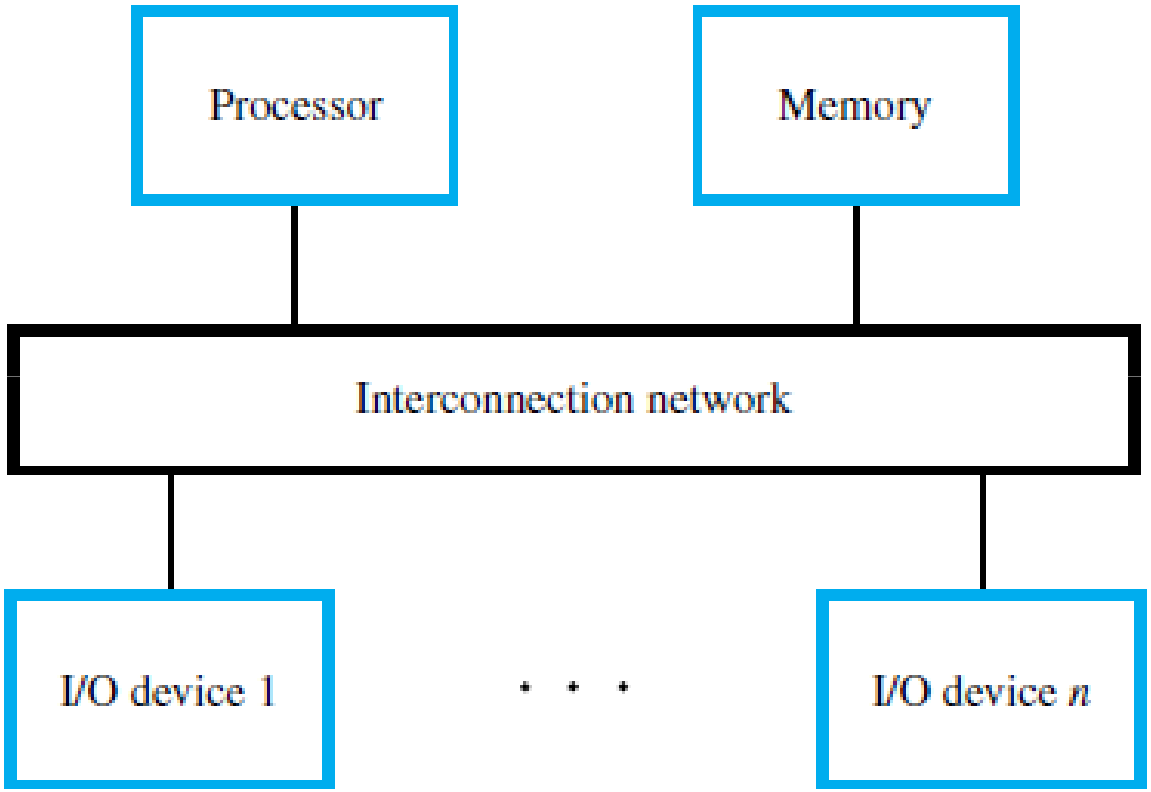
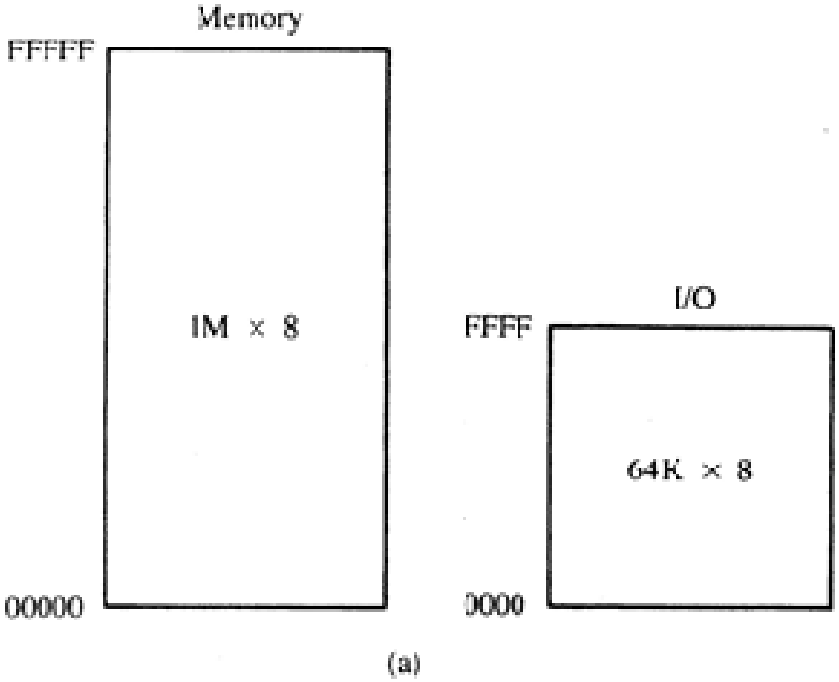
Input/Output Organization

