2	bit1	

-	IP.6	Reserved for future use.
-	IP.5	Reserved for future use.
PS	IP.4	It defines the serial port interrupt priority level.
PT1	IP.3	It defines the timer interrupt of 1 priority.
PX1	IP.2	It defines the external interrupt priority level.
PT0	IP.1	It defines the timer0 interrupt priority level.
PX0	IP.0	It defines the external interrupt of 0 priority level.

5.5.2 Interrupt Control Register:

Timer Control Register (TCON):

The interruptions that the microcontroller interfaces with (external) devices are known as external interrupts. They are received by the controller’s INTx pins. These may be triggered by edges or levels. Interrupt is enabled for a low at the INTx pin when it is level triggered, and for a high to low transition at the INTx pin when it is edge triggered. TCON is another register used to control operations of counter and timers in microcontrollers. It is an 8-bit register wherein four upper bits are responsible for timers and counters and lower bits are responsible for interrupts.



TF1	TR1	TF0	TR0	IE1	IT0	IE0	IT0

Fig 13. Timer Control Register (TCON)

TF1: The TF1 stands for ‘timer1’ flag bit. Whenever calculating the time-delay in timer1, the TH1 and TL1 reaches to the maximum value that is “FFFF” automatically.

EX: while (TF1==1)

Whenever the TF1=1, then clear the flag bit and stop the timer.

TR1: The TR1 stands for timer1 start or stop bit. This timer starting can be through software instruction or through hardware method.

EX: gate=0 (start timer 1 through software instruction) TR1=1; (Start timer)

TF0: The TF0 stands for ‘timer0’ flag-bit. Whenever calculating the time delay in timer1, the TH0 and TL0 reaches to a maximum value that is ‘FFFF’, automatically.

EX: while (TF0==1) Whenever the TF0=1, then clear the flag bit and stop the timer.

TR0: The TR0 stands for ‘timer0’ start or stop bit; this timer starting can be through software instruction or through hardware method.

EX: gate=0 (start timer 1 through software instruction) TR0=1; (Start timer)

5.5.3 Execution of Interrupts:

Types of 8051 Microcontroller Interrupts

8051 Microcontroller suffers five different types of interrupts that hampers the main program execution. These five types of interrupts are:

- Timer 0 overflow interrupt- TF0
- Timer 1 overflow interrupt-TF1
- External hardware interrupt- INT0
- External hardware interrupt- INT1
- Serial communication interrupt- RI/TI

External Hardware Interrupt- (INT0 & INT1)

The 8051 microcontrollers are able to respond to external events through its external interrupts, INT0 and INT1.

External Interrupt 0 (INT0)

- It is connected to the 8051's pin PORT3.2.
- An interrupt request is issued when this pin transitions from low to high in response to an external signal.
- It is possible to program the microcontroller to carry out a particular Interrupt Service Routine (ISR) in response to this interrupt.
- Set the IE (Interrupt Enable) bit for INT0 in the TCON register and configure the IT0 (Interrupt Type 0) bit in the TCON register corresponding to the desired triggering condition (edge or level-triggered) in order to enable and configure INT0.

External Interrupt 1 (INT1)

- It is connected to the 8051's pin PORT3.3
- When that particular pin encounters a low-to-high transitions, INT1, like INT0, creates an interrupt request.
- By configuring the IT1 (Interrupt Type 1) bit in the TCON register and setting the IE bit for INT1 in the TCON register, one can enable and configure INT1.
- A specific ISR can be executed by the microcontroller in response to INT1.

Timer Interrupts (Timer0 and Timer1)

Timer 0 and Timer 1 are hardware timers with internal timer interrupts featured in the 8051 microcontrollers. In microcontroller applications, these timers are used to measure time intervals and generate precise delays. The interrupt system of the microcontroller enables it to react quickly to outside events. Interrupts for Timer 0 and Timer 1 are produced when their respective timers exceed their limit. The microcontroller will run the interrupt service routine (ISR) for that timer if the related interrupt is enabled, and the associated interrupt flag is set upon overflow.

