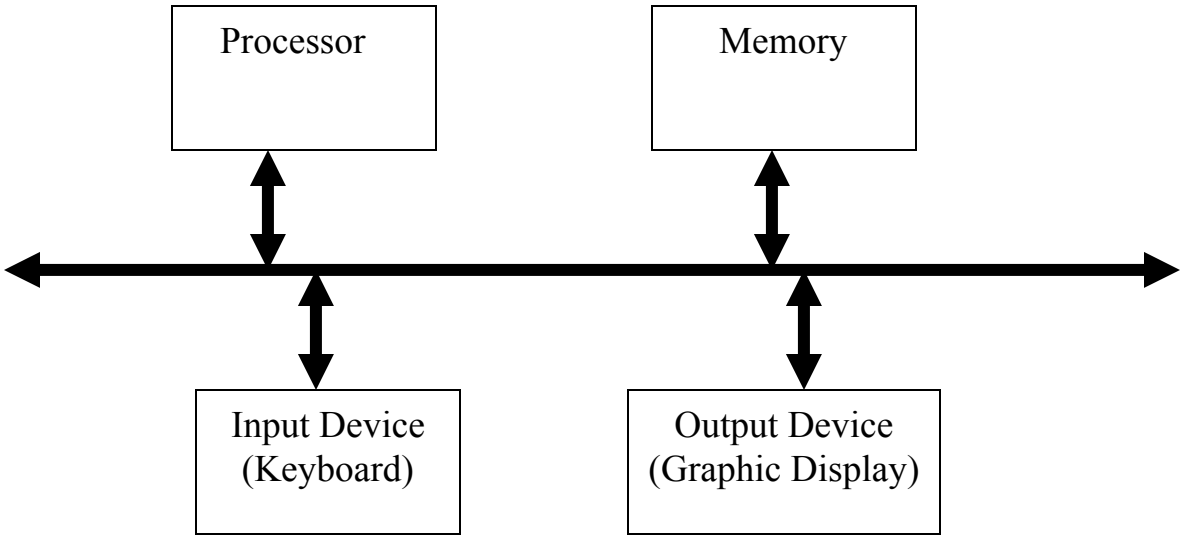
Chapter 8

Input – Output Design and Organization

8.1 Basic Concepts

- The Figure below shows a simple arrangement for connecting the processor and the memory in a given computer system to an input device and an output device.
- A single bus consisting of the required address, data, and control lines is used to connect the system's components.



8.1 Basic Concepts

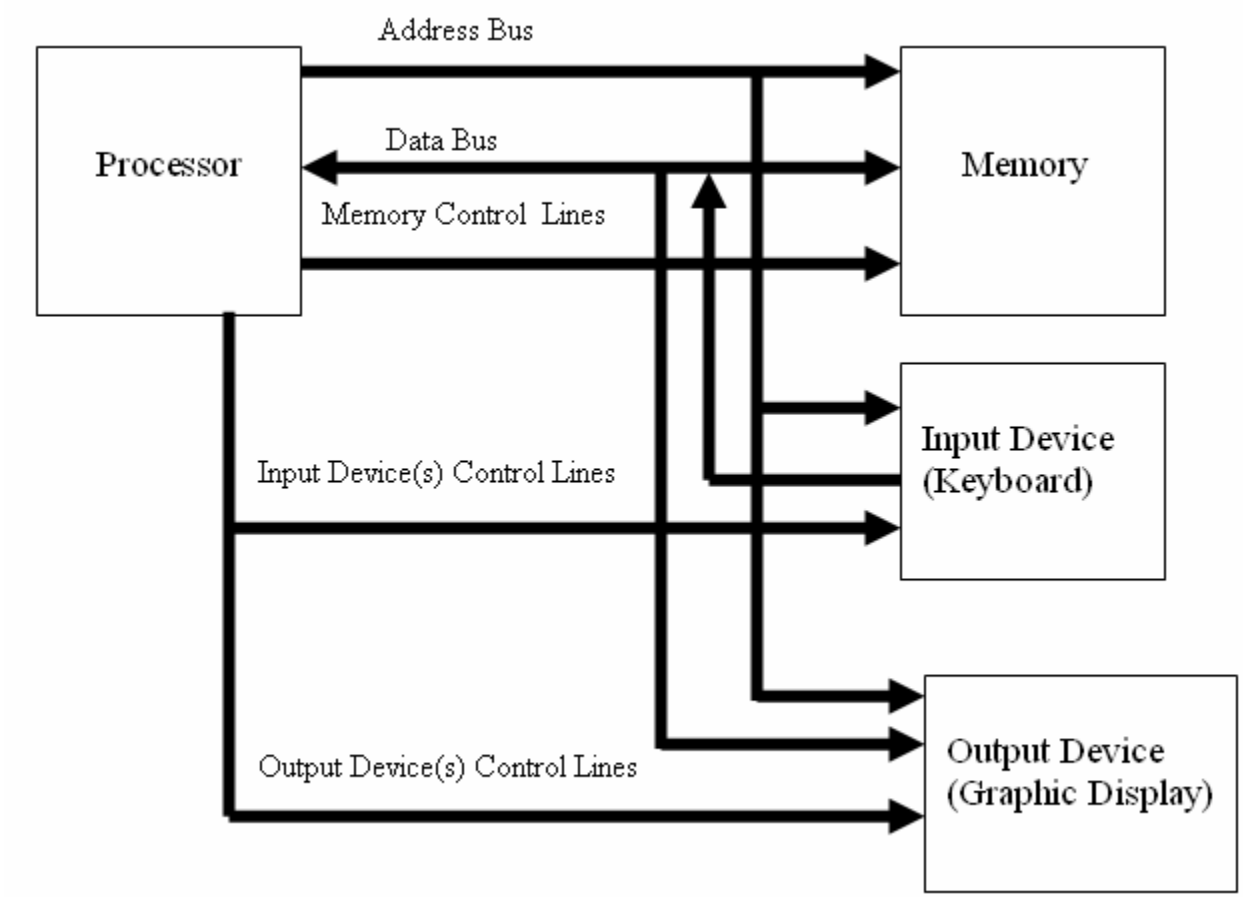
- There exists a big difference in the rate at which a processor can process information and those of input and output devices.
- A simple way of communication between the processor and I/O devices, called the *I/O protocol*, requires the availability of the input and output registers.
- A mechanism according to which the processor can address those input and output registers must be adopted.
- More than one arrangement exists to satisfy the above mentioned requirements.

