# Microcontrollers Programming
# (Semester -II)

| Year | I | CourseCode:CSCS106<br>Course Title: Microcontrollers Programming | Credits | 4 |
|---|---|---|---|---|
| Sem. | II | | Hours | 75 |
| | | | Category | C |

| Unit No. | Course Content | Hours |
|---|---|---|
| | **Theory Component** | |
| Unit I | **Microprocessors and Microcontrollers**<br>Microprocessors vs Microcontrollers –8051 Architecture – Input / Output Pins–Ports–External Memory–Counter and Timers – Serial Data I/O – Interrupts | 9 |
| Unit II | **Programming8051**<br>Addressing Modes – External Data Moves – Code Memory Read-Only Data Moves – PUSH and POP Opcodes – Data Exchanges – Logical Operations – Arithmetic Operations – Jump and Call Opcodes | 9 |
| Unit III | **8051 Microcontroller Design**<br>Microcontroller Specification – Design – Testing – Timing Subroutines – Lookup Tables for 8051 – Serial Data Transmission | 9 |
| Unit IV | **Applications**<br>Keyboards – Displays – Pulse Measurement – D/A and A/D Conversions – Multiple Interrupts | 9 |
| Unit V | **Serial Data Communication**<br>NetworkConfigurations–8051DataCommunication Modes | 9 |
| | **Recommended Learning Resources** | |
| Print Resources | 1. Kenneth J. Ayala, "The 8051 Microcontroller Architecture, Programming, and Applications", Delmar Cengage Learning, Third Edition, 2004.<br>2. Martin Bates," PIC Micro controllers- An Introduction to Microelectronics", Third Edition,Newnes,Elsevier,2011.<br>3. Hubert Henry Ward, "C Programming for the PIC Microcontroller-DemystifyCodingwithEmbeddedProgramming",Apress,UK,2020. https://doi.org/10.1007/978-1-4842-5525-4 | |
| | | |

## Unit - 1: Microprocessors and Microcontrollers

Microprocessors vs Microcontrollers –8051 Architecture – Input / Output Pins–Ports–External Memory–Counter and Timers – Serial Data I/O – Interrupts

---

# Microprocessors vs. Microcontrollers

## Microprocessors

- A microprocessor is a central processing unit (CPU) that performs arithmetic and logic operations, fetches instructions from memory, and processes data.
- It requires external components such as memory (RAM, ROM), input/output ports, and other peripherals for functioning.
- Typically used in computers, servers, and complex processing systems.

**Examples:** Intel 8085, Intel 8086, Intel Pentium, AMD Ryzen.

## Microcontrollers

- A microcontroller is a compact integrated circuit that combines a CPU, memory (RAM and ROM), input/output ports, timers, and communication modules into a single chip.
- It is designed for specific tasks like controlling appliances, vehicles, robots, and embedded systems.
- Consumes less power and is more cost-effective for dedicated applications.

**Examples:** Intel 8051, AVR, PIC, ARM Cortex, Arduino.

## Comparison

| Microprocessor | Microcontroller |
|---|---|
| Contains only CPU; needs external memory and I/O devices. | CPU, memory, I/O ports, and peripherals are integrated. |
| General-purpose; used in computers and servers. | Task-specific; used in embedded systems. |
| Faster and more powerful but consumes more power. | Slower but power-efficient. |
| Complex circuitry due to external components. | Simple and compact design. |
| Higher cost and complexity. | Cost-effective and easy to implement. |

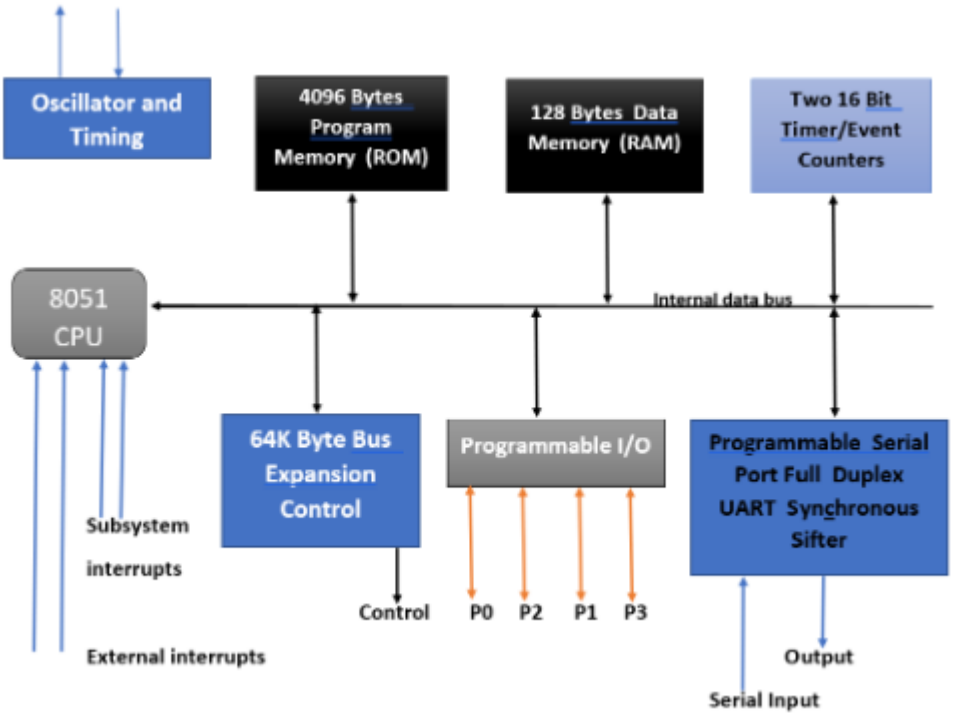
---

# 8051 Microcontroller Architecture

The **Intel 8051** is an 8-bit microcontroller with Harvard architecture, meaning it has separate memory spaces for program and data. It includes a CPU, 128 bytes of RAM, 4 KB of ROM, 32 I/O lines, two timers/counters, a serial port, and five interrupt sources. The 8051 operates on a crystal oscillator (commonly 12 MHz) and supports on-chip peripheral functions for embedded control. The internal RAM is divided into general-purpose RAM, bit-addressable RAM, and special function registers (SFRs), making it highly efficient for I/O control and real-time processing.