Timer0 Interrupt

- Since Timer 0 is an 8-bit timer, its count range is 0 to 255.
- There are two modes of operation for it, 13-bit and 16-bit. It employs the TH0 (Timer 0 High) and TL0 (Timer 0 Low) registers in 13-bit mode and only the TH0 register in 16-bit mode.
- It is possible to set timer 0 to interrupt when it approaches zero instead of staying at its maximum value. The microcontroller can perform a particular interrupt service routine (ISR) in response to the interrupt request that this overflow generates.

Timer1 Interrupt

- Timer 1 is a 16-bit timer with a counting range of 0 to 65,535.
- It can operate in 16- or 8-bit mode. It employs the TL1 (Timer 1 Low) and TH1 (Timer 1 High) registers in 8-bit mode and only the TH1 register in 16-bit mode.
- Timer 1 can be set up to produce an interrupt when it overflows, just like Timer 0. This interruption may cause a certain ISR to be executed.

Serial Communication Interrupts (UART)

UART (Universal Asynchronous Receiver/Transmitter) is a serial communication protocol used with 8051 microcontrollers. Data is sent over a single cable, bit by bit, in serial transmission. In this sense, “interrupts” refers to the processes that enable the microcontroller to react quickly to external events.

Addressing UART communication with the 8051’s interrupts:

- **Initialization of UART-** Set the data format, baud rate, and enable the UART module by configuring the UART registers.
- **Interrupt Enable**– Depending on the operation you wish to interrupt for, enable the UART’s transmit interrupt (TI) or receive interrupt (RI).
- **ISR (interrupt service routine)-** To handle the interrupt, write an ISR. The ISR in UART communication normally verifies whether the transmit buffer is ready (TI) or whether data has been received (RI).

- **Clearing the Flag- To** recognize the interrupt and get ready for the next one, in the ISR, clear the associated interrupt flag (RI or TI).

Interrupt Vector Table

The addresses of different interrupt service routines (ISRs) are stored in a table called the Interrupt Vector Table (IVT) in an 8051 microcontroller. It is a vital aspect of the interrupt handling mechanism in the microcontroller. When an interrupt occurs the interrupt specific ISR is executed by jumping the program counter to the corresponding address in the IVT. There are memory areas set aside specifically for the IVT in the 8051 microprocessors.

Interrupt	**Flag**	**Interrupt Vector Address**
Reset	–	0000H
INT0 (External Interrupt 0)	IE0	0003H
Timer0	TF0	000BH
INT1 (External Interrupt 1)	IE1	0013H
Timer 1	TF1	001BH
Serial Interrupt	TI/RI	0023H

5.5.4 Application of 8051 Interrupts:

- In real-time systems, interrupts are essential for speedy responses to external occurrences. Due to its ability to swap tasks fast through interrupts, the 8051 is a good choice for applications that need exact timing.
- Interrupts serve in the control of incoming data in communication applications, assuring timely transmission or rapid handling of received information.

- In embedded systems, 8051 interrupts are frequently utilized for activities like sensor interfacing, where the microcontroller must react quickly to environmental changes.
- Interruptions assist in ensuring timely data sampling and processing in situations when data must be obtained from sensors or external devices.

Let us sum up:

➢ **Time Control Register (TCON)**

- TCON manages external interrupts from the INTx pins and indicates edge or level triggering.
- The upper four bits control timers and counters; the lower bits manage interrupts.

➢ **Interrupts in 8051**

- The 8051 has five interrupt signals: INT0, TF0, INT1, TF1, and RI/TI.
- Interrupts can be controlled via the Interrupt Enable (IE) register.
- The EA bit in the IE register can disable all interrupts when cleared.
- The Interrupt Priority (IP) register manages interrupt priority levels

Check Your Progress

1. **What does an interrupt allow the microcontroller to do during program execution?**

   A) Terminate the program

   B) Suspend the main program to execute the ISR

   C) Increase the program speed

   D) Ignore peripheral requests

   Answer: B) Suspend the main program to execute the ISR

2. **Which register is responsible for enabling and disabling interrupts in the 8051 microcontroller?**

   A) IP Register

B) TCON Register

C) IE Register

D) PSW Register

Answer: C) IE Register

3. **What type of interrupt cannot be disabled by the programmer?**

A) Maskable Interrupt

B) Non-maskable Interrupt

C) Vectored Interrupt

D) Internal Interrupt

Answer: B) Non-maskable Interrupt

4. **How does the 8051 microcontroller identify which ISR to execute when an interrupt occurs?**

