# CS/MG/2043/09/21

# Embedded assignment

a) Port 1 and Port 2 in 8051 Microcontroller:

* Port 1 (P1): Consists of 8 pins (P1.0 to P1.7). These pins serve as bidirectional I/O ports. When logic 0 is applied, they act as output pins, and when logic 1 is applied, they act as input pins. P1 pins are often used for simple I/O operations.
* Port 2 (P2): Also has 8 pins (P2.0 to P2.7). Like P1, these pins can be configured as input or output. Additionally, P2 pins have some special functions:
  + P2.0 (RXD): Serial data receive pin for serial communication.
  + P2.1 (TXD): Serial data transmit pin for serial communication.
  + P2.2 (INT0): External hardware interrupt 0.
  + P2.3 (INT1): External hardware interrupt 1.
  + P2.4 (T0): Timer 0 external input.
  + P2.5 (T1): Timer 1 external input.
* [These ports play a crucial role in interfacing with external devices and peripherals](https://www.geeksforgeeks.org/microcontrollers-8051-interrupts/) [1](https://www.geeksforgeeks.org/microcontrollers-8051-interrupts/).

b) Challenges in Embedded Computing System Design:

* Limited Hardware: Deciding how much hardware (CPU power, memory, peripherals) to use without overspending or compromising performance.
* Meeting Deadlines: Ensuring timely execution of tasks, considering CPU clock rate and memory system limitations.
* Power Consumption: Minimizing power usage, especially critical in battery-powered applications.
* Design for Upgradability: Creating hardware that can be used across product generations or versions.
* [Reliability: Ensuring products work as expected](https://www.geeksforgeeks.org/microcontrollers-8051-interrupts/) [2](https://www.brainkart.com/article/Challenges-in-Embedded-Computing-System-Design_7783/).

c) Rules Followed by Interrupt Routines in RTOS:

* Rule 1: Keep interrupt routines short and efficient. Minimize execution time to prevent blocking other tasks.
* Rule 2: Avoid using blocking calls (such as waiting for I/O) within interrupt routines. [Use non-blocking mechanisms](https://www.geeksforgeeks.org/microcontrollers-8051-interrupts/) [3](https://electro4u.net/blog/interrupts-in-8051-microcontroller-810).

d) Push and Pull Opcodes:

* PUSH: Saves the value of the accumulator (A) or another register onto the stack.
* POP: Retrieves the value from the stack and loads it into the accumulator (A) or another register.
* [These instructions are essential for managing the stack during subroutine calls and context switching](https://www.geeksforgeeks.org/microcontrollers-8051-interrupts/) [4](https://ww1.microchip.com/downloads/en/DeviceDoc/doc0509.pdf).

e) Major Hardware Components in Embedded Computer Systems:

* CPU (Central Processing Unit): Executes instructions.
* Memory (RAM and ROM): Stores data and program code.
* I/O Ports: Connect to external devices.
* Timers and Counters: Handle timing and synchronization.
* Interrupt Controller: Manages interrupts.
* [Serial Communication Interfaces: Enable communication with other devices](https://www.geeksforgeeks.org/microcontrollers-8051-interrupts/) [5](https://www.geeksforgeeks.org/introduction-to-8051-microcontroller/).

f) Why Use Assembly Language in Implementing Microcontrollers:

* Efficiency: Assembly code allows fine-grained control over hardware resources.
* Low-Level Access: Directly manipulate registers, memory, and I/O ports.
* Size Optimization: Assembly code is compact.
* [Real-Time Constraints: Critical for real-time systems](https://www.geeksforgeeks.org/microcontrollers-8051-interrupts/) [6](https://technobyte.org/data-transfer-instructions-in-8051/).

g) Counting Semaphores vs. Mutex Semaphores:

* Counting Semaphores: Used for resource counting (e.g., available buffers). Can have multiple owners.
* Mutex Semaphores (Binary Semaphores): Used for mutual exclusion. Only one owner at a time.
* [Both manage access to shared resources but have different use cases](https://www.geeksforgeeks.org/microcontrollers-8051-interrupts/) [3](https://electro4u.net/blog/interrupts-in-8051-microcontroller-810).

h) Sequence of Events During CALL Instruction:

1. Save the return address (address of the next instruction) on the stack.
2. Jump to the subroutine (specified by the CALL instruction).
3. Execute the subroutine.
4. [Return from the subroutine using the RET instruction, which pops the return address from the stack](https://www.geeksforgeeks.org/microcontrollers-8051-interrupts/) [4](https://ww1.microchip.com/downloads/en/DeviceDoc/doc0509.pdf).

i) Voltmeter and Ohmmeter:

* Voltmeter: Measures voltage (potential difference) between two points.
* [Ohmmeter: Measures resistance (ohms) of a component or circuit](https://www.geeksforgeeks.org/microcontrollers-8051-interrupts/) [4](https://ww1.microchip.com/downloads/en/DeviceDoc/doc0509.pdf).

j) Ways to Protect Shared Data:

* Mutexes/Semaphores: Ensure exclusive access to shared resources.
* Atomic Operations: Use atomic read-modify-write instructions.
* Critical Sections: Temporarily disable interrupts during critical code sections.
* [Lock-Free Algorithms: Design algorithms that avoid locks 3](https://electro4u.net/blog/interrupts-in-8051-microcontroller-810).

### k) Strategies for I/O Software in Embedded System

* Polling: Regularly checking the status of an I/O device in a loop.
* Interrupt-Driven: Using hardware interrupts to signal when I/O operations are needed, reducing CPU load.
* DMA (Direct Memory Access): Allows hardware devices to transfer data to/from memory without CPU intervention, freeing up the CPU.

l) PSoC Architecture

* PSoC Core: The central processing unit, executing the main program logic.
* Digital System: Includes programmable digital blocks that can be configured for various digital functions and interfaces.
* Analog System: Consists of programmable analog blocks for operations like amplification, filtering, and AD/DA conversion.
* System Resources: Encompass resources like clocks, power management, and communication blocks that support the operation of the core, digital, and analog systems.

m) Limitations of Testing Embedded Systems on Host Machine

* Differences in Hardware: The host machine does not replicate the embedded system's hardware constraints and behaviors accurately.
* Real-Time Behavior: The timing and real-time responses can be very different on a host machine.
* Peripheral Interactions: Direct interactions with specific hardware peripherals cannot be accurately simulated.
* Environmental Conditions: The host environment cannot replicate the physical conditions (temperature, vibration, etc.) in which the embedded system operates.