8.1 Basic Concepts

- In the first arrangement, I/O devices are assigned particular addresses, isolated from the address space assigned to the memory.
- The Execution of an *Input Instruction* at an input device address will cause the character stored in the *Input Register* of that device to be transferred to a specific register in the CPU.
- Similarly, the execution of an *Output Instruction* at an output device address will cause the character stored in a specific register in the CPU to be transferred to the *Output Register* of that output device.
- This arrangement is called *Shared I/O.*

8.1 Basic Concepts

- Shared I/O arrangement:



8.1 Basic Concepts

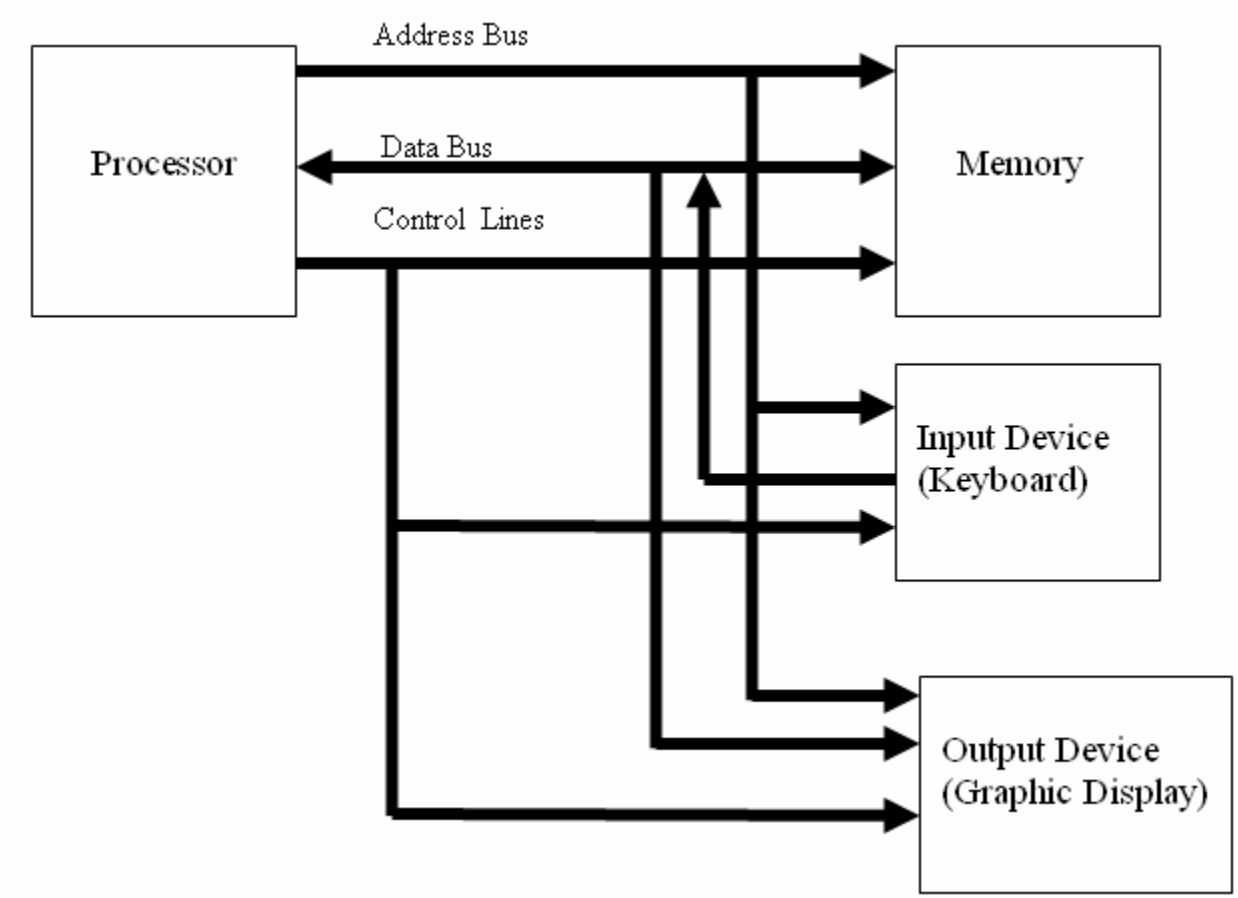
- The main advantage of the shared I/O arrangement is the separation between the memory address space and that of the I/O devices.
- Its main disadvantage is the need to have special *Input* and *Output Instructions* in the processor instruction set.
- The Shared I/O arrangement is mostly adopted by *Intel.*

8.1 Basic Concepts

- The second possible I/O arrangement is to deal with *Input* and *Output Registers* as if they are regular memory locations.
- In this case, a *Read* operation from the address corresponding to the *Input Register* of an input device, e.g., *Read Device6,* is equivalent to performing an *Input Operation* from the input register in Device #6.
- Similarly, a *Write* operation to the address corresponding to the *Output Register* of an output device, e.g., *Write Device9,* is equivalent to performing an *Output Operation* into the output register in Device #9.
- This arrangement is called *Memory-Mapped I/O*.

