PART 9: HARDWARE INTERRUPTS

→ allow events to occur asynchronously (not part of executing program)

↓

notifies CPU to take action

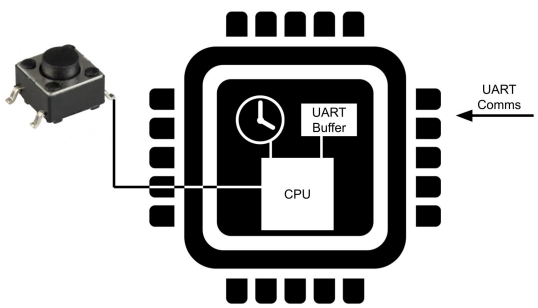
↓

CPU stops its current execution

↓

runs interrupt service routine (ISR)

example: button presses, hardware timer expiring, communication buffer being filled.



→ hardware interrupts have higher priority than any tasks

ISR rules:

(1) ISRs should not block itself

↳ because it is not executed as part of a task ∴ cannot be blocked.

↳ use function calls that end in *FromISR inside an ISR

↳ cannot wait for queue, mutex, semaphore

① Keep ISRs as short as possible

↳ so we don't delay the execution of waiting tasks

② if a variable is updated inside an ISR, declare it as "volatile"

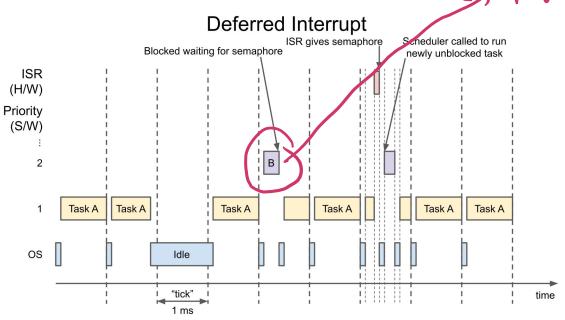
↳ lets compiler knows that the variable can change outside the current thread of execution

↳ otherwise, compilers might remove the variable as they don't think its being used.

Deferred Interrupt → to synchronise a task with an ISR

↳ the data captured inside the ISR is deferred to another task

↳ when the data is captured, we can give a semaphore to let the task know the data is ready for processing.



task B is blocked

↳ waits for semaphore from ISR

↓

when ISR finished executing, Task B is unblocked & processes the newly collected data.