Module 7: Microcontrollers: The 8051 System

This module introduces the fascinating world of microcontrollers, focusing specifically on the ubiquitous and foundational 8051 family. We will begin by clearly delineating the Introduction to Microcontrollers, highlighting their fundamental distinctions from general-purpose microprocessors and firmly positioning them within the broader context of embedded systems. Subsequently, we will undertake a thorough exploration of The 8051 Microcontroller Architecture, meticulously examining its internal organization, comprehensive memory map (distinguishing between program and data memory), and its crucial set of internal registers. Our journey will then progress to an in-depth analysis of the 8051 Instruction Set, categorizing and explaining its diverse commands for data transfer, arithmetic operations, logical manipulations, Boolean processing, and intricate program control. Following this, we will dive into the functionality of the 8051 On-Chip Peripherals, dedicating significant attention to its versatile Timers/Counters, the indispensable Serial Port (UART) for communication, its robust Interrupt system for event handling, and its flexible I/O Ports for external interfacing. Finally, the module culminates in practical insights into 8051 Programming in Assembly and C, providing concrete examples to illustrate effective peripheral control and system interaction.

7.1 Introduction to Microcontrollers: Distinction from Microprocessors, Embedded Systems Context

To truly appreciate the significance of the 8051 and its place in modern technology, it's essential to understand what a microcontroller is, how it differs from a microprocessor, and why it's the heart of countless embedded systems.

7.1.1 What is a Microcontroller (MCU)? A microcontroller (MCU) is a compact, highly integrated computing device designed to perform specific control functions within a larger system. Unlike a general-purpose computer that can run various software applications, a microcontroller is typically dedicated to one specific task or a set of closely related tasks. It is essentially a "computer on a chip," containing not just a Central Processing Unit (CPU) but also essential support components that are typically external to a microprocessor.

Key Components Integrated within a Single MCU Chip:

1. **Central Processing Unit (CPU): The brain of the MCU, responsible for executing instructions and performing arithmetic and logical operations.**
2. **Program Memory (ROM/Flash): Stores the program (firmware) that the MCU executes. This memory is typically non-volatile, meaning it retains its contents even when power is off.**
3. **Data Memory (RAM): Used for temporary storage of data during program execution, such as variables, stack data, and intermediate results. This memory is volatile.**

4. **I/O Ports: Digital pins that allow the MCU to interact with the outside world by reading inputs (e.g., from sensors, switches) and controlling outputs (e.g., LEDs, motors, relays).**
5. **Timers/Counters: Specialized circuits used for precise timing, generating delays, counting external events, or producing waveforms (e.g., Pulse Width Modulation - PWM).**
6. **Serial Communication Interfaces: Dedicated hardware for transmitting and receiving data serially, such as UART (Universal Asynchronous Receiver/Transmitter) for RS-232, SPI (Serial Peripheral Interface), I2C (Inter-Integrated Circuit).**
7. **Analog-to-Digital Converters (ADCs) / Digital-to-Analog Converters (DACs): (Common in many modern MCUs, though not all basic 8051 variants). ADCs convert analog sensor signals into digital values, while DACs convert digital values into analog outputs.**
8. **Watchdog Timer: A safety feature that resets the MCU if the program gets stuck in an infinite loop or encounters an unhandled error, ensuring system reliability.**
9. **Clock Circuit: Provides the timing pulses for all operations.**

7.1.2 Distinction from Microprocessors (MPU):

While both microcontrollers and microprocessors contain a CPU, their architectural philosophy and intended applications are fundamentally different.

Feature	Microprocessor (MPU)	Microcontroller (MCU)
Primary Role	General-purpose computing, high-performance tasks.	Dedicated control tasks, specific embedded applications.
Components	CPU only (or CPU + Cache). Requires external memory, I/O, etc.	CPU + On-chip RAM, ROM/Flash, I/O ports, Timers, Serial Ports, etc.
External Components	Requires significant external support chips (memory, I/O controllers, clock generator) to form a functional computer system.	Designed to be self-contained. Requires minimal external components (e.g., crystal oscillator, power supply, few discrete components).
Integration	Low integration (CPU core is main component).	High integration (many peripherals on one chip).
Memory	Large external memory space (GBs or TBs), often with sophisticated memory management units (MMUs).	Smaller on-chip memory (KBs or MBs), specific fixed address maps.

Speed/Performance	**Typically higher clock speeds (GHz), optimized for raw processing power, complex OS support.**	**Lower to moderate clock speeds (MHz to hundreds of MHz), optimized for real-time control.**
Power Consumption	**Generally higher power consumption due to external components and high clock speeds.**	**Generally lower power consumption, optimized for battery-powered or low-power applications.**
Cost	**CPU chip itself can be expensive; total system cost higher due to external components.**	**MCU chip is relatively inexpensive; total system cost is lower.**
Operating System	**Designed to run complex operating systems (Windows, Linux, macOS).**	**Often runs bare-metal code, simple real-time operating systems (RTOS), or firmware.**
Examples	**Intel Core i7, AMD Ryzen, ARM Cortex-A series.**	**8051, PIC, Atmel AVR, ARM Cortex-M series (e.g., STM32, ESP32).**

Export to Sheets

Analogy: A microprocessor is like a powerful, specialized engine that needs a full car chassis, transmission, wheels, and fuel tank to function as a vehicle. A microcontroller is like a complete, compact motorized scooter that has all its essential components integrated to perform a specific function of transport.

7.1.3 Embedded Systems Context:

Microcontrollers are the cornerstone of embedded systems. An embedded system is a specialized computer system designed to perform one or a few dedicated functions, often with real-time computing constraints. It is "embedded" as part of a complete device, including hardware and mechanical parts.

Key Characteristics of Embedded Systems:

1. **Dedicated Functionality: Performs a specific task rather than being a general-purpose computer.**
2. **Real-Time Constraints: Often must respond to external events within strict time limits (e.g., controlling a motor, reading a sensor rapidly).**
3. **Resource Constraints: Limited memory, processing power, and energy consumption, leading to optimized code and hardware.**
4. **Reliability and Stability: Expected to operate continuously and reliably for long periods without human intervention.**
5. **Cost-Effectiveness: Often designed for mass production, so cost per unit is a major consideration.**

Where Microcontrollers (and thus Embedded Systems) are Found:

- **Consumer Electronics: Remote controls, washing machines, microwaves, digital cameras, smart TVs, home security systems.**
- **Automotive: Engine control units (ECUs), anti-lock braking systems (ABS), airbag systems, infotainment systems.**
- **Industrial Control: PLCs (Programmable Logic Controllers), robotic arms, factory automation equipment.**
- **Medical Devices: Insulin pumps, heart rate monitors, diagnostic equipment.**
- **IoT (Internet of Things) Devices: Smart home sensors, wearables, smart lighting.**
- **Communication Devices: Routers, modems, mobile phones (baseband processors).**
- **Toys and Gadgets: Drones, electronic toys.**

The 8051, despite its age, remains highly relevant due to its clear, simple architecture, extensive documentation, and widespread use as an educational tool and in many legacy industrial applications. Understanding the 8051 provides a solid foundation for grasping the principles that underpin more complex modern microcontrollers.