8.1 Basic Concepts

- Memory-mapped I/O arrangement:

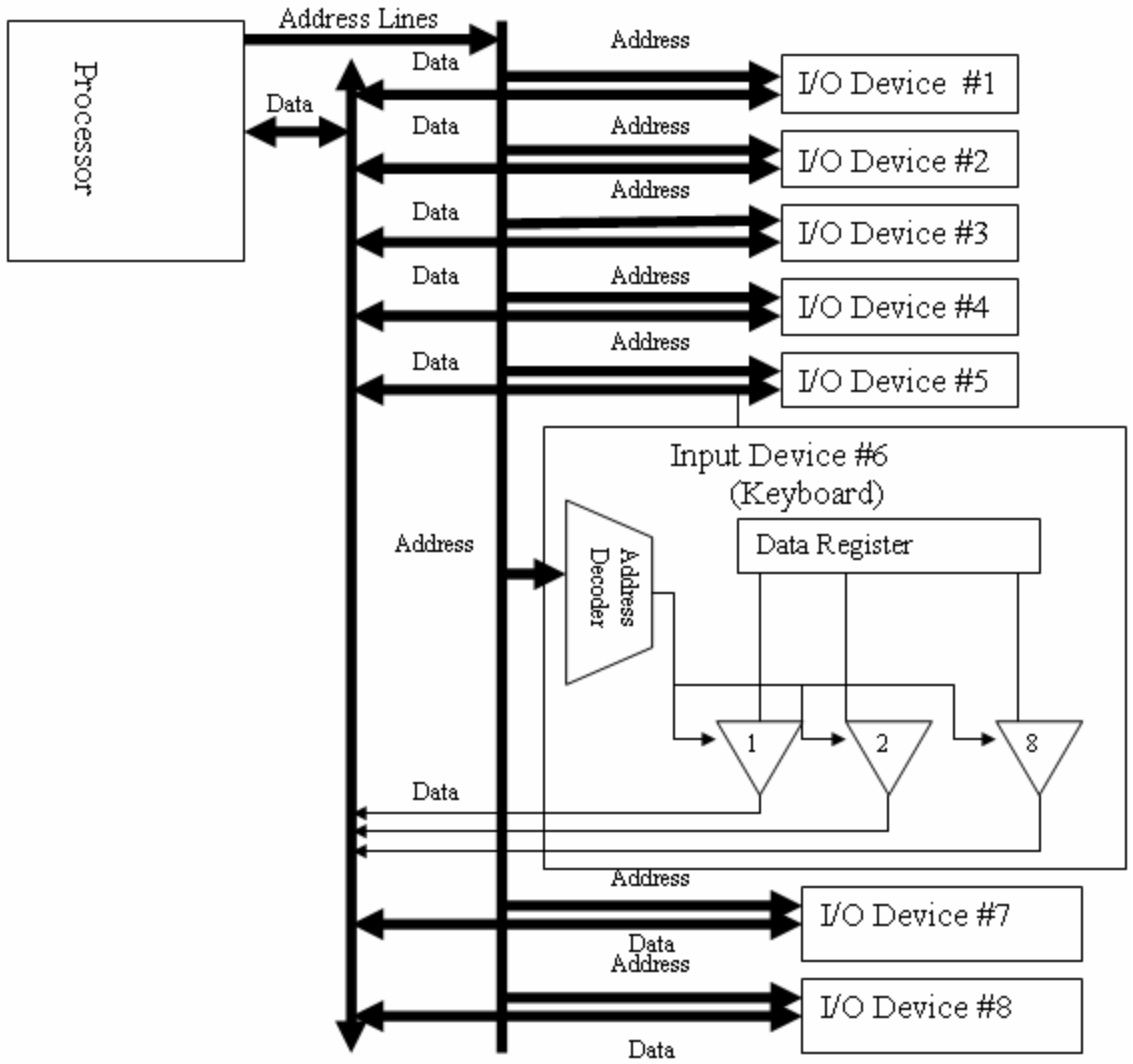


8.1 Basic Concepts

- The main advantage of the memory-mapped I/O is the use of the read and write instructions of the processor to perform the input and output operations, respectively.
- It eliminates the need for introducing special I/O instructions.
- The main disadvantage of the memory-mapped I/O is the need to reserve a certain part of the memory address space for addressing I/O devices, i.e., a reduction in the available memory address space.
- The memory-mapped I/O has been mostly adopted by Motorola.