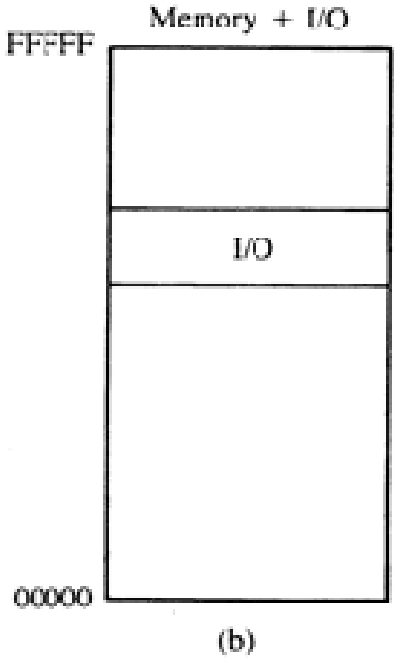
Accessing I/O Devices



Memory-mapped I/O

Accessing I/O Devices

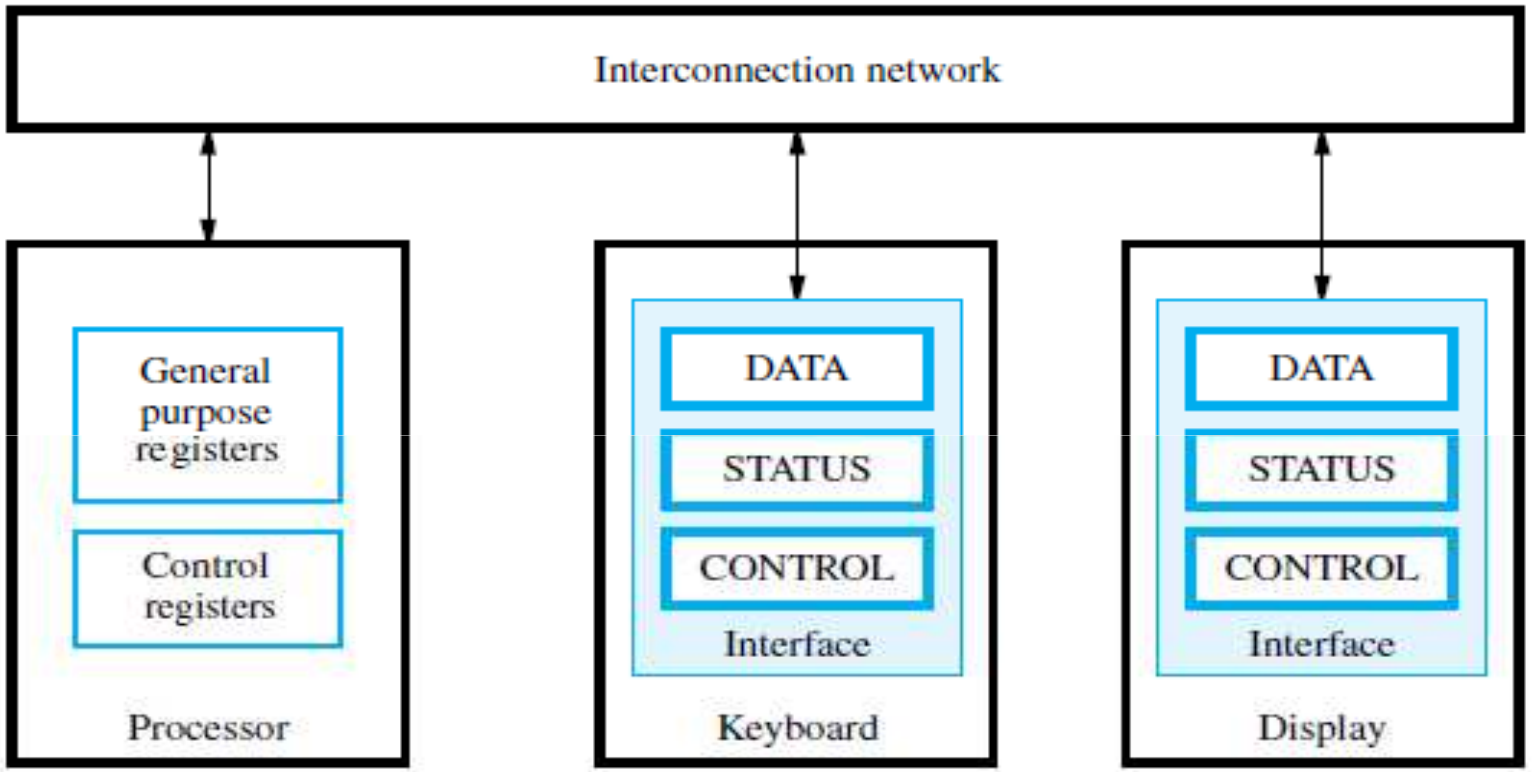




I/O Mapped I/O (Isolated I/O)

Memory Mapped I/O

I/O Device Interface



- **Load R2, DATAIN**
- **Store R2, DATAOUT**

Hardware needed to access I/O devices

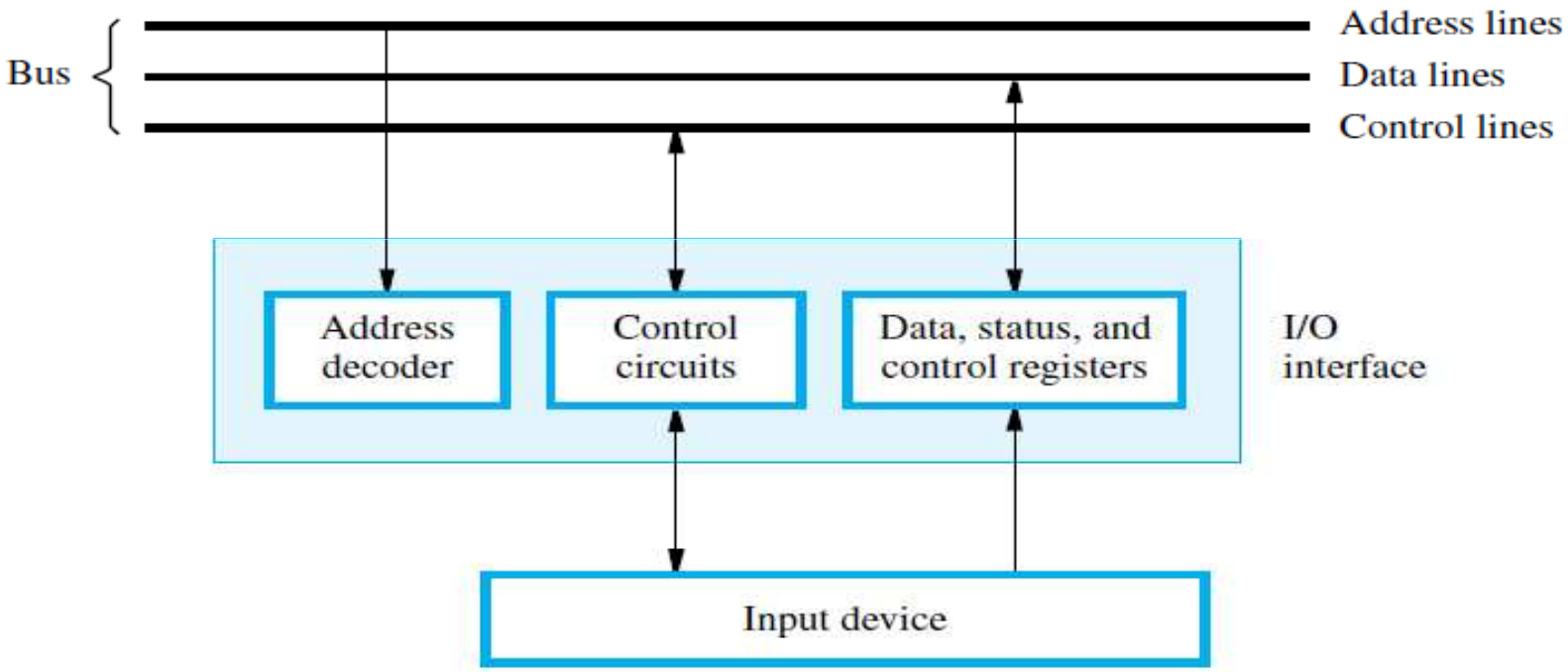


Figure 7.2 I/O interface for an input device.