7.2 The 8051 Microcontroller Architecture: Internal Organization, Memory Map (Program and Data Memory), Registers

The 8051, a member of the MCS-51 family, is an 8-bit microcontroller introduced by Intel in 1980. Its success lies in its relatively simple yet powerful architecture, making it easy to understand and program for a wide range of embedded applications.

7.2.1 Internal Organization: The Blocks on the Chip

The 8051 is a single-chip device containing all the essential components of a computer system. Here's a breakdown of its key functional blocks:

1. **Central Processing Unit (CPU):**
    - **The heart of the 8051, an 8-bit processor. This means it can process 8 bits of data at a time.**
    - **Includes an Arithmetic Logic Unit (ALU) for arithmetic and logical operations, control unit for instruction decoding and execution, and various registers.**
2. **On-Chip Program Memory (ROM/Flash):**
    - **Typically 4textKB for the original 8051 (like the 8051/8052), used to store the user's program code (firmware).**
    - **It is Non-Volatile Memory (NVM), meaning its contents are retained even when power is off. Modern derivatives offer larger sizes (e.g., 8textKB, 16textKB, 64textKB or more).**
3. **On-Chip Data Memory (RAM):**
    - **128textbytes for the original 8051, used for temporary data storage (variables, stack, intermediate results).**
    - **It is Volatile Memory, meaning its contents are lost when power is off.**

    - **This 128textbytes is partitioned into various areas: Register Banks, Bit Addressable Area, and General Purpose RAM. (More detail in memory map).**
4. **I/O Ports (Port 0, Port 1, Port 2, Port 3):**
    - **Four 8-bit bidirectional I/O ports. These are digital pins that can be configured as inputs or outputs to interface with external devices.**
    - **Each port has its own internal latch, output driver, and input buffer.**
5. **Timers/Counters (Timer 0, Timer 1):**
    - **Two 16-bit timers/counters (Timer 0 and Timer 1).**
    - **Can be used to generate time delays, measure pulse widths, count external events, or generate baud rates for the serial port.**
    - **The 8052 variant adds a third timer (Timer 2).**
6. **Serial Port (UART):**
    - **A full-duplex Universal Asynchronous Receiver/Transmitter (UART).**
    - **Allows the 8051 to communicate serially with other devices (e.g., PCs, other microcontrollers) using protocols like RS-232.**
7. **Interrupt Control Unit:**
    - **Manages the 8051's interrupt system. The original 8051 has 5 interrupt sources (2 external, 2 timer, 1 serial).**
    - **Allows the MCU to respond quickly to asynchronous events from peripherals or external signals.**
8. **Clock Circuit:**
    - **Requires an external crystal oscillator to provide the clock pulses that synchronize all internal operations of the 8051. The operating frequency is typically in MHz (e.g., 11.0592textMHz, 12textMHz).**

7.2.2 Memory Map: Organizing the Digital Landscape

The 8051 has a Harvard architecture variant, meaning it has separate address spaces for Program Memory and Data Memory. This allows simultaneous fetching of instructions and accessing data, improving performance.

7.2.2.1 Program Memory (Code Memory):

- **Address Space: Up to 64textKB (from 0000textH to FFFFtextH).**
- **Storage: Stores the machine code instructions of the program, along with any constant data (e.g., lookup tables) that are part of the program.**
- **On-Chip ROM/Flash: The original 8051 has 4textKB of on-chip ROM, mapped from 0000textH to 0FFFtextH.**
- **External Program Memory: If the program size exceeds the on-chip memory, or if no on-chip ROM is available (e.g., 8031 variant), external program memory chips can be connected. These are typically accessed by enabling the overlinetextEA (External Access) pin.**
- **Access: Program memory is read-only during normal execution. Instructions are fetched from here using the Program Counter (PC).**

7.2.2.2 Data Memory: The 8051 distinguishes between Internal Data Memory and External Data Memory.

a) Internal Data Memory (RAM):

- **Address Space: 128textbytes (from 00textH to 7FtextH) in the original 8051. Some enhanced versions (like 8052) have 256textbytes.**
- **Volatile: Contents are lost on power off.**
- **Structure of the 128 Bytes (for original 8051):**
    - **Register Banks (00H to 1FH - 32 bytes):**
        - **Divided into 4 banks (Bank 0, Bank 1, Bank 2, Bank 3), each containing 8 working registers (R0 to R7).**
        - **Only one bank is active at a time, selected by bits RS1 and RS0 in the Program Status Word (PSW) register. This provides fast access to 8 registers at any given time, context switching, or for subroutine calls.**
        - **Example: If Bank 0 is selected, R0 corresponds to internal RAM address 00textH, R1 to 01textH, etc. If Bank 1 is selected, R0 then corresponds to 08textH, R1 to 09textH, etc.**
    - **Bit-Addressable RAM (20H to 2FH - 16 bytes):**
        - **This area allows individual bits to be addressed and manipulated directly. There are 16textbytestimes8textbits/byte=128textindividuallyaddressablebits.**
        - **Addresses are 00textH to 7FtextH for bits.**
        - **Extremely useful for setting/clearing flags, controlling individual I/O pins, or for Boolean logic operations.**
        - **Formula: Bit address = (Byte address - 20textH)times8+textbitposition(0−7).**
        - **Numerical Example: Byte address 20textH contains bits 00textH to 07textH. Byte address 21textH contains bits 08textH to 0FtextH, and so on, up to 2FtextH containing bits 78textH to 7FtextH. To access bit 0 of byte 20textH, its bit address is 00textH. To access bit 7 of byte 2FtextH, its bit address is 7FtextH.**
    - **General Purpose RAM (30H to 7FH - 80 bytes):**
        - **Used for general data storage, local variables, or as a software stack.**
        - **Accessed by byte address.**
    - **Special Function Registers (SFRs) (80H to FFFFH, conceptually):**
        - **Not part of the 128textbytes of general RAM. These are registers that control or monitor the 8051's on-chip peripherals and CPU functions. They reside in a separate address space (though often accessed as if they were RAM locations above 7FtextH).**
        - **Some SFRs are also bit-addressable.**

b) External Data Memory:

- **Address Space: Up to 64textKB (from 0000textH to FFFFtextH).**
- **Connection: Accessed via Port 0 (multiplexed data/low-order address) and Port 2 (high-order address).**

- **Purpose: Used when the internal RAM is insufficient for large data buffers or external peripheral registers.**
- **Instructions: Accessed using `MOVX` (Move External) instructions.**

7.2.3 Registers: The CPU's Workspace

The 8051's CPU includes several dedicated registers that are crucial for its operation.

