Module 5- Accessing I/O devices

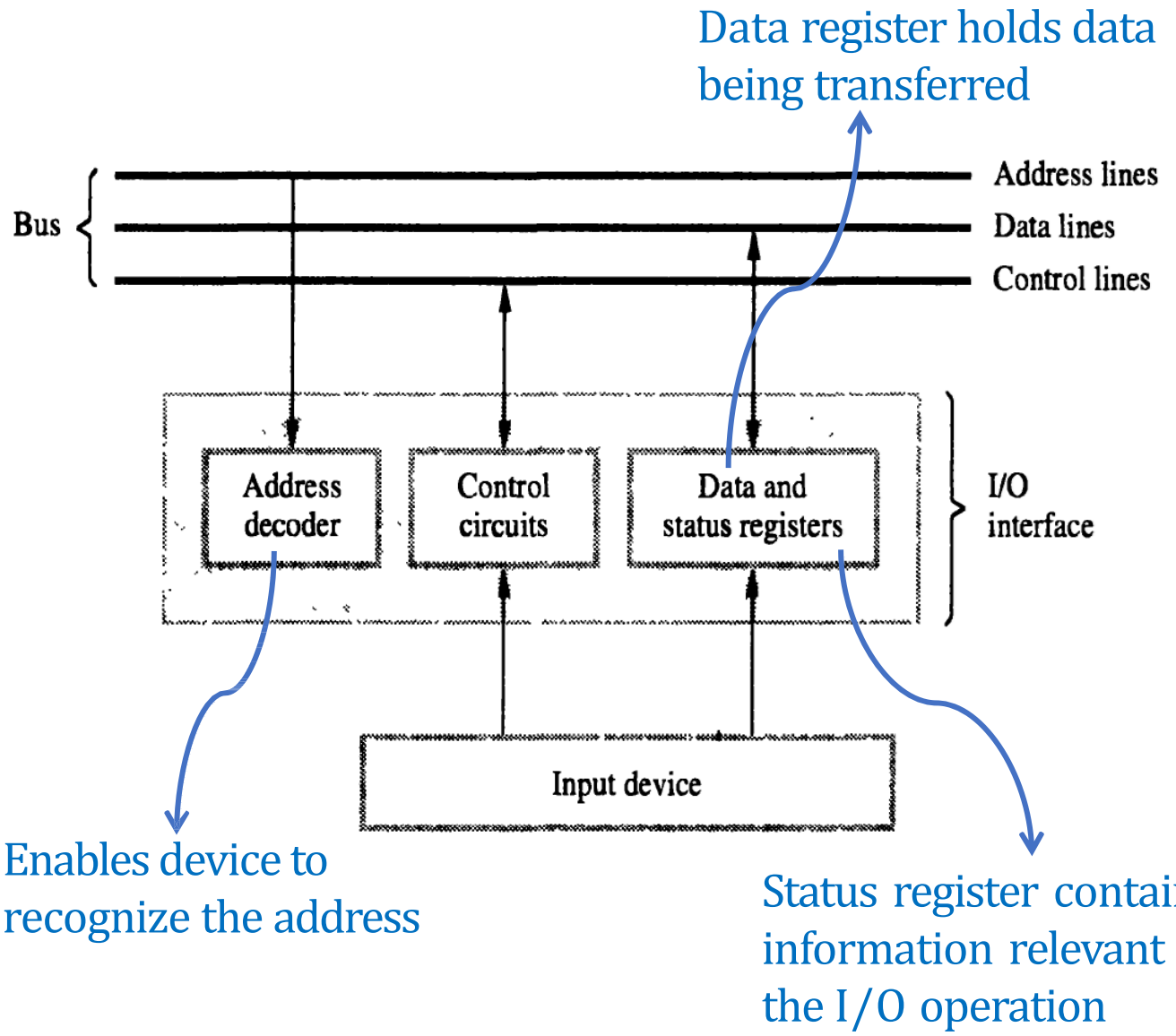
Part 1- Accessing I/O, Interrupts, DMA

Accessing I/O Devices

- Simple arrangement to connect I/O devices to a computer – single bus
  - Enables all devices connected to it to exchange information
  - Consists of three sets of lines – address, data, control
- Each device is assigned a unique set of addresses.
- The I/O devices and the memory may share the same address space (memory-mapped I/O) or may be have separate spaces.
  - Same address space → any instruction that can access the memory can be used to access I/O devices as well (e.g.: Move DATAIN, R0)
    - Simpler software
  - Different address space → special In and Out instructions for I/O transfers
    - Second approach – I/O devices deal with fewer address lines

Accessing I/O Devices

- When the processor places a particular address on the address lines, the device that recognizes this address responds to the commands (read/write) sent over the command lines.
- Data is transferred over the data lines.
- I/O interface unit – address decoder, control circuits, data and status registers



