**INPUT-OUTPUT SYSTEMS**

**Types of I/O Devices**

* Block device
* Character device

Block device:-

* The **device** with which the **driver** communicates by sending **blocks of data**.
* Like **hard disk** & **USB cameras** etc.

Character device:-

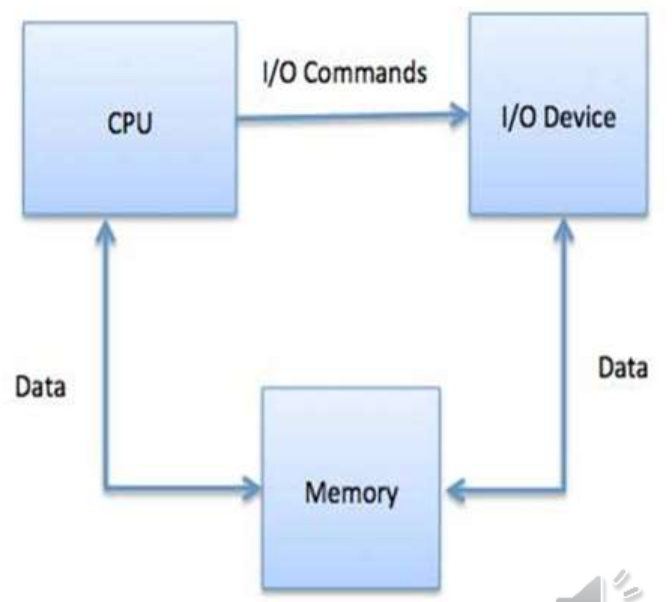
* The **device** with which the **driver** communicates **byte by byte**.
* Like **serial ports**, **parallel ports** & **sound card** etc.

**Device Controller**

* **Electronics** having capability to **operate** port, bus or device.
* Computer devices are connected to **plug** & **socket**.
* And **socket** is connected to ***device controller***.
* Each of them contains their own ***buffer*** & ***command register***.
* **Command register:** Stores executable commands in it.
* ***Local buffers*** store messages passed between ***device controller*** & **CPU**.
* And this communication is done via ***interrupts*** (signals).
* Examples include: **Network card**, **disk controller**, **USB**, **serial ports** & **sound card**.

**Memory Mapped I/O**

* A **communication triangle** among ***CPU***, ***memory*** & ***peripherals***.
* Uses **common bus**, **control systems** & **address space**.
* The **address space** is further divided into two parts:
  + Computer’s memory space
  + Peripheral’s register memory space



Advantages:-

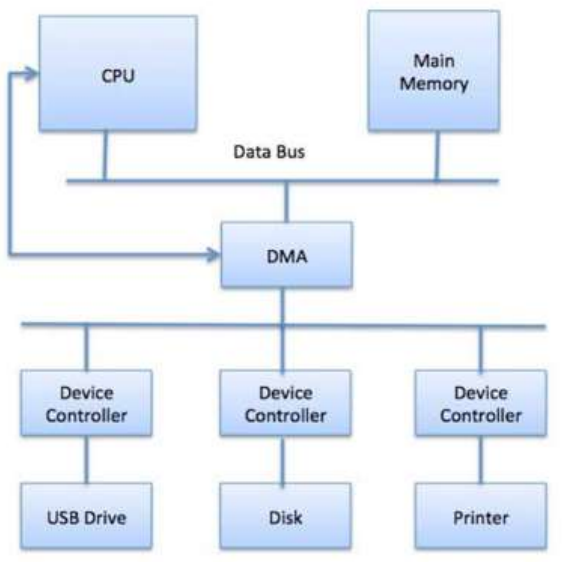
* **Reduced programming efforts**, as peripheral memory can be accessed as regular memory space.
* **No separate commands are required** to program it.

Disadvantages:-

* **Poor memory utilization**, as some of the computer’s memory is reserved for the peripheral (peripheral only has buffer here).
* **Not suitable** with **large** number of **ports**.

**Direct Memory Access (DMA)**

* **Direct transfer** of data between ***main memory*** & ***peripheral***.
* So, CPU intervention is **not** required in this.
* **Multiple bytes of data** can be transferred at once using **DMA**.
* If it involves an **interrupt**, then **CPU** is notified after complete data transfer.
* **Without** **DMA**, **CPU** has to make physical transfer of data.



