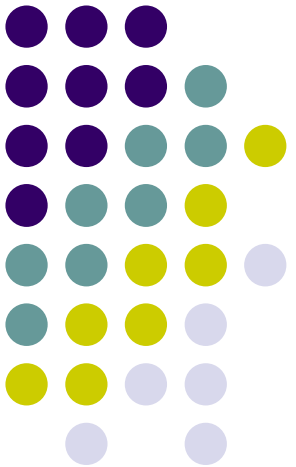
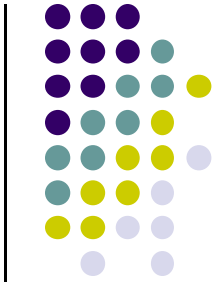
Unit V – Input/Output Organization

C.Bala Subramanian,
AP/CSE,
KLU

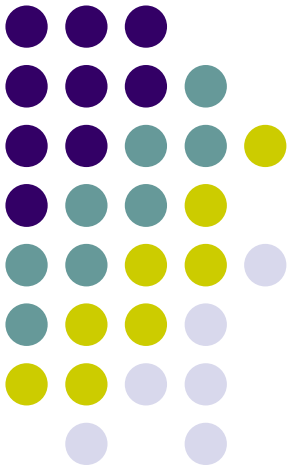


Overview



- Computer has ability to exchange data with other devices.
- Human-computer communication
- Computer-computer communication
- Computer-device communication
- …

Accessing I/O Devices



Single Bus

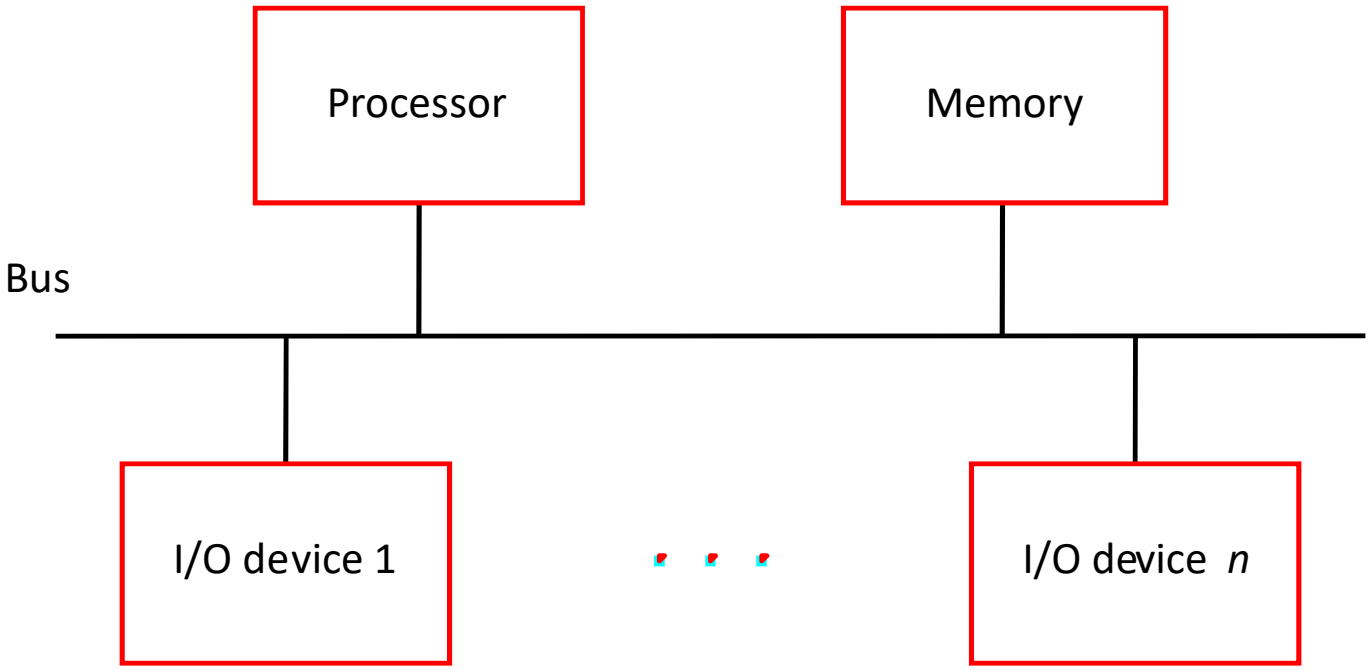
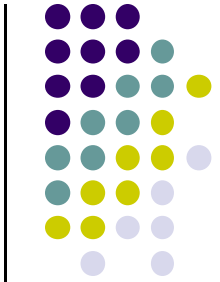


Figure 4.1. A single-bus structure.

Memory-Mapped I/O



- When I/O devices and the memory share the same address space, the arrangement is called memory-mapped I/O.
- Any machine instruction that can access memory can be used to transfer data to or from an I/O device.

```
Move  DATAIN, R0
Move  R0, DATAOUT
```

- Some processors have special In and Out instructions to perform I/O transfer.

Interface

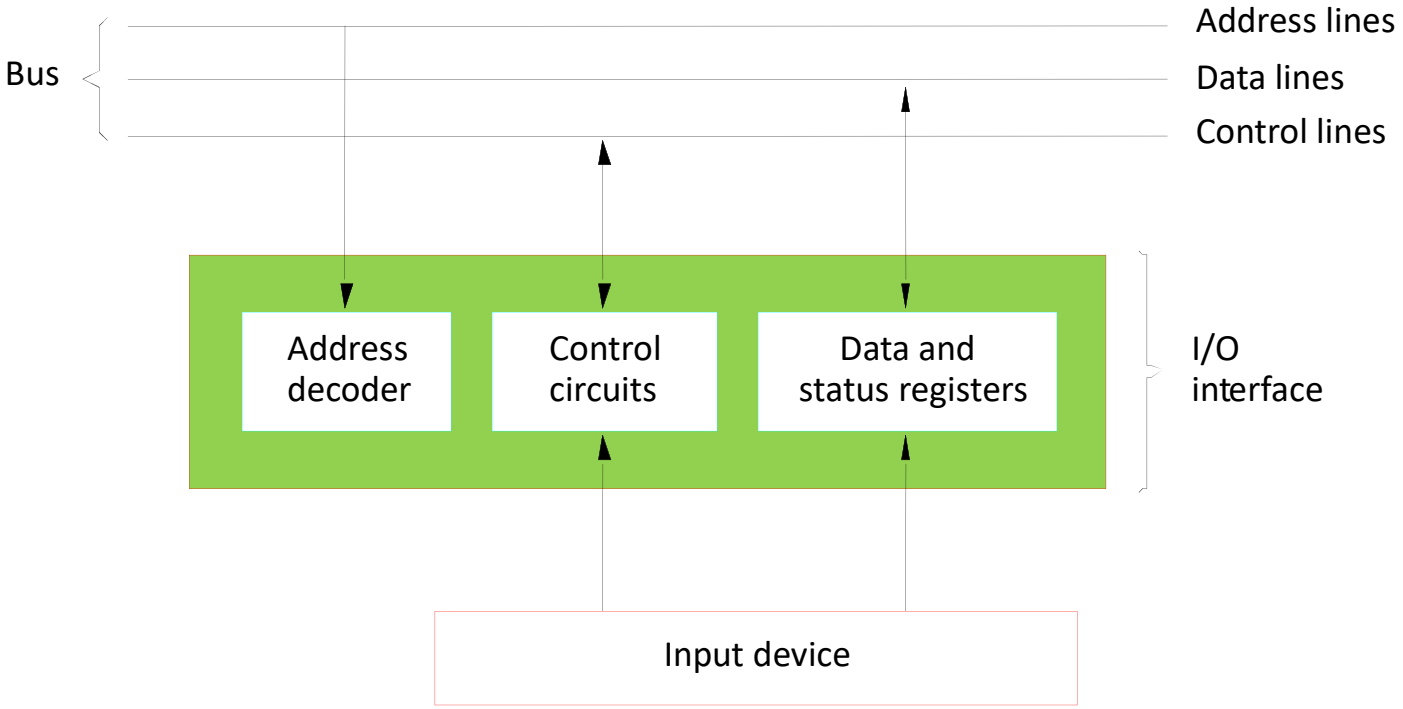
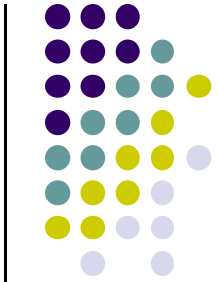


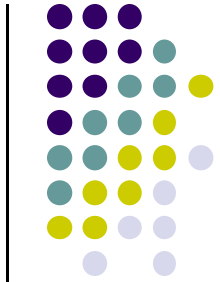
Figure 4.2. I/O interface for an input device.