### Diagram

The following diagram shows the overall idea and working principles of the 8051 microcontroller.




## Main Components

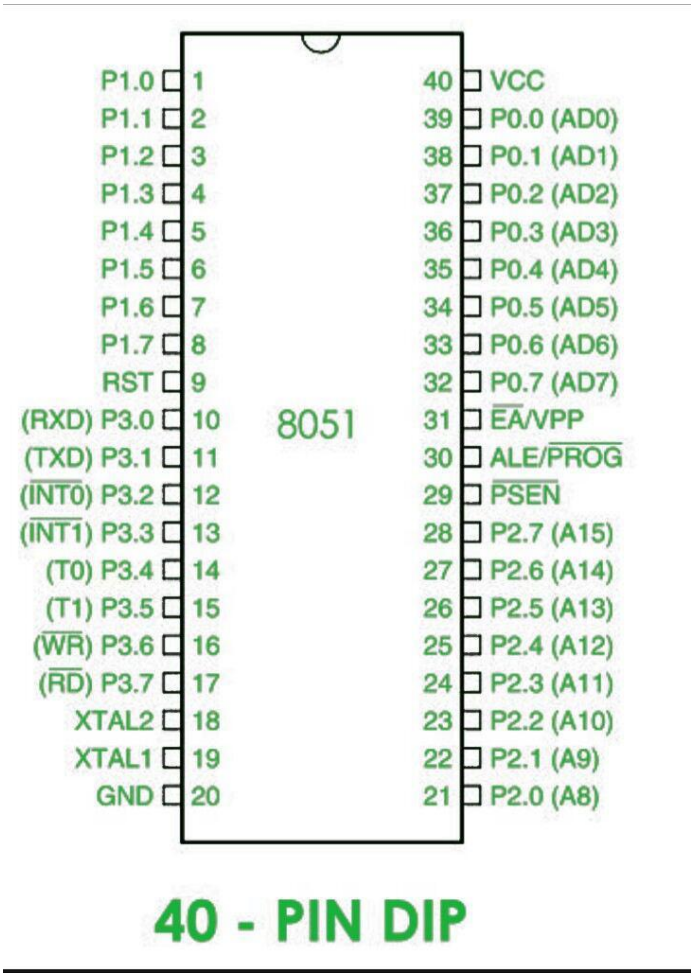
1. **Central Processing Unit (CPU):**
   - Controls and processes all operations.
   - Fetches, decodes, and executes instructions.
2. **Memory:**
   - **128 Bytes of Internal RAM:** Stores temporary data and variables.
   - **4 KB of Internal ROM:** Stores program instructions permanently.
   - External memory can be connected if needed.
3. **Input/Output Ports:**
   - Four 8-bit ports (P0, P1, P2, P3), each with 8 pins, are used to connect external devices like sensors, LEDs, and displays.
4. **Timers and Counters:**
   - Two 16-bit timers (Timer 0 and Timer 1) are used for timing events, generating delays, and counting external pulses.

5. **Serial Communication:**
    - A Universal Asynchronous Receiver-Transmitter (UART) module enables serial data transmission and reception.
6. **Interrupt System:**
    - Five interrupt sources: two external interrupts, two timer interrupts, and one serial communication interrupt.
    - Interrupts improve efficiency by allowing the CPU to respond to events without constantly polling.
7. **Bus System:**
    - **8-bit Data Bus:** Transfers data between CPU, memory, and peripherals.
    - **16-bit Address Bus:** Accesses memory locations, supporting up to 64 KB of external memory.

---

# Input / Output Pins and Ports

The 8051 has **40 pins**, of which **32 pins** are used as input/output. These are organized into **4 ports (P0 to P3)**, each having 8 bits. Each pin can be programmed as input or output depending on the logic written into the program. The remaining 8 pins are used for control (like RESET, XTAL1/2 for oscillator, and control signals such as ALE, PSEN, EA) and power supply (Vcc and GND). The versatility of I/O pins makes 8051 suitable for interfacing with LEDs, sensors, displays, etc.




**40 - PIN DIP**

The 8051 microcontroller has **32 input/output pins** divided into four 8-bit ports:

1. **Port 0 (P0.0 - P0.7):**
    - Open-drain, meaning it requires external pull-up resistors for proper operation.
    - Can be used as an input/output port or as the lower-order address and data bus (AD0 - AD7) for external memory interfacing.
2. **Port 1 (P1.0 - P1.7):**
    - Dedicated input/output port with internal pull-up resistors.
    - Commonly used for general-purpose input/output operations.
3. **Port 2 (P2.0 - P2.7):**
    - Can function as a general-purpose I/O port or as the higher-order address bus (A8 - A15) when interfacing with external memory.
4. **Port 3 (P3.0 - P3.7):**
    - Functions as both a general-purpose I/O port and a control port with specialized functions such as:
        - P3.0: Serial input (RXD)
        - P3.1: Serial output (TXD)
        - P3.2: External interrupt 0 (INT0)
        - P3.3: External interrupt 1 (INT1)
        - P3.4: Timer 0 external input (T0)
        - P3.5: Timer 1 external input (T1)
        - P3.6: External data memory write (WR)
        - P3.7: External data memory read (RD)

---

# External Memory

The 8051 microcontroller has limited internal memory, but external memory can be connected using the address and data buses. The 8051 can address up to 64 KB of external program memory (ROM) and 64 KB of external data memory (RAM). External memory is accessed using Port 0 (lower 8-bit address/data multiplexed) and Port 2 (higher 8-bit address), along with control signals /PSEN (Program Store Enable) for ROM and /RD, /WR for RAM. The EA (External Access) pin determines whether the microcontroller uses internal or external memory.

