Chapter 4. Input/Output Organization

Computer Architecture and Organization

Instructor: Mustafa Mohamed

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

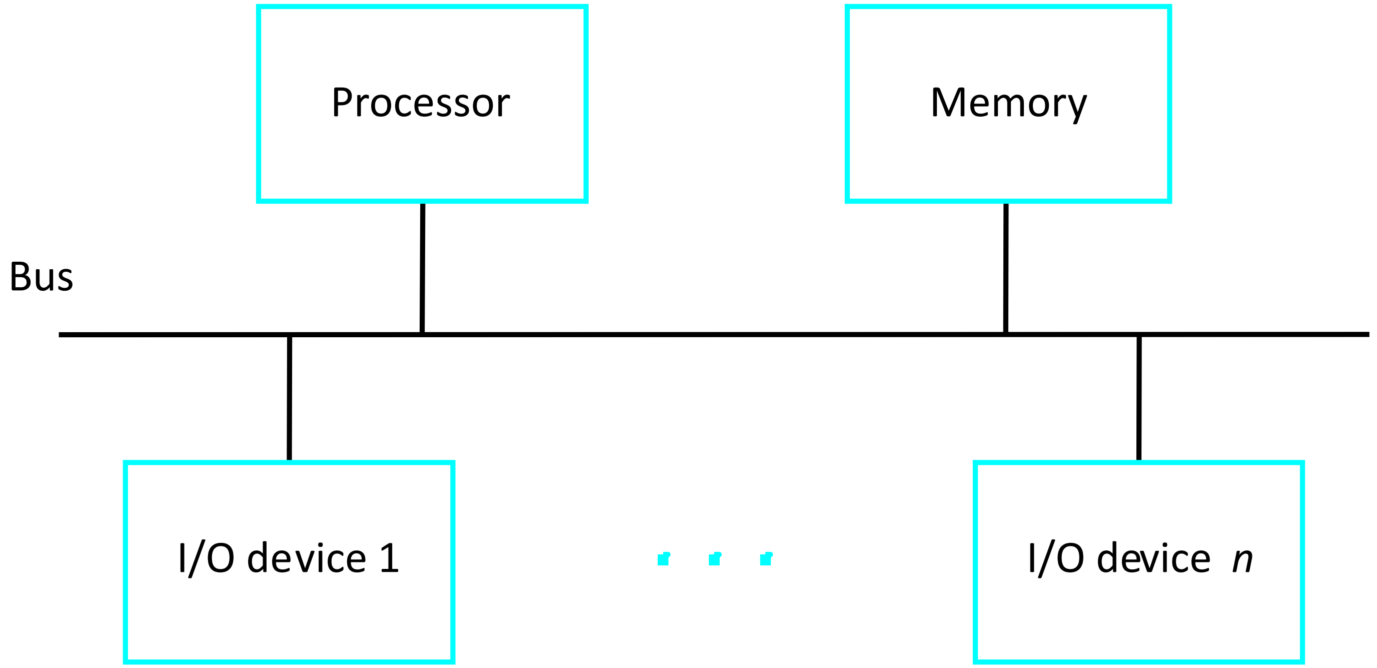


Figure 4.1. A single-bus structure.

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

Program-Controlled I/O

- I/O devices operate at speeds that are very much 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.

Three Major Mechanisms

- Program-controlled I/O – processor polls the device.
- Interrupt
- Direct Memory Access (DMA)

Interrupts

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.

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.

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.

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

Simultaneous Requests

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
  - Breakpoint
- Privilege exception

Use of Interrupts in Operating Systems

- The OS and the application program pass control back and forth using software interrupts.
- Supervisor mode / user mode
- Multitasking (time-slicing)
- Process – running, runnable, blocked
- Program state

