# Interrupt handling

COS10004 Lecture 11.1

## Interrupts

- An interrupt is a signal or process that requests the CPU’s attention
- Sources of interrupts include any peripheral device, clocked events, or program induced events
- *Interrupt handling* is the process of temporarily halting a processor’s execution of a task in order to handle an event
  - Handled by a defined “Interrupt Handler” for that specific event
  - The halt allows peripheral devices to access the microprocessor
- Once interrupt handling is complete, the processor state is restored back to what it was before the interrupt

## Interrupts and the Fetch-Execute cycle

**Interrupts and the Fetch-Execute cycle**

Interrupts are processed only at the end of each fetch-execute cycle. Why? To allow interrupts *during* fetch, decode, or execute would not permit the state of the processor to be saved in a consistent manner, and therefore it would not possible to resume processing, safely, once the interrupt had been processed.

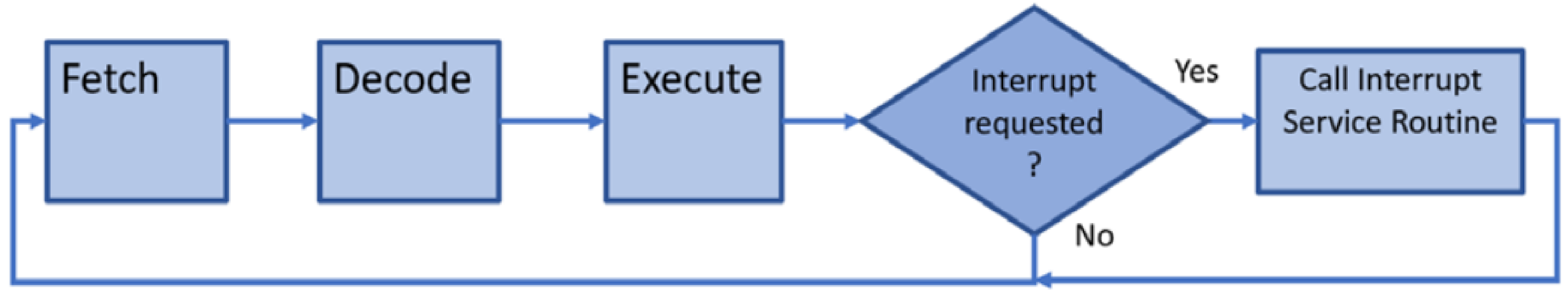
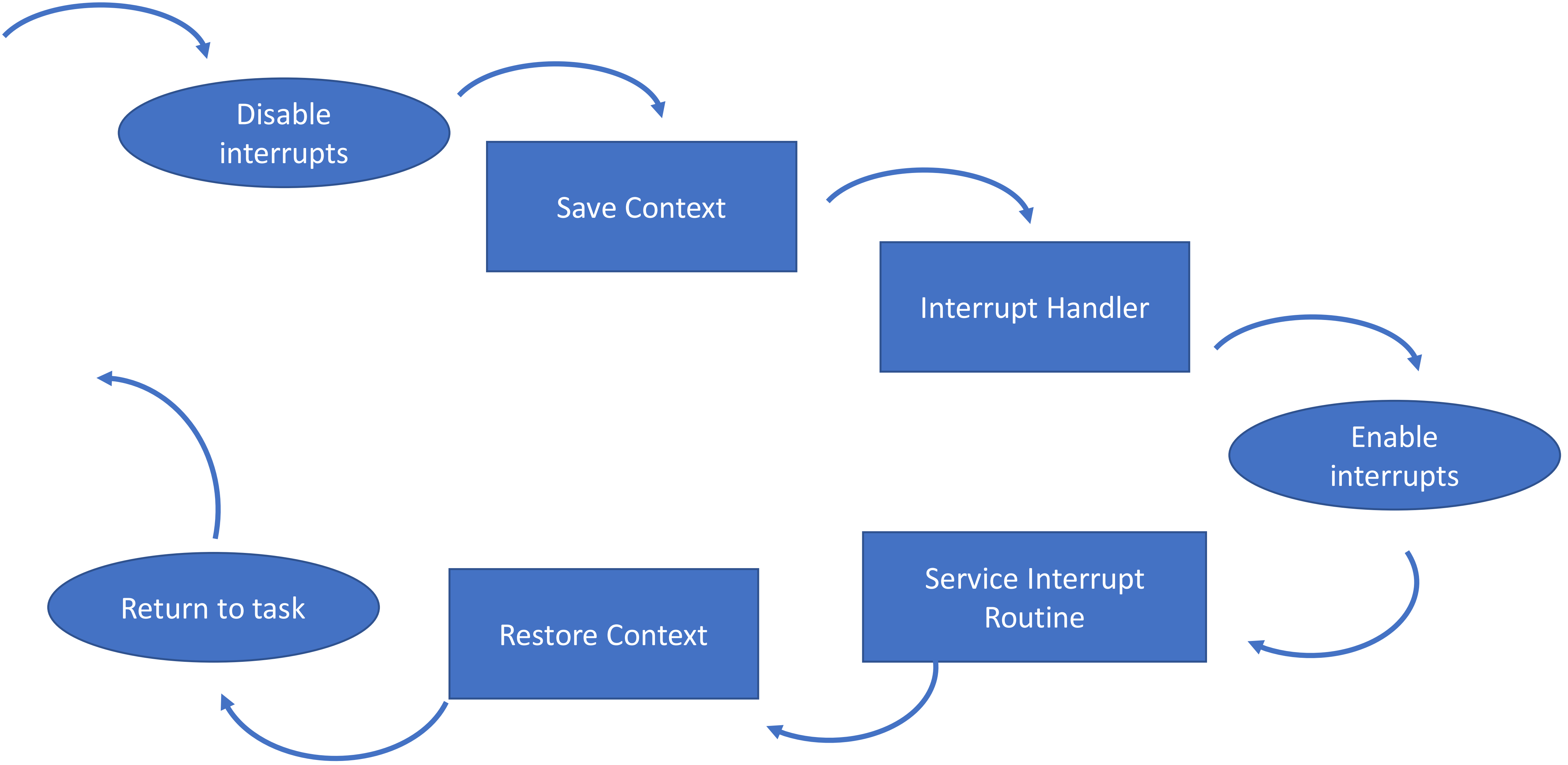


