2. (a) Explain the mechanisms to handle multiple interrupts. [3 marks]

(b) Using a block diagram, explain a typical instruction cycle with

interrupts. [4 marks]

(c) Differentiate between centralized and distributed bus arbitration.

[3 marks]  
  
  
  
Here’s a beginner-friendly explanation for part (a):

**(a) Mechanisms to Handle Multiple Interrupts**  
**[3 Marks]**

1. **Disable Interrupts:**
   * When the CPU is handling one interrupt, it can temporarily disable other interrupts.
   * Other interrupts remain *pending* and will be checked once the current one is finished.
   * This ensures *sequential* (one-by-one) interrupt handling.
2. **Interrupt Priority Levels:**
   * Each interrupt can be assigned a *priority*.
   * A *higher priority* interrupt can interrupt (preempt) a lower priority one.
   * After handling the higher priority task, the CPU returns to finish the lower one.
   * This helps handle *more important* tasks first.

These two mechanisms help manage and organize multiple interrupt requests efficiently.

Let me know if you want a diagram or example to understand it more clearly!

b)

A diagram of a diagram

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

c)

Centralized and distributed bus arbitration differ in how control over the bus is managed:

* **Centralized Arbitration:** A single hardware component, called the bus controller or arbiter, is responsible for managing access to the bus. This controller allocates time on the bus and ensures orderly communication between connected devices. It can be integrated into the CPU or exist as a separate unit.
* **Distributed Arbitration:** There is no central controller; instead, each module connected to the bus contains its own access control logic. These modules work together to determine bus access, leading to a more decentralized and cooperative approach.

A centralized system offers simplicity and efficiency but can create bottlenecks if the arbiter fails. Distributed arbitration increases system resilience and scalability but requires more complex coordination among modules.

Here's a beginner-friendly comparison to answer part (c) of your question:

**(c) Difference between Centralized and Distributed Bus Arbitration:**  
**[3 Marks]**

| **Feature** | **Centralized Arbitration** | **Distributed Arbitration** |
| --- | --- | --- |
| **Control** | A single device (like a bus controller or CPU) controls who gets access to the bus. | No single controller; all devices participate in deciding who gets access. |
| **Complexity** | Simple to implement. | More complex because each module needs its own logic. |
| **Speed** | Usually faster because decisions are made by one device. | Slower due to communication between modules. |

Let me know if you want an example or diagram to understand better.

1. B)   
   Differentiate among programmed I/O, interrupt driven I/O, and

direct memory access (DMA). Explain operation of direct memory

access (DMA). [5 marks]  
  
  
Here’s a clear and beginner-friendly answer for your **5-mark** question based on the given information:

**(a) Difference among Programmed I/O, Interrupt Driven I/O, and Direct Memory Access (DMA):**

**[3 Marks]**

| **Technique** | **Programmed I/O** | **Interrupt Driven I/O** | **Direct Memory Access (DMA)** |
| --- | --- | --- | --- |
| **CPU Role** | CPU handles the entire I/O operation and waits for it to finish. | CPU starts I/O and continues other tasks; gets interrupted when done. | CPU starts I/O, then DMA controller handles the data transfer. |
| **Efficiency** | Least efficient — CPU time is wasted waiting. | Better — CPU can do other work during I/O. | Most efficient — CPU is free while DMA transfers data. |
| **Data Transfer** | Done by CPU. | Done by CPU after interrupt. | Done by DMA controller directly between memory and I/O. |
| **CPU Involvement** | Fully involved during the whole process. | Involved only at start and end. | Involved only to start and receive final signal (interrupt). |

**(b) Operation of Direct Memory Access (DMA):**

**[2 Marks]**

1. **Initiation:** The **CPU** sends a request to the **DMA controller** with:
   * Type of operation (read/write)
   * Source and destination addresses
   * Size of data
2. **Transfer:**
   * The **DMA controller** takes control of the system bus.
   * It transfers data **directly between I/O device and memory**, bypassing the CPU.
3. **Completion:**
   * Once the transfer is complete, the **DMA controller sends an interrupt** to the CPU.
   * CPU then continues with post-transfer operations.

✅ This reduces CPU workload and increases overall system performance, especially for large data transfers.

Let me know if you want a simple diagram to go with this!