Processor Examples

```
Main  program

          MOVE.L     #LINE,PNTR        Initialize buffer pointer.
          CLR        EOL               Clear end-of-line indicator.
          ORI.B      #4,CONTROL        Set bit KEN.
          MOVE       #$100,SR          Set processor priority to 1.
          :

Interrupt-service  routine

READ      MOVEM.L    A0/D0,−(A7)       Save registers A0, D0 on stack.
          MOVEA.L    PNTR,A0           Load address pointer.
          MOVE.B     DATAIN,D0         Get input character.
          MOVE.B     D0,(A0)+          Store it in memory buffer.
          MOVE.L     A0,PNTR           Update pointer.
          CMPI.B     #$0D,D0           Check if Carriage Return.
          BNE        RTRN
          MOVE       #1,EOL            Indicate end of line.
          ANDI.B     #$FB,CONTROL      Clear bit KEN.
RTRN      MOVEM.L    (A7)+,A0/D0       Restore registers D0, A0.
          RTE
```

Figure 4.15. A 68000 interrupt-service routine to read an input line from a keyboard based on Figure 4.9.

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

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

Centralized Arbitration

Distributed Arbitration

Buses

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

Synchronous Bus Detailed Timing

Multiple-Cycle Transfers

Asynchronous Bus – Handshaking Protocol for Input Operation

Asynchronous Bus – Handshaking Protocol for 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.

Interface Circuits

Function of I/O Interface

- Provide a storage buffer for at least one word of data;
- Contain status flags that can be accessed by the processor to determine whether the buffer is full or empty;
- Contain address-decoding circuitry to determine when it is being addressed by the processor;
- Generate the appropriate timing signals required by the bus control scheme;
- Perform any format conversion that may be necessary to transfer data between the bus and the I/O device.

Parallel Port

- A parallel port transfers data in the form of a number of bits, typically 8 or 16, simultaneously to or from the device.
- For faster communications

Parallel Port – Input Interface (Keyboard to Processor Connection)

Parallel Port – Input Interface (Keyboard to Processor Connection)