Image src: https://peterhigginson.co.uk/ARMlite/Assembly%20Language%20-%20Student%20version.pdf

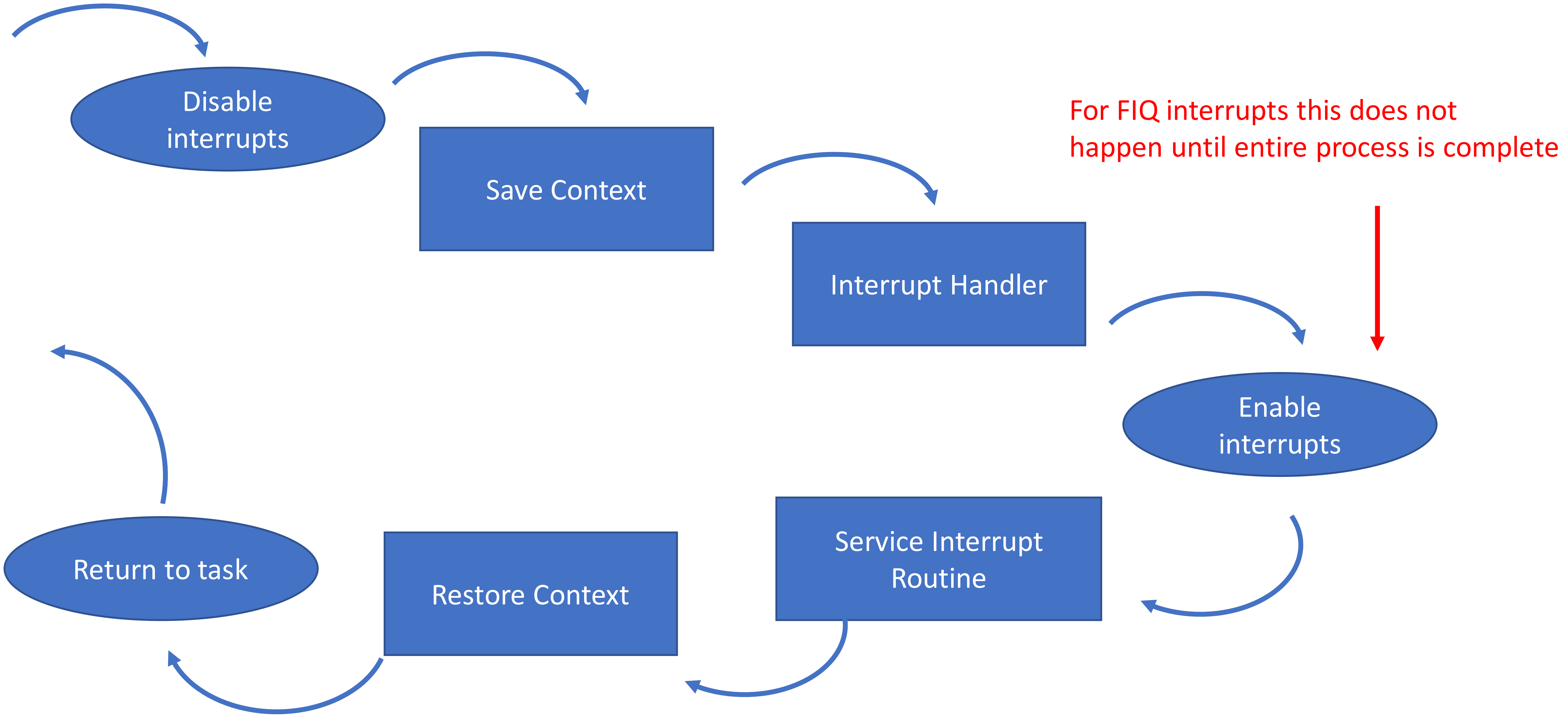
# ARM Interrupt types

- ARM CPUs support two types of interrupts: IRQ and FIQ
  - IRQ: **I**nterrupt **R**e**Q**uests are general interrupts
  - FIQ: **F**ast **I**nterrupt Re**Q**uests
- FIQs have a higher priority than IRQs in two ways:
  - Serviced first when multiple interrupts arise
    - Think keyboard strokes, mouse actions, network card etc
  - Servicing an FIQ disables IRQs (and further FIQs)
  - IRQs will not be serviced until after FIQ handler exits
- IRQs for general interrupts:
  - Program generated, or other peripheral devices.