1. **Accumulator (A or ACC):**
    - **An 8-bit register, the most versatile in the 8051.**
    - **All arithmetic operations (addition, subtraction, multiplication, division) involve the Accumulator.**
    - **Many data transfer and logical operations also use the Accumulator as a source or destination.**
2. **B Register (B):**
    - **An 8-bit register, used primarily in multiplication and division operations.**
        - **For `MUL AB` (A times B), the 16-bit result is stored in B (high byte) and A (low byte).**
        - **For `DIV AB` (A divided by B), the quotient is stored in A, and the remainder in B.**
    - **Can also be used as a general-purpose scratchpad register.**
3. **Program Status Word (PSW):**
    - **An 8-bit register containing various flag bits that reflect the status of CPU operations or control certain functions.**
    - **Bits:**
        - **P (PSW.0 - Parity Flag): Set to 1 if the Accumulator contains an odd number of 1s (odd parity), or 0 if an even number of 1s (even parity).**
        - **--- (PSW.1 - User Defined Flag): Available for general programming use.**
        - **OV (PSW.2 - Overflow Flag): Set to 1 if an arithmetic operation (signed addition/subtraction) results in an overflow, meaning the result exceeds the register's signed capacity.**
        - **RS0 (PSW.3 - Register Bank Select Bit 0)**
        - **RS1 (PSW.4 - Register Bank Select Bit 1)**
            - **These two bits (`RS1 RS0`) select the active register bank (00=Bank 0, 01=Bank 1, 10=Bank 2, 11=Bank 3).**
        - **F0 (PSW.5 - User Defined Flag): Another general-purpose flag.**
        - **AC (PSW.6 - Auxiliary Carry Flag): Set if there's a carry out from bit 3 to bit 4 during an 8-bit addition (or borrow into bit 3). Primarily used for BCD arithmetic.**
        - **CY (PSW.7 - Carry Flag): Set if there's a carry out from the MSB (bit 7) during an 8-bit addition, or a borrow into the MSB during subtraction. Also used for bit operations.**
4. **Data Pointer (DPTR):**
    - **A 16-bit register composed of two 8-bit registers: DPH (Data Pointer High byte) and DPL (Data Pointer Low byte).**

    - Used to hold a 16-bit address for accessing external program memory or external data memory.
    - Can also be used as a 16-bit general-purpose register.
5. Program Counter (PC):
    - A 16-bit register that holds the address of the *next* instruction to be fetched from program memory.
    - Automatically increments after each instruction fetch.
    - Modified by jump, call, and return instructions.
6. Stack Pointer (SP):
    - An 8-bit register that holds the address of the top of the stack within the internal RAM.
    - Initially set to 07textH after a reset. The stack grows upwards, so the first byte pushed will be at 08textH.
    - `PUSH` instruction increments SP then stores data at SP's new address.
    - `POP` instruction retrieves data from SP's current address then decrements SP.

Relationship between Registers and Memory:

- **ACC, B, PSW, SP: These are all Special Function Registers (SFRs) that are part of the CPU's direct access architecture and are also mapped into the SFR address space.**
- **R0-R7: These 8 registers are part of the internal RAM and their actual physical addresses depend on the selected register bank.**
- **DPTR, PC: These are 16-bit registers that are not directly mapped to SFR addresses for byte access, but their values can be loaded/stored using specific instructions.**

Understanding this internal organization and memory map is crucial for efficient 8051 programming, allowing developers to optimize code for speed, memory usage, and direct hardware control.

7.3 8051 Instruction Set: Data Transfer, Arithmetic, Logical, Boolean, and Program Control Instructions

The 8051's instruction set defines the complete list of commands that the CPU can understand and execute. It is designed to be efficient for control-oriented applications, featuring specialized instructions for bit manipulation and direct hardware access. Most instructions operate on 8-bit data.

Instruction Format: `MNEMONIC Destination, Source` (Operand order can vary).

7.3.1 Data Transfer Instructions (MOV Family): These instructions move data between registers, memory locations, and I/O ports. They do not affect flag bits.

- **Internal Data RAM Moves:**
    - `MOV A, Rn`**: Move content of register Rn (R0-R7) to Accumulator A.**

        - *Numerical Example:* `MOV A, R5` (If R5 contains 55textH, A will be 55textH).
    - `MOV Rn, A`: Move content of Accumulator A to register Rn.
    - `MOV A, direct_address`: Move content of internal RAM or SFR at `direct_address` to A.
        - *Numerical Example:* `MOV A, 30H` (If RAM location 30textH contains 12textH, A will be 12textH).
    - `MOV direct_address, A`: Move content of A to internal RAM or SFR at `direct_address`.
    - `MOV Rn, #data`: Move immediate `data` to register Rn.
        - *Numerical Example:* `MOV R0, #0AH` (R0 becomes 0AtextH).
    - `MOV A, @Ri`: Move content of internal RAM pointed to by Ri (R0 or R1) to A (indirect addressing).
        - *Numerical Example:* If R0 contains 40textH and RAM location 40textH contains F0textH, `MOV A, @R0` results in A being F0textH.
    - `MOV @Ri, A`: Move content of A to internal RAM pointed to by Ri.
- External Data RAM Moves (using DPTR):
    - `MOVX A, @DPTR`: Move byte from external data RAM pointed to by DPTR to A.
        - *Numerical Example:* If DPTR contains 1000textH and external RAM at 1000textH contains AAtextH, `MOVX A, @DPTR` results in A being AAtextH.
    - `MOVX @DPTR, A`: Move byte from A to external data RAM pointed to by DPTR.
    - `MOVX A, @Ri`: Move byte from external data RAM pointed to by R0/R1 (only lower 256textbytes of external memory).
    - `MOVX @Ri, A`: Move byte to external data RAM pointed to by R0/R1.
- Program Memory Moves (using DPTR for lookup tables):
    - `MOVC A, @A+DPTR`: Move byte from code memory (ROM) pointed to by (Accumulator + DPTR) to A. Used for reading lookup tables.
        - *Numerical Example:* If DPTR = 1000textH, A = 05textH, and location 1005textH in program memory contains C3textH, `MOVC A, @A+DPTR` results in A being C3textH.
    - `MOVC A, @A+PC`: Move byte from code memory pointed to by (Accumulator + PC) to A.
- Stack Operations:
    - `PUSH direct_address`: Increment SP, then store content of `direct_address` (internal RAM or SFR) onto stack.
    - `POP direct_address`: Retrieve content from stack (at SP), then decrement SP, and store into `direct_address`.