Accessing I/O Devices

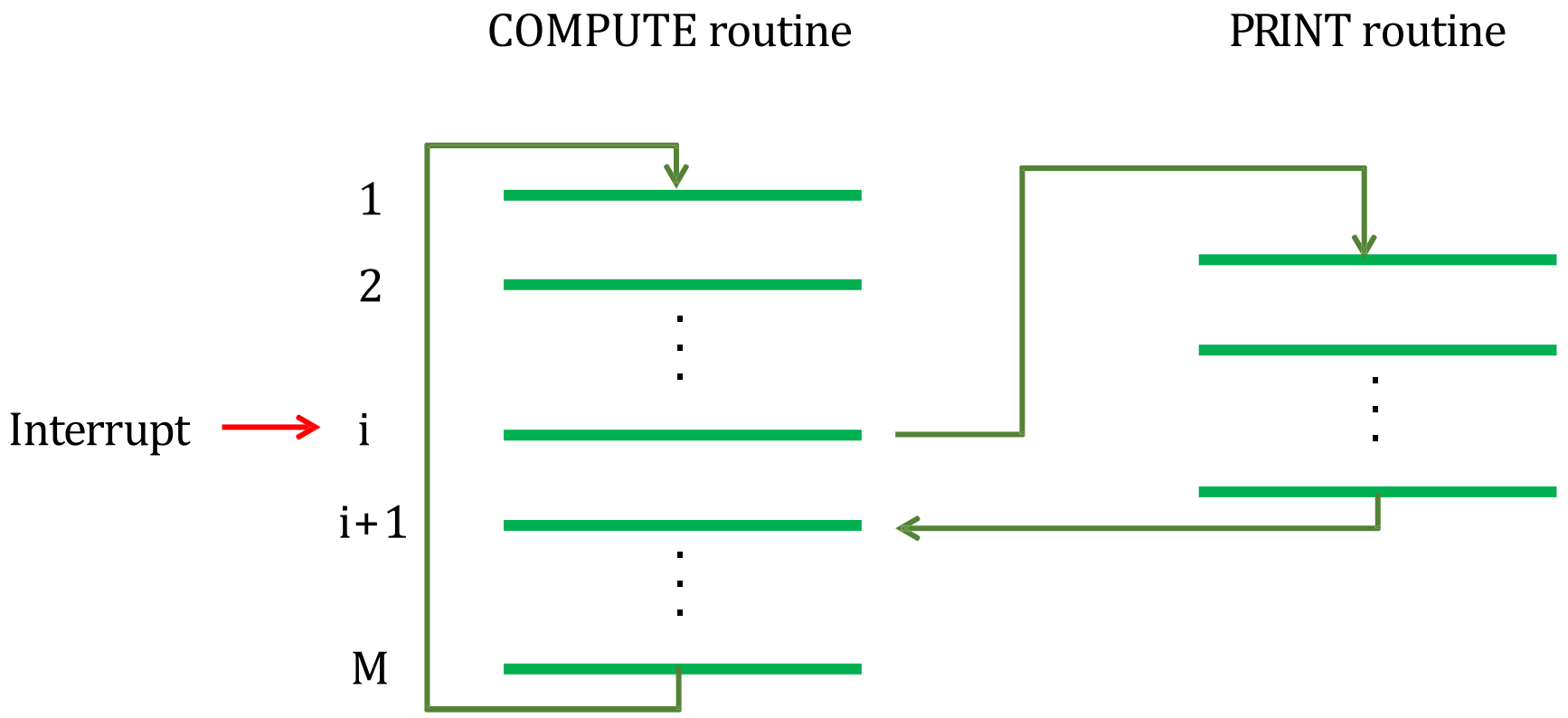
- Methods for communicating with I/O devices/implementing I/O operations:
  - Program-controlled I/O
    - Processor repeatedly checks a status flag (e.g.: SIN or SOUT) to achieve synchronization between the processor and the I/O device. – Polling
  - Interrupts
    - Involves the I/O device sending a special signal over the bus whenever it is ready for data transfer
  - Direct memory access (DMA)
    - Used for high-speed I/O devices
    - Involves data transfer without continuous involvement of the processor

Interrupts

Interrupts

- In program-controlled I/O, the processor is put through a waiting period, where it constantly tests the device status and does not perform any useful task.
- To allow the processor to perform some other task till the I/O device is ready, the I/O device could alert the processor when ready
  - Can be accomplished by sending a hardware signal called interrupt
- One of the bus control lines allotted for this – interrupt-request line
- Processor need not continuously check the status of the I/O devices.
  - Waiting periods eliminated
- Routine executed in response to an interrupt – Interrupt Service Routine (ISR)

Interrupt - Example



Handling Interrupts

- Once an interrupt request is received, the processor must stop executing the current program and execute an interrupt service routine (PRINT routine in the example).
- If the interrupt request is received when the i^{th} instruction is being executed, the processor does the following:
  - It first completes the execution of the i^{th} instruction.
  - It stores the current contents of the PC in a special register (Link register)or a stack.
  - It loads the PC with the address of the first instruction of the ISR.
- After the ISR has finished executing, the return-from-interrupt signal at the end of the ISR reloads the previous content of the PC from the temporary storage.