Three Major Mechanisms

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

Program-Controlled I/O

- **Processor Polls the device**

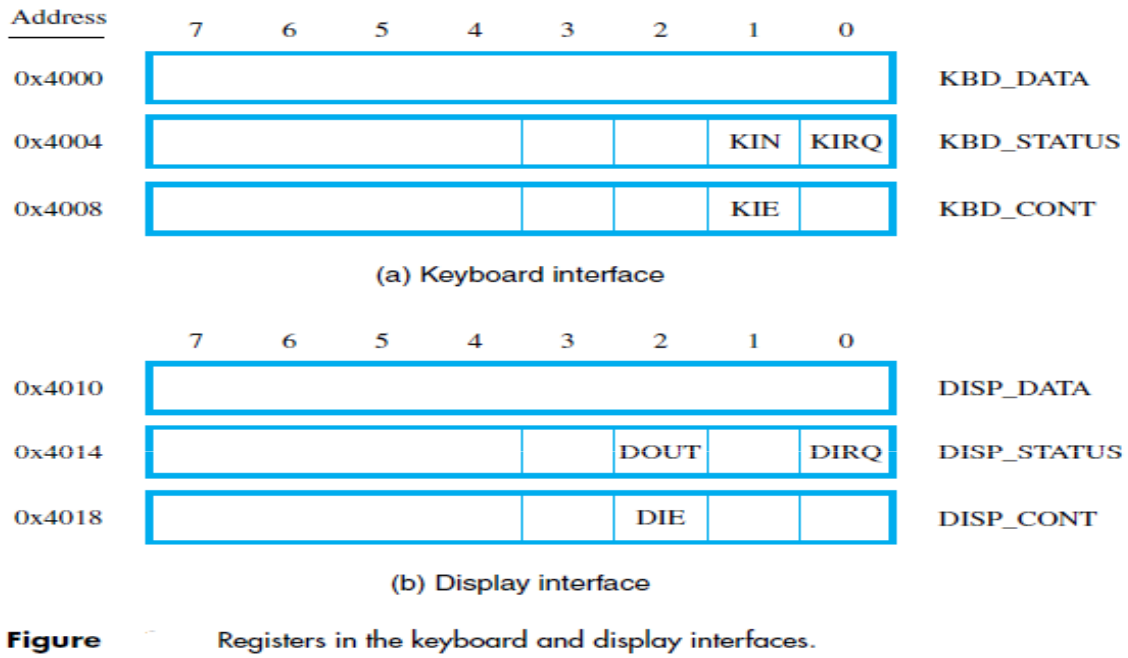


Figure Registers in the keyboard and display interfaces.

Initial state (power on)
KIN is 0
DOUT is 1

```
READWAIT    Read the KIN flag
            Branch to READWAIT if KIN = 0
            Transfer data from KBD_DATA to R5
```

```
WRITEWAIT   Read the DOUT flag
            Branch to WRITEWAIT if DOUT = 0
            Transfer data from R5 to DISP_DATA
```

Program-Controlled I/O

```
READWAIT    Read the KIN flag
            Branch to READWAIT if KIN = 0
            Transfer data from KBD_DATA to R5
```

```
READWAIT:   LoadByte        R4, KBD_STATUS
            And             R4, R4, #2
            Branch_if_[R4]=0 READWAIT
            LoadByte        R5, KBD_DATA
```

```
WRITEWAIT   Read the DOUT flag
            Branch to WRITEWAIT if DOUT = 0
            Transfer data from R5 to DISP_DATA
```

```
WRITEWAIT:  LoadByte        R4, DISP_STATUS
            And             R4, R4, #4
            Branch_if_[R4]=0 WRITEWAIT
            StoreByte       R5, DISP_DATA
```