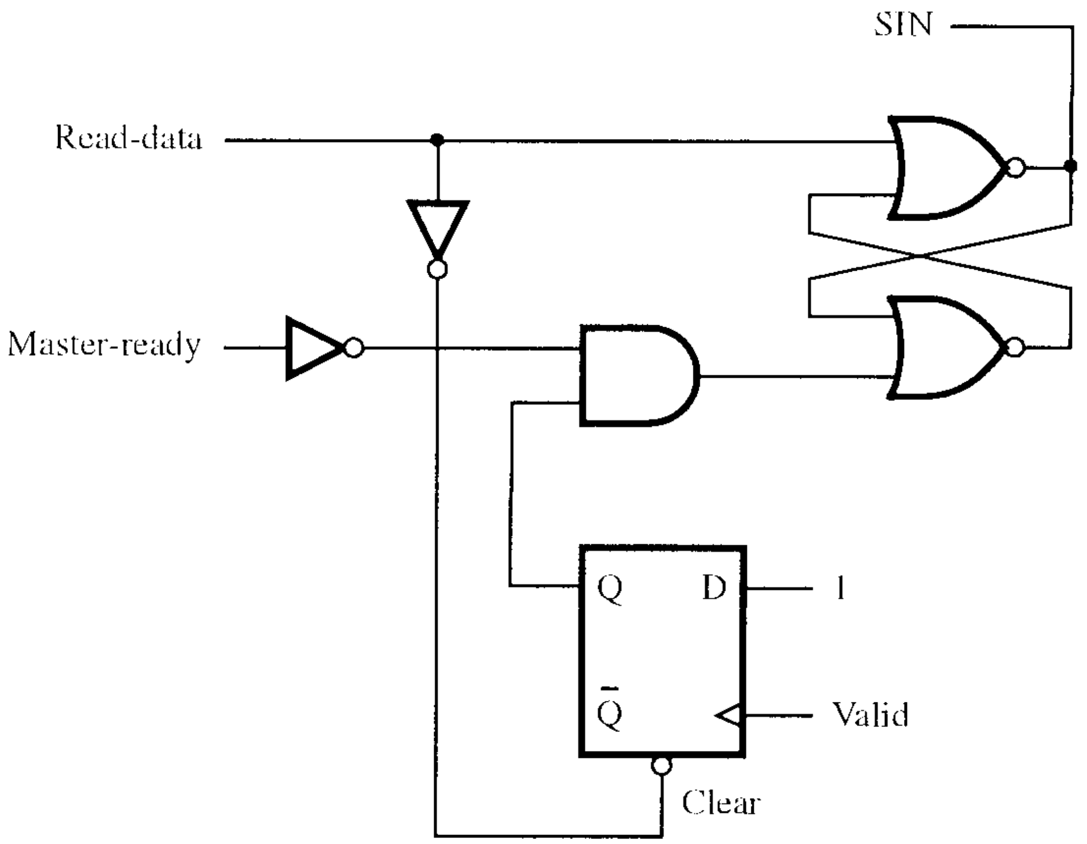


Figure 4.30. Circuit for the status flag block in Figure 4.29.

Parallel Port – Output Interface (Printer to Processor Connection)

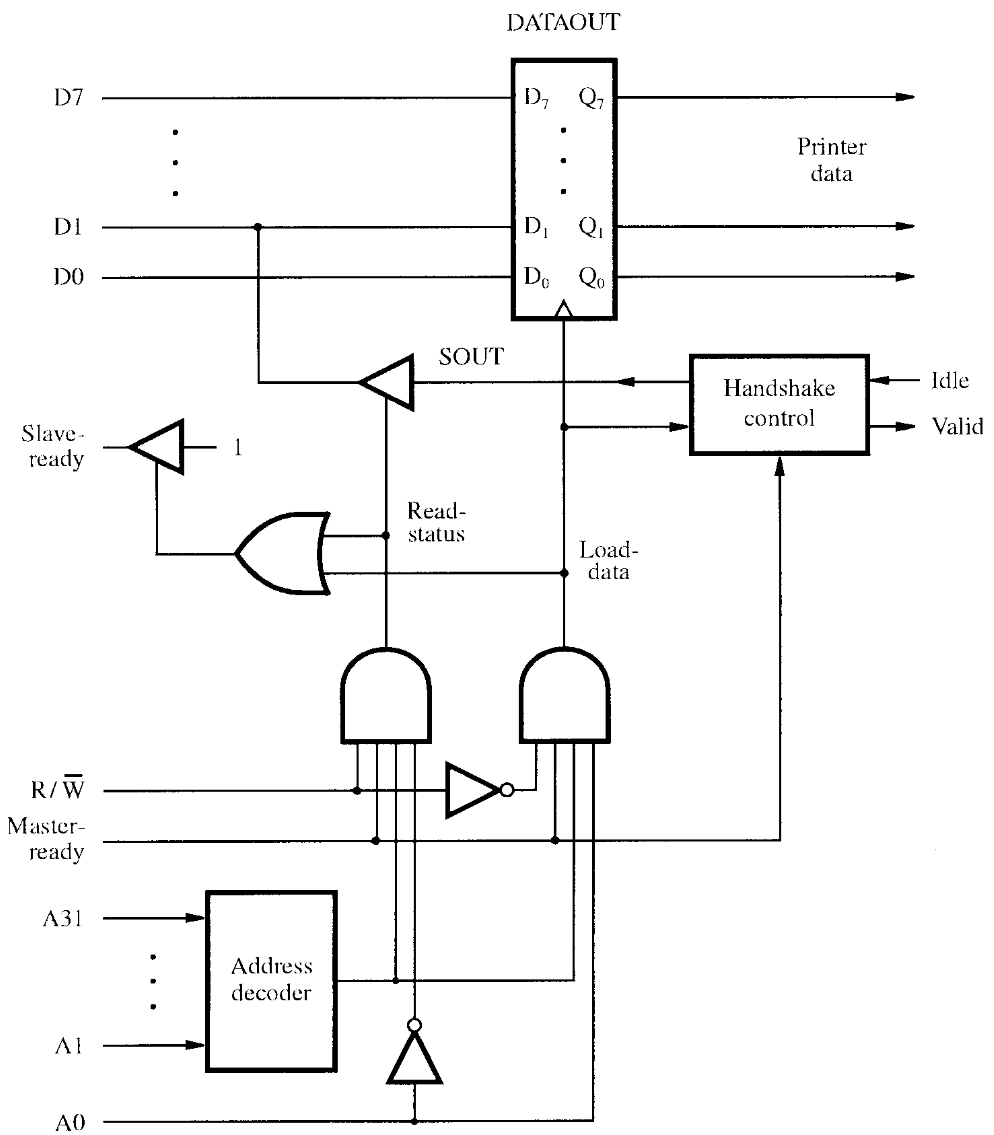


Figure 4.32. Output interface circuit.