Program-Controlled I/O



- I/O devices operate at speeds that are vastly different from that of the processor.
- Keyboard, for example, is very slow.
- It needs to make sure that only after a character is available in the input buffer of the keyboard interface; also, this character must be read only once.
- The basic idea is to set a status flag, SIN, as part of the status register.

Keyboard Control Example



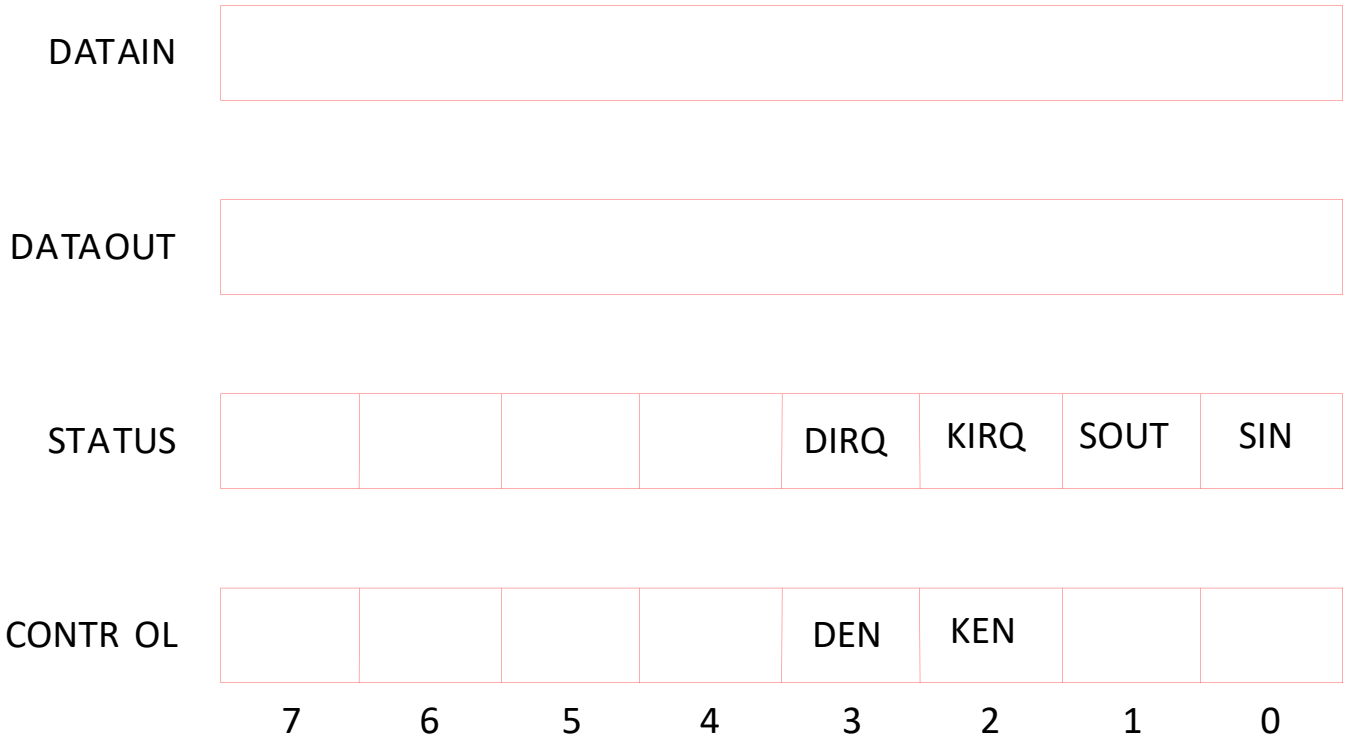
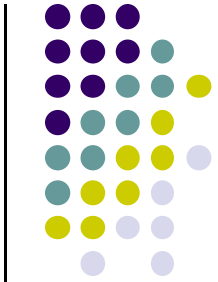


Figure 4.3. Registers in keyboard and display interfaces.

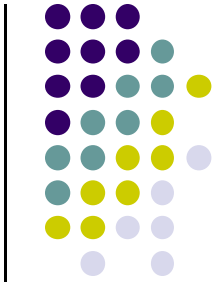
Keyboard Control Example



```
        Move      #LINE,R0        Initialize memory pointer.
WAITK   TestBit   #0,STATUS       Test SIN.
        Branch=0  WAITK           Wait for character to be entered.
        Move      DATAIN,R1       Read character.
WAITD   TestBit   #1,STATUS       Test SOUT.
        Branch=0  WAITD           Wait for display to become ready.
        Move      R1,DATAOUT      Send character to display.
        Move      R1,(R0)+        Store character and advance pointer.
        Compare   #$0D,R1         Check if Carriage Return.
        Branch≠0  WAITK           If not, get another character.
        Move      #$0A,DATAOUT    Otherwise, send Line Feed.
        Call      PROCESS         Call a subroutine to process
                                    the input line.
```

Figure 4.4 A program that reads one line from the keyboard stores it in memory buffer, and echoes it back to the display.

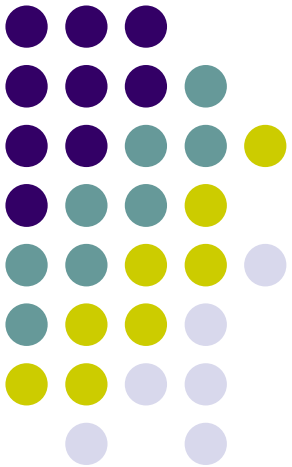
Two Major Mechanisms for implementing I/O operations



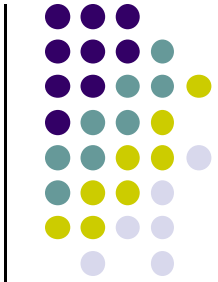
- Interrupt
- Direct Memory Access (DMA)

Interrupts

1. Interrupt Hardware
2. Enabling and disabling interrupts
3. Handling multiple devices
    1. Vectored interrupts
    2. Interrupt nesting
    3. Simultaneous nesting
4. Controlling device request
5. Exceptions
    1. Recovery from errors
    2. Debugging
    3. Privilege exception

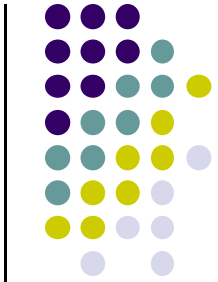


Overview



- In program-controlled I/O, the program enters a wait loop in which it repeatedly tests the device status. During the period, the processor is not performing any useful computation.
- However, in many situations other tasks can be performed while waiting for an I/O device to become ready.
- Let the device alert the processor.

Example



- Some computations + print
- Two subroutines: COMPUTE and PRINT
- The printer accepts only one line of text at a time.
- Try to overlap printing and computation.

  - COMPUTE produces first *n* lines of text;
  - PRINT sends the first line to the printer; then PRINT is suspended; COMPUTE continues to perform other computations;
  - After the printer finishes printing the first line, it send an interrupt-request signal to the processor;
  - In response, the processor interrupts execution of COMPUTE and transfers control to PRINT to send the next line;
  - COMPUTE resumes;
  - …

Example

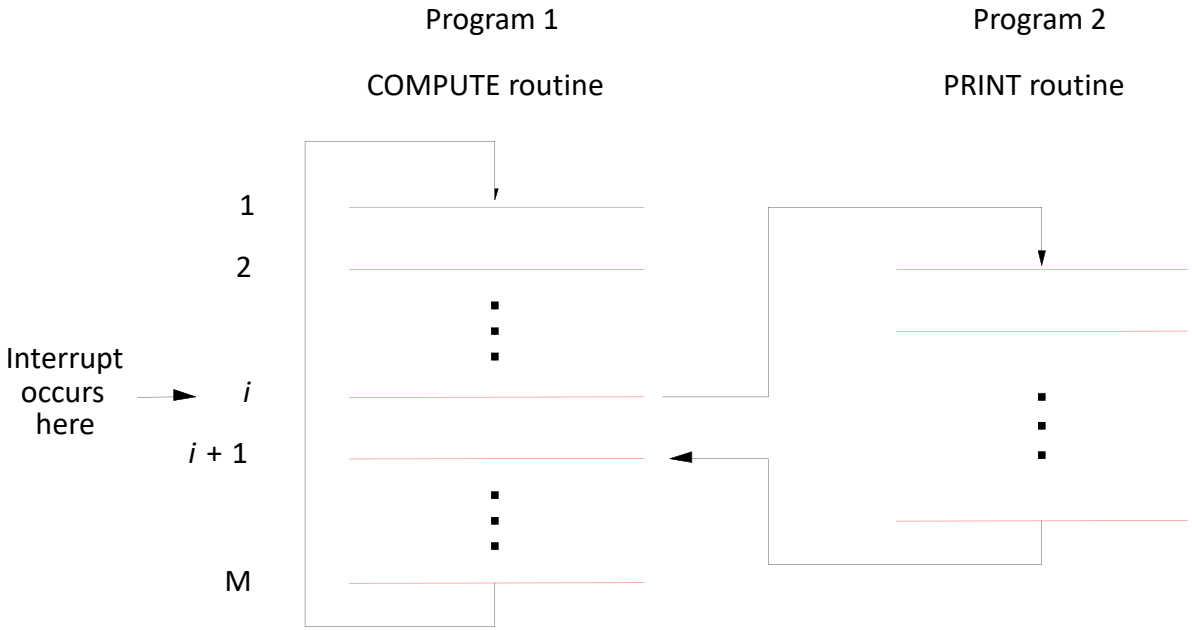
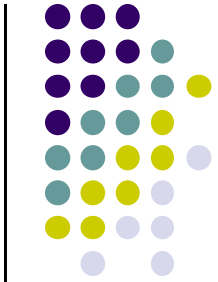


Figure 4.5. Transfer of control through the use of interrupts.