7.3.2 Arithmetic Instructions: These instructions perform addition, subtraction, multiplication, and division. They typically affect the Carry (CY), Auxiliary Carry (AC), and Overflow (OV) flags in PSW.

- `ADD A, Rn`: Add content of Rn to A.
    - *Numerical Example:* A=10textH, R0=05textH. `ADD A, R0` results in A=15textH.
- `ADD A, #data`: Add immediate `data` to A.
- `ADD A, direct_address`: Add content of `direct_address` to A.
- `ADD A, @Ri`: Add content of internal RAM pointed to by Ri to A.
- `ADDC A, Rn`: Add content of Rn to A with Carry flag. (A = A + Rn + CY)
    - *Numerical Example:* A=10textH, R0=F0textH, CY=1. `ADDC A, R0` results in A=01textH (since 10textH+F0textH+1textH=101textH, so 01textH goes to A, CY becomes 1).
- `SUBB A, Rn`: Subtract content of Rn from A with Borrow (A = A - Rn - CY).
- `INC operand`: Increment `operand` (A, Rn, direct_address, @Ri) by 1. Does not affect flags.
- `DEC operand`: Decrement `operand` by 1. Does not affect flags.
- `MUL AB`: Multiply A by B. The 16-bit result is stored in B (high byte) and A (low byte).
    - *Numerical Example:* A=05textH, B=04textH. `MUL AB` results in A=14textH and B=00textH (5times4=20, which is 14textH).
- `DIV AB`: Divide A by B. The quotient is stored in A, and the remainder in B.
    - *Numerical Example:* A=0AtextH (10 decimal), B=03textH (3 decimal). `DIV AB` results in A=03textH (quotient) and B=01textH (remainder).
- `DA A`: Decimal Adjust Accumulator. Used after binary addition to convert the result into a correct BCD number.
    - *Numerical Example:* A=19textH (binary 0001_1001_2), `ADD A, #01H` -> A=1AtextH. `DA A` -> A=20textH (since 19+1=20 in BCD).

7.3.3 Logical Instructions: Perform bitwise logical operations (AND, OR, XOR, NOT) on operands. They do not affect Carry, Auxiliary Carry, or Overflow flags.

- `ANL A, Rn`: Logical AND of Rn with A, result in A.
    - *Numerical Example:* A=F0textH (1111_0000_2), R0=0FtextH (0000_1111_2). `ANL A, R0` results in A=00textH (0000_0000_2).
- `ANL A, #data`: Logical AND of immediate `data` with A.
- `ANL A, direct_address`: Logical AND of `direct_address` with A.
- `ORL A, Rn`: Logical OR of Rn with A, result in A.
- `ORL A, #data`: Logical OR of immediate `data` with A.
- `XRL A, Rn`: Logical XOR of Rn with A, result in A.
- `CLR A`: Clear Accumulator (A = 00textH).
- `CPL A`: Complement Accumulator (bitwise NOT A).
- `SWAP A`: Swap nibbles in Accumulator (upper 4 bits with lower 4 bits).

    - *Numerical Example:* A=A5textH (1010_0101_2). `SWAP A` results in A=5AtextH (0101_1010_2).
- Rotate Instructions:
    - `RL A`: Rotate Accumulator Left. (Bit 7 moves to Bit 0).
        - *Numerical Example:* A=81textH (1000_0001_2). `RL A` results in A=03textH (0000_0011_2).
    - `RLC A`: Rotate Accumulator Left through Carry flag. (Bit 7 moves to CY, CY moves to Bit 0).
    - `RR A`: Rotate Accumulator Right. (Bit 0 moves to Bit 7).
    - `RRC A`: Rotate Accumulator Right through Carry flag. (Bit 0 moves to CY, CY moves to Bit 7).

7.3.4 Boolean (Bit Manipulation) Instructions: These instructions operate directly on individual bits, a powerful feature of the 8051. They are critical for I/O control and flag manipulation.

- `CLR bit`: Clear specified bit to 0. (Bit can be an I/O pin, a bit-addressable RAM location, or a flag).
    - *Numerical Example:* `CLR P1.0` (Set Port 1 Pin 0 to Low). `CLR C` (Clear Carry flag).
- `SETB bit`: Set specified bit to 1.
    - *Numerical Example:* `SETB P1.0` (Set Port 1 Pin 0 to High). `SETB C` (Set Carry flag).
- `CPL bit`: Complement specified bit.
- `ANL C, bit`: Logical AND of Carry flag with specified bit, result in Carry.
- `ORL C, bit`: Logical OR of Carry flag with specified bit, result in Carry.
- `MOV C, bit`: Move specified bit to Carry flag.
- `MOV bit, C`: Move Carry flag to specified bit.

7.3.5 Program Control Instructions (Jumps, Calls, Returns): These instructions alter the flow of program execution by modifying the Program Counter (PC).

- Unconditional Jumps:
    - `LJMP address16`: Long Jump. Jumps to a 16-bit absolute address anywhere in the 64textKB program memory space.
    - `AJMP address11`: Absolute Jump. Jumps to an 11-bit address within the current 2textKB block of program memory. Faster and shorter instruction than `LJMP`.
    - `SJMP relative_address`: Short Jump. Jumps relative to the current PC (from -128 to +127 bytes). Most common for local jumps.
        - *Numerical Example:* If PC points to `SJMP LABEL` and `LABEL` is 10 bytes ahead, PC becomes PC+10.
- Conditional Jumps: All conditional jumps are short (PC-relative) 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 `bit` is Set.**
    - **`JNB bit, label`: Jump if `bit` is Not Set.**
    - **`JBC bit, label`: Jump if `bit` is Set, then Clear `bit`.**
    - **`CJNE A, #data, label`: Compare Accumulator with immediate `data` and Jump if Not Equal.**
    - **`CJNE A, direct_address, label`: Compare Accumulator with content of `direct_address` and Jump if Not Equal.**
    - **`DJNZ Rn, label`: Decrement register Rn, and Jump if Not Zero. Used frequently for loops.**
        - ***Numerical Example:* `MOV R0, #10`. `LOOP: ... DJNZ R0, LOOP`. This loop will execute 10 times.**
    - **`DJNZ direct_address, label`: Decrement `direct_address`, and Jump if Not Zero.**
- **Call and Return Instructions: Used for subroutine calls.**
    - **`LCALL address16`: Long Call. Pushes the (PC+3) onto the stack (return address), then jumps to a 16-bit absolute address.**
    - **`ACALL address11`: Absolute Call. Pushes the (PC+2) onto the stack, then jumps to an 11-bit address within the current 2textKB block.**
    - **`RET`: Return from subroutine. Pops the 16-bit address from the stack into PC.**
    - **`RETI`: Return from Interrupt. Pops the 16-bit address from the stack into PC and also restores interrupt priority bits, if necessary.**