Handling Interrupts

- Apart from storing the contents of the PC, condition codes and contents of registers used by both the interrupted program and the ISR may be saved and restored.
- Task of saving and restoring can be done automatically by the processor or by program instructions.
- The process of saving and restoring is time-consuming (involves memory transfers) → Most modern processors only save and restore the bare minimum, typically PC and status register.
- Remaining information if required can be saved by program instructions at the beginning of the ISR.
- Time between when an interrupt request is received and ISR execution starts – interrupt latency

Enabling and Disabling Interrupts

- Programmer must be able to control the events that take place during program execution.
- Arrival of interrupts may alter the sequence envisaged by the programmer.
- Therefore, there is a facility to enable and disable interrupts as desired.
- Processor might have to ignore interrupts in certain cases.
- Examples:
  - In the Compute-Print example, interrupt request from the printer should only be accepted if there is content to be printed.
  - The ISR may change data used by a program being executed, in which case a particular sequence of instructions must be executed without interruption.

Enabling and Disabling Interrupts

- When a device activates the interrupt-request signal, it keeps this signal activated, till it learns that the processor has accepted its request.
- It is essential to ensure that the active request signal is not interpreted as successive interruptions.
- Three ways for the processor to handle the above:
  - Ignore interrupt request till the ISR has started executing and the let the first instruction of the ISR be an interrupt-disable and the last be an interrupt-enable instruction.
  - Save the contents of PC and status registers and disable the interrupt flag in the status register, then start the ISR. When the registers are restored after the ISR, interrupt flag will automatically be enabled.
  - Make the interrupt-handling circuit respond only to the leading edge of the signal – edge-triggered

Interrupt Handling - Summarized

- Sequence of steps involved in handling interrupt from a single device:
  - The device raises an interrupt request.
  - The processor interrupts the program currently being executed.
  - PC and status register contents are stored.
  - Interrupts are disabled.
  - Device is informed that its request has been recognized.
  - Device deactivates interrupt request signal.
  - Action requested is performed by the interrupt service routine.
  - Interrupts are enabled, PC and status register are restored, and interrupted program is resumed.

Handling Multiple Devices

- How can the processor recognize the device requesting an interrupt?
- How will it know the starting address of the interrupt service routine?
- Can a second device interrupt while the first is being serviced?
- How should simultaneous interrupt requests be handled?

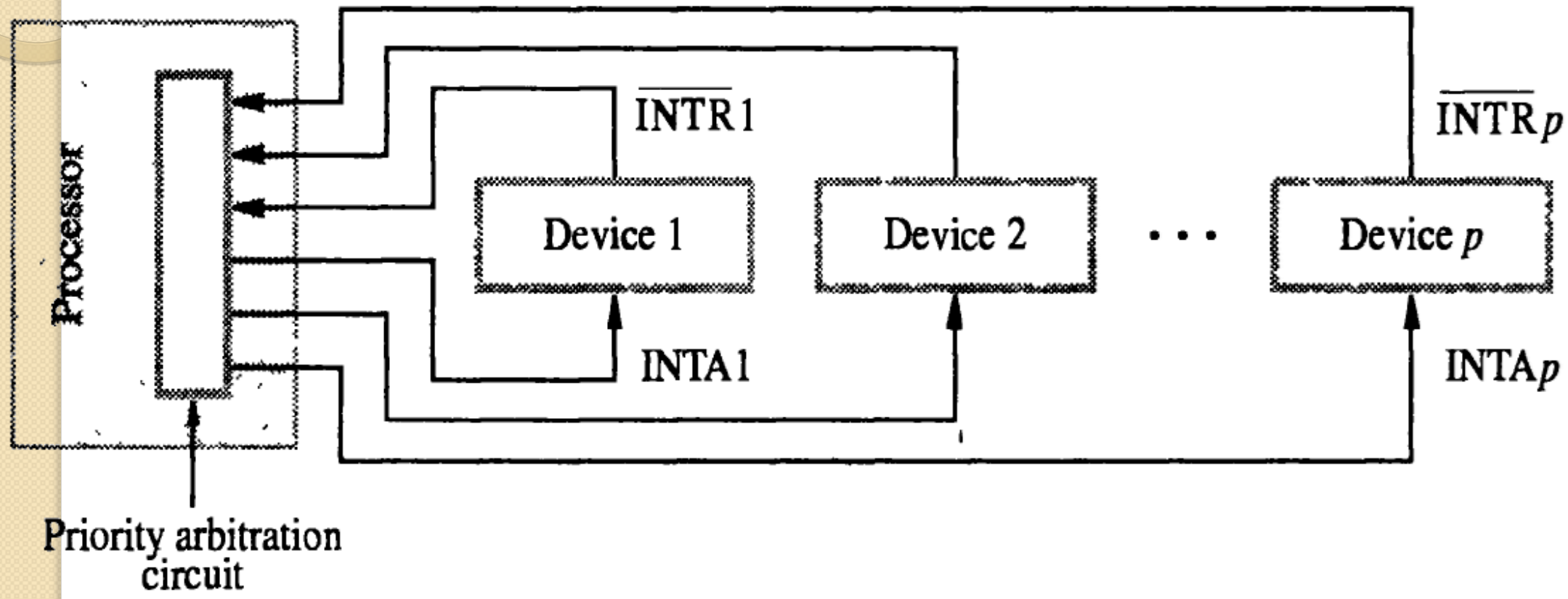
Vectored Interrupts

- How can the processor recognize the device requesting an interrupt?
- How will it know the starting address of the interrupt service routine?
- Polling all the devices – Time consuming
- Vectored Interrupts
  - Device requesting interrupt should identify itself.
  - Should send a special code, which can also represent the address of the interrupt service routine
    - Code length – typically 4 to 8 bits, remaining bits identified by the processor