Several Points



- Interrupt-service routine – PRINT
- Store the current PC
- Return-from-interrupt
- Interrupt-acknowledge signal
- Interrupt latency
- Real-time processing

- Difference between an interrupt-service routine and regular subroutine?

Interrupt Hardware

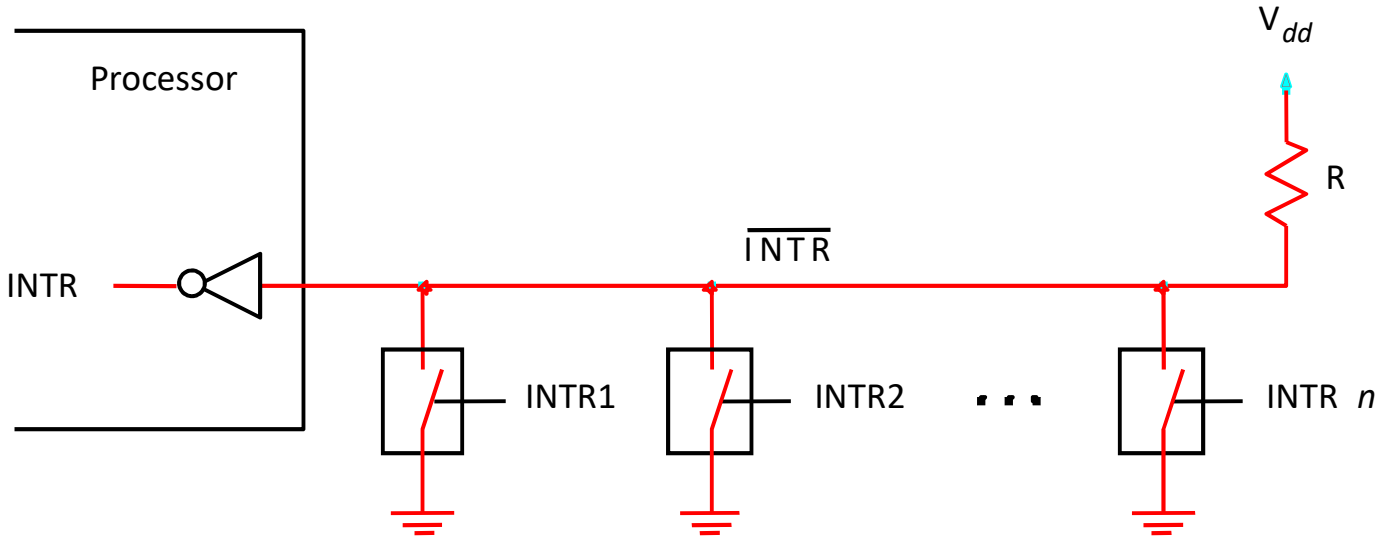
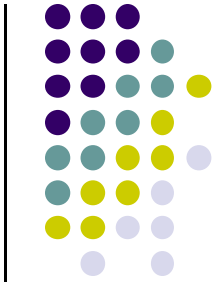


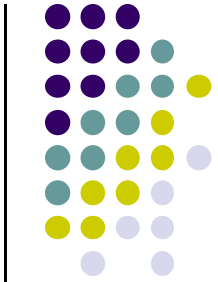
Figure 4.6. An equivalent circuit for an open-drain bus used to implement a common interrupt-request line.

Enabling and Disabling Interrupts



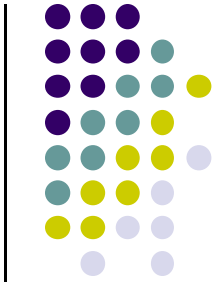
- Since the interrupt request can come at any time, it may alter the sequence of events from that envisaged by the programmer.
- Interrupts must be controlled.
- Ignoring an interrupt happens in the previous example.

Enabling and Disabling Interrupts



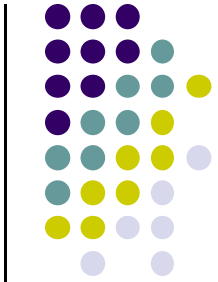
- The interrupt request signal will be active until it learns that the processor has responded to its request. This must be handled to avoid successive interruptions.
  - Let the interrupt be disabled/enabled in the interrupt-service routine.
  - Let the processor automatically disable interrupts before starting the execution of the interrupt-service routine.
  - Edge-triggered

Handling Multiple Devices



- How can the processor recognize the device requesting an interrupt?
- Given that different devices are likely to require different interrupt-service routines, how can the processor obtain the starting address of the appropriate routine in each case?
- (Vectored interrupts)
- Should a device be allowed to interrupt the processor while another interrupt is being serviced?
- (Interrupt nesting)
- How should two or more simultaneous interrupt requests be handled?
- (Daisy-chain)

Vectored Interrupts



- A device requesting an interrupt can identify itself by sending a special code to the processor over the bus.
- Interrupt vector
- Avoid bus collision

Interrupt Nesting

- Simple solution: only accept one interrupt at a time, then disable all others.
- Problem: some interrupts cannot be held too long.
- Priority structure

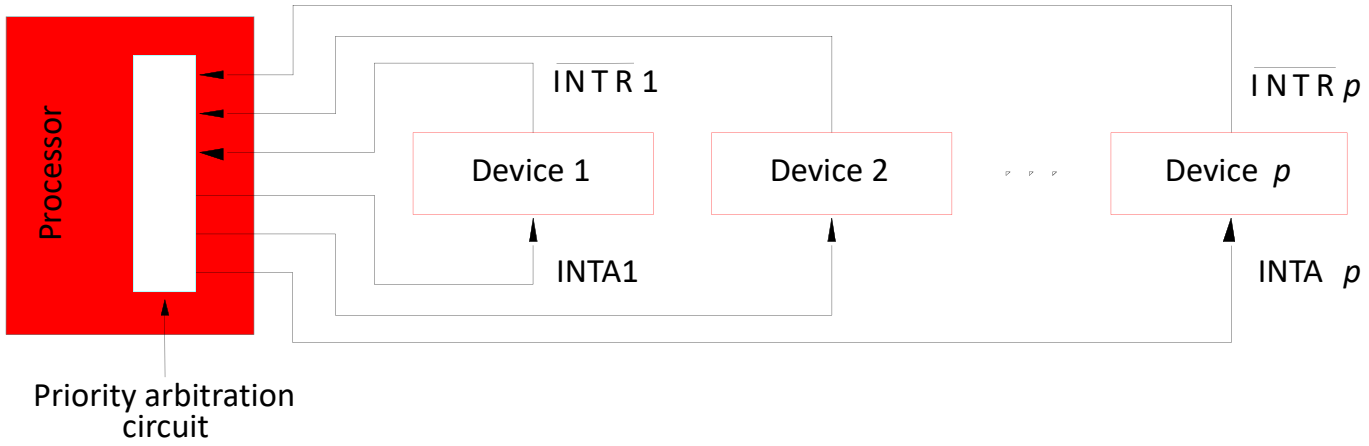
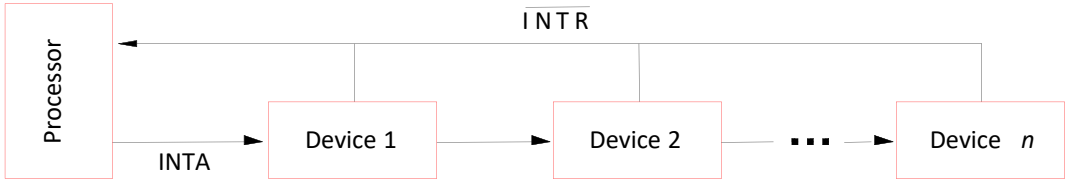
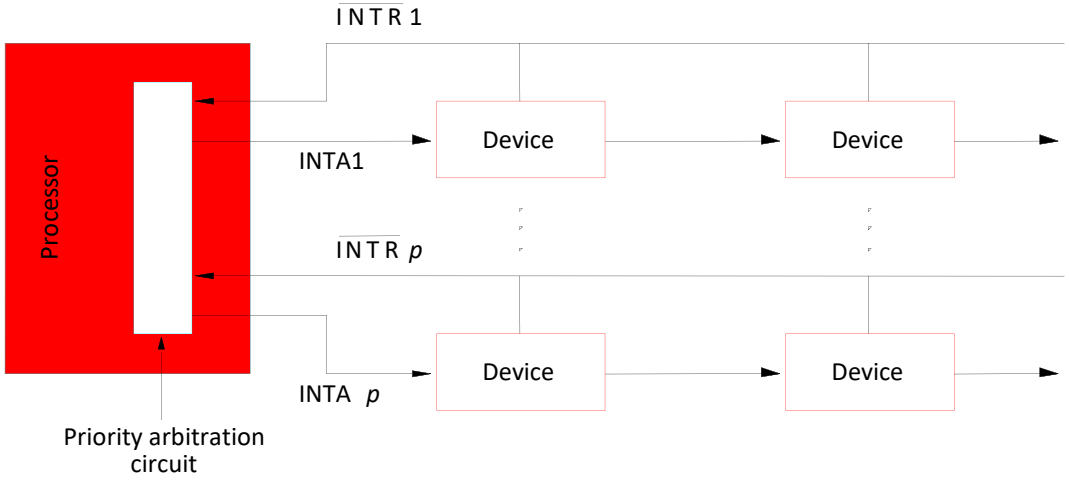