Program-Controlled I/O

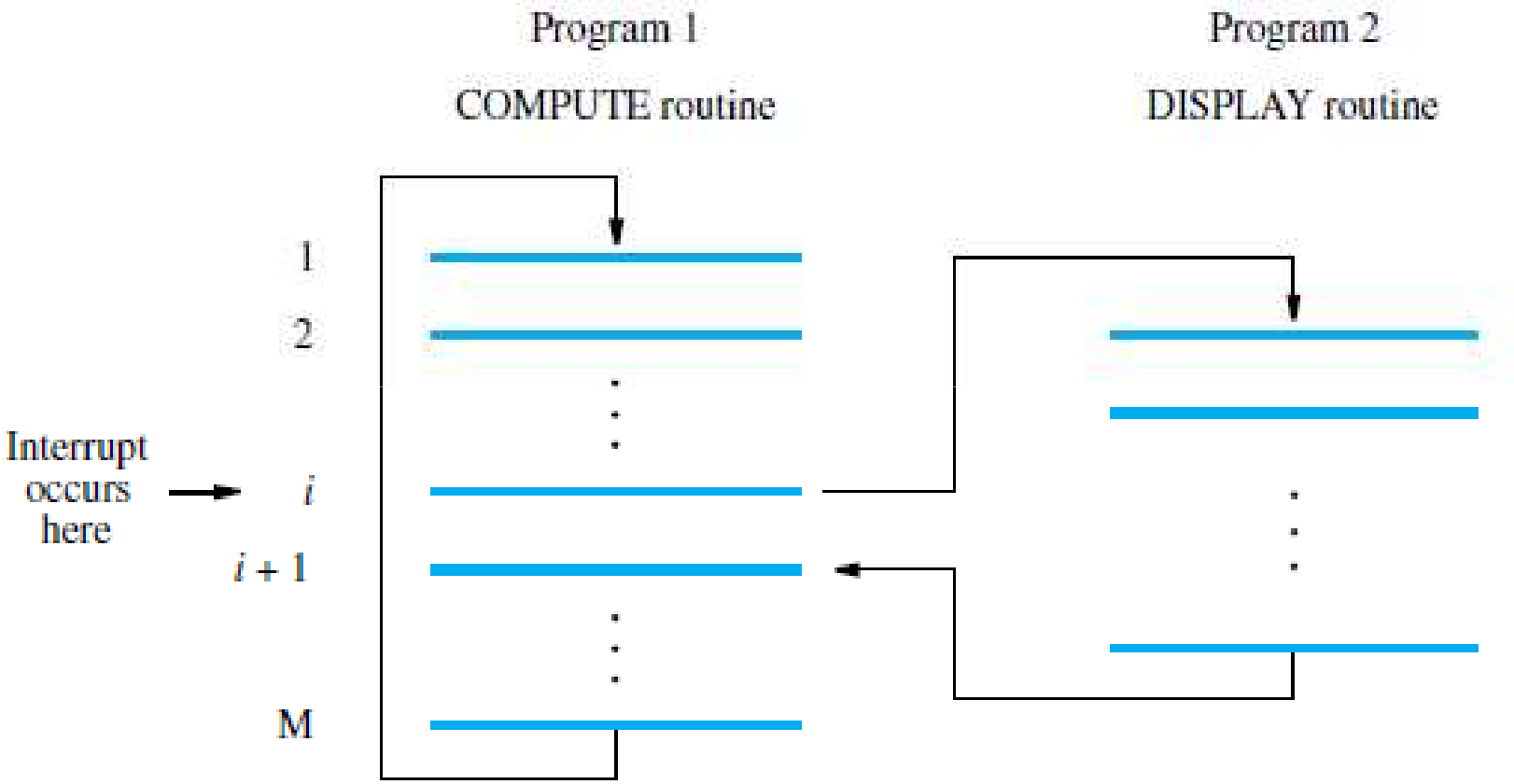
```
        Move          R2, #LOC        Initialize pointer register R2 to point to the
                                        address of the first location in main memory
                                        where the characters are to be stored.
        MoveByte      R3, #CR         Load ASCII code for Carriage Return into R3.
READ:   LoadByte      R4, KBD_STATUS  Wait for a character to be entered.
        And           R4, R4, #2      Check the KIN flag.
        Branch_if_[R4]=0  READ
        LoadByte      R5, KBD_DATA    Read the character from KBD_DATA
                                        (this clears KIN to 0).
        StoreByte     R5, (R2)        Write the character into the main memory and
        Add           R2, R2, #1        increment the pointer to main memory.
ECHO:   LoadByte      R4, DISP_STATUS Wait for the display to become ready.
        And           R4, R4, #4      Check the DOUT flag.
        Branch_if_[R4]=0  ECHO
        StoreByte     R5, DISP_DATA   Move the character just read to the display
                                        buffer register (this clears DOUT to 0).
        Branch_if_[R5]≠[R3]  READ     Check if the character just read is the
                                        Carriage Return. If it is not, then
                                        branch back and read another character.
```

Figure A RISC-style program that reads a line of characters and displays it.

Interrupts

- I/O device
  - raise the ***Interrupt request***
    - alert the processor when ready
    - hardware signal to the processor
- Processor
  - No longer poll the status of I/O devices
  - Use the waiting period to perform other useful tasks

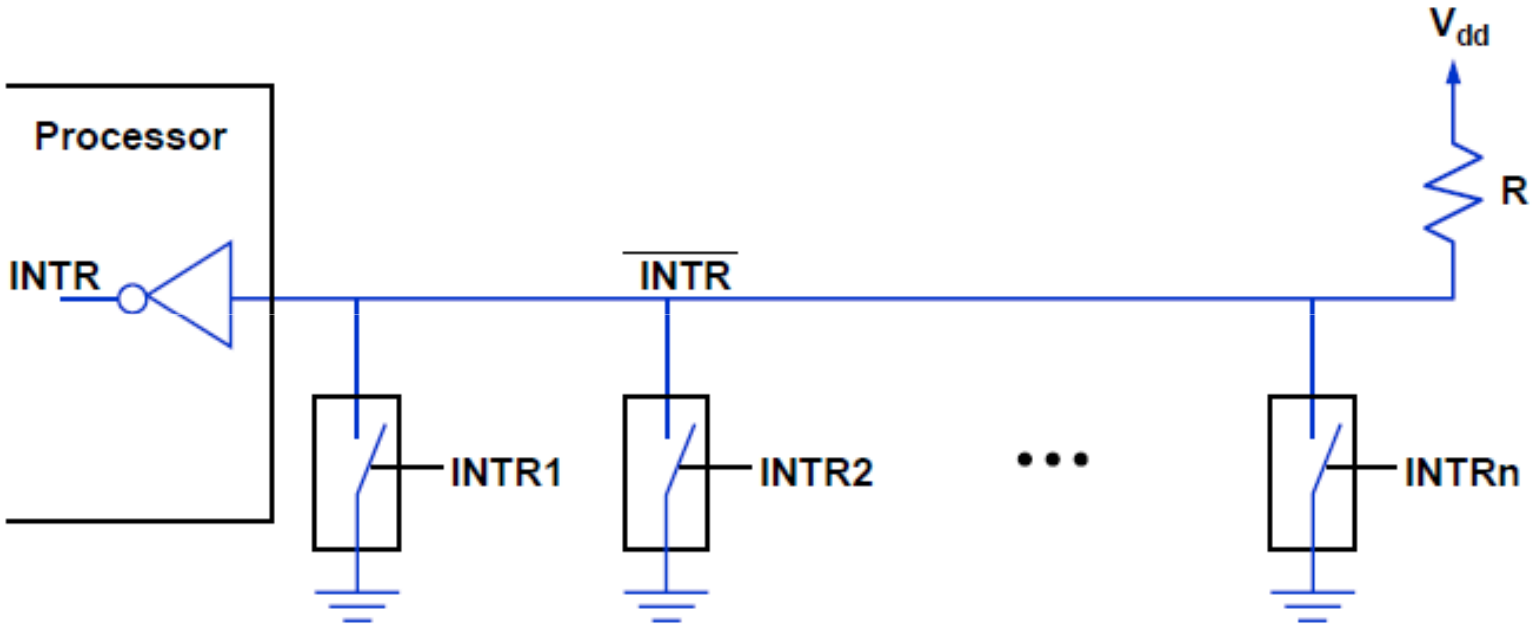
Interrupt


Program 1
COMPUTE routine
Program 2
DISPLAY routine
1
2
Interrupt occurs here
i
i + 1
M

Interrupt

- **Interrupt-service routine**
- **Store the current PC**
- **Return-from-interrupt**
- **Interrupt-acknowledge signal**
  - Implicitly : processor access to data register and status register
  - Explicitly : Interrupt acknowledge signal send to device through interconnection network
- ***Interrupt latency***
  - **Real-time processing –latency should be low**
- **Shadow register**

Interrupt Hardware



INTR=INTR1+INTR2+…+INTRn

Enabling and Disabling Interrupt

- Interrupt-request signal from a device will be active until the request completes

- So three possibilities used to handle one or more interrupt request.
  - 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

Summarize the sequence of events involved in handling an interrupt request from a single device

1. The device raises an interrupt request.
2. The processor interrupts the program currently being executed and saves the contents of the PC and PS registers.
3. Interrupts are disabled by clearing the IE bit in the PS to 0.
4. The action requested by the interrupt is performed by the interrupt-service routine, during which time the device is informed that its request has been recognized, and in response, it deactivates the interrupt-request signal.
5. Upon completion of the interrupt-service routine, the saved contents of the PC and PS registers are restored (enabling interrupts by setting the IE bit to 1), and execution of the interrupted program is resumed.