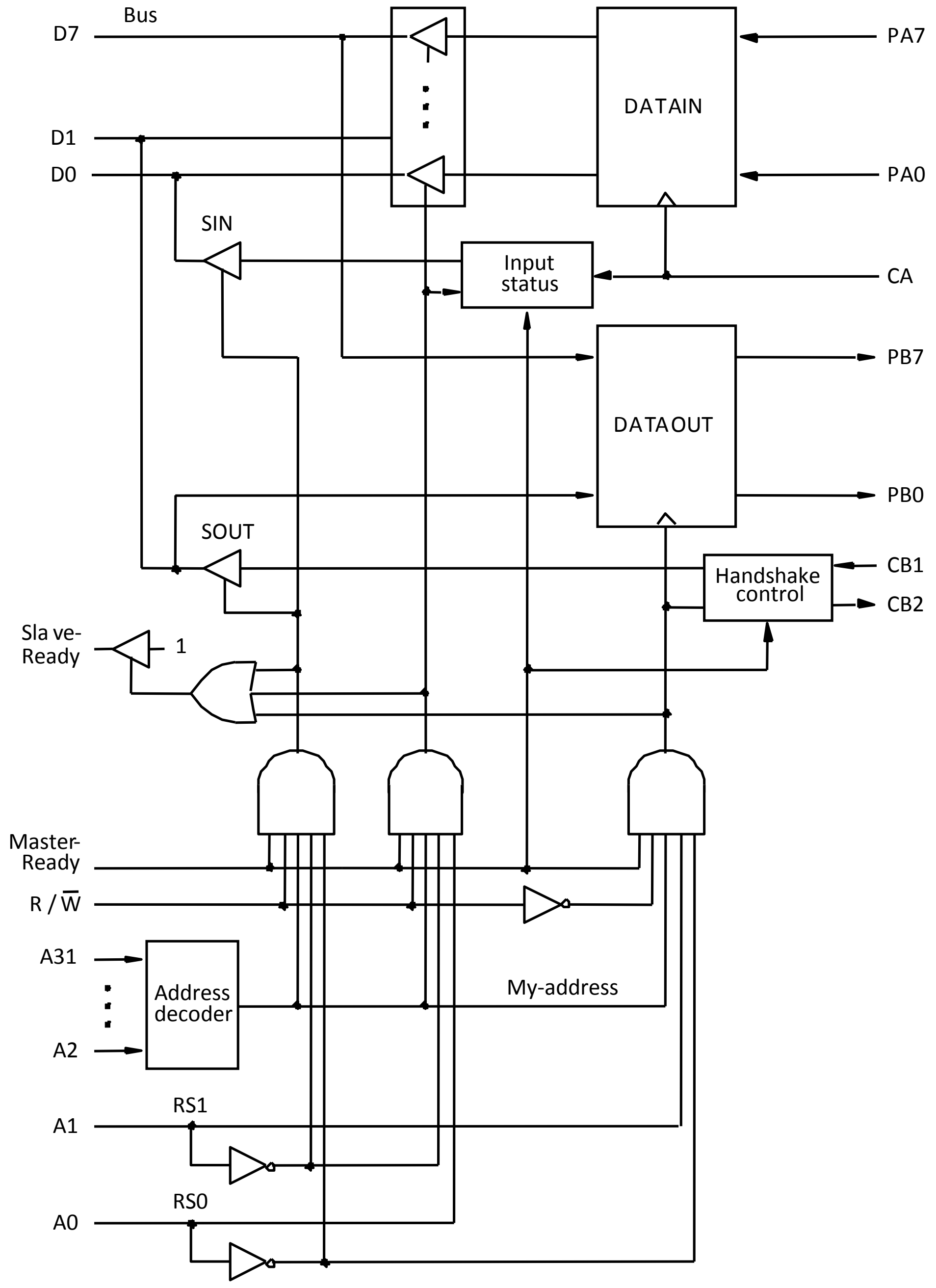