This comprehensive instruction set provides the programmer with fine-grained control over the 8051's internal resources and its interaction with the external environment, enabling the creation of robust and efficient embedded applications.

7.4 8051 On-Chip Peripherals: Timers/Counters, Serial Port (UART), Interrupts, and I/O Ports

The true power of the 8051 as a microcontroller lies in its integrated on-chip peripherals. These dedicated hardware modules perform common tasks, offloading work from the CPU and enabling the 8051 to interact effectively with the outside world.

7.4.1 Timers/Counters (TMOD, TCON, TLx, THx): The 8051 features two (8051) or three (8052) 16-bit Timers/Counters: Timer 0, Timer 1 (and Timer 2 on 8052). They can be used for two primary functions:

1. **Timer: Measures time delays by counting internal machine cycles.**
2. **Counter: Counts external events by counting pulses on an external input pin.**

Key Special Function Registers (SFRs) for Timers:

- **TMOD (Timer Mode Control Register): An 8-bit SFR at address 89textH. Not bit-addressable.**
    - **Controls the operating mode and function (timer/counter) for Timer 0 and Timer 1.**
    - **Structure: | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 | | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | | GATE1 | C/T1 | M11 | M01 | GATE0 | C/T0 | M10 | M00 |**
        - **`GATEx` (x=0 or 1): When set, timer/counter runs only when INTx (external interrupt pin) is high AND TRx (Timer Run bit in TCON) is set. This allows external control of the timer.**
        - **`C/Tx` (x=0 or 1): Counter/ Timer select.**
            - **`C/Tx = 0`: Timer operation (counts internal machine cycles).**
            - **`C/Tx = 1`: Counter operation (counts pulses on Tx external pin).**
        - **`M1x, M0x`: Mode select bits for Timer x. Define the timer's operating mode.**
            - **Mode 0 (M1x=0, M0x=0): 13-bit Timer/Counter**
                - **Uses TLx (lower 8 bits) and 5 bits of THx (upper 5 bits). The remaining 3 bits of THx are unused.**
                - **Max count = 213−1=8191.**
                - **Rolls over from all 1s to all 0s, setting the TFx (Timer Flag) in TCON.**
            - **Mode 1 (M1x=0, M0x=1): 16-bit Timer/Counter**
                - **Uses all 16 bits of TLx and THx.**
                - **Max count = 216−1=65535.**
                - **Rolls over from all 1s to all 0s, setting TFx.**
            - **Mode 2 (M1x=1, M0x=0): 8-bit Auto-Reload Timer/Counter**
                - **TLx acts as the counter, THx stores the reload value.**
                - **When TLx overflows from FFH to 00H, it sets TFx AND automatically reloads TLx with the value from THx.**
                - **Ideal for generating precise baud rates or periodic interrupts.**
            - **Mode 3 (M1x=1, M0x=1): Split Timer Mode**
                - **Timer 0 in Mode 3: Splits Timer 0 into two separate 8-bit timers/counters (TL0 and TH0). TL0 uses Timer 0 control bits (TR0, TF0), while TH0 uses Timer 1 control bits (TR1, TF1). This means Timer 1 can only operate as a UART baud rate generator in this configuration.**
                - **Timer 1 in Mode 3: Retains its Mode 0, 1, or 2 operation.**
- **TCON (Timer Control Register): An 8-bit SFR at address 88textH. Bit-addressable.**

    - Controls and monitors the run status and interrupt flags for Timer 0 and Timer 1.
    - Structure: | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 | | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | | TF1 | TR1 | TF0 | TR0 | IE1 | IT1 | IE0 | IT0 |
        - `TFx` (x=0 or 1): Timer Flag. Set to 1 when Timer x overflows. Cleared by hardware when the CPU vectors to the interrupt service routine (ISR) or can be cleared by software.
        - `TRx` (x=0 or 1): Timer Run control bit. Set to 1 to start Timer x, 0 to stop.
        - `IEx` (x=0 or 1): Interrupt Edge flag. Set when a negative edge is detected on external interrupt pin INTx. Cleared when CPU vectors to ISR.
        - `ITx` (x=0 or 1): Interrupt Type control bit.
            - `ITx = 0`: Interrupt on low level (level triggered).
            - `ITx = 1`: Interrupt on falling edge (edge triggered).
- TL0, TH0, TL1, TH1: These are the 8-bit registers that hold the current count value for Timer 0 (TL0, TH0) and Timer 1 (TL1, TH1). They are read/write registers.

Timing Calculation (Timer Mode):

- Machine Cycle: The 8051 typically executes 1 machine cycle for every 12 oscillator periods.
    - Formula: T_machine_cycle=12times(1/F_oscillator)
    - Numerical Example: If F_oscillator=12textMHz. T_machine_cycle=12times(1/12textMHz)=12times(1/12,000,000texts)=1textmicrosecond(mus).
- Delay Calculation:
    - To achieve a delay `D` in microseconds using a 16-bit timer (Mode 1):
        - The timer needs to count `D` ticks.
        - The timer counts up from an initial value `X` to 65535. So, 65536−X=D.
        - Initial count value X=65536−D.
        - This value X is loaded into the 16-bit (THx:TLx) register pair.
    - Numerical Example: Generate a 10textms delay using a 12textMHz crystal (1 mus per tick).
        - Total ticks required = 10textms/1mus/tick=10,000 ticks.
        - Initial value for 16-bit timer = 65536−10000=55536.
        - Convert to hex: 55536=D8F0textH. So, `THx = D8H`, `TLx = F0H`.

7.4.2 Serial Port (UART): The 8051 has a built-in full-duplex (can transmit and receive simultaneously) Universal Asynchronous Receiver/Transmitter (UART), making it easy to communicate with other devices that use serial protocols like RS-232.