Handling Multiple Devices

- The situation where a number of devices capable of initiating interrupts are connected to the processor

Handling Multiple Devices

- Several devices may request interrupts at exactly the same time. This gives rise to a number of questions:

1. How can the processor determine which device is requesting an interrupt?
2. 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?
3. Should a device be allowed to interrupt the processor while another interrupt is being serviced?
4. How should two or more simultaneous interrupt requests be handled?

Handling Multiple Devices

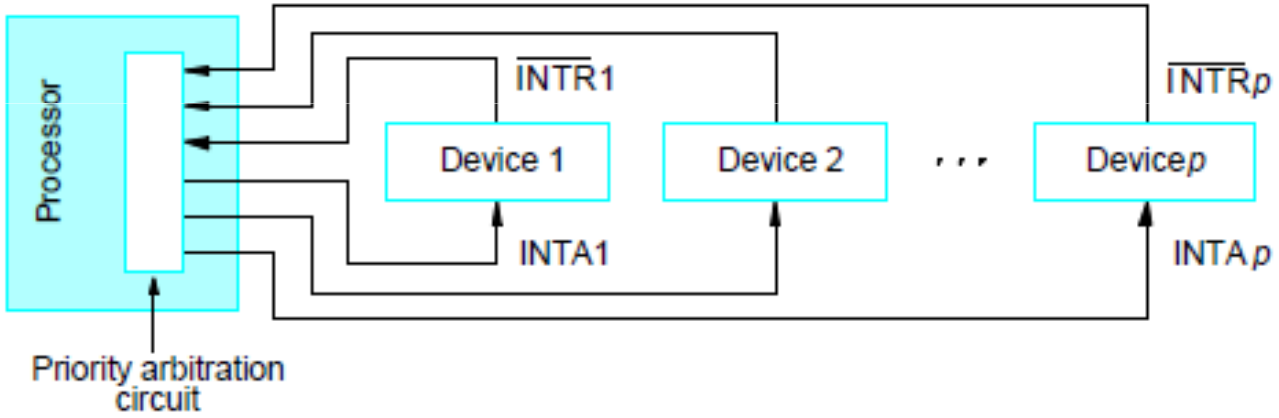
- Processor serve the device using
- Polling method
  - But polling need more time
  - Request is serviced in the order of polling
- So go for Vectored Interrupt
  - A device requesting an interrupt can identify itself by sending a special code to the processor over the bus

Handling Multiple Devices

- **Interrupt Nesting**
  - Accept one interrupt at a time, then disable all others.
  - Problem: some interrupts cannot be held too long
  - Priority structure

Handling Multiple Devices

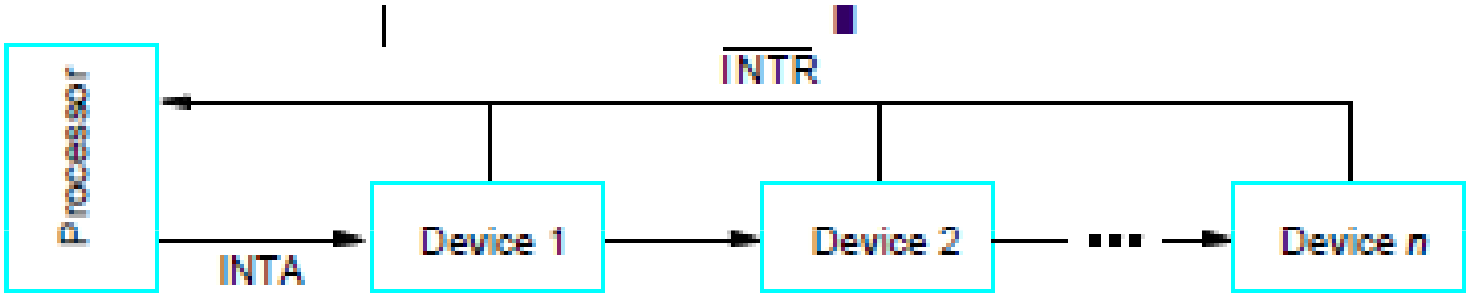
- Multiple priority scheme implemented using individual INTR and INTA lines.



- Priority arbitration circuit: A logic circuit which combines all interrupts but allows only the highest-priority request.

Handling Multiple Devices

- **Daisy Chain**

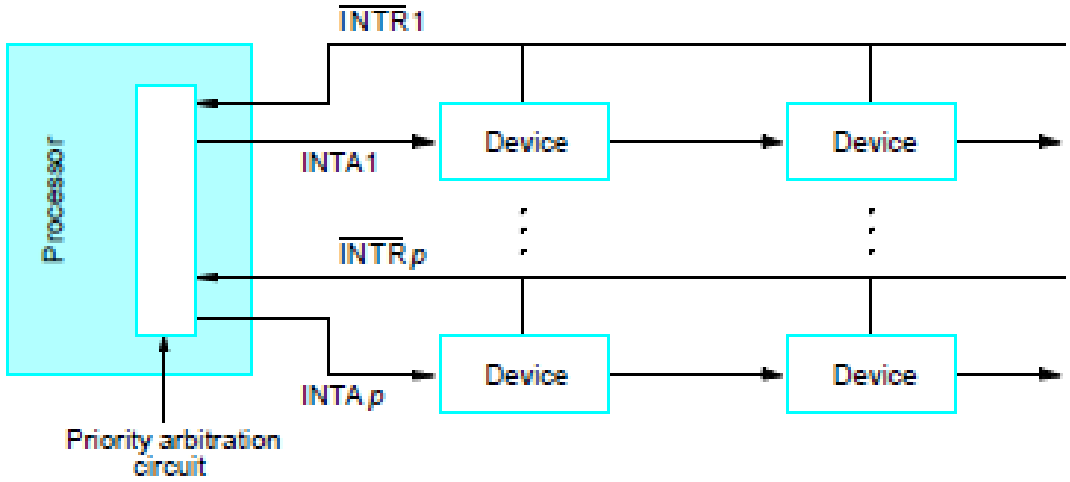


(a) Daisy chain

- INTR is common to all device
- INTA connected in daisy chain fashion where INTA signal propagate serially through the devices.