Figure 4.7. Implementation of interrupt priority using individual interrupt-request and acknowledge lines.

Simultaneous Requests



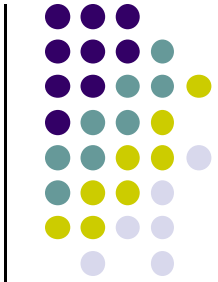
(a) Daisy chain



(b) Arrangement of priority groups

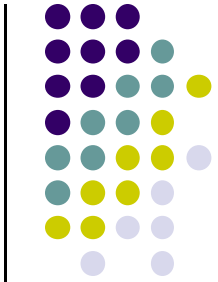
Figure 4.8. Interrupt priority schemes.

Controlling Device Requests



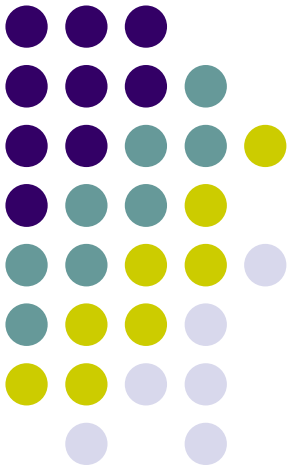
- Some I/O devices may not be allowed to issue interrupt requests to the processor.
- At device end, an interrupt-enable bit in a control register determines whether the device is allowed to generate an interrupt request.
- At processor end, either an interrupt enable bit in the PS register or a priority structure determines whether a given interrupt request will be accepted.

Exceptions

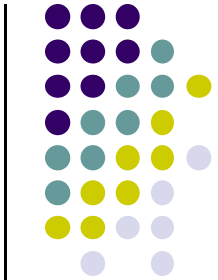


- Recovery from errors
- Debugging
  - Trace
  - Breakpoints
- Privilege exception

Direct Memory Access

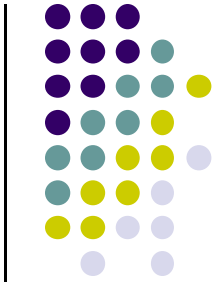


DMA



- Think about the overhead in both polling and interrupting mechanisms when a large block of data need to be transferred between the processor and the I/O device.
- A special control unit may be provided to allow transfer of a block of data directly between an external device and the main memory, without continuous intervention by the processor – direct memory access (DMA).
- The DMA controller provides the memory address and all the bus signals needed for data transfer, increment the memory address for successive words, and keep track of the number of transfers.

DMA Procedure



- Processor sends the starting address, the number of data, and the direction of transfer to DMA controller.
- Processor suspends the application program requesting DMA, starts DMA transfer, and starts another program.
- After the DMA transfer is done, DMA controller sends an interrupt signal to the processor.
- The processor puts the suspended program in the Runnable state.

DMA Register

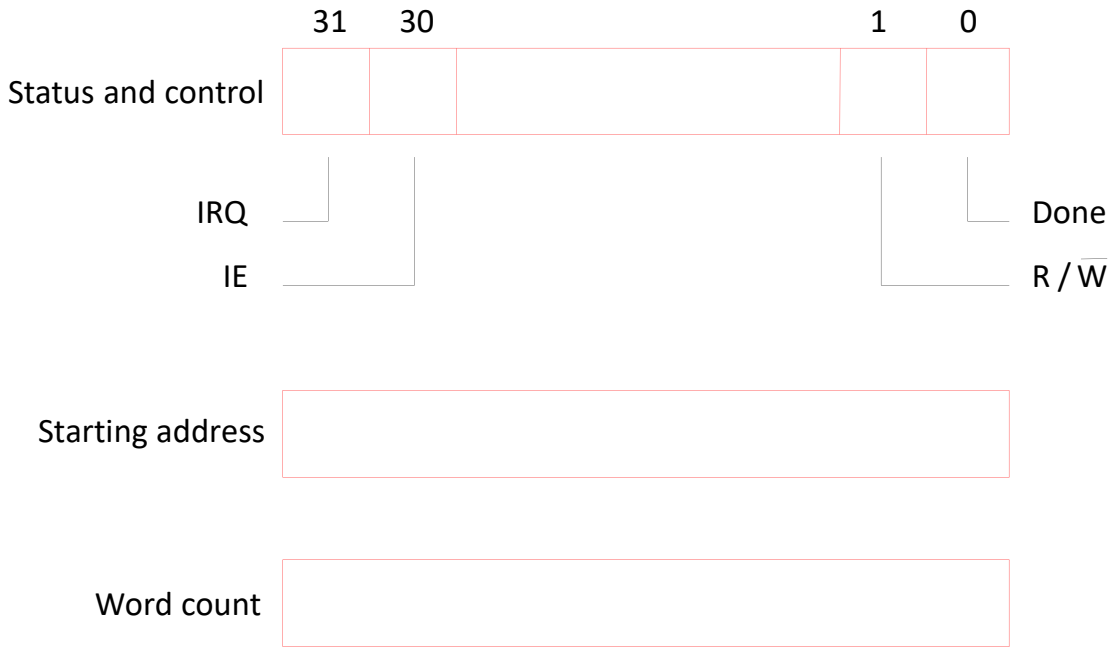


Figure 4.18. Registers in a DMA interf ace.

System

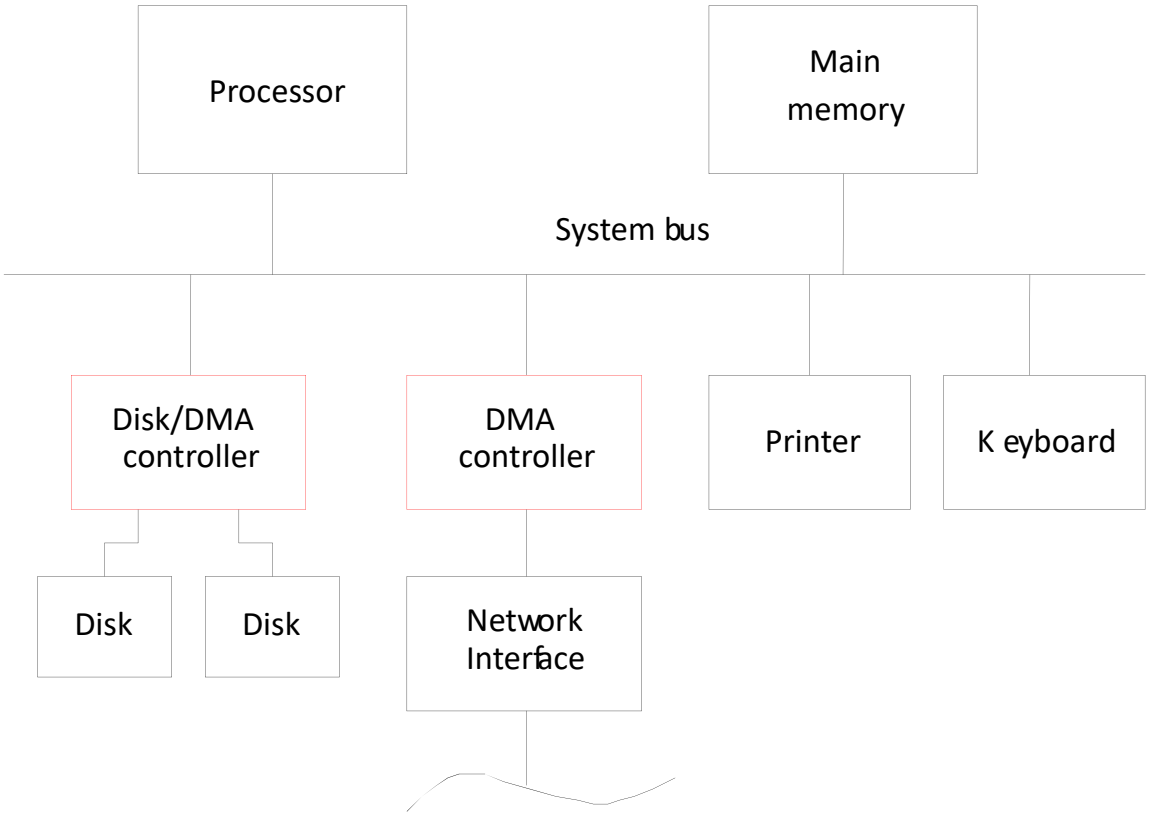
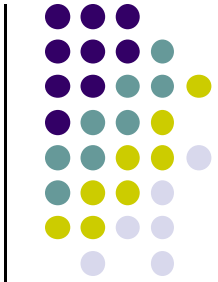