8.2 Programmed I/O

- Example of an 8 I/O device connection to a processor



8.2 Programmed I/O

- The following protocol steps (program) have to be followed:
  - The processor executes an input instruction from device 6, e.g., *INPUT 6.* The effect of executing this instruction is to send the device number to the address decoder circuitry in each input device in order to identify the specific input device to be involved. In this case, the output of the decoder in Device #6 will be enabled, while the outputs of all other decoders will be disabled.
  - The buffers (in the figure we assumed that there are 8 such buffers) holding the data in the specified input device (Device #6) will be enabled by the output of the address decoder circuitry.
  - The data output of the enabled buffers will be available on the data bus.
  - The instruction decoding will gate the data available on the data bus into the input of a particular register in the CPU, normally the *accumulator*.

8.2 Programmed I/O

- Output operations can be performed in a similar way.
  - The only difference will be the direction of data transfer, which will be from a specific CPU register to the output register in the specified output device.
- I/O operations performed in this manner are called *Programmed I/O*. They are performed under the CPU control.
- A complete instruction fetch, decode, and execute cycle will have to be executed for every input and every output operation. Programmed I/O is useful in cases whereby one character at a time is to be transferred.
- Although simple, programmed I/O is slow.

8.2 Programmed I/O

- A mechanism should be adopted in order to handle the substantial speed difference between I/O devices and the processor.
  - For example, ensure that a character sent to the output register of an output device, such as a screen, is not over written by the processor (due to the processor's high speed) before it is displayed and that a character available in the input register of a keyboard is read only once by the processor.
- A mechanism that can be implemented requires the availability of a *Status Bit* (B_{in}) in the interface of each input device and a *Status Bit* (B_{in}) in the interface of each output device.

8.2 Programmed I/O

- Whenever an input device has a character available in its input register, it indicates that by setting B_{in} =1.
- A program in the processor can be used to continuously monitor B_{in}.
- When the program sees that B_{in} =1, it will interpret that to mean a character is available in the input register of that device.
- Reading such character will require executing that protocol explained before.

8.2 Programmed I/O

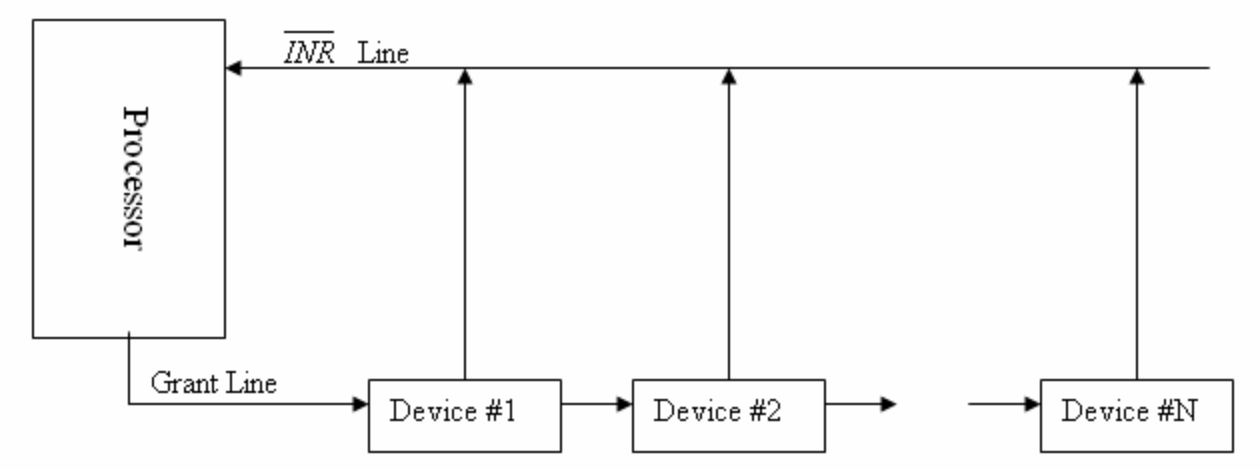
- Whenever the character is read, the program can reset B_{in} = 0, thus avoiding multiple read of the same character.
- In a similar manner, the processor can deposit a character in the output register of an output device only when B_{in} = 0.
- It is only after the output device has displayed the character that it sets B_{in} =1, indicating to the monitoring program that the output device is ready to receive the next character.
- The process of checking the status of I/O devices in order to determine their readiness for receiving and/or sending characters, is called *Software I/O Polling.*
- In addition to the I/O polling, two other mechanisms can be used to carry out I/O operations:
  - Interrupt-driven I/O and
  - Direct Memory Access (DMA).

8.3 Interrupt-Driven I/O

- Interrupt Hardware
  - Computers are provided with *Interrupt Hardware* capability in the form of specialized *Interrupt Lines* to the processor.
  - These lines are used to send interrupt signals to the processor.
  - In the case of I/O, there exists more than one I/O device. The processor should be provided with a mechanism that enables it to handle simultaneous interrupt requests and to recognize the interrupting device.
    - Two basic schemes can be implemented to achieve this task:
      - *Daisy Chain Bus Arbitration* (DCBA) and
      - *Independent Source Bus Arbitration* (ISBA)