1. **External RAM (Data Memory):**
    - Up to 64 KB of external RAM can be interfaced using the address bus and Port 0 for data transfer.
    - The ALE (Address Latch Enable) pin is used to separate the address and data signals.
2. **External ROM (Program Memory):**
    - If the internal 4 KB ROM is insufficient, up to 64 KB of external ROM can be connected using Port 2 for higher-order addresses and Port 0 for lower-order addresses and data.
    - The PSEN (Program Store Enable) pin is used to read data from external ROM.

---

# Timers and Counters

The 8051 microcontroller has two 16-bit timers: **Timer 0** and **Timer 1.** Each timer has four modes of operation. The 8051 includes two 16-bit timers/counters (Timer 0 and Timer 1). These can operate in various modes:

Mode 0: 13-bit Timer

Mode 1: 16-bit Timer

Mode 2: 8-bit Auto-reload

Mode 3: Split Timer mode

Timers can be used to create time delays, count external events, or generate baud rates for serial communication. Control registers like TCON and TMOD are used to configure their operation.

1. **Timer Mode:**
    - Generates time delays by counting internal clock pulses.
    - The clock frequency is divided by 12 to determine the timer increment rate.
2. **Counter Mode:**
    - Counts external pulses applied to specific pins (T0 for Timer 0, T1 for Timer 1).
3. **Timer Registers:**
    - Each timer uses two 8-bit registers: TLx (lower byte) and THx (higher byte), where x = 0 or 1.
    - These registers store the current count value.
4. **Timer Control (TCON) Register:**
    - Controls timer operation and flags overflow events.
    - Bits: TF0 (Timer 0 overflow flag), TR0 (Timer 0 run control), TF1 (Timer 1 overflow flag), TR1 (Timer 1 run control).

---

# Serial Data Input/Output (UART Communication)

The 8051 microcontroller includes a built-in UART module that enables serial communication using two pins: The 8051 features full-duplex serial communication via the Serial Control (SCON) register. It supports different modes like asynchronous and synchronous data transmission. The TXD (transmit) and RXD (receive) pins located on Port 3 handle serial data transfer. The baud rate is typically generated using Timer 1. It supports serial data transfer using interrupt-driven or polling methods.

- **RXD (P3.0):** Receives serial data.
- **TXD (P3.1):** Transmits serial data.

1. **Baud Rate:**

    - The speed of data transfer is determined by the baud rate, which is controlled using Timer 1 in mode 2 (auto-reload mode).
2. **Data Format:**
    - Standard formats include 8-bit data with 1 start bit, 1 stop bit, and no parity bit.
3. **SBUF Register:**
    - SBUF (Serial Buffer Register) is used to store data during transmission and reception.
    - Writing data to SBUF transmits it via TXD, and reading from SBUF receives data from RXD.
4. **SCON Register:**
    - The Serial Control (SCON) register configures and controls the serial port.
    - Important bits:
        - REN (Receive Enable): Enables reception.
        - TI (Transmit Interrupt Flag): Set when transmission is complete.
        - RI (Receive Interrupt Flag): Set when a character is received.

---

# Interrupts in 8051 Microcontroller

Interrupts allow the CPU to pause its current task and respond to urgent events. The 8051 supports five interrupt sources: two external interrupts (INT0 and INT1), two timer interrupts (T0 and T1), and one serial communication interrupt. Interrupts allow the microcontroller to respond to external or internal events immediately, improving real-time performance. The IE (Interrupt Enable) and IP (Interrupt Priority) registers are used to enable and prioritize these interrupts. This mechanism allows multitasking by interrupting the main program to service high-priority tasks.

1. **External Interrupt 0 (INT0) – Pin P3.2**
2. **External Interrupt 1 (INT1) – Pin P3.3**
3. **Timer 0 Overflow Interrupt (TF0)**
4. **Timer 1 Overflow Interrupt (TF1)**
5. **Serial Communication Interrupt (RI/TI)**

## Interrupt Priority and Control

- Interrupts are prioritized, with external interrupts having the highest priority, followed by timer interrupts and serial interrupts.
- The Interrupt Enable (IE) register is used to enable or disable interrupts.
- The Interrupt Priority (IP) register assigns priority levels.

**Interrupt Enable (IE) Register Bits:**

- EA: Global interrupt enable (must be set to 1 to enable any interrupt)
- ET0: Timer 0 interrupt enable
- ET1: Timer 1 interrupt enable
- EX0: External interrupt 0 enable
- EX1: External interrupt 1 enable

- ES: Serial interrupt enable

**Interrupt Priority (IP) Register Bits:**

- PT0: Timer 0 interrupt priority
- PT1: Timer 1 interrupt priority
- PX0: External interrupt 0 priority
- PX1: External interrupt 1 priority
- PS: Serial interrupt priority

## Interrupt Execution Sequence

1. When an interrupt occurs, the CPU finishes executing the current instruction.
2. The Program Counter (PC) is saved, and the CPU jumps to the interrupt vector address.
3. After executing the interrupt service routine (ISR), the CPU returns to the main program using the RETI instruction.

---

## Unit – II Programming 8051

Addressing Modes – External Data Moves – Code Memory Read-Only Data Moves – PUSH and POP Opcodes – Data Exchanges – Logical Operations – Arithmetic Operations – Jump and Call Opcodes

---

# Programming the 8051 Microcontroller

## 1. Addressing Modes

Addressing modes define how the operand (data) is specified in an instruction. The 8051 microcontroller supports the following addressing modes:

1. **Immediate Addressing Mode**
   - Data is directly specified in the instruction.
   - Syntax: `MOV A, #25H` (Move hexadecimal 25 into the accumulator A)
2. **Register Addressing Mode**
   - Data is stored in a register and is referred to by its name.
   - Syntax: `MOV A, R1` (Move data from register R1 to accumulator A)
3. **Direct Addressing Mode**
   - Access data stored in a specific internal RAM location.
   - Syntax: `MOV A, 50H` (Move data from RAM address 50H to A)
4. **Indirect Addressing Mode**
   - The address of data is held in a register (R0 or R1) or a pointer register.
   - Syntax: `MOV A, @R0` (Move data from the address pointed to by R0 to A)
5. **External Addressing Mode**
   - Access data from external memory using DPTR (Data Pointer) or register indirect mode.
   - Syntax: `MOVX A, @DPTR` (Move data from external memory address pointed by DPTR to A)
6. **Indexed Addressing Mode**
   - Used to access data from program memory (ROM) using an index.
   - Syntax: `MOVC A, @A+DPTR` (Move code byte from ROM address formed by A + DPTR to A)

---

## 2. External Data Moves

External data moves allow the transfer of data between the microcontroller and external RAM or devices. The `MOVX` instruction is used for this purpose.