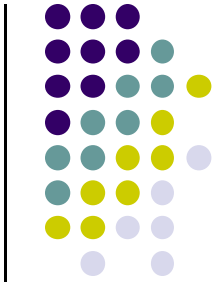
Figure 4.19. Use of DMA controllers in a computer system.

Memory Access



- Memory access by the processor and the DMA controller are interwoven.
- DMA device has higher priority.
- Among all DMA requests, top priority is given to high-speed peripherals.
- Cycle stealing
- Block (burst) mode
- Data buffer
- Conflicts

Bus Arbitration



- The device that is allowed to initiate data transfers on the bus at any given time is called the bus master.
- Bus arbitration is the process by which the next device to become the bus master is selected and bus mastership is transferred to it.
- Need to establish a priority system.
- Two approaches: centralized and distributed

Centralized Arbitration

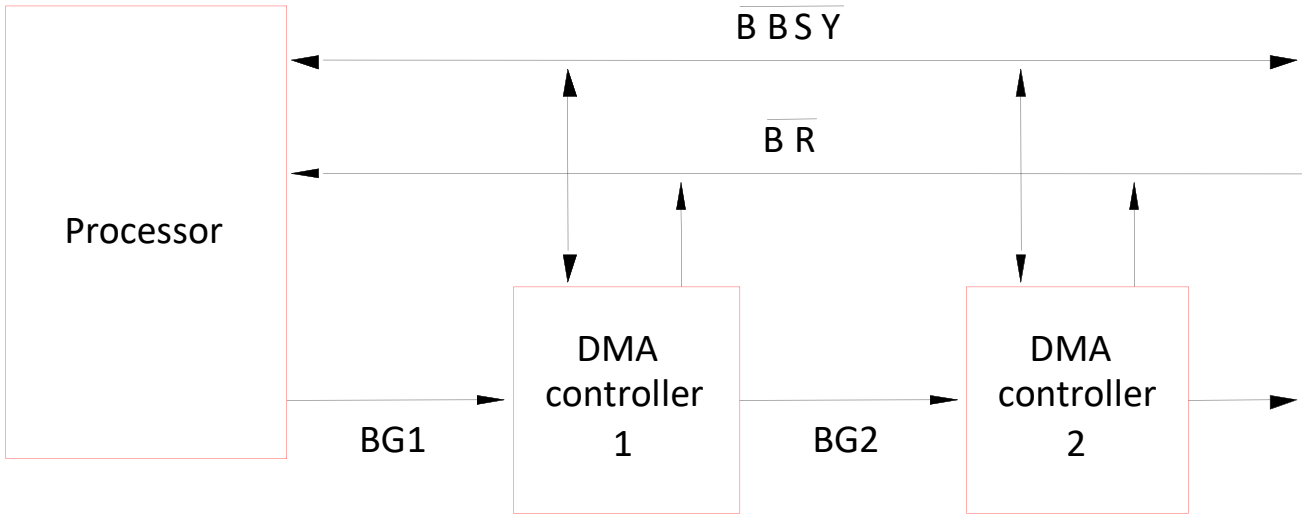


Figure 4.20. A simple arrangement for b us arbitration using a daisy chain.

Centralized Arbitration

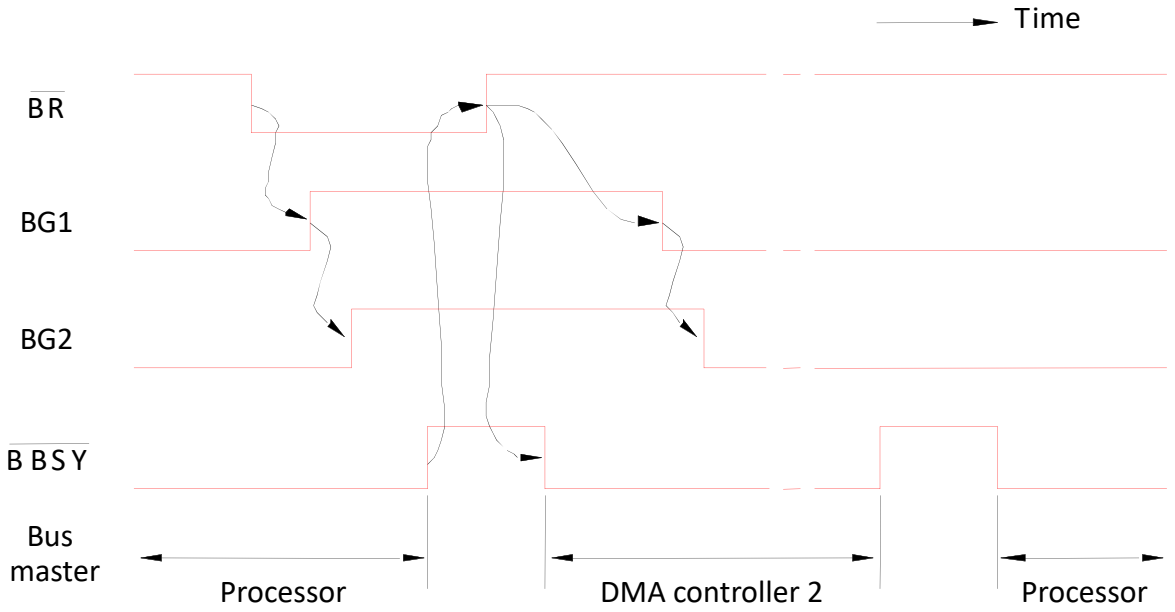


Figure 4.21. Sequence of signals during transfer of bus mastership for the devices in Figure 4.20.