Key SFRs for Serial Port:

- **SBUF (Serial Buffer Register): Address 99textH. Not bit-addressable.**
  - **A dual-purpose register:**
    - **Writing to SBUF initiates a serial transmission.**
    - **Reading from SBUF retrieves received serial data.**
- **SCON (Serial Port Control Register): Address 98textH. Bit-addressable.**
  - **Controls the serial port operating modes and holds status flags.**
  - **Structure: | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 | | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | | SM0 | SM1 | SM2 | REN | TB8 | RB8 | TI | RI |**
    - **`SM0, SM1`: Serial Port Mode Control bits. Define the 4 operating modes:**
      - **Mode 0 (SM0=0, SM1=0): Shift Register Mode**
        - **Data is transmitted/received on RXD pin. Clock is provided on TXD pin. Fixed baud rate: F_oscillator/12.**
      - **Mode 1 (SM0=0, SM1=1): 8-bit UART (Variable Baud Rate)**
        - **1 Start bit, 8 data bits, 1 Stop bit. Baud rate set by Timer 1 (or Timer 2 on 8052).**
      - **Mode 2 (SM0=1, SM1=0): 9-bit UART (Fixed Baud Rate)**
        - **1 Start bit, 8 data bits, 1 programmable 9th data bit (TB8/RB8), 1 Stop bit. Fixed baud rate: F_oscillator/32 or F_oscillator/64.**
      - **Mode 3 (SM0=1, SM1=1): 9-bit UART (Variable Baud Rate)**
        - **Same as Mode 2 but with variable baud rate set by Timer 1 (or Timer 2).**
    - **`SM2`: Multiprocessor Communication Bit. Used in Modes 2/3 for filtering addresses.**
    - **`REN` (Receive Enable): Set to 1 to enable serial reception.**
    - **`TB8`: Transmit Bit 8. The 9th data bit that will be transmitted in Modes 2/3.**
    - **`RB8`: Receive Bit 8. The 9th data bit that was received in Modes 2/3.**
    - **`TI` (Transmit Interrupt Flag): Set by hardware when a byte has been successfully transmitted. Cleared by software.**
    - **`RI` (Receive Interrupt Flag): Set by hardware when a byte has been successfully received. Cleared by software.**
- **PCON (Power Control Register): Address 87textH. Not bit-addressable.**
  - **Contains the `SMOD` bit (PCON.7) which, when set, doubles the baud rate in Modes 1, 2, and 3.**
  - **Also contains bits for power-down modes.**

Baud Rate Generation (Mode 1/3):

- **Typically, Timer 1 (in Mode 2, 8-bit auto-reload) is used to generate the baud rate.**
- **The overflow rate of Timer 1 determines the serial port baud rate.**

- **Formula (Baud Rate, when SMOD=0): BaudRate=fracF_oscillator384times(256−textTH1)**
- **Formula (Baud Rate, when SMOD=1): BaudRate=fracF_oscillator192times(256−textTH1)**
- **Numerical Example: To achieve 9600 baud with F_oscillator=11.0592textMHz and SMOD=0:**
    - **9600=frac11059200384times(256−textTH1)**
    - **384times(256−textTH1)=11059200/9600=1152**
    - **256−textTH1=1152/384=3**
    - **textTH1=256−3=253 (decimal)**
    - **textTH1=FDtextH (hexadecimal).**
    - **So, load `TH1` with `FDH` to get 9600 baud.**

7.4.3 Interrupts: Interrupts allow the 8051 to respond immediately to important events, rather than constantly polling peripherals. When an interrupt occurs, the CPU suspends its current task, executes a dedicated Interrupt Service Routine (ISR), and then returns to its original task.

8051 Interrupt Sources (Original 8051 has 5):

1. **External Interrupt 0 (overlinetextINT0): Pin P3.2**
2. **Timer 0 Overflow Interrupt: From Timer 0.**
3. **External Interrupt 1 (overlinetextINT1): Pin P3.3**
4. **Timer 1 Overflow Interrupt: From Timer 1.**
5. **Serial Port Interrupt: Triggered by TI (transmit interrupt) or RI (receive interrupt) flags.** ***(8052 adds a Timer 2 interrupt)***

Key SFRs for Interrupts:

- **IE (Interrupt Enable Register): Address A8textH. Bit-addressable.**
    - **Enables or disables individual interrupt sources.**
    - **Structure: | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 | | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | | EA | -- | ES | ET1 | EX1 | ET0 | EX0 | |**
        - **`EA` (Enable All): Global interrupt enable/disable. Must be set to 1 to enable any interrupt.**
        - **`ES` (Enable Serial): Enables/disables Serial Port interrupt.**
        - **`ET1` (Enable Timer 1): Enables/disables Timer 1 interrupt.**
        - **`EX1` (Enable External 1): Enables/disables External Interrupt 1.**
        - **`ET0` (Enable Timer 0): Enables/disables Timer 0 interrupt.**
        - **`EX0` (Enable External 0): Enables/disables External Interrupt 0.**
- **IP (Interrupt Priority Register): Address B8textH. Bit-addressable.**
    - **Sets the priority level for each interrupt source (high or low).**
    - **Structure: | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 | | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | | -- | -- | PS | PT1 | PX1 | PT0 | PX0 | |**
        - **`PS`: Serial Port interrupt priority.**
        - **`PT1`: Timer 1 interrupt priority.**
        - **`PX1`: External Interrupt 1 priority.**

        - **PT0: Timer 0 interrupt priority.**
        - **PX0: External Interrupt 0 priority.**
    - **0 = Low priority, 1 = High priority. If multiple interrupts of the same priority occur, a polling sequence determines which is serviced first.**
    - **High priority interrupts can interrupt low priority ISRs. Low priority interrupts cannot interrupt high priority ISRs.**

Interrupt Vector Table:

- **A fixed set of addresses in Program Memory where the CPU jumps when a specific interrupt occurs.**
- **The first instruction of the ISR must be located at these addresses.**
    - **External 0: 0003textH**
    - **Timer 0: 000BtextH**
    - **External 1: 0013textH**
    - **Timer 1: 001BtextH**
    - **Serial Port: 0023textH**

Interrupt Process:

1. **Interrupt flag is set (hardware).**
2. **If interrupt is enabled (in IE) and EA is set, and its priority allows, the CPU completes its current instruction.**
3. **CPU pushes (PC+1) onto the stack.**
4. **CPU pushes PSW onto the stack.**
5. **CPU loads PC with the corresponding interrupt vector address.**
6. **CPU clears the interrupt flag (hardware for TFx, IEx; software for TI, RI).**
7. **ISR executes.**
8. **RETI instruction is executed.**
9. **CPU pops PSW from stack.**
10. **CPU pops PC from stack, resuming main program.**

7.4.4 I/O Ports (P0, P1, P2, P3): The 8051 has four 8-bit bidirectional I/O ports: P0, P1, P2, P3. Each port has 8 pins, which can be configured as inputs or outputs by software.