A) By checking the program counter

B) By referencing the Interrupt Vector Table

C) By using the PSW

D) By counting the number of interrupts

Answer: B) By referencing the Interrupt Vector Table

5. **What happens when the RETI instruction is executed in an ISR?**

A) It halts the program

B) It resets the interrupt flag

C) It pops the current stack top to the PC and PSW

D) It loads the ISR address into the PC

Answer: C) It pops the current stack top to the PC and PSW

Summary:

The foundation for understanding microcontrollers, distinguishing them from microprocessors, and delving into the architecture of the widely used 8051 microcontroller is crucial. It provides essential knowledge on pin configuration, timers, counters, and interrupt handling, all vital for designing and implementing efficient embedded systems. Mastery of these concepts is fundamental for anyone pursuing a career in embedded systems and electronics engineering. Overall, it equips students with practical skills and theoretical understanding necessary for real-world applications.

In this unit module, we covered:

- The fundamental differences between microcontrollers and microprocessors.
- The architecture of the 8051 microcontroller, including its CPU, memory, I/O ports, timers/counters, serial port, and interrupt system.
- The detailed pin description of the 8051 microcontroller.
- The operating modes and control registers of the 8051's timers and counters.
- The interrupt system in the 8051, including interrupt sources, the Interrupt Enable (IE) register, and the Interrupt Priority (IP) register.

The execution process of interrupts in the 8051 microcontroller.

In this unit, an introduction about the fundamental differences between microcontrollers and microprocessors. Microcontrollers, like the 8051, integrate a CPU, memory, and I/O peripherals on a single chip, making them ideal for embedded systems, while microprocessors are standalone CPUs requiring external components.

The architecture of the 8051 microcontrollers is explored, which includes a CPU, 4 KB of ROM, 128 bytes of RAM, four 8-bit I/O ports, two 16-bit timers/counters, a serial port, and an interrupt system with five sources. Also, covered its pin configuration, highlighting key functions like power supply, ground, I/O ports, and control signals for memory interfacing.

The unit detailed the 8051's timers and counters, focusing on their operating modes and control registers (TMOD and TCON). Additionally, we examined the interrupt system, learning how the microcontroller handles events through interrupt service routines (ISRs) and how the interrupt control registers (IE and IP) manage enabling, disabling, and prioritizing interrupts.

Overall, a solid understanding of the 8051 microcontroller's architecture, pin configuration, timers, counters, and interrupt handling, essential for designing embedded systems is explained.

Activities

Activity 1: Write a program to continuously generate a square wave of 2 kHz frequency on pin P1.5 using timer 1. Assume the crystal oscillator frequency to be 12 MHz.

Activity 2: System Design Module

Objective: Design and implement a simple embedded system using the 8051 microcontrollers.

Instructions:

- Design a simple embedded system that includes:
    - An input(button)
    - An output (LED or a buzzer)
    - A timer to create a periodic event
    - An interrupt for handling the button press
- Draw the circuit diagram for the system with pseudocode
- Test the system on an 8051-development board

Check Your Progress

1. What are some common applications of the 8051 microcontroller architecture?

----

2. Are there any newer microcontroller architectures that are similar to the 8051?

----

3. Provide an example of a real-world application that uses the 8051 microcontrollers. Explain the role of the 8051 in this application

----

----

4. How would you configure Timer 0 in mode 1 to generate a delay of 1 second? Provide the calculations.

--- -------------------------

Self-Assessment Questions

1. Define the concept of a microcontroller and how it differs from a microprocessor.
2. What are the main components of a microcontroller, and what role does each play?
3. Explain the significance of interrupts in microcontroller operation.
4. Describe the steps taken by a processor when an interrupt is received.
5. What are the differences between external and internal interrupts? Provide examples.
6. Explain the distinction between maskable and non-maskable interrupts.
7. Define vectored and non-vectored interrupts, and describe how each is processed.
8. In the 8051 microcontroller, list the five types of interrupts and their characteristics.
9. Describe the function of the Interrupt Enable (IE) register in the 8051 microcontroller.
10. What is the purpose of the Interrupt Priority (IP) register, and how does it influence interrupt handling?
11. Explain the Timer Control Register (TCON) and its role in managing timer interrupts.
12. How do the external hardware interrupts (INT0 and INT1) operate in the 8051 microcontroller?