Distributed Arbitration

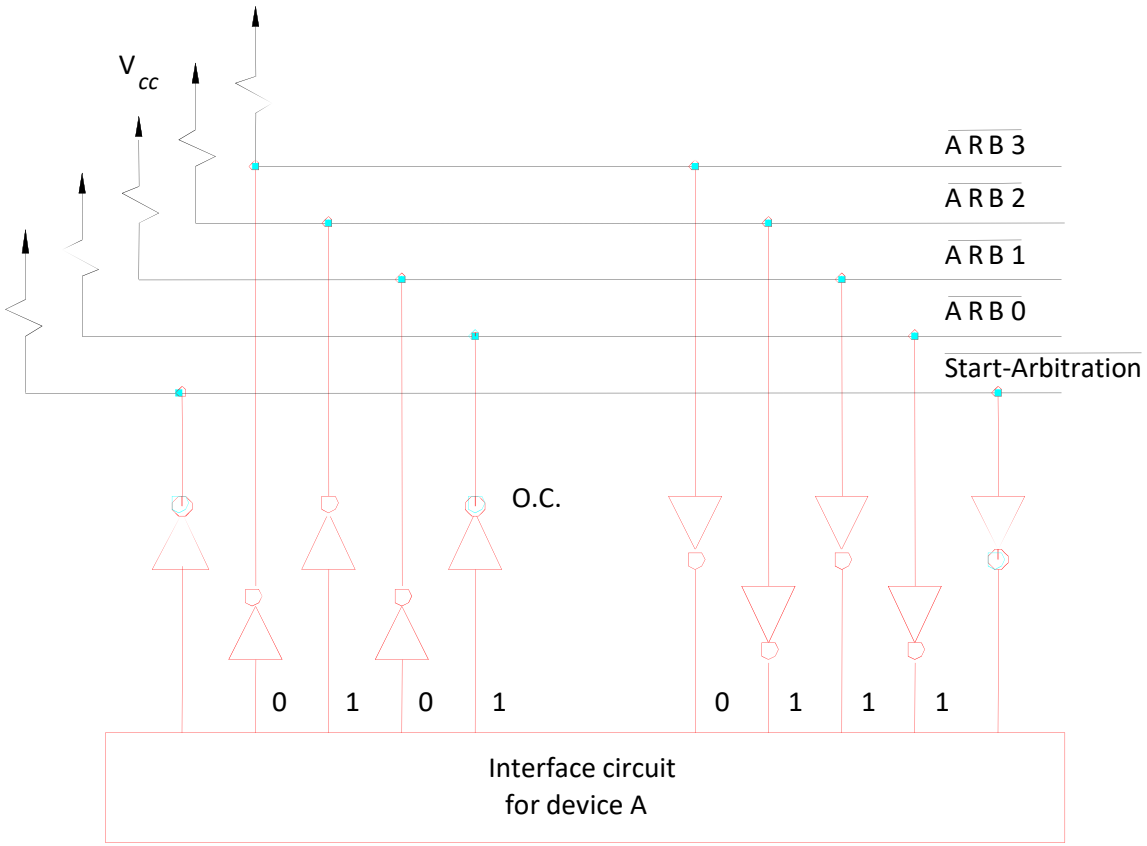
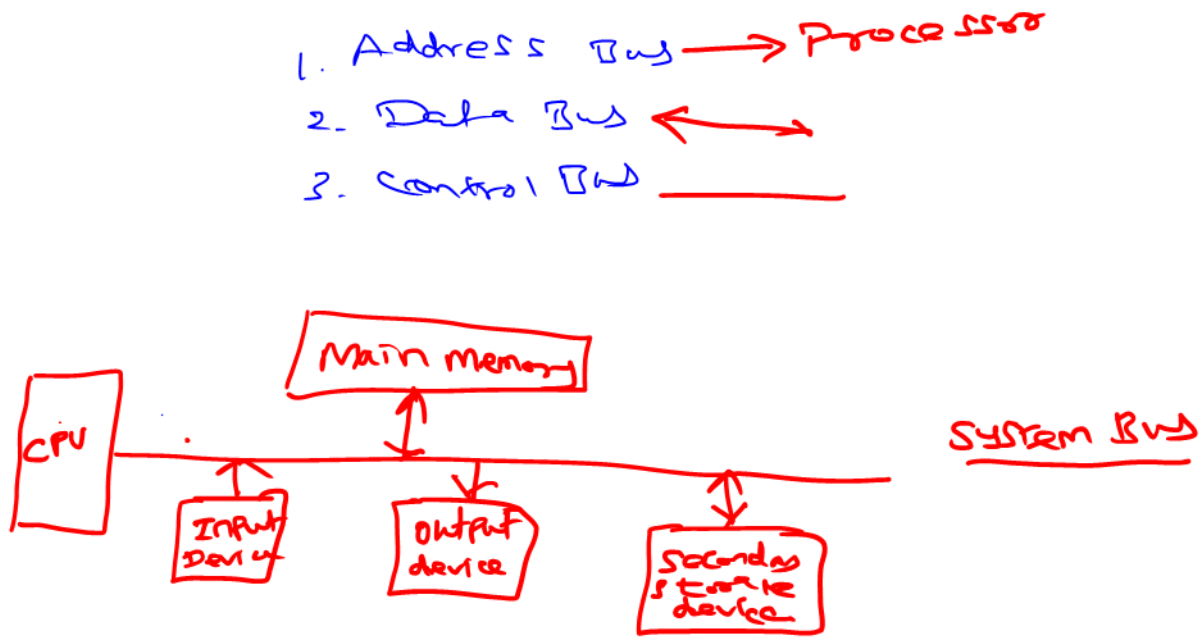
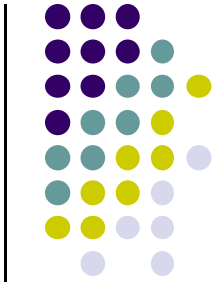


Figure 4.22. A distributed arbitration scheme.

Buses


1. Address Bus → Processor
2. Data Bus ↔
3. Control Bus
Main memory
CPU
System Bus
Input device
Output device
Secondary storage device

Overview



- The primary function of a bus is to provide a communications path for the transfer of data.
- A bus protocol is the set of rules that govern the behavior of various devices connected to the bus as to when to place information on the bus, assert control signals, etc.
- Three types of bus lines: data, address, control
- The bus control signals also carry timing information.
- Bus master (initiator) / slave (target)

Synchronous Bus Timing

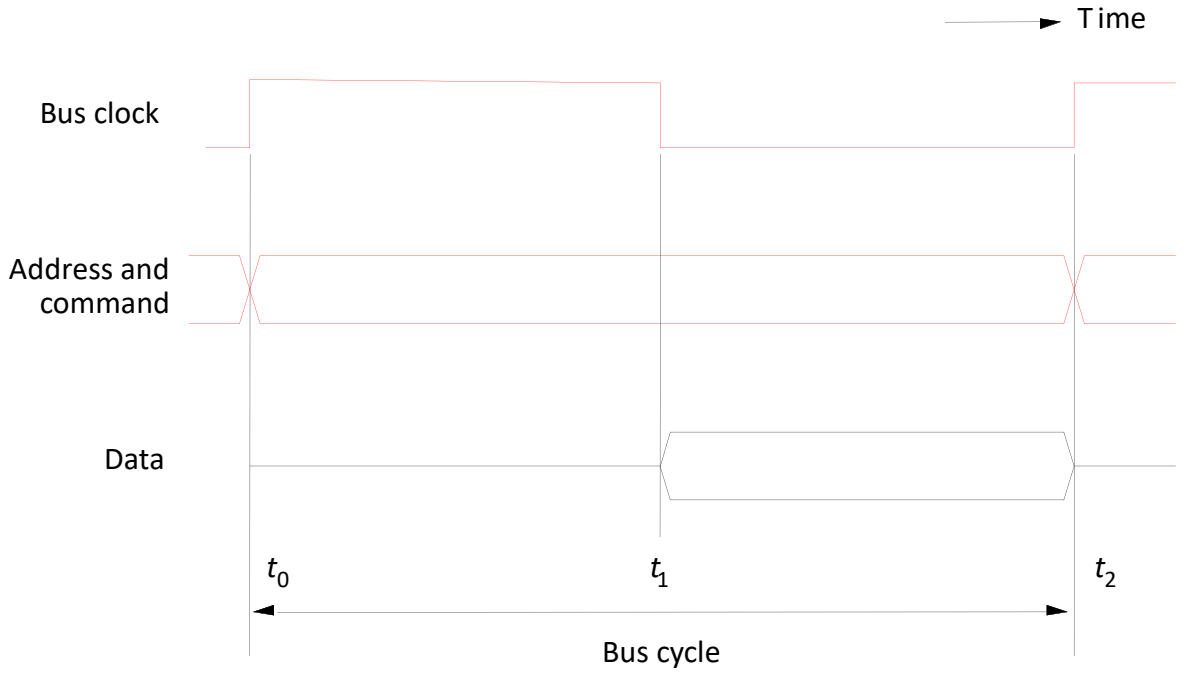


Figure 4.23. Timing of an input transfer on a synchronous bus.