Interrupt Nesting

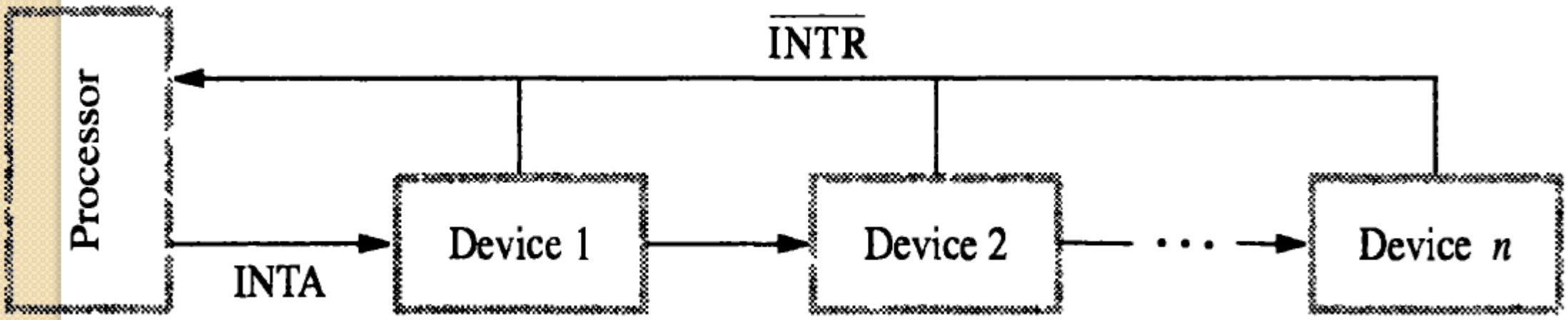
- Can a second device interrupt while the first is being serviced?
- Interrupts can be disabled when one interrupt is being serviced. However, certain interrupt requests may have to be attended to immediately. (E.g.: real-time clock)
- Assign priority
  - Interrupt-request from a high-priority device should be accepted when the processor is servicing another request from a lower-priority device.
  - A priority level can be assigned to the processor, which will be priority level of the program currently being executed.
    - This will disable interrupts from devices that have the same or lower priority.