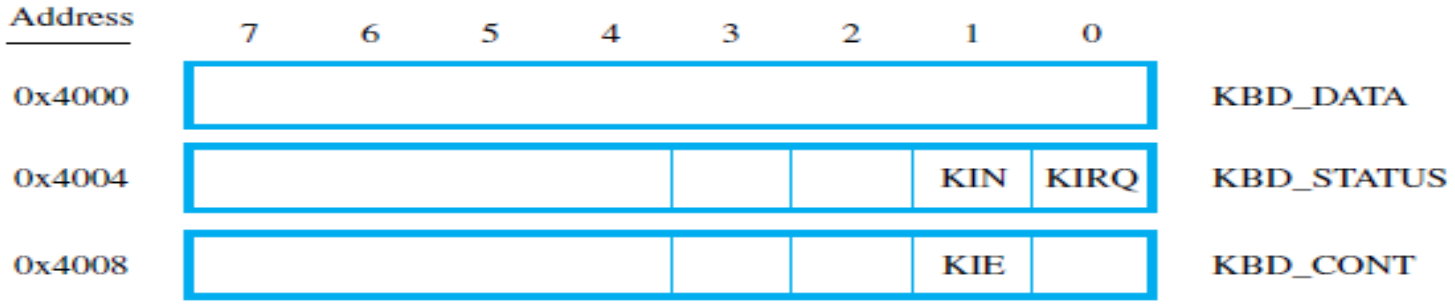
Handling Multiple Devices

- Combining the Multiple-priority and daisy chain.
- Where device organized in group and each have different priority level

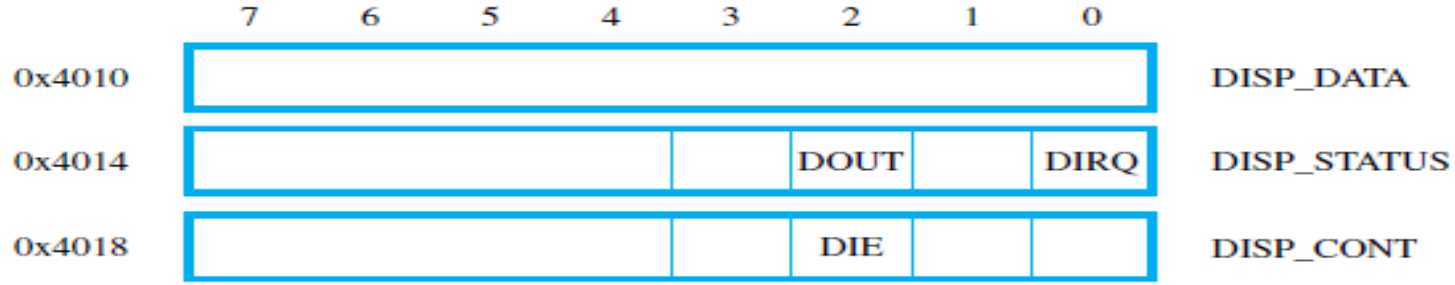


(b) Arrangement of priority groups

Controlling Device Request
Device side



(a) Keyboard interface



(b) Display interface

Figure Registers in the keyboard and display interfaces.