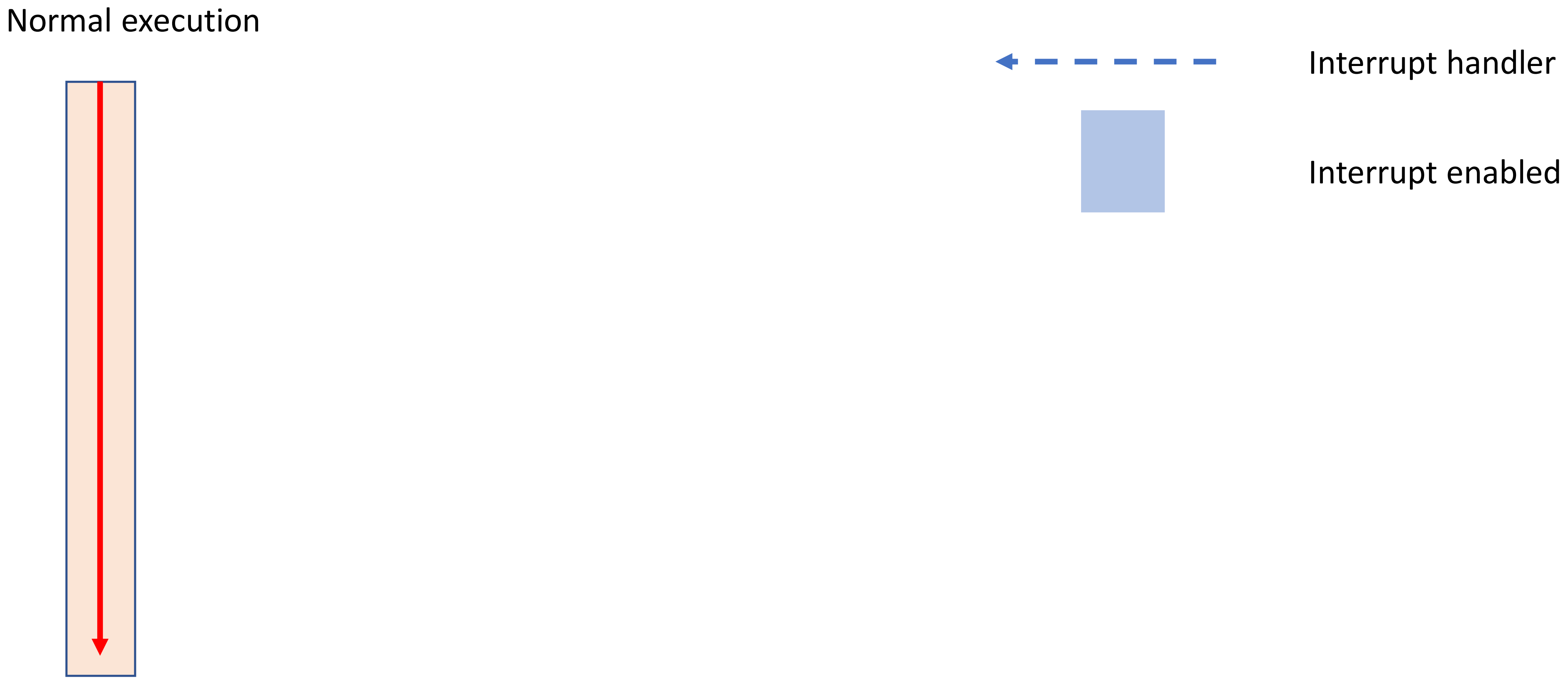
# Simple Interrupt Handling Flow



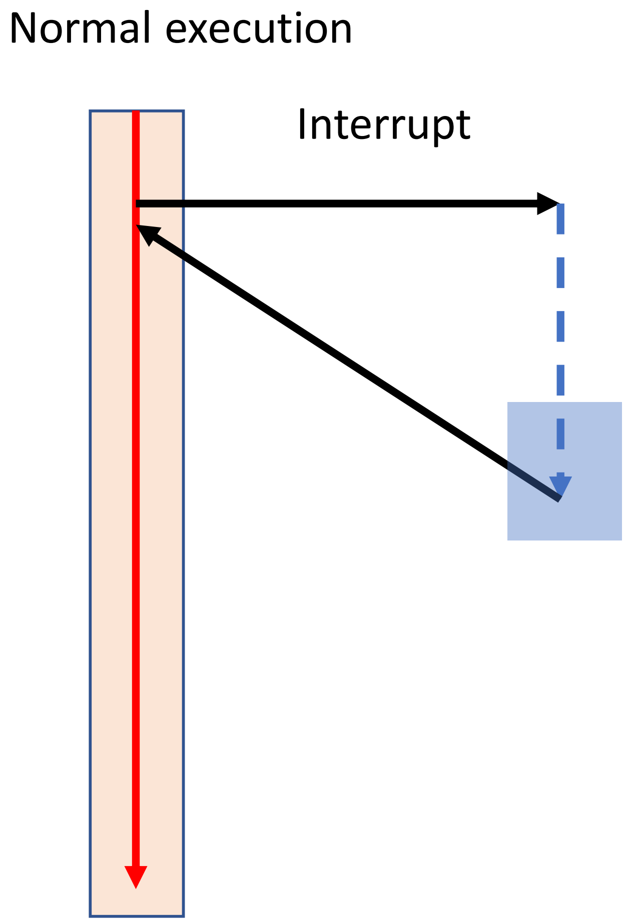
# Simple Interrupt Handling Flow

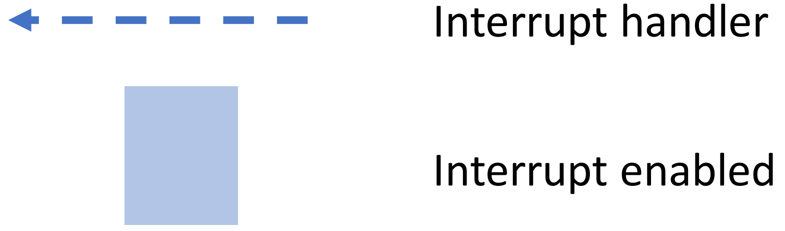

Disable interrupts
Save Context
Interrupt Handler
For FIQ interrupts this does not happen until entire process is complete
Enable interrupts
Service Interrupt Routine
Restore Context
Return to task