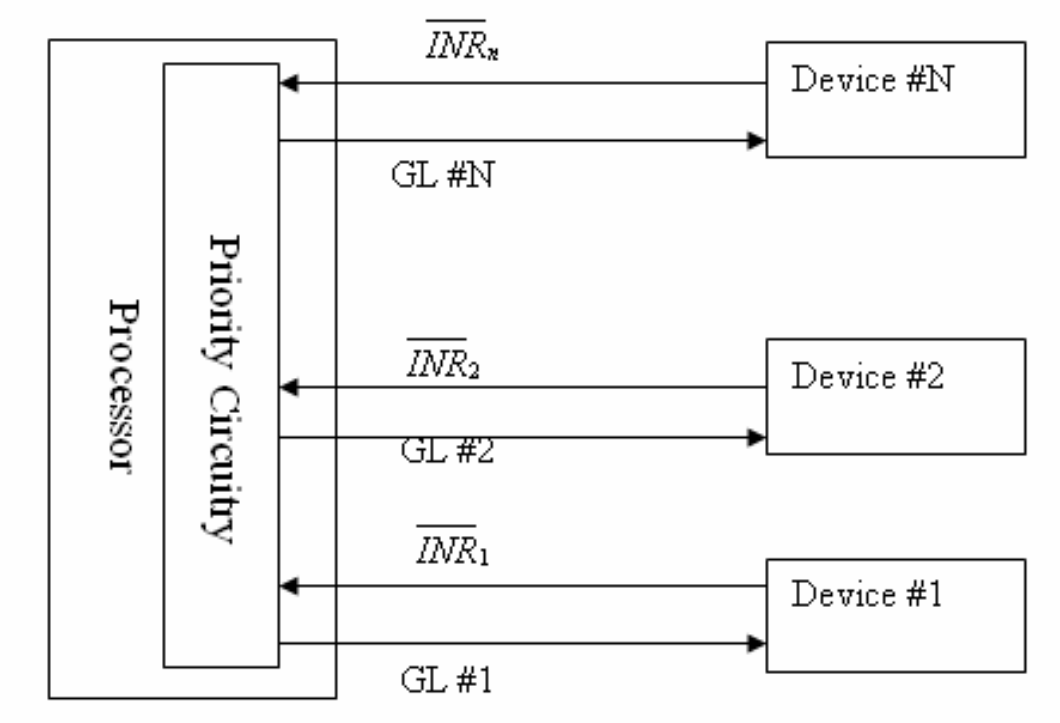
8.3 Interrupt-Driven I/O

- Interrupt Hardware: Daisy Chain Interrupt Arrangement



8.3 Interrupt-Driven I/O

- Interrupt Hardware: Independent Interrupt Arrangement

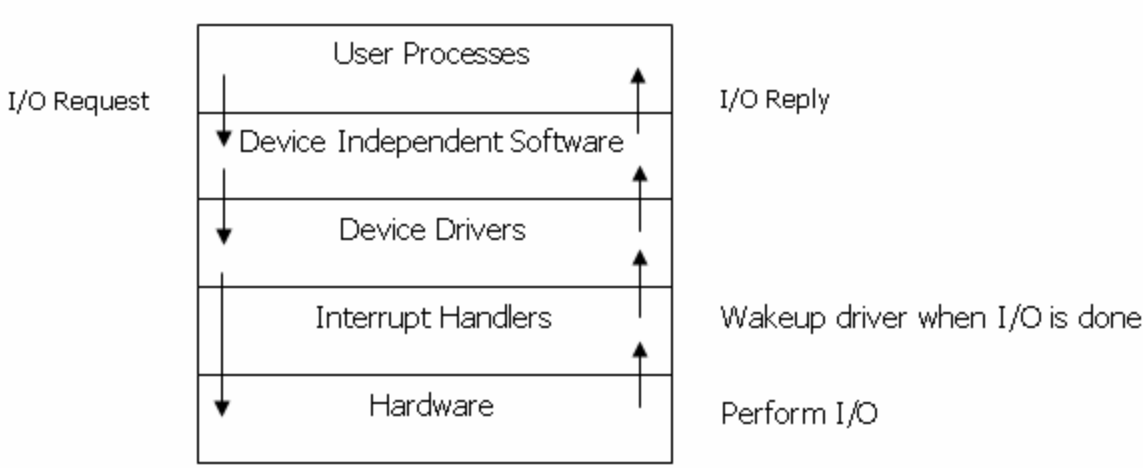


8.3 Interrupt-Driven I/O

- Interrupt in Operating Systems
  - When an interrupt occurs, the operating system gains control.
  - The operating system saves the state of the interrupted process, analyzes the interrupt, and passes control to the appropriate routine to handle the interrupt.
    - There are several types of interrupts including I/O interrupts.
    - An I/O interrupt notifies the operating system that an I/O device has completed or suspended its operation and needs some service from the CPU.
  - To process an interrupt, the context of the current process must be saved and the interrupt handling routine must be invoked. This process is called *context switching*.
  - A process' context has two parts:
    - Processor context: is the state of the CPU's registers including program counter (PC), program status words (PSWs), and other registers.
    - Memory context: is the state of the program's memory including the program and data.

8.3 Interrupt-Driven I/O

- Interrupt in Operating Systems
  - The layers of software involved in I/O operations:



  - First, the program issues an I/O request via an I/O call.
  - The request is passed through to the I/O device.
  - When the device completes the I/O, an interrupt is sent and the interrupt handler is invoked.
  - Eventually, control is relinquished back to the process that initiated the I/O.