When KIIE=1, KIN =1 then KIRQ can be set 1

Controlling Device Request
Processor side

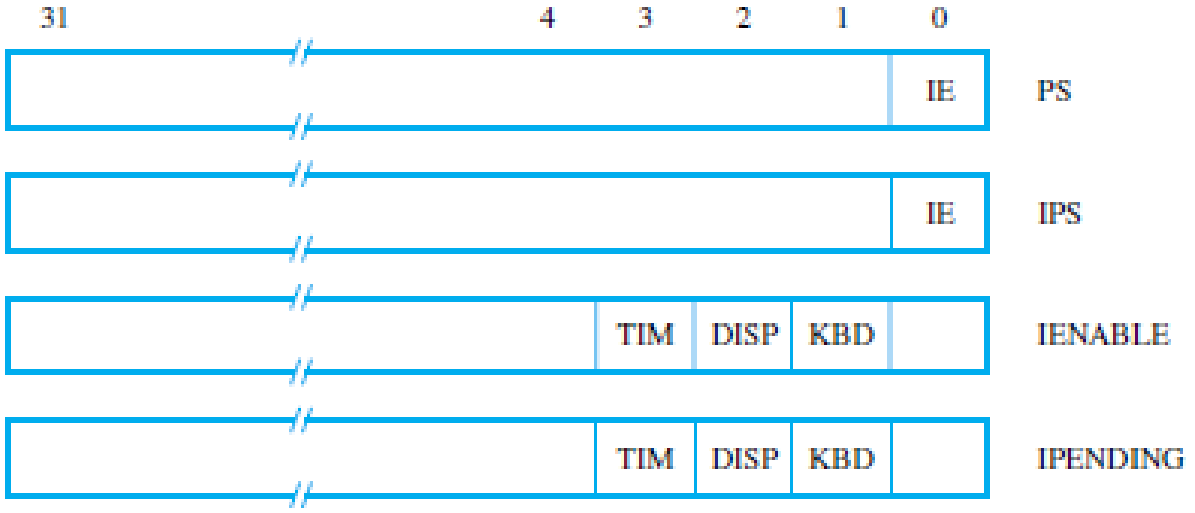
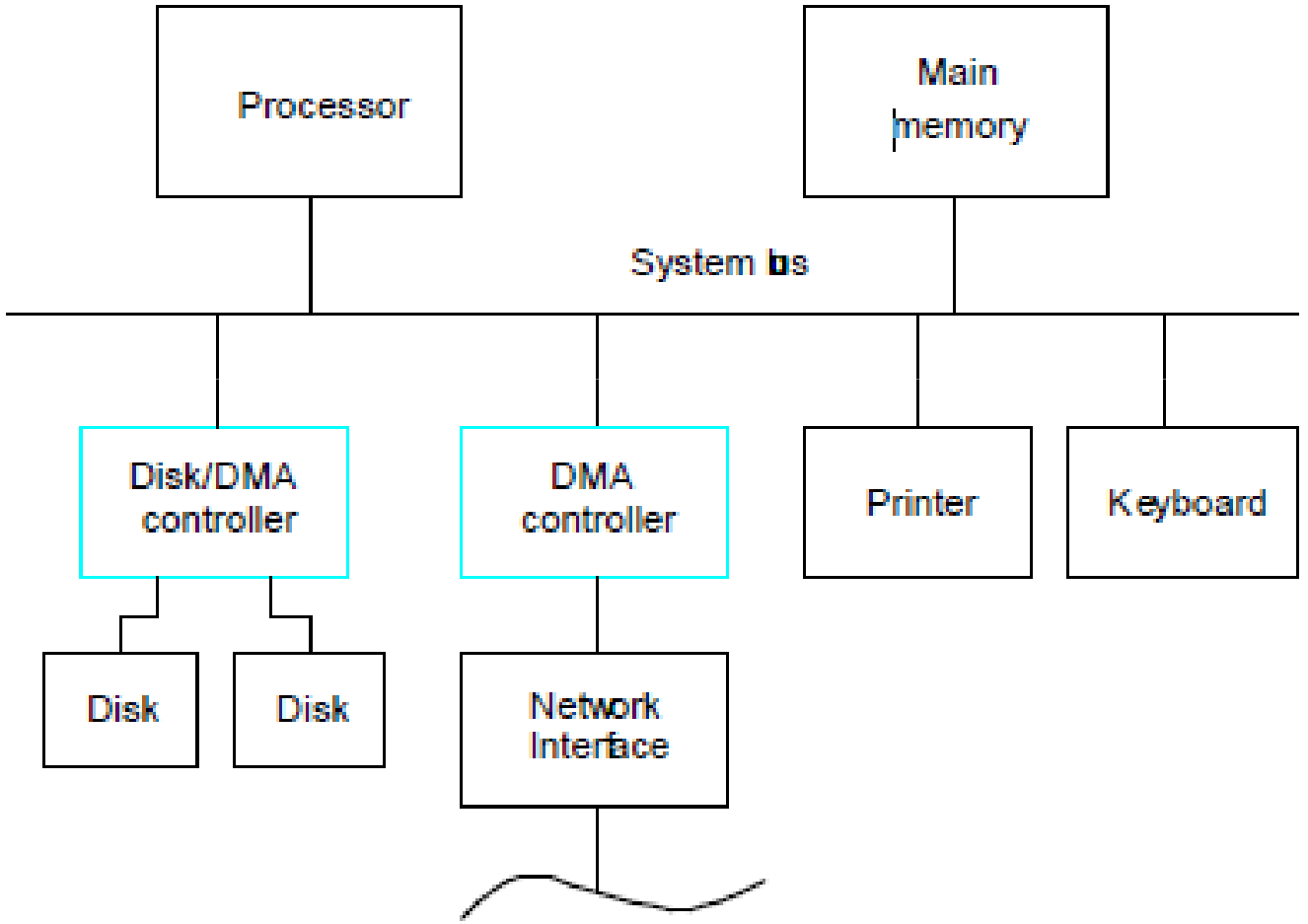


Figure Control registers in the processor.

Exceptions

- Any event that causes an interruption.
  - Recovery from errors
  - Debugging
    - Trace
    - Breakpoint
  - Privilege exception

Direct Memory Access (DMA)



Direct Memory Access (DMA)

- Special control unit (DMA Controller)
- Used to transfer large blocks of data at high speed between external device and main memory
- No need of continuous intervention by the processor
  - **the starting address, the number of words in the block, and the direction of the transfer**
- DMA controller proceeds to perform the requested operation.
  - Two modes
    - **cycle stealing**
    - **Block or burst mode**
  - **When the entire block has been transferred, the controller informs the processor by raising an interrupt signal.**
  - **Then the program that raised the transfer will resume**

Direct Memory Access (DMA)

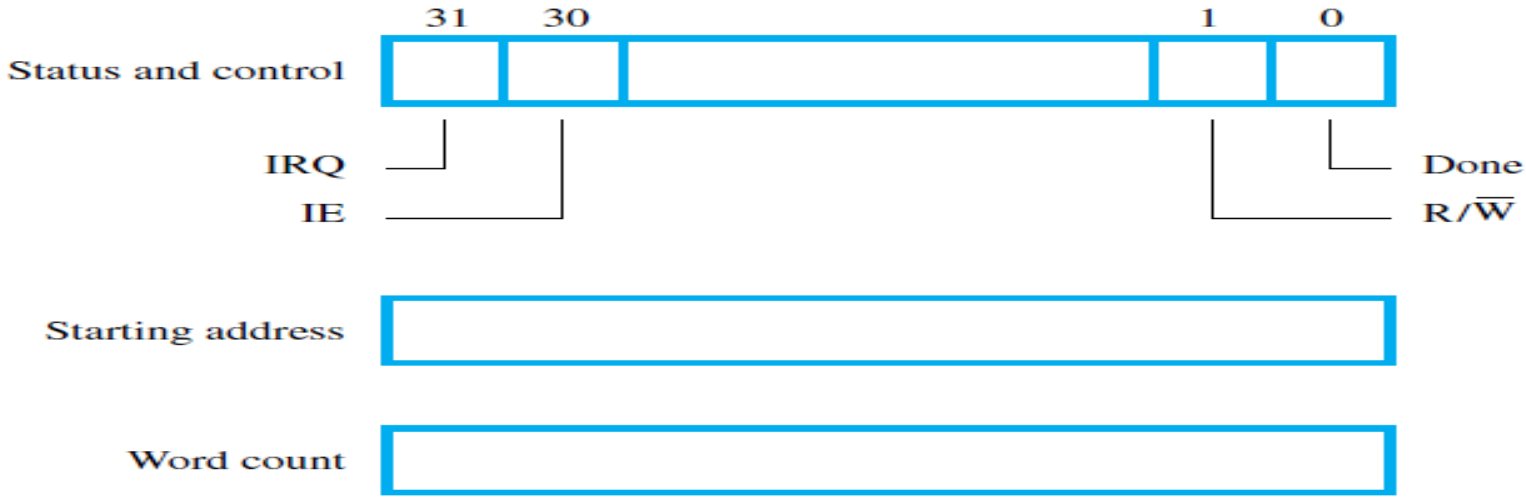


Figure 8.12 Typical registers in a DMA controller.