Synchronous Bus Detailed Timing

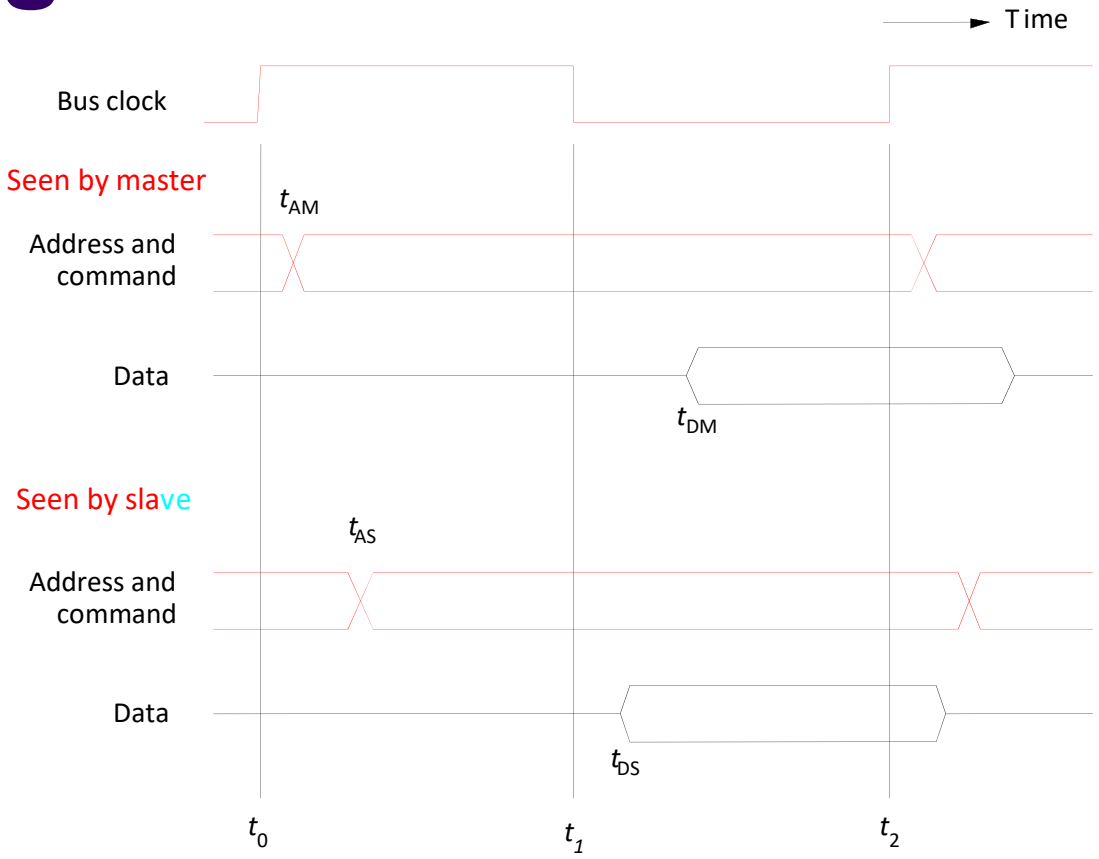


Figure 4.24. A detailed timing diagram for the input transfer of Figure 4.23.

Multiple-Cycle Transfers

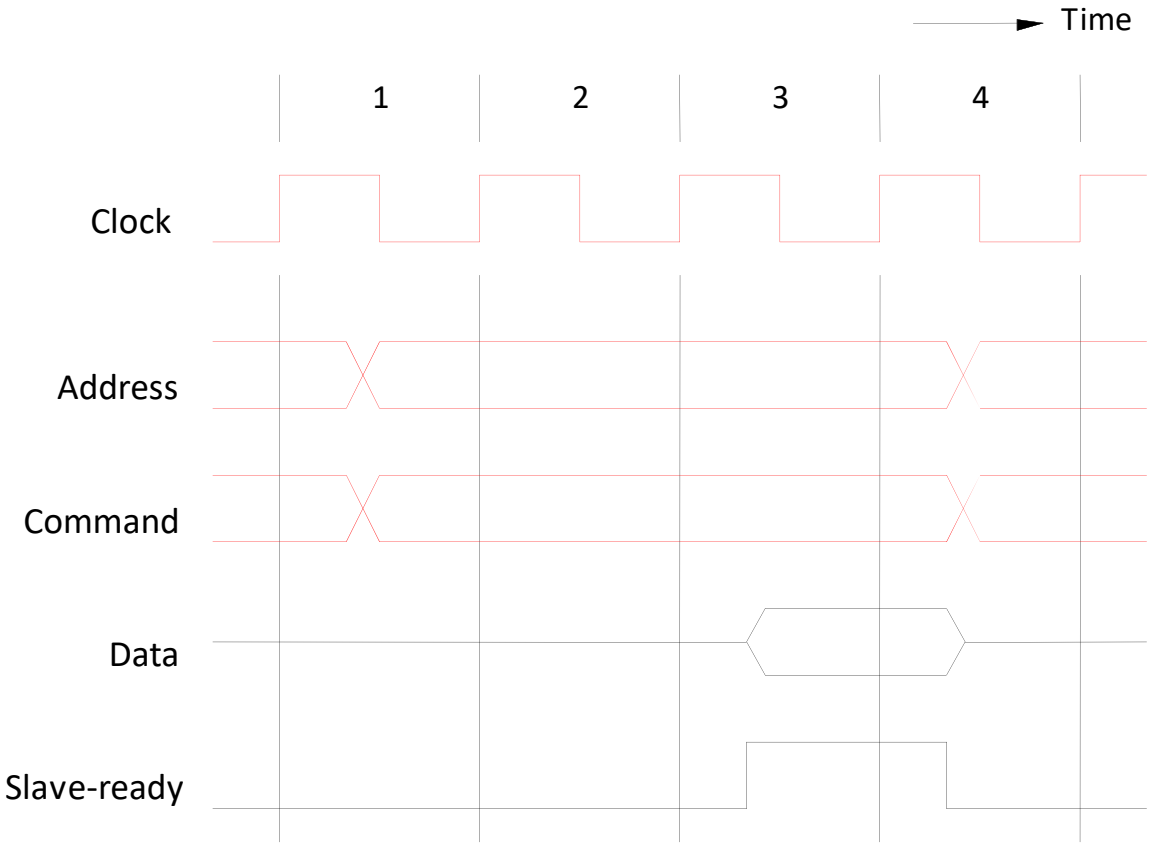


Figure 4.25. An input transfer using multiple clock cycles.

Asynchronous Bus – Handshaking Protocol for Input Operation

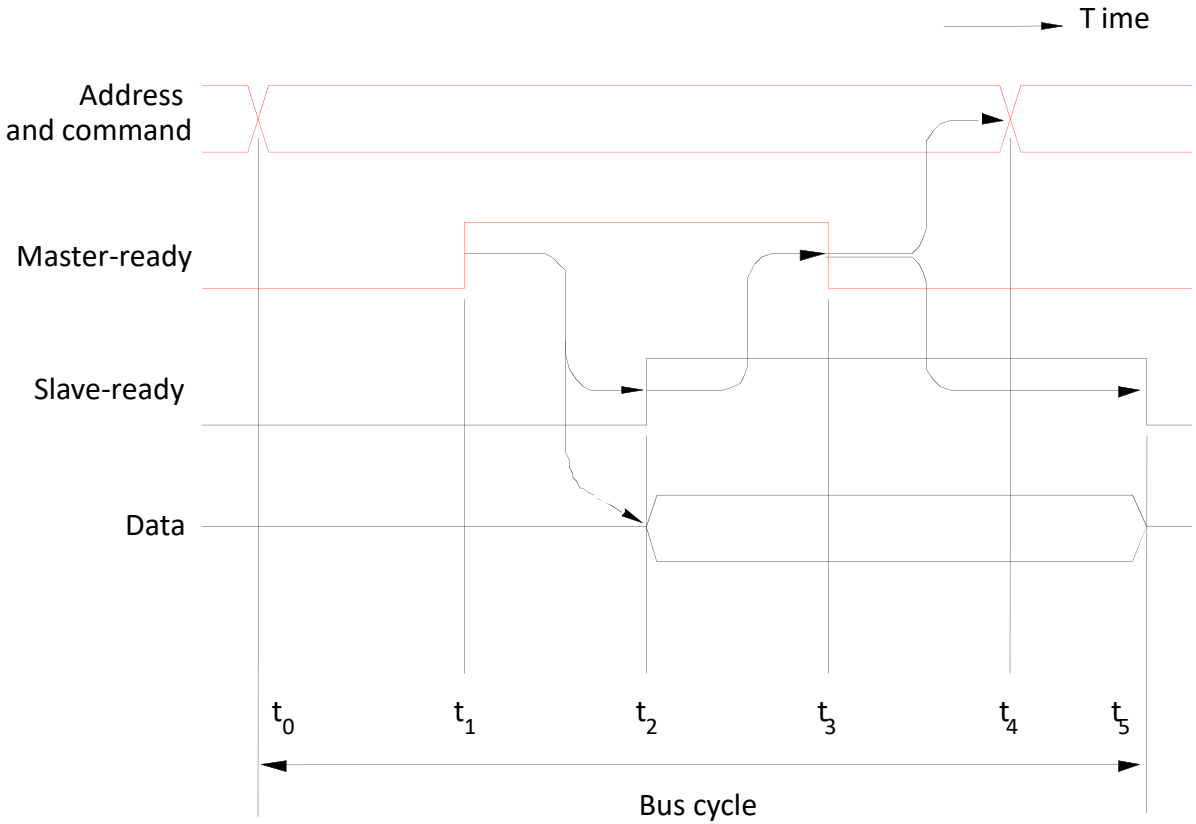


Figure 4.26. Handshake control of data transfer during an input operation.