- **Using DPTR:**
  - `MOVX A, @DPTR` (Move data from external memory pointed by DPTR to A)
  - `MOVX @DPTR, A` (Move data from A to external memory pointed by DPTR)
- **Using Register Indirect Mode:**
  - `MOVX A, @R0` or `MOVX A, @R1` (Move data from external memory addressed by R0 or R1 to A)

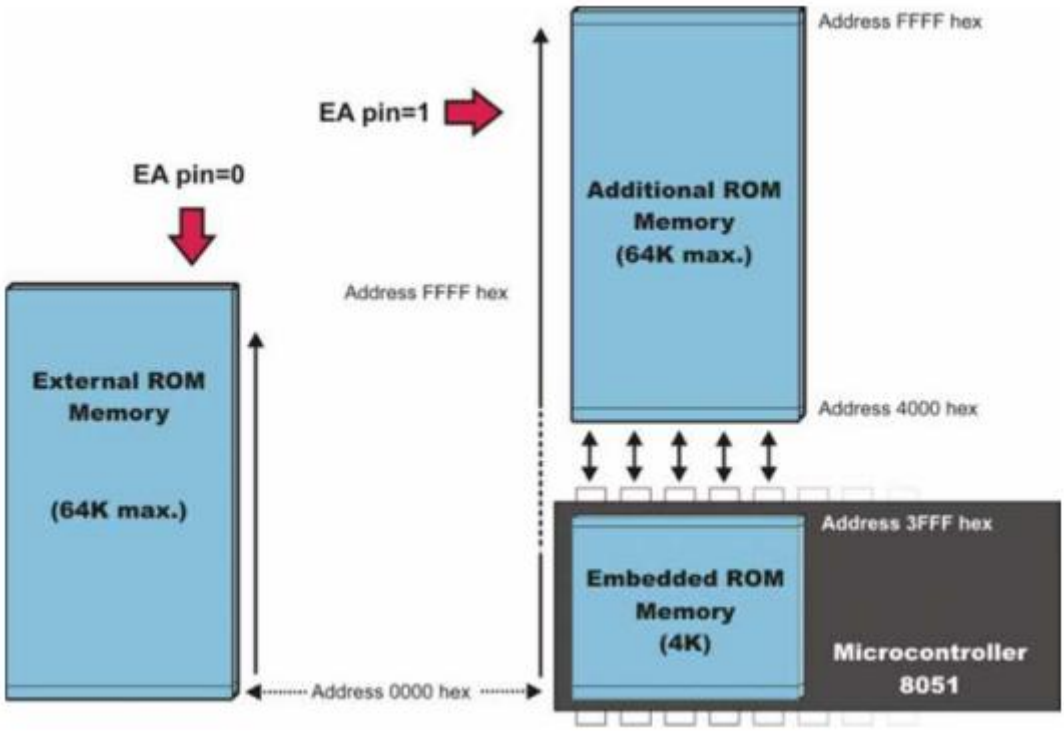
- `MOVX @R0, A` or `MOVX @R1, A` (Move data from A to external memory)

---

## 3. Code Memory Read-Only Data Moves

The instruction `MOVC` is used to move data from code memory (ROM) to the accumulator. Since ROM is read-only, data cannot be written to it.

- **Using DPTR as Base:**
  - `MOVC A, @A+DPTR` (Move code byte from ROM address formed by A + DPTR to A)
- **Using PC as Base:**
  - `MOVC A, @A+PC` (Move code byte from ROM address formed by A + PC to A)




---

## 4. PUSH and POP Opcodes

The **stack** is a region of internal RAM used for temporary data storage. It operates in **Last-In-First-Out (LIFO)** order. The **SP (Stack Pointer)** register points to the top of the stack.

- **PUSH Instruction:** Stores data onto the stack.
  - Syntax: `PUSH 50H` (Push the data from RAM address 50H onto the stack)
  - Operation: SP is incremented, and data is stored at the new stack address.
- **POP Instruction:** Retrieves data from the stack.
  - Syntax: `POP 60H` (Pop data from the stack into RAM address 60H)
  - Operation: Data is retrieved from the address pointed to by SP, and SP is decremented.

**Example:**

```
MOV SP, #07H   ; Initialize stack pointer
MOV A, #25H    ; Load accumulator with 25H
PUSH ACC       ; Push A onto the stack
MOV A, #42H    ; Load accumulator with 42H
POP R1         ; Pop from stack into R1 (R1 = 25H)
```

---

## 5. Data Exchanges

Data exchange instructions swap the contents of registers or memory locations.