Interrupt Nesting



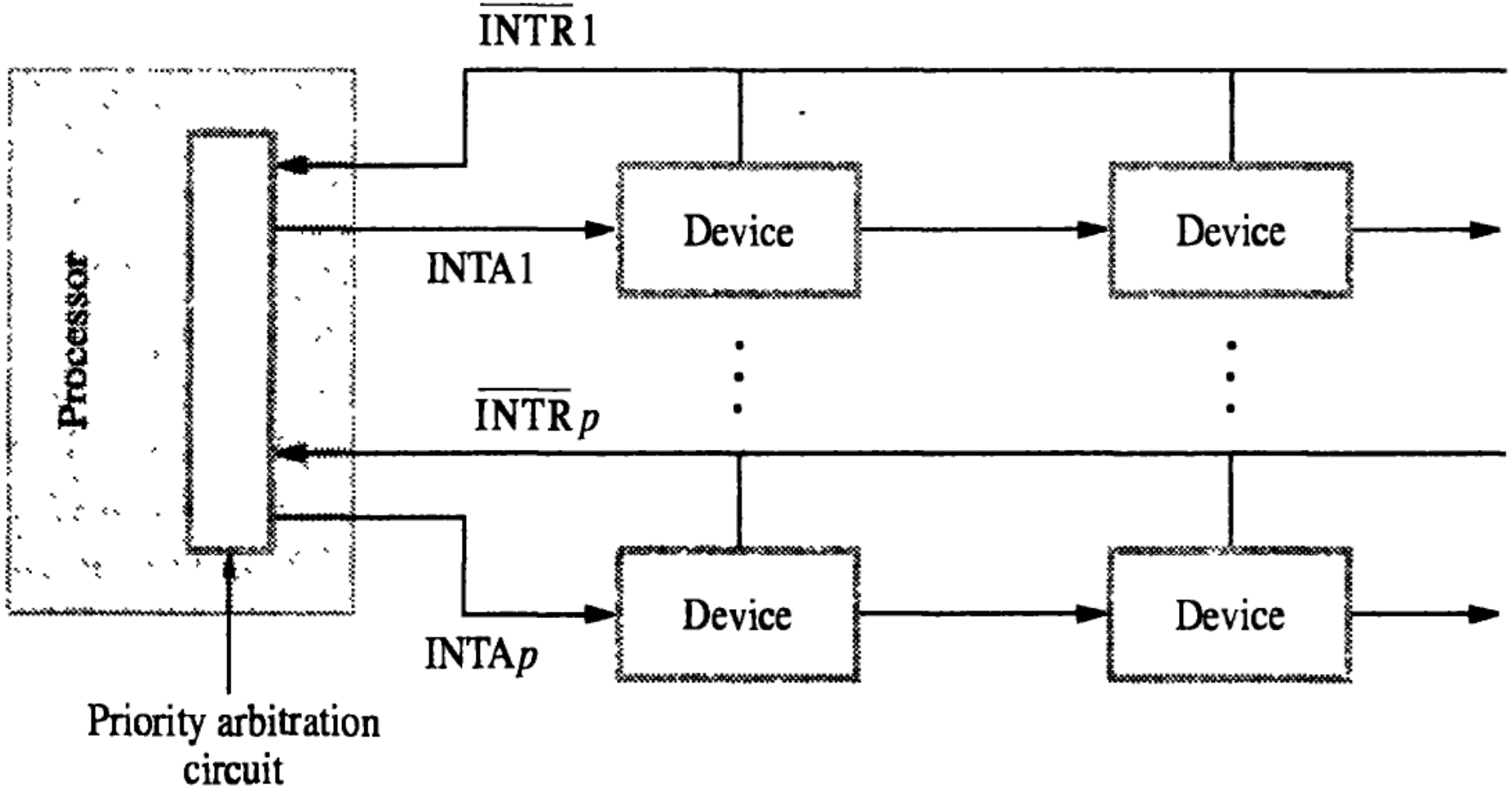
Simultaneous Requests

- How should simultaneous interrupt requests be handled?
- Priority can be assigned, or polling can be used
- Connect interrupt acknowledge line in a daisy-chain fashion
  - Common interrupt-request line (INTR) but acknowledgement (INTA) propagates serially through the devices (daisy-chain fashion).
  - Device 1 passes the signal on if it requires no service, else blocks it, and so on