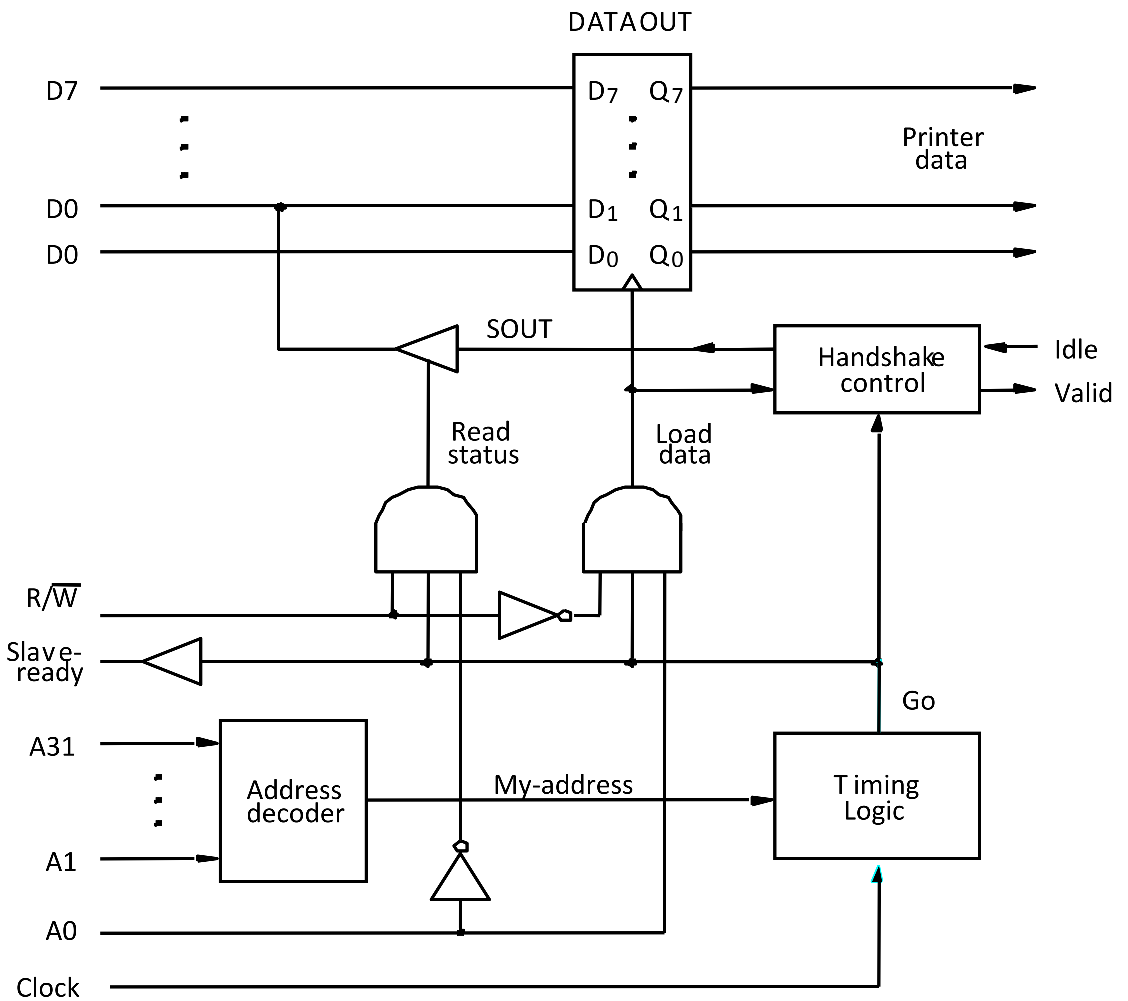