1. **XCH A, Rn:** Exchange data between accumulator and register (R0 - R7).
   - Example: `XCH A, R1` (Exchange contents of A and R1)
2. **XCH A, direct:** Exchange data between accumulator and a direct RAM location.
   - Example: `XCH A, 50H` (Exchange contents of A and RAM address 50H)
3. **XCH A, @Ri:** Exchange data between accumulator and the address pointed to by R0 or R1.
   - Example: `XCH A, @R0`
4. **XCHD A, @Ri:** Exchange only the lower nibble (4 bits) between accumulator and indirect address.
   - Example: `XCHD A, @R0`

---

## 6. Logical Operations

Logical instructions manipulate data at the bit level, performing operations like AND, OR, XOR, and NOT.

| Instruction | Description | Example |
|---|---|---|
| `ANL A, Rn` | AND register with accumulator | `ANL A, R1` |
| `ANL A, direct` | AND direct address byte with accumulator | `ANL A, 50H` |
| `ANL A, #data` | AND immediate data with accumulator | `ANL A, #0F0H` |
| `ORL A, Rn` | OR register with accumulator | `ORL A, R2` |
| `ORL A, direct` | OR direct address byte with accumulator | `ORL A, 60H` |
| `ORL A, #data` | OR immediate data with accumulator | `ORL A, #0F0H` |
| `XRL A, Rn` | XOR register with accumulator | `XRL A, R3` |
| `XRL A, direct` | XOR direct address byte with accumulator | `XRL A, 70H` |
| `XRL A, #data` | XOR immediate data with accumulator | `XRL A, #0F0H` |
| `CPL A` | Complement (invert) all bits of accumulator | `CPL A` |
| `CLR A` | Clear all bits of accumulator (set to zero) | `CLR A` |

---

## 7. Arithmetic Operations

Arithmetic instructions perform basic mathematical operations like addition, subtraction, increment, and decrement.

| Instruction | Description | Example |
|---|---|---|
| `ADD A, Rn` | Add register to accumulator | `ADD A, R4` |
| `ADD A, direct` | Add direct address byte to accumulator | `ADD A, 50H` |
| `ADD A, #data` | Add immediate data to accumulator | `ADD A, #25H` |
| `ADDC A, Rn` | Add with carry flag | `ADDC A, R5` |
| `SUBB A, Rn` | Subtract register from accumulator with borrow | `SUBB A, R6` |
| `SUBB A, #data` | Subtract immediate data from accumulator with borrow | `SUBB A, #10H` |
| `INC A` | Increment accumulator by 1 | `INC A` |
| `INC Rn` | Increment register by 1 | `INC R7` |
| `INC direct` | Increment data at direct address by 1 | `INC 60H` |
| `DEC A` | Decrement accumulator by 1 | `DEC A` |
| `DEC Rn` | Decrement register by 1 | `DEC R1` |
| `DEC direct` | Decrement data at direct address by 1 | `DEC 70H` |
| `DA A` | Decimal adjust accumulator after BCD addition | `DA A` |
| `MUL AB` | Multiply A and B registers (16-bit result in A and B) | `MUL AB` |
| `DIV AB` | Divide A by B (Quotient in A, remainder in B) | `DIV AB` |

---

## 8. Jump and Call Opcodes

Jump and call instructions control the flow of a program. They allow the program to make decisions and reuse code by calling subroutines.

### *Jump Instructions (JMP)*

1. **Unconditional Jumps:**
   - `SJMP label` (Short jump within ±127 bytes)
   - `LJMP address` (Long jump to any 16-bit address)
   - `AJMP address` (Absolute jump within 2 KB page)
2. **Conditional Jumps:**
   - `JZ label` (Jump if accumulator is zero)
   - `JNZ label` (Jump if accumulator is not zero)
   - `JC label` (Jump if carry flag is set)
   - `JNC label` (Jump if carry flag is not set)
   - `JB bit, label` (Jump if specified bit is set)
   - `JNB bit, label` (Jump if specified bit is not set)
   - `JBC bit, label` (Jump if bit is set and then clear it)
3. **Looping Instruction:**
   - `DJNZ Rn, label` (Decrement register and jump if not zero)
   - `DJNZ direct, label` (Decrement direct address and jump if not zero)

### *Call Instructions (CALL)*

1. **LCALL (Long Call):**
   - `LCALL address` (Call subroutine at any 16-bit address)
   - Saves the return address onto the stack and jumps to the subroutine.
2. **ACALL (Absolute Call):**
   - `ACALL address` (Call subroutine within 2 KB page)

3. **RET (Return):**
    - `RET` (Return from subroutine)
    - Pops the return address from the stack and resumes execution.
4. **RETI (Return from Interrupt):**
    - `RETI` (Return from interrupt service routine)

## Example: Subroutine Using CALL and RET

```
ORG 0000H
MOV A, #10H
LCALL SUBROUTINE
SJMP END

SUBROUTINE:
    ADD A, #20H
    RET

END:
    END
```

- The main program calls the `SUBROUTINE`, which adds 20H to the accumulator and returns.

**Unit III 8051 Microcontroller Design**

Microcontroller Specification – Design – Testing – Timing Subroutines – Lookup Tables for 8051 – Serial Data Transmission

---

## 8051 Microcontroller Design

## 1. Microcontroller Specification

The **8051 microcontroller** is an 8-bit microcontroller developed by Intel, widely used in embedded systems for its simplicity and efficiency. Key specifications include:

- **Processor:** 8-bit CPU
- **Clock Frequency:** 12 MHz (Standard, can be increased using external oscillators)
- **Program Memory (ROM):** 4 KB (On-chip, can be expanded externally)
- **Data Memory (RAM):** 128 bytes (Internal, expandable up to 64 KB externally)
- **I/O Ports:** 4 ports (P0, P1, P2, P3), each with 8 pins
- **Timers/Counters:** 2 (Timer 0, Timer 1)
- **Interrupts:** 5 (2 external, 3 internal)
- **Serial Communication:** UART (Universal Asynchronous Receiver-Transmitter)

---

## 2. Design

Designing an 8051-based system involves the following steps:

### *A. Selecting Microcontroller and Components*

- Choose the appropriate 8051 variant (AT89C51, AT89S52, etc.) based on the application.
- Include components like crystal oscillators, capacitors, reset circuits, and power supply units.

### *B. Circuit Design*

1. **Power Supply:**
    - Typically 5V DC is used.
2. **Clock Oscillator:**
    - An external crystal oscillator (usually 11.0592 MHz or 12 MHz) ensures timing accuracy.
3. **Reset Circuit:**
    - A reset pin (RST) connected with a resistor and capacitor ensures the system resets on power-up.
4. **Input/Output Interfaces:**
    - Use GPIO pins for sensors, actuators, LEDs, and displays.
5. **Memory Expansion:**
    - Connect external RAM or ROM if required.
6. **Serial Communication Interface:**
    - Connect RS-232 or USB interfaces using MAX232 IC.