8.4 Direct Memory Access (DMA)

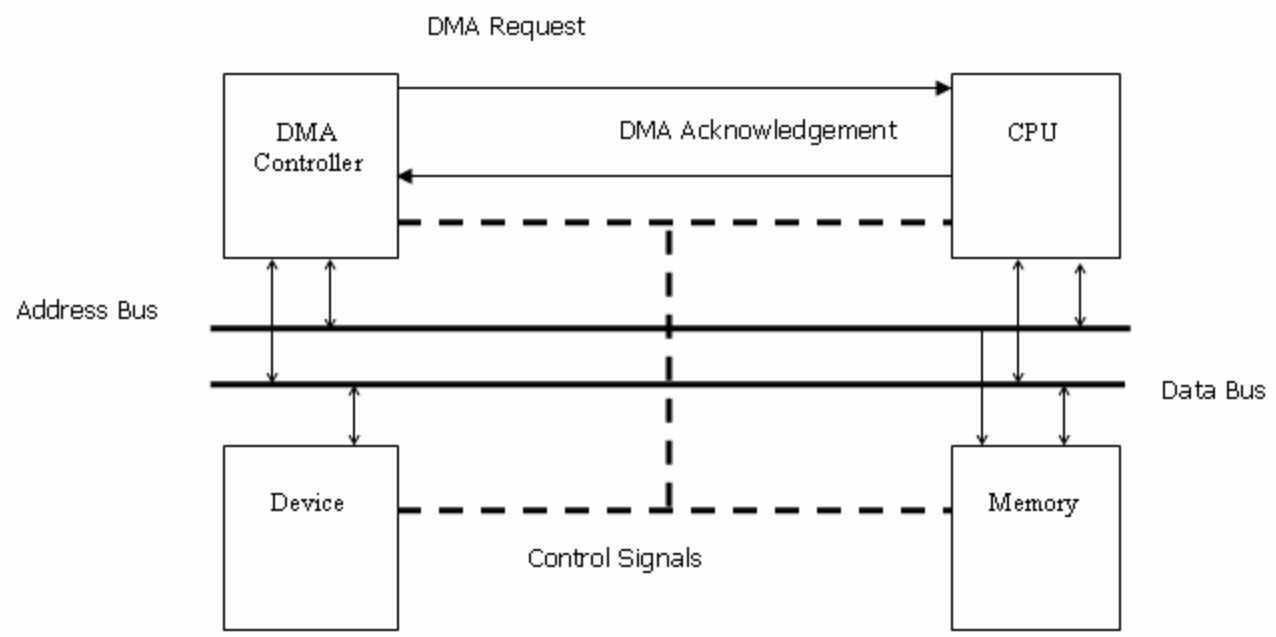
- The main idea of Direct Memory Access (DMA) is to enable peripheral devices to cut out the "middle man" role of the CPU in data transfer.
- It allows peripheral devices to transfer data directly from and to memory without the intervention of the CPU.
- Having peripheral devices access memory directly would allow the CPU to do other work, which would lead to improved performance, especially in the cases of large transfers.
- The DMA controller is a piece of hardware that controls one or more peripheral devices.
  - It allows devices to transfer data to or from the system's memory without the help of the processor.

8.4 Direct Memory Access (DMA)

- In a typical DMA transfer, some event notifies the *DMA controller* that data needs to be transferred to or from memory.
- Both the DMA and CPU use memory bus and only one or the other can use the memory at the same time.
- The DMA controller then sends a request to the CPU asking its permission to use the bus.
- The CPU returns an acknowledgment to the DMA controller granting it bus access.
- The DMA can now take control of the bus to independently conduct memory transfer.
- When the transfer is complete, the DMA relinquishes its control of the bus to the CPU.
- Processors that support DMA provide one or more input signals that the bus requester can assert to gain control of the bus and one or more output signals that the CPU asserts to indicate it has relinquished the bus.

8.4 Direct Memory Access (DMA)

- DMA controller shares the CPU's memory bus:



8.4 Direct Memory Access (DMA)

- The following steps summarize the DMA Operations:
  - DMA Controller initiates data transfer.
  - Data is moved (increasing the address in memory, and reducing the count of words to be moved).
  - When word count reaches zero, the DMA informs the CPU of the termination by means of an interrupt.
  - The CPU regains access to the memory bus.

8.5 Buses

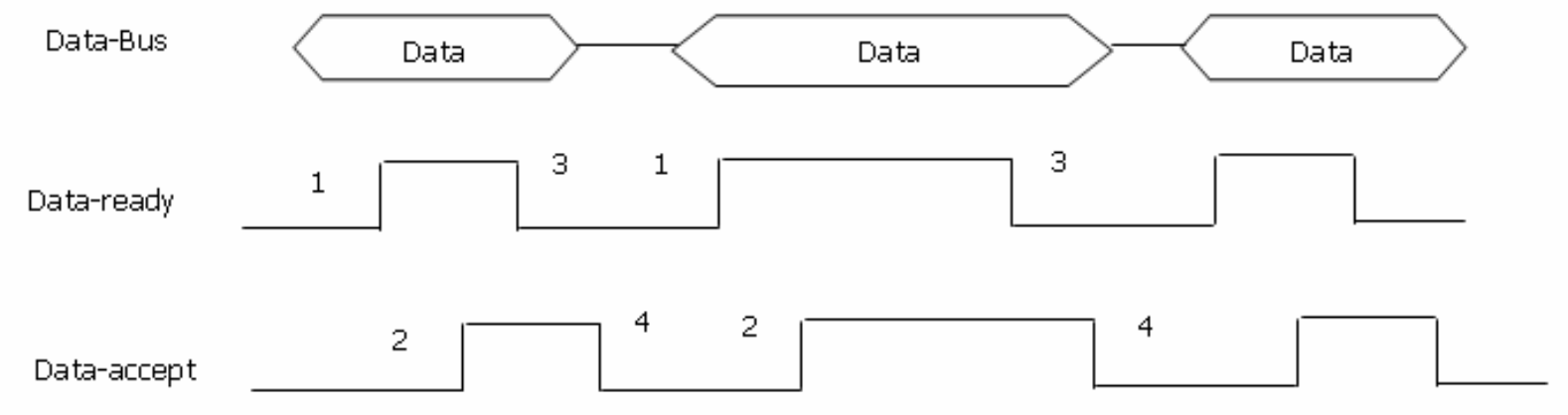
- Synchronous Buses
  - In synchronous busses, the steps of data transfer take place at fixed clock cycles.
  - Everything is synchronized to the bus clock and clock signals are made available to both master and slave.
  - A transfer may take multiple bus cycles depending on the speed parameters of the bus and the two ends of the transfer.
  - Synchronous buses are simple and easily implemented.
  - However, when connecting devices with varying speeds to a synchronous bus, the slowest device will determine the speed of the bus.
  - Also, the synchronous bus length could be limited to avoid clock-skewing problem.