Handling Multiple Devices

Priority Groups – Combination of the last two methods

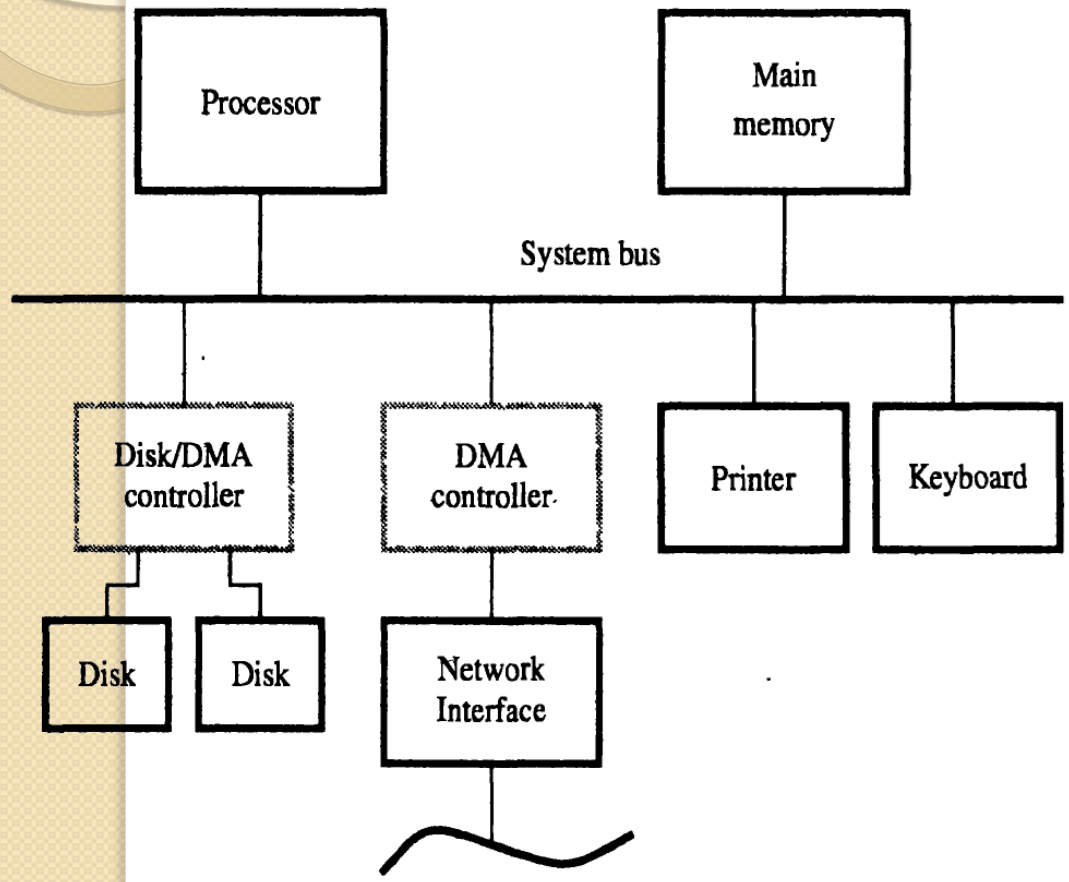


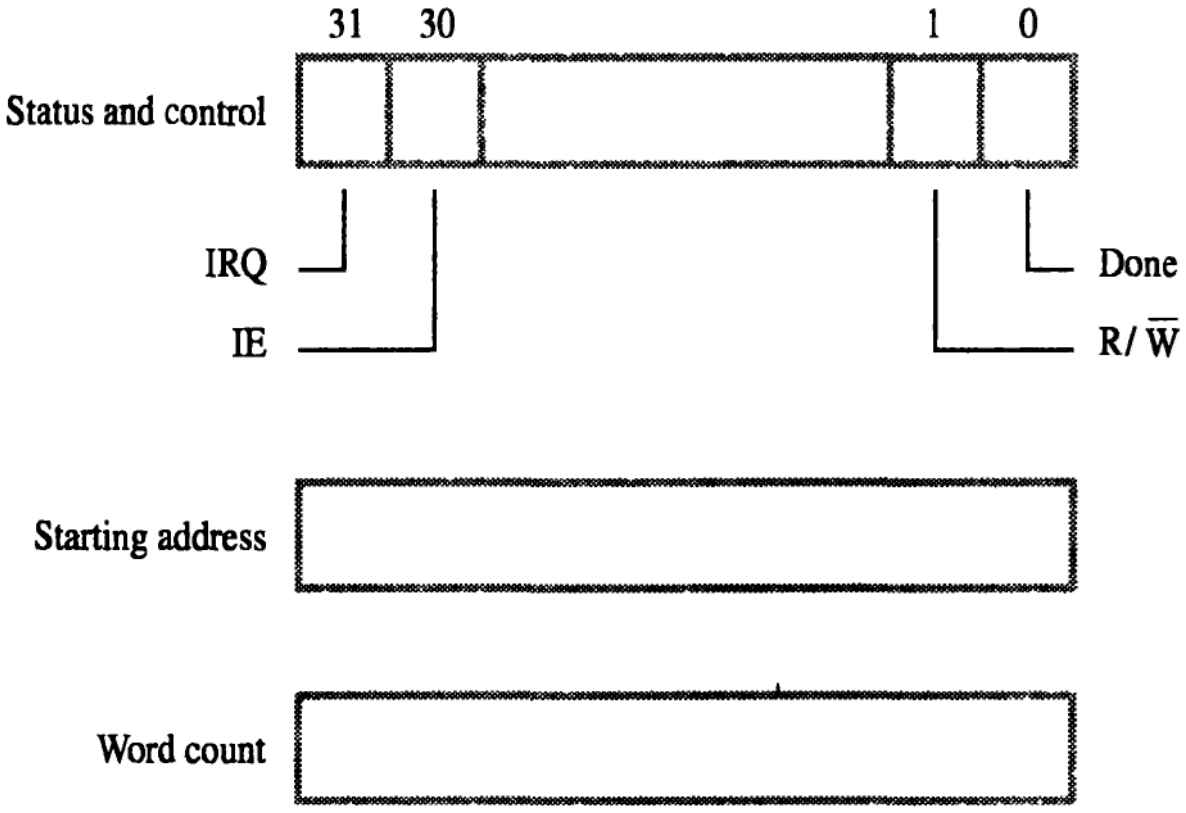
Direct memory access

Direct Memory Access

- Transfer of input/output data happens only after I/O device is ready.
- Processor polls or waits for an interrupt.
- Considerable overhead incurred when a block of data is to be transferred
- Solution:
  - Special control unit to allow transfer of a block of data between main
  - memory and external device without processor intervention
    - Direct Memory Access (DMA)
    - Control circuit that is part of I/O device – DMA controller
- Operation must be under the control of a program executed by the processor.

Use of DMA Controller





Sequence of Steps

- To initiate transfer, processor sends starting address, number of words in the block, and direction of transfer.
  - Starting address and word count are placed in the appropriate registers.
  - Direction of transfer (read/write) indicated in the status and control register
  - Processor puts the program that requested transfer in blocked state and executes another program.
- DMA controller performs the required process and sends an interrupt signal on completion.
  - Once a block of data has been transferred, controller sets "Done" to 1.
  - When IE = 1, controller raises an interrupt after the transfer of a block.
  - IRQ = 1 when the controller has requested an interrupt.
- Processor puts suspended program in runnable state.