### *C. Programming*

- Write programs using Assembly or Embedded C.
- Use Keil uVision IDE or MPLAB for coding and simulation.

### *D. PCB Layout*

- Design PCB using software like Proteus, Eagle, or Altium.

---

## 3. Testing

Testing ensures that the designed system works correctly and meets performance requirements. Steps include:

1. **Hardware Testing:**
    - Check power supply, clock frequency, reset circuit, and GPIO connections.
2. **Software Testing:**
    - Verify the logic of programs, instruction execution, and data flow.
3. **Functional Testing:**
    - Ensure all inputs and outputs function as expected.
4. **Performance Testing:**
    - Measure timing accuracy, interrupt handling, and data transmission rates.
5. **Debugging:**
    - Use tools like simulators and logic analyzers to identify and fix issues.

---

## 4. Timing Subroutines

Timing subroutines generate precise delays using timers in the 8051. Timer 0 and Timer 1 are 16-bit timers, and their modes determine how they count.

### *Timer Modes:*

- **Mode 0:** 13-bit timer (counts 0 to 8191)
- **Mode 1:** 16-bit timer (counts 0 to 65535)
- **Mode 2:** 8-bit auto-reload timer
- **Mode 3:** Split timer mode (Timer 0 divided into two 8-bit timers)

### *Delay Subroutine Example (Using Timer 0 in Mode 1):*

```
ORG 0000H
MOV TMOD, #01H  ; Timer 0, Mode 1
MOV TH0, #0FCH  ; Load initial values
MOV TL0, #018H
SETB TR0         ; Start Timer 0

HERE: JNB TF0, HERE ; Wait until overflow
CLR TR0           ; Stop Timer
CLR TF0           ; Clear overflow flag
RET
END
```

- This generates approximately a 1 ms delay using a 12 MHz clock.

---

## 5. Lookup Tables for 8051

Lookup tables store predefined data, which the microcontroller can quickly access using indexed addressing. They are commonly used for character patterns, waveforms, or mathematical functions like sine or logarithms.

### *Example: Displaying Characters Using Lookup Tables*

```
ORG 0000H
MOV DPTR, #TABLE  ; Load table address into DPTR
MOV A, #02H       ; Load index (character 2)
MOVC A, @A+DPTR   ; Retrieve character from table
MOV P1, A         ; Output character to port 1
SJMP $

TABLE: DB 41H, 42H, 43H, 44H  ; ASCII for A, B, C, D
END
```

- This code retrieves the ASCII code of the character "C" from the table and outputs it on Port 1.

---

## 6. Serial Data Transmission

The 8051 microcontroller uses UART for serial communication. It transmits and receives data in asynchronous mode using TXD (P3.1) and RXD (P3.0) pins.

### *Configuration Steps:*

1. Configure the Timer 1 in Mode 2 (8-bit auto-reload) for baud rate generation.
2. Set the SM0 and SM1 bits of the SCON register to select mode 1 (8-bit UART).
3. Enable reception by setting REN bit.
4. Use SBUF to transmit and receive data.
5. Monitor TI and RI flags for transmission and reception completion.

### *Baud Rate Calculation:*

Baud Rate=Oscillator Frequency12×(256−TH1)Baud\ Rate = \frac{Oscillator\ Frequency}{12 \times (256 - TH1)}

### *Example: Sending and Receiving Data*

```
ORG 0000H
MOV TMOD, #20H  ; Timer 1, Mode 2 (Auto-reload)
MOV TH1, #0FDH  ; Baud rate 9600 (11.0592 MHz)
MOV SCON, #50H  ; Mode 1, 8-bit UART, REN enabled
SETB TR1        ; Start Timer 1

MOV SBUF, #41H  ; Send ASCII 'A'
HERE: JNB TI, HERE
CLR TI          ; Clear transmit flag

HERE1: JNB RI, HERE1
MOV A, SBUF     ; Receive data into accumulator
CLR RI          ; Clear receive flag
SJMP $

END
```

- This program transmits the character 'A' and waits to receive a character.

---

## Summary

- **Microcontroller Specification:** Defines 8051's key features and architecture.
- **Design:** Includes hardware selection, circuit design, and programming.
- **Testing:** Ensures the system works correctly through hardware and software tests.
- **Timing Subroutines:** Provide precise delays using timers.
- **Lookup Tables:** Store predefined data for quick access.
- **Serial Data Transmission:** Enables communication using UART.

---

# Unit IV Applications

Keyboards – Displays – Pulse Measurement – D/A and A/D Conversions – Multiple Interrupts

---

## 8051 Microcontroller Applications

The **8051 microcontroller** is extensively used in embedded systems due to its versatility and efficiency. Here are detailed explanations of its key applications:

---

## 1. Keyboards

**Application:** Interface matrix keyboards with the 8051 for data entry in systems like calculators, security systems, and ATMs.

### *Working Principle:*

- A matrix keyboard has rows and columns connected to I/O ports.
- The microcontroller scans each row and column to detect a pressed key.

### *Example: 4x4 Matrix Keyboard Interface*

- Rows (R0-R3) connected to Port 1 outputs.
- Columns (C0-C3) connected to Port 2 inputs.

### *Algorithm:*

1. Initialize ports.
2. Output logic LOW on each row sequentially.
3. Read column inputs to detect which key is pressed.
4. Decode the key based on the row and column combination.

```
ORG 0000H
MOV P1, #0FFH   ; Configure Port 1 as input (Rows)
MOV P2, #0FFH   ; Configure Port 2 as input (Columns)

CHECK: MOV P1, #0FEH  ; Set Row 0 LOW
NOP
MOV A, P2
CJNE A, #0FFH, FOUND

MOV P1, #0FDH  ; Set Row 1 LOW
NOP
MOV A, P2
CJNE A, #0FFH, FOUND

SJMP CHECK

FOUND: MOV P3, A  ; Display pressed key on Port 3
SJMP CHECK
END
```

- This program scans each row and checks the columns to detect the pressed key.