8.5 Buses

- Asynchronous Buses
  - There are no fixed clock cycles in asynchronous busses.
  - Handshaking is used instead.
  - The master asserts the data-ready line (point 1 in the figure) until it sees a data-accept signal.
  - When the slave sees data-ready signal, it will assert the data-accept line (point 2 in the figure).
  - The rising of the data-accept line will trigger the falling of the data-ready line and the removal of data from the bus.
  - The falling of the data-ready line (point 3 in the figure) will trigger the falling of the data-accept line (point 4 in the figure).
  - This handshaking, which is called fully interlocked, is repeated until the data is completely transferred.
  - Asynchronous bus is appropriate for different speed devices.

8.5 Buses

- Asynchronous Buses

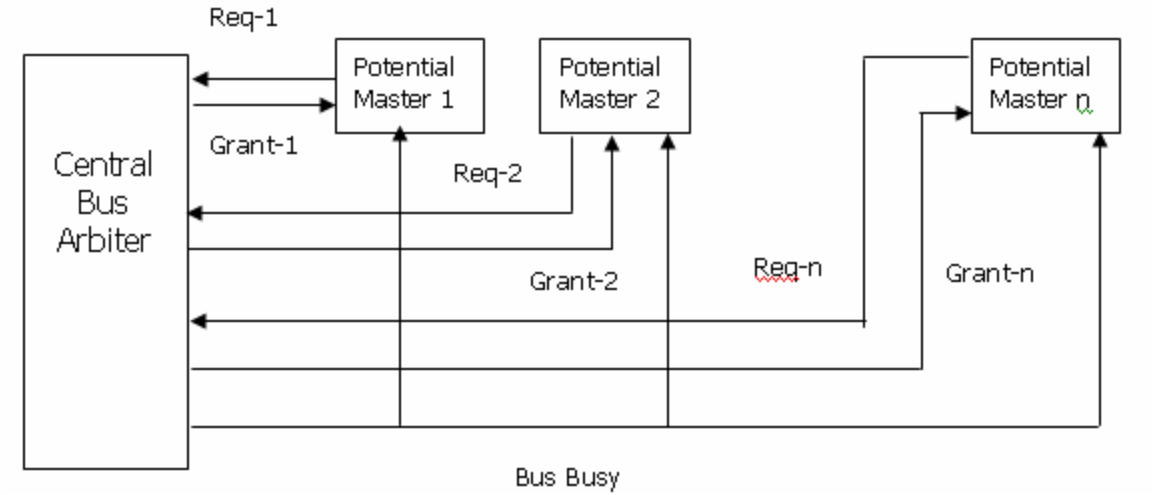


8.5 Buses

- Bus Arbitration
  - Centralized Arbitration
    - In centralized arbitration schemes, a single arbiter is used to select the next master.
    - A simple form of centralized arbitration uses a bus request line, a bus grant line, and a bus busy line.
    - Each of these lines is shared by potential masters, which are daisy-chained in a cascade.
    - each of the potential masters can submit a bus request at any time.
    - A fixed priority is set among the masters from left to right.

8.5 Buses

- Bus Arbitration
  - Centralized Arbitration
    - When a bus request is received at a central bus arbiter, it issues a bus grant by asserting the bus grant line.
    - When the potential master that is closest to the arbiter (Potential master 1) sees the bus-grant signal, it checks to see if it had made a bus request.
    - If yes, it takes over the bus and stops propagation of the bus grant signal any further.
    - If it has not made a request, it will simple turn the bus grant signal to the next master to the right (Potential master 2), and so on.
    - When the transaction is complete, the busy line is deasserted.