**Working of DMA**

* When a small part of user program is executed, **CPU** initializes **DMA** using **control signals**.
* **CPU** initializes **DMA** with address of both the ***peripheral*** & ***main memory*** & a ***counter***.
* This small part of user program is executed by **CPU**.
* After this, **DMA** sends request signal to ***peripheral*** for becoming **operable**.
* Then the ***peripheral*** sends acknowledgment signal to **DMA** for that.
* If **DMA** receives acknowledgement, then it enables **HOLD** signal which allows **DMA** to work with ***data bus***.
* Then again it waits for **HOLD ACK** message by the **peripheral**.
* **HOLD ACK** is also known as **HLDA**.
* When **DMA** receives **HLDA**, the data transfer process begins.
* This transfer happens until the ***counter*** (containing variable **"total data transferred"** becomes **zero**.
* **CPU** is **blocked** during this transfer process & is **unblocked** after process completes.

**Modes of DMA**

* **Burst mode:** Transfers **large amount of data** from I/O to memory after receiving **HLDA**.
* **Cycle stealing mode:** DMA forcefully **snatches** control over **bus** from CPU, even without requiring **HLDA**. After transferring data, it gives control **back** to CPU.
* **Blocked mode:** DMA transfers data in **blocks** to memory.

**Merits & Demerits of DMA**

Advantages:-

* **Fast** data transfer.
* Does transfers in **few clock cycles**.

Disadvantages:-

* Expensive
* **No** syncing mechanism to avoid access to **non-updated** data in RAM.
* This is because syncing mechanism **avoids** processes from interfering data.

**Polling v/s Interrupt**

* **Polling** and **interrupt** are two ways to detect a coming **input**.

Polling:-

* Periodically checking **if its time** to perform an I/O operation.
* I/O device puts **status** about if its time for an I/O operation in a ***status register***.
* Then the processor checks the **status** from there & performs accordingly.

Interrupt:-

* A **device controller** places its **interrupt signal** on the bus when calling CPU.
* After CPU receives this signal, it **saves** its own **current state** & calls a required type of **interrupt handler** using ***interrupt vector***.
* **Interrupt vector:** A **matrix** containing **addresses** of various OS processes which perform some **system-level** tasks.
* After being done with the interrupting device, the CPU continues doing what it was doing.

**Dedicated v/s Shared Device**

* **Dedicated device:** Devices that can be shared by only **one process** at a time.
* For example, **printer** can print only one page at a time.
* **Shared device:** Devices that can be shared by **multiple processes** without causing **deadlock**.
* For example, there can be **multiple applications** running in **background** which reside in **hard disk**.

**Goals of I/O Software**

* **Device independence –** I/O software makes I/O devices of same category are treated equally.
* For example, an **application** gets itself stored in **any kind** of **hard disk**.
* **Error handling –** I/O software are **programmed** to **protect** I/O device’s resources against application errors.
* **Buffering –** I/O software are also **programmed** to **handle buffer** that most I/O devices possess.

**I/O Devices**

Interrupt:-

* CPU has ***interrupt request bus*** for receiving **interrupt signals**.
* CPU checks the **presence** of interrupt there after executing **each** instruction.
* Means it responds only after having execute the **current instruction**.

Interrupt handler:-

* **Interrupt handler** detects the **cause** of interrupt & **executes** necessary processes accordingly.
* And then returns from the **interrupt instruction** & to get back and **continue** CPU’s work.

Interrupt instruction’s instruction cycle:-

* Fetch cycle
* Execution cycle
* Interrupt cycle

Interrupt flag:-

* If there was an **interrupt** during execution of an instruction, then after execution it **pushes** the value of **program counter** to **stack** temporarily.
* Now the **interrupt** subprogram is executed until **IRET** instruction comes.
* Then it **pops** it back after the interrupt cycle is over using **IRET** instruction.
* **Vector address:** Entry point address in **interrupt** subprogram.

**Device Driver**

* ***Device driver*** controls the interaction with ***device controllers***.
* It is a software component enabling **OS** to communicate with **hardware devices**.
* This communication is done through **bus**.
* It is also a **chip-based** device which controls signals on the wire of ***serial ports***.
* There can be **platform specific** device drivers.
* **Device data sheet:** A document containing **technical information** about the device.

**Types of Device Drivers**

* Serial port drivers
* Network device drivers
* Flash
* Bus driver

**Buffering**

* It’s a technique by which the device manager allows **slow I/O devices** to perform operations when the process is **idle**.
* **Input buffering:** **Data** in device is **read first** before saving it to the **main memory**.
* **Output buffering:** Device **first saves** data to main memory & then is affected by **input** (if any), while the process is executing.

**Types of Buffering**

* Single buffering
* Double buffering
* Circular buffering
* No buffering