---

## 2. Displays

**Application:** Interface 7-segment displays, LCDs, and LEDs for visual output.

### *A. 7-Segment Display*

- The display is driven by GPIO ports using BCD codes.
- Common cathode or common anode configurations are used.

```
ORG 0000H
MOV P1, #3FH  ; Display '0' on 7-seg
SJMP $
END
```

- This example sends the hexadecimal code `3F` to display '0' on a common cathode 7-seg display.

---

### *B. 16x2 LCD Display*

- Controlled using 8-bit or 4-bit data mode.
- Commands and data are sent through GPIO pins.

### Pin Configuration:

- RS (Register Select), RW (Read/Write), and E (Enable) connected to control pins.
- D0-D7 connected to data pins.

### Command Examples:

- `0x38` - Initialize in 8-bit mode
- `0x0C` - Display ON, Cursor OFF
- `0x01` - Clear Display

```
ORG 0000H
MOV P2, #38H   ; Initialize LCD
CALL COMMAND
MOV P2, #0CH   ; Display ON, Cursor OFF
CALL COMMAND
MOV P2, #41H   ; Display 'A'
CALL DATA
SJMP $

COMMAND: CLR P3.0
SETB P3.1
ACALL ENABLE
RET

DATA: SETB P3.0
```

```
CLR P3.1
ACALL ENABLE
RET

ENABLE: SETB P3.2
ACALL DELAY
CLR P3.2
ACALL DELAY
RET
END
```

- This code initializes an LCD and displays the character 'A'.

---

## 3. Pulse Measurement

**Application:** Measure pulse width and frequency using Timer 0 and Timer 1.

### Working Principle:

- Use an external interrupt (INT0 or INT1) to start and stop the timer.
- Count pulses using the timer's overflow events.

```
ORG 0000H
MOV TMOD, #01H  ; Timer 0, Mode 1
SETB IT0        ; Configure INT0 as falling edge trigger
SETB EX0        ; Enable External Interrupt 0
SETB EA         ; Enable global interrupts

START: SETB TR0  ; Start Timer
SJMP START

ORG 0003H       ; INT0 Interrupt Vector
CLR TR0         ; Stop Timer
MOV A, TL0      ; Store low byte of count
MOV B, TH0      ; Store high byte of count
RET
END
```

- This code measures the duration between two falling edges using Timer 0.

---

## 4. D/A and A/D Conversions

### *A. Digital-to-Analog (D/A) Conversion*

**Application:** Generate analog signals for motor control, audio signals, and sensor simulation.

- DAC0808 IC is used with the 8051.
- Digital input (8 bits) is converted into an analog voltage.

```
ORG 0000H
MOV P1, #0FFH  ; Maximum digital value
```

```
SJMP $
END
```

- Sending `0xFF` produces maximum output voltage from the DAC.

---

### *B. Analog-to-Digital (A/D) Conversion*

**Application:** Interface sensors like temperature, pressure, and light intensity with the 8051.

- ADC0804 IC is commonly used.
- The microcontroller controls the start conversion (WR pin) and reads the digital output.

```
ORG 0000H
CLR P3.0       ; Start conversion (WR = 0)
SETB P3.0
WAIT: JB P3.1, WAIT  ; Wait until conversion complete (INTR = 0)
MOV A, P1       ; Read digital value
SJMP $
END
```

- This code starts the ADC conversion and reads the digital output from Port 1.

---

## 5. Multiple Interrupts

**Application:** Handle multiple simultaneous events like button presses, timers, and serial data reception.

### Interrupt Sources:

1. **External Interrupt 0 (INT0)** - Priority 1
2. **Timer 0 Overflow** - Priority 2
3. **External Interrupt 1 (INT1)** - Priority 3
4. **Timer 1 Overflow** - Priority 4
5. **Serial Communication (RI/TI)** - Priority 5

### Interrupt Vector Table:

- `0003H` - INT0
- `000BH` - Timer 0
- `0013H` - INT1
- `001BH` - Timer 1
- `0023H` - Serial Communication

```
ORG 0000H
SETB EX0   ; Enable External Interrupt 0
SETB EX1   ; Enable External Interrupt 1
SETB ET0   ; Enable Timer 0 Interrupt
SETB ET1   ; Enable Timer 1 Interrupt
SETB ES    ; Enable Serial Interrupt
```

```
SETB EA    ; Enable Global Interrupt

SJMP $

ORG 0003H
MOV P1, #01H  ; INT0 Interrupt
RET

ORG 000BH
MOV P1, #02H  ; Timer 0 Overflow
RET

ORG 0013H
MOV P1, #03H  ; INT1 Interrupt
RET

ORG 001BH
MOV P1, #04H  ; Timer 1 Overflow
RET

ORG 0023H
MOV P1, #05H  ; Serial Interrupt
RET
END
```

- This program handles multiple interrupts and outputs a different value on Port 1 for each interrupt source.

---

## Summary

- **Keyboards:** Interface matrix keyboards using GPIO ports and scanning techniques.
- **Displays:** Drive 7-segment and LCD displays using BCD codes and control signals.
- **Pulse Measurement:** Use timers and external interrupts to measure pulse width and frequency.
- **D/A and A/D Conversions:** Interface DAC0808 and ADC0804 for analog signal generation and measurement.
- **Multiple Interrupts:** Manage simultaneous events using external and internal interrupts with priority control.

---

# 8051 Microcontroller: Serial Data Communication

Serial data communication refers to transmitting data one bit at a time over a single communication line, which reduces wiring and simplifies long-distance communication. Unlike parallel communication (which sends multiple bits simultaneously), serial communication is more efficient for embedded systems. In the 8051 microcontroller, serial communication is handled by the Serial Control (SCON) register and Timer 1 (used for baud rate generation). Data is sent and received using the SBUF register, and communication can be either synchronous or asynchronous. The TXD (P3.1) pin is used for transmission, and RXD (P3.0) is used for receiving data. Serial interrupts can also be enabled for efficient data handling.