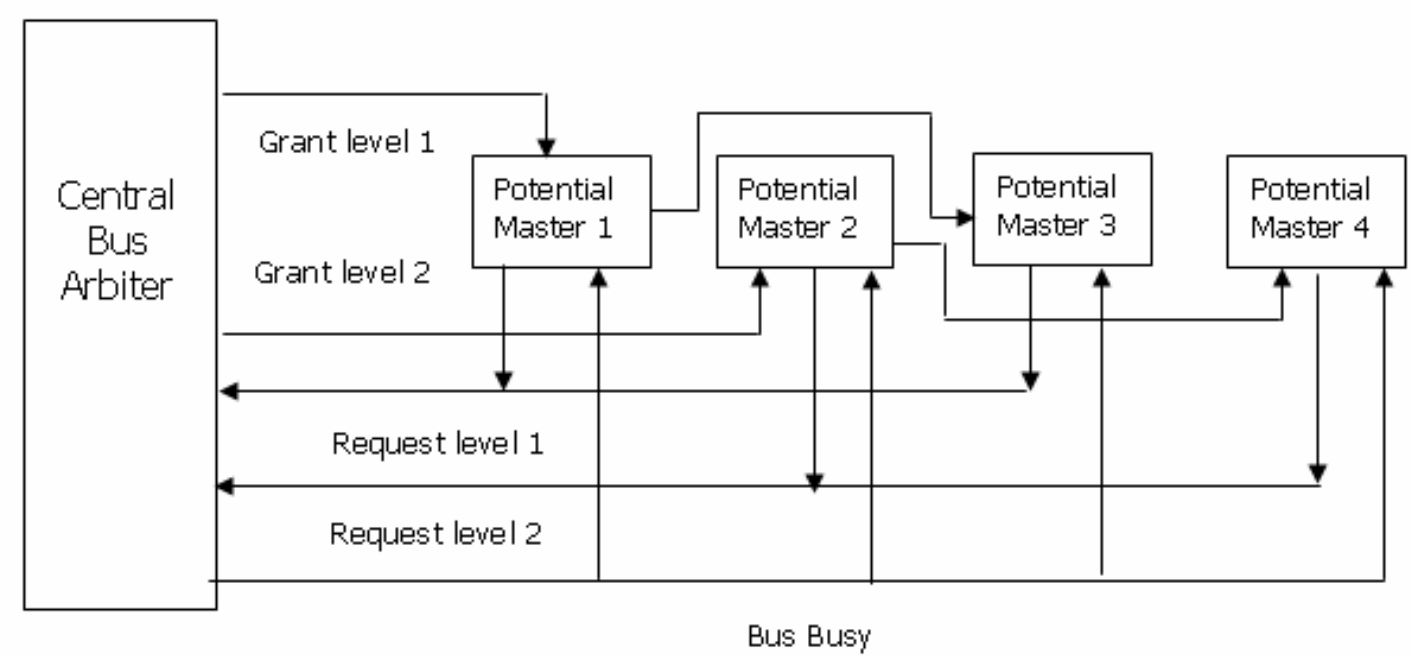


8.5 Buses

- Bus Arbitration
  - Centralized Arbitration
    - Instead of using shared request and grant lines, multiple bus request and bus grant lines can be used.
    - In one scheme, each master will have its own independent request and grant line.
    - The central arbiter can employ any priority based or fairness based tiebreaker.
    - Another scheme allows the masters to have multiple priority levels.
      - For each priority level, there is a bus request and a bus grant lines.
        - » Within each priority level, daisy chain is used.
        - » In this scheme, each device is attached to the daisy chain of one priority level.
        - » If the arbiter receives multiple bus requests from different levels, it grants the bus to the level with the highest priority.
        - » Daisy chaining is used among the devices of that level.

8.5 Buses

- Bus Arbitration
  - Centralized Arbitration
    - » Four devices included in two priority levels.
    - » Potential Master 1 and Potential master 3 are daisy chained in level 1 and Potential Master 2 and Potential Master 4 are daisy chained in level 2.



8.5 Buses

- Bus Arbitration
  - Decentralized Arbitration
    - In decentralized arbitration schemes, priority based arbitration is usually used in a distributed fashion.
    - Each potential master has a unique arbitration number, which is used in resolving conflicts when multiple requests are submitted.
    - For example, a conflict can always be resolved in favor of the device with the highest arbitration number.
    - The question now is how to determine which device has the highest arbitration number?
      - One method is that a requesting device would make its unique arbitration number available to all other devices.
      - Each device compares that number with its own arbitration number.
      - The device with the smaller number is always dismissed.
      - Eventually, the requester with the highest arbitration number will survive and be granted bus access.

8.6 Input-Output Interfaces

- An interface is a data path between two separate devices in a computer system.
- Interface to buses can be classified based on the number of bits that are transmitted at a given time to serial versus parallel ports.
- In a serial port, only 1 bit of data is transferred at a time.
- Mice and modems are usually connected to serial ports.
- A parallel port allows more than 1 bit of data to be processed at once.
- Printers are the most common peripheral devices connected to parallel ports.

8.7 Summary

- One of the major features in a computer system is its ability to exchange data with other devices and to allow the user to interact with the system.
- This chapter focused on the I/O system and the way the processor and the I/O devices exchange data in a computer system.
- The chapter described three ways of organizing I/O:
  - Programmed I/O
    - The CPU handles the transfers, which take place between registers and the devices.
  - Interrupt driven I/O
    - CPU handles data transfers and an I/O module is running concurrently.
  - DMA.
    - Data are transferred between memory and I/O devices without intervention of the CPU.

8.7 Summary

- We also studied two methods for synchronization:
  - Polling
    - the processor polls the device while waiting for I/O to complete. Clearly processor cycles are wasted in this method.
  - Interrupts
    - Processors are free to switch to other tasks during I/O.
    - Devices assert interrupts when I/O is complete.
    - Interrupts incur some delay penalty with them.
  - The chapter also covered busses and interfaces.