# Nested Interrupts

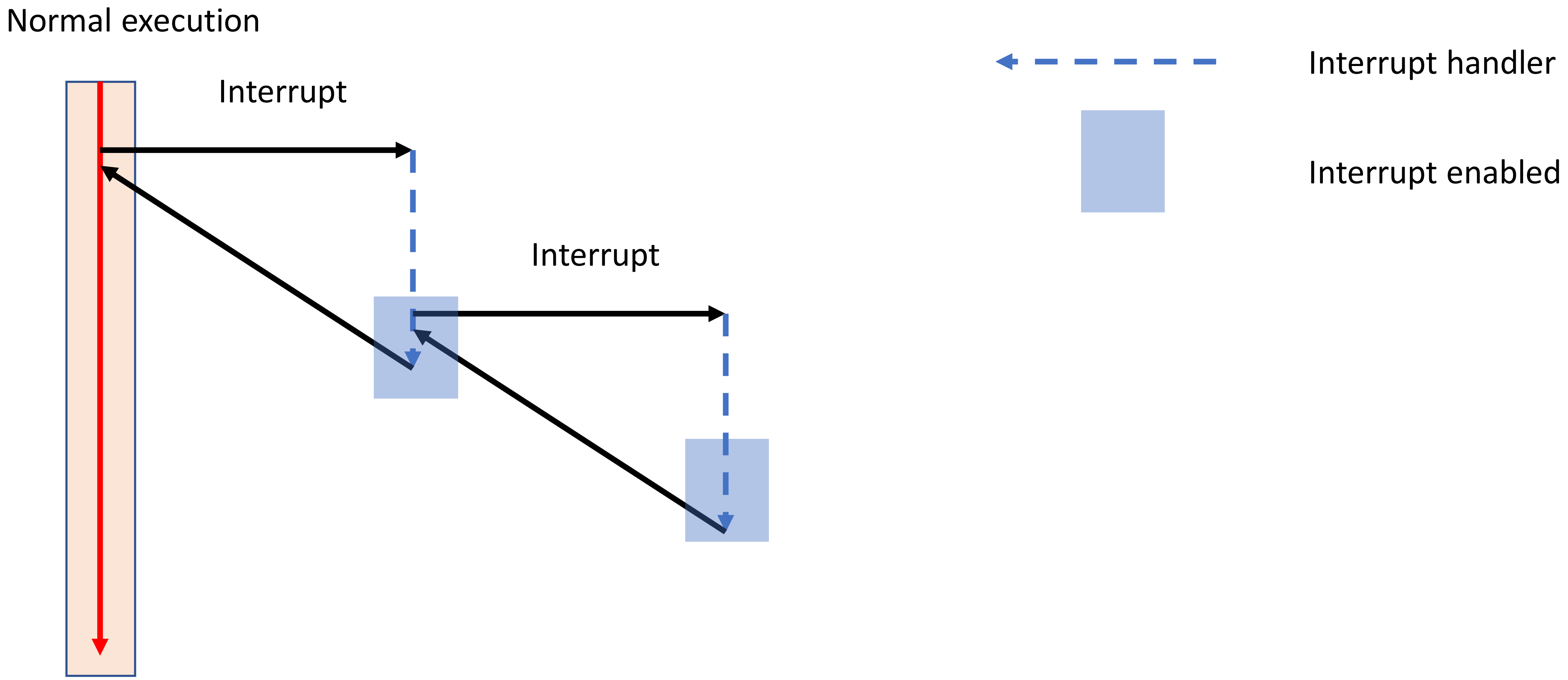


# Nested Interrupts

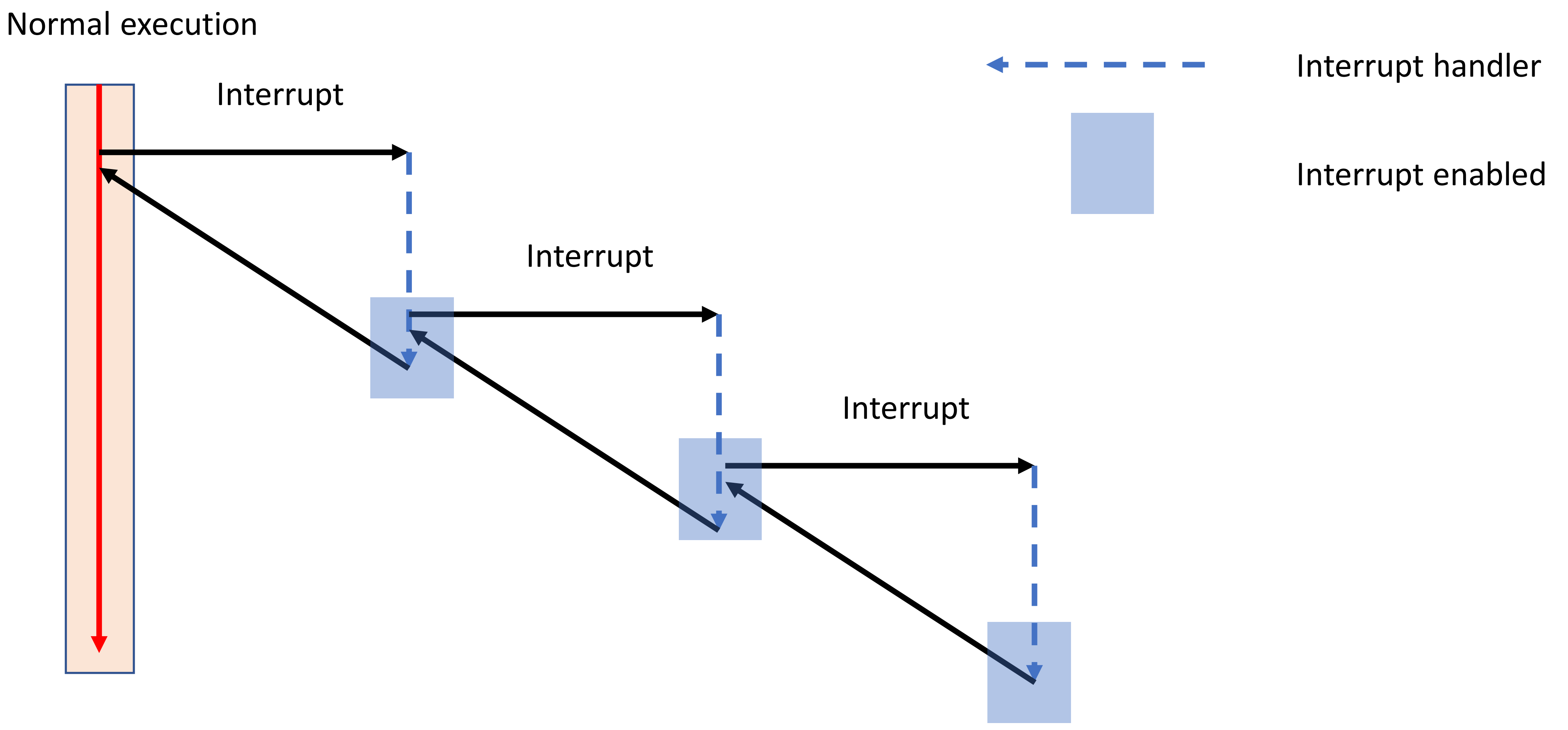




# Nested Interrupts



# Nested Interrupts



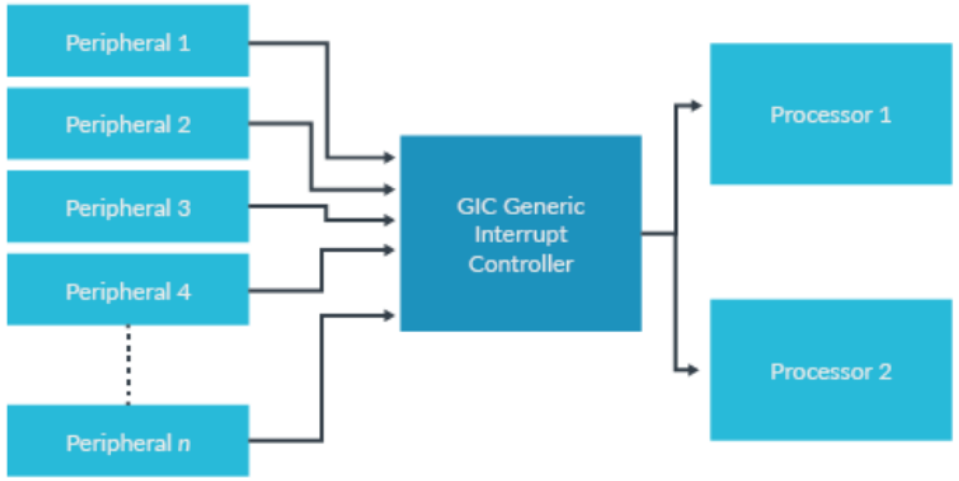
## Handling Interrupts – the vector table

- Exception/Interrupt handling on ARM processors is controlled through the use of an area of memory called the vector table.
- Within this table one word is allocated to each of the various exception/interrupt types.
- This word will contain some form of branch instruction to an interrupt handler:
  - Code specifically provided to handle that type of interrupt
- When one of these exceptions is taken, the ARM goes through a set of actions in order to invoke the appropriate interrupt handler.

# Handling interrupts – storing and recalling context

- Interrupt handling requires a different program/function to be executed.