---

## 1. Network Configurations

Serial communication in embedded systems can follow different network configurations depending on the application. The key configurations are:

### *A. Point-to-Point Communication*

- **Description:** Direct communication between two devices using a dedicated connection.
- **Example:** Connecting an 8051 microcontroller to a computer using RS-232.
- **Advantages:** Simple, fast, and reliable for short distances.

---

### *B. Multi-Drop Communication (One-to-Many)*

- **Description:** One device acts as the master, and multiple devices act as slaves, all connected on a single communication line.
- **Example:** 8051 communicating with multiple sensors using RS-485.
- **Advantages:** Cost-effective for long distances with multiple devices.

---

### *C. Full-Duplex and Half-Duplex Communication*

- **Full-Duplex:** Simultaneous transmission and reception of data using two separate lines (TXD and RXD).

- **Half-Duplex:** Transmission and reception occur alternately over a single line.

---

### *D. Synchronous and Asynchronous Communication*

- **Synchronous:** Transmitter and receiver share a common clock signal for synchronized data transfer.
- **Asynchronous:** Each byte is sent with start and stop bits, and no common clock is needed (used in 8051).

---

## 2. 8051 Data Communication Modes

The 8051's **UART module** allows asynchronous serial communication using two pins:

- **TXD (P3.1):** Transmits data
- **RXD (P3.0):** Receives data

The communication is controlled using the **SCON (Serial Control)** register.

---

### *SCON Register Format (Bit Functions):*

| Bit | Name | Description |
|---|---|---|
| 7 | SM0 | Serial Mode Control Bit 0 |
| 6 | SM1 | Serial Mode Control Bit 1 |
| 5 | SM2 | Multiprocessor Communication Enable |
| 4 | REN | Receive Enable |
| 3 | TB8 | Transmit Bit 8 (9th bit in Mode 2 & 3) |
| 2 | RB8 | Receive Bit 8 (9th bit in Mode 2 & 3) |
| 1 | TI | Transmit Interrupt Flag |
| 0 | RI | Receive Interrupt Flag |

---

### *Data Communication Modes:*

The 8051 supports **4 modes of serial communication:**

---

## Mode 0: 8-bit Shift Register (Synchronous Mode)

- Data is transmitted and received bit-by-bit using the TXD pin.
- A clock signal is output through the RXD pin.
- Transmission speed: **1/12th of the oscillator frequency**.

**Use Case:** Simple synchronous communication with peripherals.

---

## Mode 1: 8-bit UART (Asynchronous Mode)

- Data is transmitted with **1 start bit, 8 data bits, and 1 stop bit**.
- Baud rate is variable and determined by **Timer 1**.
- Commonly used for standard serial communication (RS-232).

### Baud Rate Calculation:

Baud Rate=Oscillator Frequency12×(256−TH1)Baud\ Rate = \frac{Oscillator\ Frequency}{12 \times (256 - TH1)}

For a **11.0592 MHz** crystal and **TH1 = 0xFD**, the baud rate is **9600 bps**.

---

## Mode 2: 9-bit UART (Fixed Baud Rate)

- Data frame consists of **1 start bit, 8 data bits, and 1 stop bit**.
- 9th bit (TB8/RB8) can be used for parity or control information.
- Baud rate is fixed at **1/32 or 1/64 of the oscillator frequency**.

**Use Case:** Multiprocessor communication.

---

## Mode 3: 9-bit UART (Variable Baud Rate)

- Similar to **Mode 2** but with a **variable baud rate** determined by **Timer 1**.
- Supports long-distance communication and higher speeds.

---

## Baud Rate Generation using Timer 1

- Timer 1 is configured in **Mode 2 (8-bit auto-reload)** to generate the desired baud rate.
- **TH1** register is loaded with a value that sets the baud rate.

---

## Serial Communication Example (Mode 1, 9600 Baud, Full Duplex)

```
ORG 0000H
MOV TMOD, #20H  ; Timer 1 in Mode 2 (Auto-reload)
MOV TH1, #0FDH  ; Baud rate 9600 (11.0592 MHz)
MOV SCON, #50H  ; Mode 1, REN enabled
SETB TR1        ; Start Timer 1

MOV SBUF, #41H  ; Transmit 'A'
WAIT_TX: JNB TI, WAIT_TX
```

```
CLR TI

WAIT_RX: JNB RI, WAIT_RX
MOV A, SBUF     ; Receive data into Accumulator
CLR RI
SJMP $

END
```

- This program transmits the character **'A'** and waits to receive data.

---

## Multiprocessor Communication (Using SM2 Bit)

- **SM2 bit** enables **Multiprocessor Communication Mode** in Modes 2 and 3.
- When **SM2 = 1**, the receiver ignores frames with the 9th bit = 0, allowing communication only with specific devices.

---

## Interrupt-Driven Serial Communication

- The **TI (Transmit Interrupt)** flag is set when transmission is complete.
- The **RI (Receive Interrupt)** flag is set when a byte is received.
- Using interrupts allows non-blocking communication, improving efficiency.

### Enable Interrupts:

```
SETB ES  ; Enable Serial Interrupt
SETB EA  ; Enable Global Interrupt
```

---

## Summary

- **Network Configurations:** Point-to-Point, Multi-Drop, Full-Duplex, and Half-Duplex.
- **Data Communication Modes:**
  - **Mode 0:** 8-bit synchronous shift register.
  - **Mode 1:** 8-bit UART with variable baud rate.
  - **Mode 2:** 9-bit UART with fixed baud rate.
  - **Mode 3:** 9-bit UART with variable baud rate.
- **Baud Rate Generation:** Controlled using **Timer 1** in Mode 2.
- **Interrupts:** TI and RI flags enable efficient, non-blocking communication.

---