Internal Structure of 8051 I/O Pins (Conceptual): Each pin consists of:

- **Internal Latch (SFR): An 8-bit SFR (e.g., P0, P1) stores the data written to the port.**
- **Output Driver: A pair of FETs (Field Effect Transistors) that can drive the pin HIGH or LOW.**
- **Input Buffer: Reads the logic level on the physical pin.**
- **Pull-up Resistor: Internal pull-up resistor (except for Port 0, which requires external pull-ups when used as I/O or for external memory).**
- **Port 0 (P0): Address 80textH. Bit-addressable.**

    - **Dual Purpose: Can be used as a general-purpose 8-bit I/O port OR as a multiplexed address/data bus (A0-A7 / D0-D7) for accessing external memory.**
    - **Open-Drain Outputs: Unlike other ports, P0 pins are open-drain when used as I/O. This means they can sink current to ground but require external pull-up resistors to drive a logic HIGH.**
    - **When used for external memory access, its internal pull-ups are disabled.**
- **Port 1 (P1): Address 90textH. Bit-addressable.**
    - **Dedicated I/O: Can only be used as a general-purpose 8-bit I/O port.**
    - **Internal Pull-ups: Has internal pull-up resistors, so no external pull-ups are needed for simple output (driving LEDs) or input from pushbuttons (active low).**
- **Port 2 (P2): Address A0textH. Bit-addressable.**
    - **Dual Purpose: Can be used as a general-purpose 8-bit I/O port OR as the high-order address bus (A8-A15) for accessing external memory.**
    - **Internal Pull-ups: Has internal pull-up resistors.**
- **Port 3 (P3): Address B0textH. Bit-addressable.**
    - **Multi-Purpose: Can be used as a general-purpose 8-bit I/O port.**
    - **Special Alternate Functions: Each pin of Port 3 has an alternate special function:**
        - **P3.0: RXD (Serial receive data)**
        - **P3.1: TXD (Serial transmit data)**
        - **P3.2: overlinetextINT0 (External Interrupt 0)**
        - **P3.3: overlinetextINT1 (External Interrupt 1)**
        - **P3.4: T0 (Timer 0 external input)**
        - **P3.5: T1 (Timer 1 external input)**
        - **P3.6: overlinetextWR (External Data Memory Write strobe)**
        - **P3.7: overlinetextRD (External Data Memory Read strobe)**
    - **Internal Pull-ups: Has internal pull-up resistors.**

Configuring I/O Pins:

- **As Output: To output a HIGH, write '1' to the corresponding port latch bit. To output a LOW, write '0'.**
- **As Input: To configure a pin as an input, a '1' must be written to its corresponding port latch bit. This disables the output driver, enabling the internal pull-up (for P1, P2, P3) and allowing the input buffer to read the external pin state. The actual input value is then read directly from the port pin (e.g., `MOV A, P1`).**
    - ***Important Note:* Writing `0` to a pin makes it an output and pulls it low (drives it low). This is useful for driving LEDs.**

The combination of these on-chip peripherals allows the 8051 to perform complex control, communication, and timing tasks autonomously, minimizing the need for external components and reducing system cost and complexity.

7.5 8051 Programming in Assembly and C: Practical Examples for Peripheral Control

Effective control of the 8051's peripherals requires programming. While modern development largely favors C, understanding 8051 Assembly language provides critical insight into the hardware's low-level operations.

7.5.1 8051 Assembly Language Programming: Assembly language provides direct, low-level control over the 8051's registers and memory. Each assembly instruction typically corresponds to one machine code instruction.

Advantages of Assembly:

- **Maximum Performance: Code can be highly optimized for speed and efficiency.**
- **Minimal Code Size: Generates very compact code, crucial for MCUs with limited program memory.**
- **Direct Hardware Control: Offers precise control over every bit and register.**
- **Debugging Low-Level Issues: Essential for understanding exact hardware behavior.**

Disadvantages of Assembly:

- **Development Time: Slower to write and debug compared to C.**
- **Portability: Code is not portable to other microcontroller architectures.**
- **Maintainability: Difficult to read and maintain for complex projects.**

Assembly Example: Toggling an LED on Port P1.0 with a Delay

This example demonstrates setting up a timer for a delay and then toggling an LED connected to P1.0.

Code snippet

```
; Program to toggle an LED connected to P1.0 with a 500ms delay
; Crystal Frequency: 11.0592 MHz
; Machine Cycle: 12/11.0592MHz = 1.085 us (approx)
; Desired Delay = 500ms = 500,000 us
; Timer Ticks for 500ms = 500,000 us / 1.085 us/tick = 460830 ticks
; This is too large for a single 16-bit timer (max 65536 ticks).
; We need to generate 50ms delay 10 times to get 500ms.
; Ticks for 50ms = 50,000 us / 1.085 us/tick = 46083 ticks
; TH0:TL0 value for 46083 ticks (Mode 1, 16-bit) = 65536 - 46083 = 19453 (decimal)
; 19453 decimal = 4BFD H
; TH0 = 4BH, TL0 = FDH

ORG 0000H        ; Program starts at address 0000H
LJMP MAIN        ; Jump to the main program
ORG 000BH        ; Timer 0 Interrupt Vector Address
LJMP TIMER0_ISR   ; Jump to Timer 0 Interrupt Service Routine
```

```
MAIN:
   MOV P1, #00H      ; Initialize Port 1 (LED is OFF)
   SETB P1.0         ; Turn LED ON initially (P1.0 = 1)

   ; Configure Timer 0 for Mode 1 (16-bit timer)
   MOV TMOD, #01H    ; Timer 0, Mode 1 (0000_0001B)

   ; Load initial value for 50ms delay (4BFDH)
   MOV TL0, #0FDH
   MOV TH0, #04BH

   ; Enable Timer 0 Interrupt
   SETB ET0          ; Enable Timer 0 interrupt
   SETB EA           ; Enable Global interrupts

   ; Start Timer 0
   SETB TR0          ; Start Timer 0

LOOP:
   SJMP LOOP         ; Infinite loop, CPU waits for interrupts
                     ; (or perform other tasks while timer runs)

TIMER0_ISR:
   CLR TR0           ; Stop Timer 0 (before re-loading)
   CLR TF0           ; Clear Timer 0 overflow flag (hardware does this on vector, but
good practice)

   ; Reload Timer 0 for next 50ms delay
   MOV TL0, #0FDH
   MOV TH0, #04BH

   CPL P1.0          ; Complement P1.0 (toggle LED)

   SETB TR0          ; Restart Timer 0
   RETI              ; Return from Interrupt
```