Asynchronous Bus – Handshaking Protocol for Output Operation

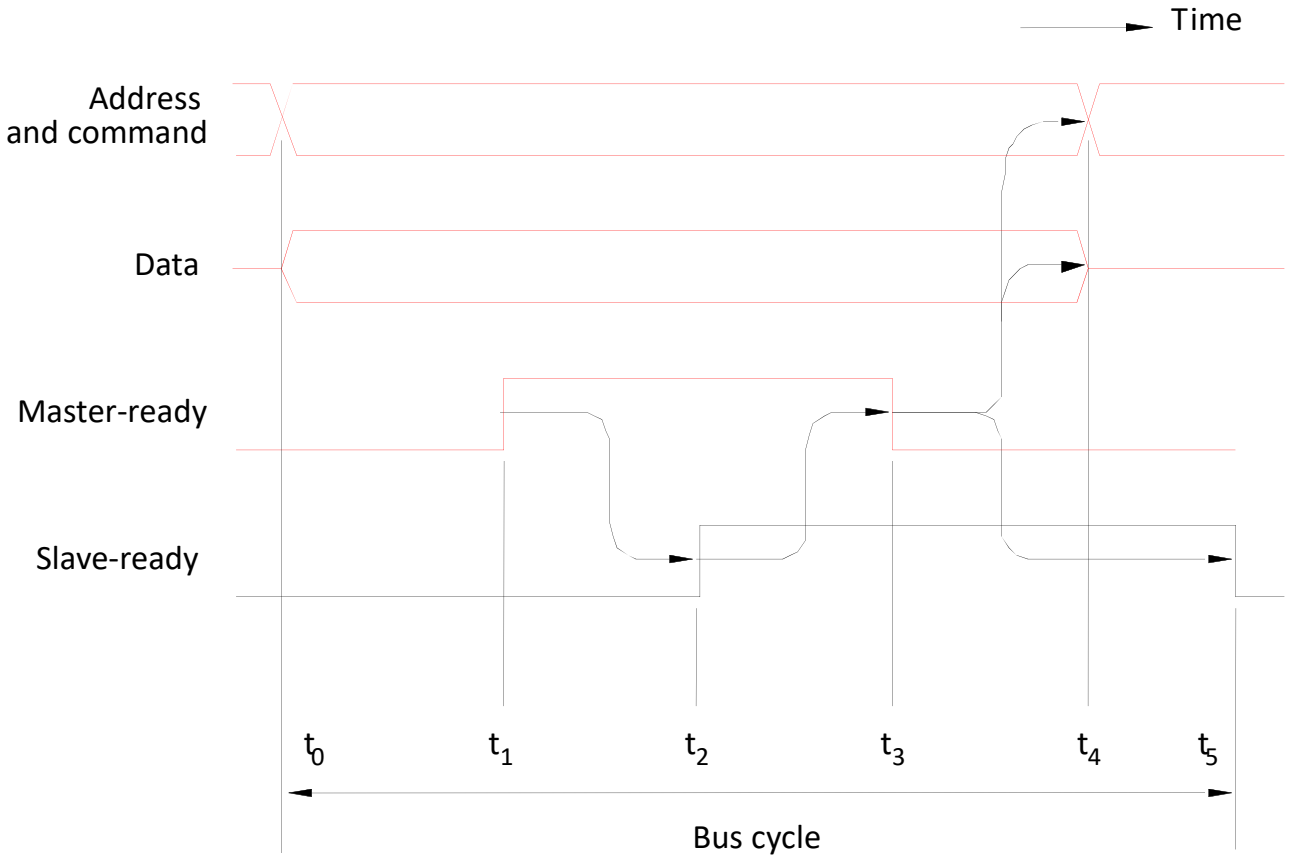
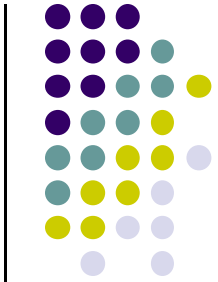


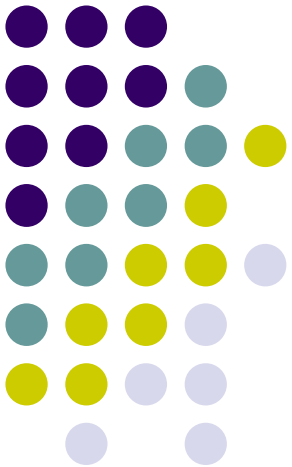
Figure 4.27. Handshake control of data transfer during an output operation.

Discussion

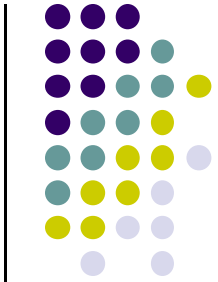


- Trade-offs
  - Simplicity of the device interface
  - Ability to accommodate device interfaces that introduce different amounts of delay
  - Total time required for a bus transfer
  - Ability to detect errors resulting from addressing a nonexistent device or from an interface malfunction
- Asynchronous bus is simpler to design.
- Synchronous bus is faster.

Standard I/O Interfaces



Overview



- The needs for standardized interface signals and protocols.
- Motherboard
- Bridge: circuit to connect two buses
- Expansion bus
- ISA, PCI, SCSI, USB,…

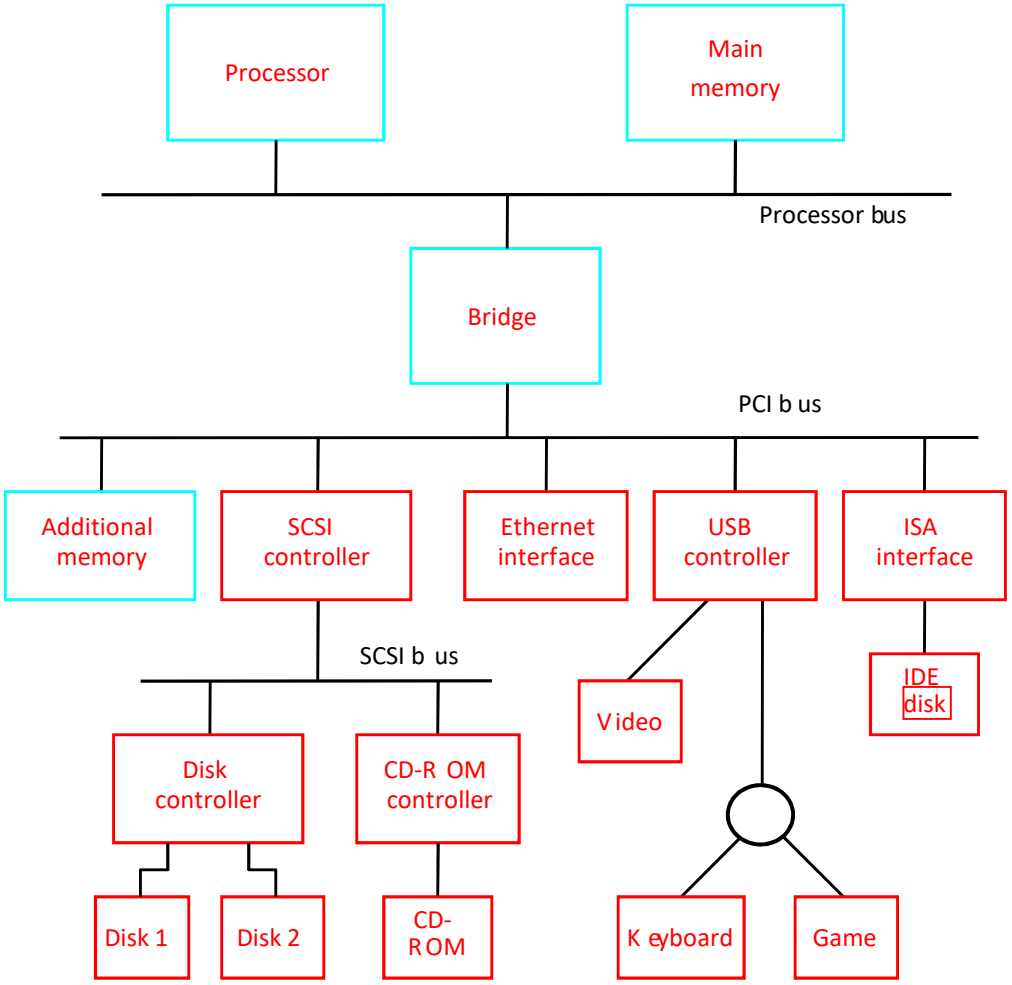
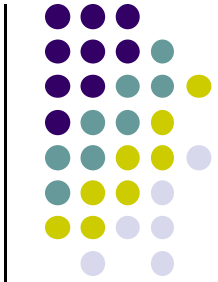
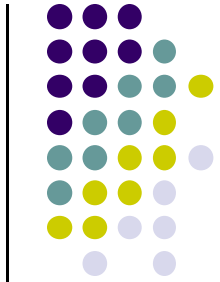


Figure 4.38. An example of a computer system using different interface standards.

Standard I/O Interfaces



- PCI
  - Data transfer
  - Device Configuration
  - Electrical Characteristics
- SCSI Bus
  - Bus Signals
  - Arbitration
  - Selection
  - Information Transfer
  - Reselection



- USB
  - Port Limitation
  - Device Characteristics
  - Plug-and –Play
  - USB Architecture
  - Addressing
  - USB Protocols