Figure 4.33. Combined input/output interface circuit.

Recall the Timing Protocol



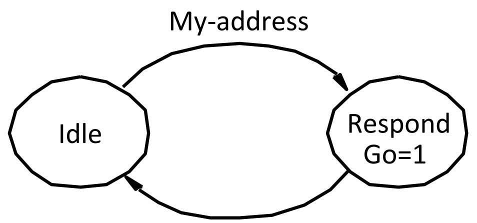


Figure 4.35. A parallel point interface for the bus of Figure 4.25, with a state-diagram for the timing logic.

Serial Port

- A serial port is used to connect the processor to I/O devices that require transmission of data one bit at a time.
- The key feature of an interface circuit for a serial port is that it is capable of communicating in bit-serial fashion on the device side and in a bit-parallel fashion on the bus side.
- Capable of longer distance communication than parallel transmission.

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,…

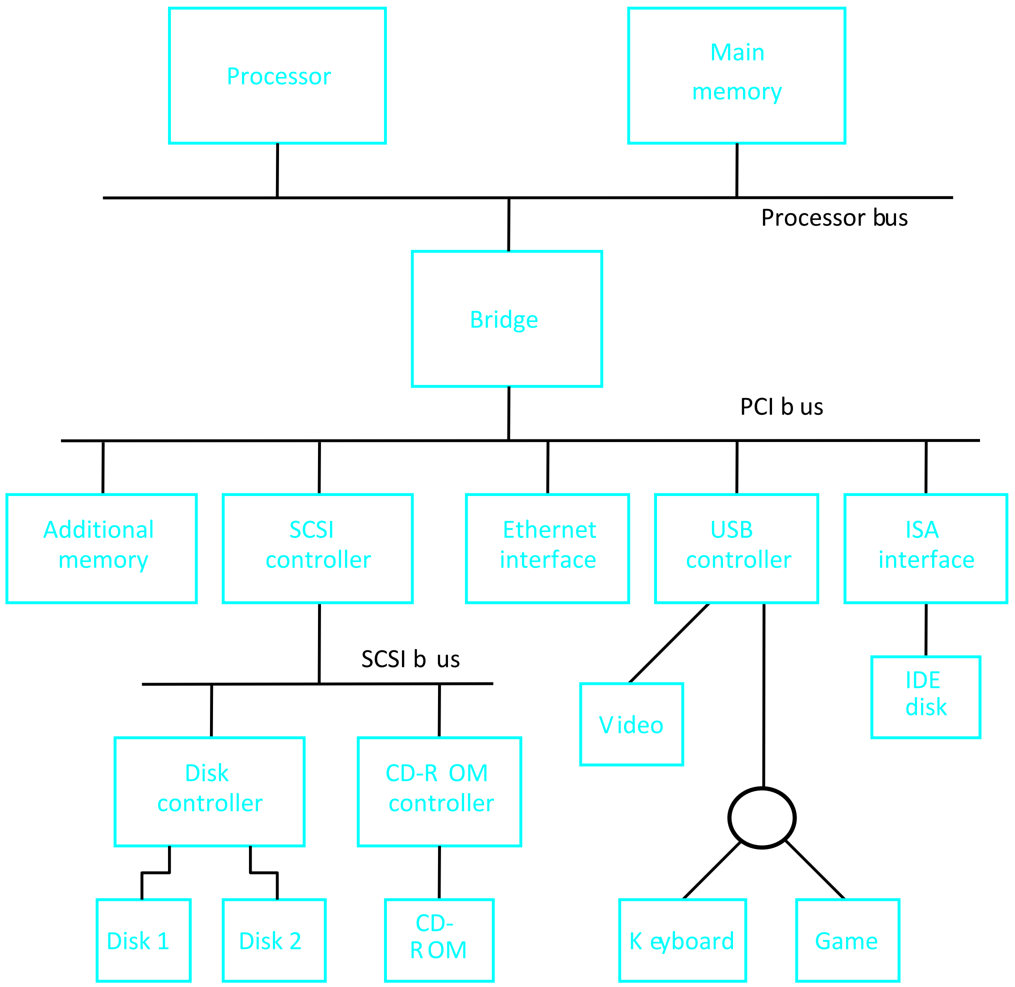


Figure 4.38. An example of a computer system using different interface standards.