Explanation of Assembly Example:

1. `ORG 0000H`, `LJMP MAIN`: Sets program start and jumps to `MAIN`.
2. `ORG 000BH`, `LJMP TIMER0_ISR`: Sets Timer 0 interrupt vector.
3. `MAIN`:
    - `MOV P1, #00H`, `SETB P1.0`: Initializes Port 1 and turns on the LED on P1.0.
    - `MOV TMOD, #01H`: Sets Timer 0 to 16-bit mode.
    - `MOV TL0, #0FDH`, `MOV TH0, #04BH`: Loads the calculated initial count for a 50textms delay.

- `SETB ET0`, `SETB EA`: Enables the Timer 0 specific interrupt and the global interrupt enable.
- `SETB TR0`: Starts Timer 0.
- `SJMP LOOP`: The main program enters an infinite loop, allowing the CPU to service interrupts.

4. `TIMER0_ISR`:
   - This routine is executed every 50textms when Timer 0 overflows.
   - `CLR TR0`, `CLR TF0`: Stops the timer and clears its overflow flag.
   - Reloads `TL0` and `TH0` with the same value to ensure continuous 50textms delays.
   - `CPL P1.0`: Toggles the state of the P1.0 pin.
   - `SETB TR0`: Restarts the timer for the next interval.
   - `RETI`: Returns from the interrupt, restoring PC and PSW, and re-enabling interrupts.

7.5.2 8051 C Language Programming: C is the preferred language for 8051 development due to its readability, maintainability, and greater portability (though specific 8051 extensions are used). Compilers like Keil uVision (C51 compiler) provide optimized code generation for the 8051.

Advantages of C:

- **Higher Abstraction: Easier to write and understand complex logic.**
- **Faster Development Cycle: Reduced coding and debugging time.**
- **Modularity and Reusability: Code can be easily organized into functions and modules.**
- **Portability (Limited): More portable than assembly; some code can be reused on other C-supported MCUs.**

Disadvantages of C:

- **Less Optimal Code Size/Speed: Generated machine code might be larger and slower than hand-optimized assembly.**
- **Less Direct Control: Compiler might hide some low-level hardware details.**

C Example: Toggling an LED on Port P1.0 with a Delay (using Keil C51 specific syntax)

C

```
#include <reg51.h> // Include standard 8051 register definitions

// Define the LED pin
sbit LED_PIN = P1^0; // P1.0 (Port 1, bit 0)

// Global variable for delay count (if 50ms delay needs to happen multiple times for a longer delay)
unsigned int ms_count = 0;

// Timer 0 Interrupt Service Routine prototype
```

```
void Timer0_ISR(void) interrupt 1 // 'interrupt 1' means it's Timer 0 interrupt
{
   // Reload Timer 0 for 50ms delay
   TH0 = 0x4B; // Load high byte
   TL0 = 0xFD; // Load low byte

   ms_count++; // Increment millisecond counter (each increment is 50ms)

   if (ms_count >= 10) // Check if 500ms (10 * 50ms) has passed
   {
      LED_PIN = ~LED_PIN; // Toggle LED
      ms_count = 0;       // Reset counter
   }
}

void main()
{
   // --- Configure Timer 0 for 50ms delay ---
   TMOD = 0x01; // Timer 0, Mode 1 (16-bit timer)

   // Load initial value for 50ms delay (4BFDH)
   // 65536 - (50ms / (12 / 11.0592MHz)) = 65536 - 46083 = 19453 (0x4BFD)
   TH0 = 0x4B;
   TL0 = 0xFD;

   // --- Configure Interrupts ---
   ET0 = 1;     // Enable Timer 0 Interrupt
   EA = 1;      // Enable Global Interrupts

   // --- Start Timer ---
   TR0 = 1;     // Start Timer 0

   // Initial LED state
   LED_PIN = 0; // Turn LED OFF initially (assuming active high LED, or low for active
low)

   while (1)
   {
      // Main loop, can perform other tasks here while timer runs
      // For this example, it's just an empty loop.
   }
}
```

Explanation of C Example:

1. `#include <reg51.h>`: Includes a header file provided by the compiler (e.g., Keil C51) that defines all the 8051 SFRs (e.g., `P1`, `TMOD`, `TH0`, `EA`, `ET0`, `TR0`) as C variables or bit addresses.
2. `sbit LED_PIN = P1^0;`: This is a Keil C51 specific extension that allows direct bit manipulation. It defines `LED_PIN` as a symbolic name for bit 0 of Port 1.
3. `unsigned int ms_count = 0;`: A global variable to count multiple 50ms intervals to achieve a 500ms toggle.
4. `void Timer0_ISR(void) interrupt 1`: This defines the Interrupt Service Routine for Timer 0. The `interrupt 1` keyword tells the compiler to place the function at the Timer 0 interrupt vector address (000BtextH) and generate the `RETI` instruction at the end.
    - Inside the ISR, `TH0` and `TL0` are reloaded for the next 50textms period.
    - `ms_count` is incremented.
    - When `ms_count` reaches 10 (meaning 10times50textms=500textms has passed), `LED_PIN` is toggled using the bitwise NOT operator (`~`), and `ms_count` is reset.
5. `main()` function:
    - `TMOD = 0x01;`: Configures Timer 0 for Mode 1 (16-bit timer).
    - `TH0 = 0x4B; TL0 = 0xFD;`: Loads the initial count value for 50textms.
    - `ET0 = 1; EA = 1;`: Enables the Timer 0 interrupt and the global interrupt enable.
    - `TR0 = 1;`: Starts Timer 0.
    - `LED_PIN = 0;`: Sets the initial state of the LED.
    - `while (1)`: An infinite loop, the typical structure for embedded programs. The CPU stays in this loop, waiting for interrupts to occur.

This module provides a solid foundation in the 8051 microcontroller, from its architectural nuances and memory organization to its instruction set and on-chip peripherals. The practical examples in both Assembly and C illustrate how these concepts translate into real-world control applications, equipping you with the knowledge to begin developing embedded systems with the 8051.