- This involves what is known as a *context switch:*
  - the process of storing the state of a process or thread, so that it can be restored and resume execution at a later point.
  - Allows a single CPU to multi-task
- The state of the currently executing process must be saved so it can be restored
  - This includes the register values, including the PC and the LR addresses.
- The stack is typically used to achieve this.

## Interrupt Controller



- Multiplexes a number of possible interrupt sources for presentation to processor.
- ARM defines the *Generic Interrupt Controller* to handle large numbers of interrupts and CPUs with more than 1 core.
- Consists of two primary functionalities:
  - Distributor:
    - peripheral facing component.
    - responsible for managing interrupts in the whole system and decides priorities between them and routing mechanism of the same.
  - CPU interfaces:
    - For each CPU core, there is a corresponding CPU Interface that bridges the Distributor interface with the core.
    - It implements the priority masking for the processor.

# Interrupts versus polling

| Interrupts | Polling |
|---|---|
| An event that is triggered by external components other than the CPU that alerts the CPU to perform a certain action | An activity of sampling the status of an external device as part of a synchronous activity |
| When an interrupt occurs, the interrupt handler executes | CPU handles the service as required |
| Can occur at any time | Occurs at regular intervals |
| Interrupt-request line indicates that device needs a service | Command ready bit indicates the device needs a service |
| Minimal CPU cycle wastage | Wastes significant CPU cycles |
| Can be inefficient when device interrupts the CPU frequently | Inefficient when CPU rarely finds service requests from devices |