Cycle Stealing and Burst Mode

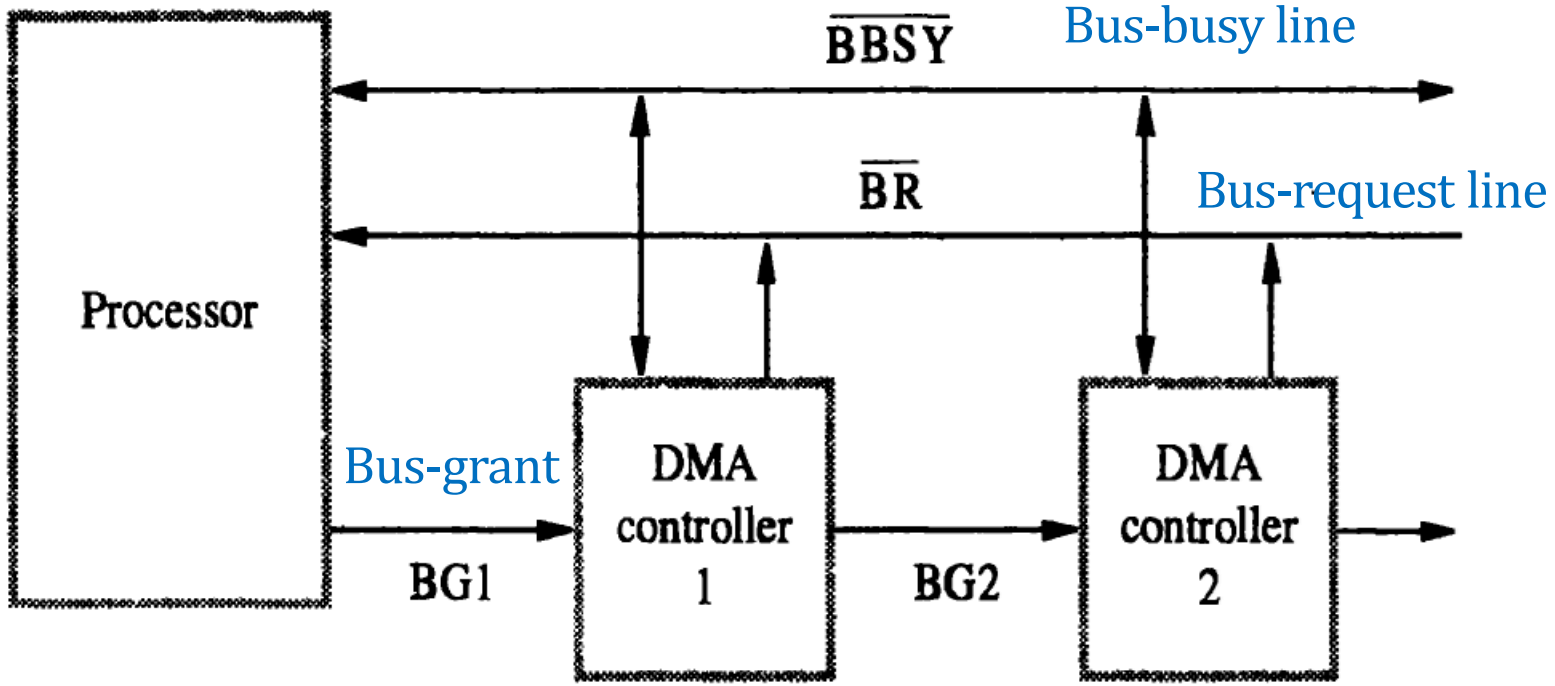
- Memory access by processor and DMA controllers are interwoven.
- Requests from DMA controller are given higher priority than processor requests.
- Among different DMA devices, top priority is given to high-speed peripherals.
- Since the processor originates most memory access cycles, the DMA controller can be said to steal memory cycles from processor – Cycle stealing
- DMA controller may also be given exclusive access to the main memory without any interruption – Block or burst mode
- A conflict may arise if both the processor and a DMA controller or two DMA controllers attempt to use the bus at the same time to access the main memory.
- To resolve these conflicts, an arbitration procedure is implemented.

Bus Arbitration

- Device allowed to initiate data transfers on the bus at any given time – bus master
- When the current master relinquishes control of the bus, another device can acquire the status.
- Bus arbitration – process by which the next bus master is selected, and the bus mastership is transferred to it
- Two approaches to bus arbitration:
  - Centralized – Single bus arbiter performs the arbitration
  - Distributed – All devices participate in the process of selection

Centralized Arbitration

- The bus arbiter may be the processor, or a separate unit connected to the bus.
- A simple arrangement for bus arbitration using a daisy chain:



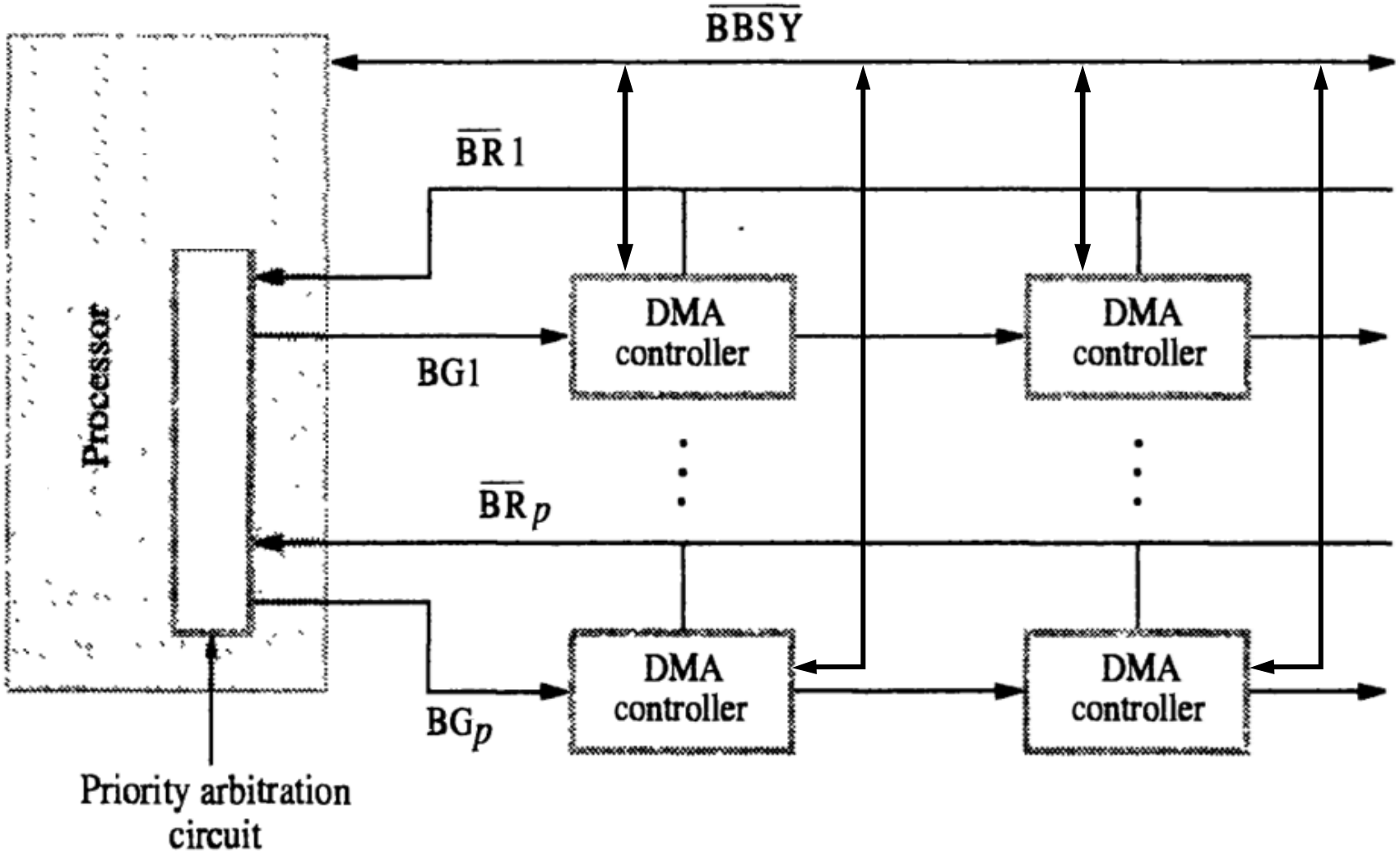
Centralized Arbitration

- The processor will be the bus master till it grants mastership to one of the DMA controllers.
- A DMA controller indicates that it needs to become bus master by sending out a bus-request signal ($\overline{BR}$).
- The processor then sends out the bus-grant signal (BG) in a daisy-chain arrangement.
- The DMA controller that requested to be bus master accepts the bus-grant signal, waits for the bus-busy line ($\overline{BBSY}$) to become inactive and then assumes mastership of the bus.
- This DMA controller also activates the $\overline{BBSY}$ line again to keep other DMA controllers from accessing the bus.

Centralized Arbitration

- Multiple DMA controllers may send out a bus-request signal at a time.
- The signal on the $\overline{BR}$ line is an OR of all the bus-request signals.
- First DMA controller on the daisy-chain gets granted the bus mastership.
- The devices can also be arranged in priority groups.
  - The arbiter ensures that only one request is granted at a time, based on priority – $\overline{BR1}$ with the highest priority and $\overline{BR_p}$ with the lowest
  - Rotating priority scheme could also be used.
    - If $\overline{BR1}$ is granted first, then the priority becomes 2, 3, …, p, 1

Centralized Arbitration



Distributed Arbitration

- All devices waiting to use the bus have equal responsibility in carrying out the arbitration process, without a central arbiter.
- Each device on the bus is assigned an identification number.
- When one or more devices request the bus, they assert the Start – arbitration signal and place their ID numbers on the ID lines.
  - The code on the ID lines ultimately represents the request that has the highest ID number.
  - For example:
  - Let each device ID be 4 bits long.
  - Say, devices A and B, of IDs 0101 and 0110, are contending for the bus.

Distributed Arbitration

- A transmits 0101 and B transmits 0110.
- The connection performs an OR operation, so the code seen by the two devices will be 0111.
- Each device compares the pattern on the arbitration lines to its own ID, starting from the most significant bit.
- If it detects a difference at any bit position, it disables drivers (makes the value 0) at that bit position and all lower bits.
- This will ultimately cause the arbitration lines to reflect the highest ID, namely, 0110.