13. Discuss the operational differences between Timer 0 and Timer 1 interrupts in the 8051 microcontroller.
14. What is UART, and how is it integrated into the interrupt system of the 8051 microcontroller?
15. Describe the structure and function of the Interrupt Vector Table (IVT) in the 8051 microcontroller.
16. What are some practical applications of interrupts in embedded systems and real-time applications?
17. How can the interrupt handling system improve the performance of microcontroller-based applications?

Further Reading and References

Textbooks

1. "The 8051 Microcontroller and Embedded Systems: Using Assembly and C" by Muhammad Ali Mazidi

- Comprehensive guide to 8051 architecture, programming, and applications.

- ISBN: 978-0130344681

2. "Programming and Customizing the 8051 Microcontroller" by Myke Predko

- Focuses on programming techniques and customization for various applications.

- ISBN:978-0079115975

3. "Microcontrollers: Principles and Applications" by William J. Agranoff

- Covers microcontroller basics, architecture, and applications, including 8051.

- ISBN:978-0130189437

4. "Microcontroller Theory and Applications with the 68HC12 and 68HC11" by Daniel J. Pack and Steven F. Barrett

- While focused on other microcontrollers, it provides insights into architecture relevant to understanding 8051.

- ISBN: 978-0130146898

5. "Embedded Systems: Real-Time Operating Systems for Arm Cortex M Microcontrollers" by Jonathan Valvano

- Discusses real-time systems that can also be applied to 8051 microcontroller concepts.

- ISBN: 978-1475791767

Online Resources

1. TutorialsPoint - 8051 Microcontroller Architecture

- Detailed tutorials covering architecture, programming, and interfacing.

- [TutorialsPoint - 8051 Microcontroller Architecture] (https://www.tutorialspoint.com/microprocessor/microcontrollers_8051_architecture.htm)

2. Microcontroller Central

- Articles and resources on various microcontrollers, including 8051.

- [Microcontroller Central] (http://www.microcontrollercentral.com)

3. All About Circuits - 8051 Microcontroller

- A comprehensive overview of 8051 microcontroller features, applications, and tutorials.

- [All About Circuits - 8051] (https://www.allaboutcircuits.com/technical-articles/8051-microcontroller-architecture/)

4. Embedded.com

- Offers articles, tutorials, and forums focused on embedded systems, including the 8051.

- [Embedded.com] (https://www.embedded.com)

5. Hackster.io - 8051 Projects

- Project ideas and implementations using the 8051 microcontroller.

- [Hackster.io - 8051 Projects] (https://www.hackster.io/search?i=projects&q=8051)

Video Material

1. YouTube - "Introduction to 8051 Microcontroller" by Neso Academy

- Comprehensive introduction and tutorials on 8051 architecture and programming.

- [Neso Academy - 8051 Microcontroller]
(https://www.youtube.com/watch?v=RxlQ6X19AzA)

2. YouTube - "8051 Microcontroller Basics" by The Engineering Mindset

- Provides clear explanations of the architecture, pin configurations, and applications.

- [The Engineering Mindset - 8051 Basics]
(https://www.youtube.com/watch?v=H3x35DwnrF0)

3. Coursera - "Introduction to Embedded Systems"

- Covers basics of embedded systems including microcontroller concepts.

- [Coursera - Introduction to Embedded Systems]
(https://www.coursera.org/learn/embedded-systems)

4. Udemy - "Embedded Systems Programming on ARM Cortex-M3/M4 Processor"

- While focused on ARM, it helps in understanding embedded programming principles applicable to 8051.

- [Udemy - Embedded Systems Programming]
(https://www.udemy.com/course/embedded-systems-programming-on-arm-cortex-m3-m4-processor/)

5. freeCodeCamp - "Microcontroller Basics"

- Video tutorials covering the fundamentals of microcontroller programming and applications.

- [freeCodeCamp- Microcontroller Basics
(https://www.youtube.com/watch?v=6HTrT0ynKCE)