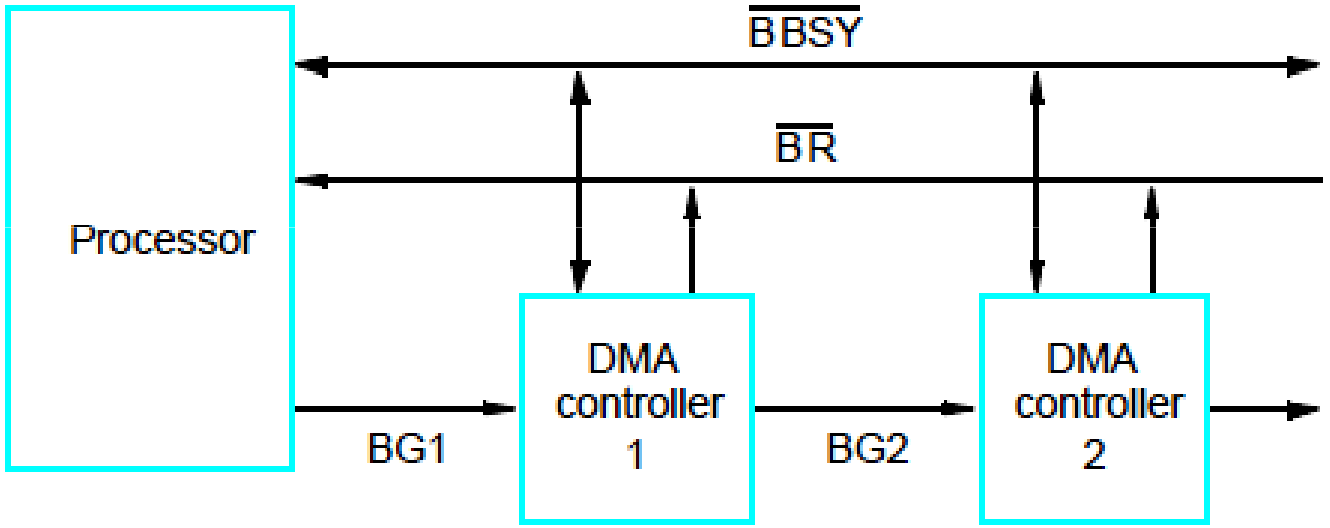
Direct Memory Access (DMA)

- **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**

Direct Memory Access (DMA)

Bus Arbitration

Centralized Arbitration

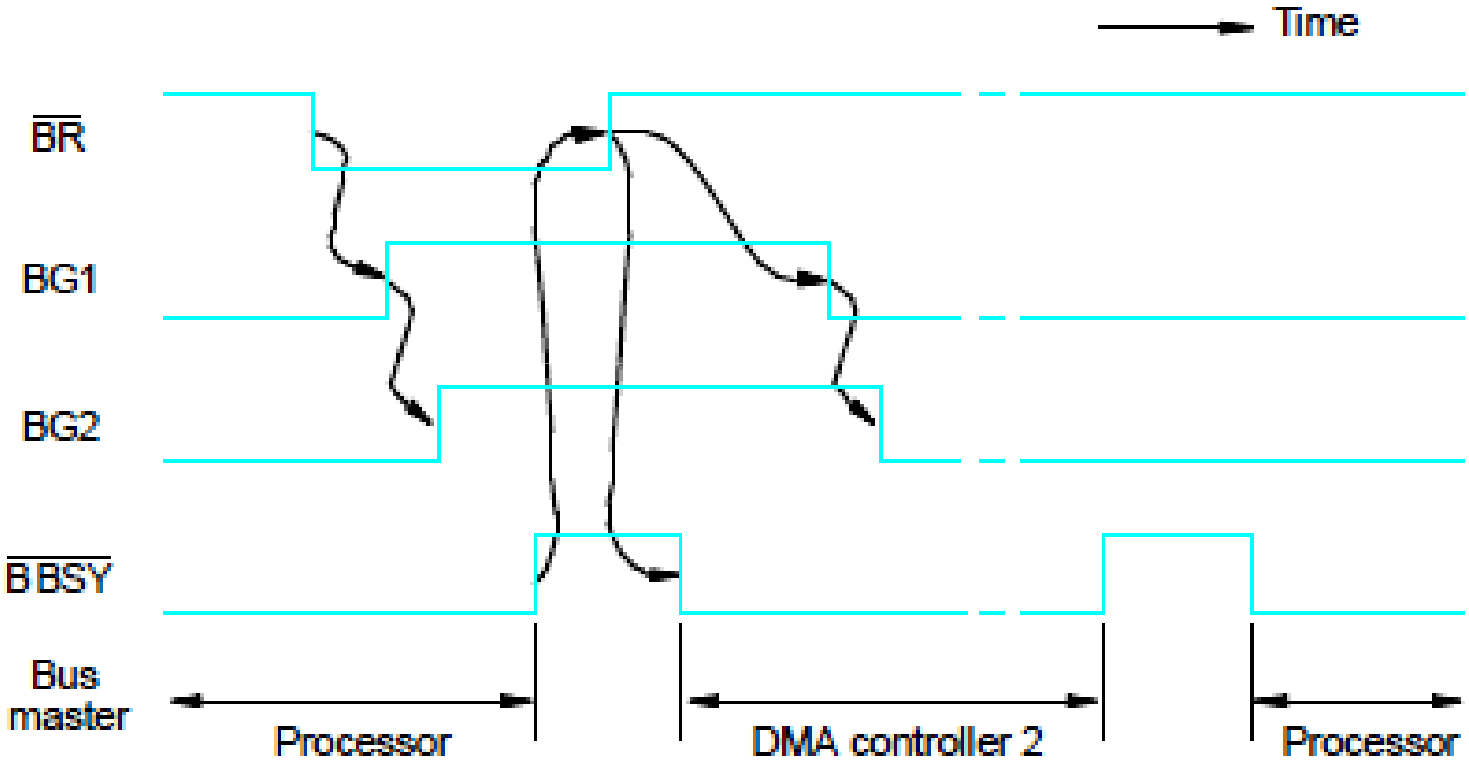


- The bus arbiter may be the processor or a separate unit
- The processor is normally the bus master unless it grants bus mastership to one of the DMA controllers

Direct Memory Access (DMA)

Bus Arbitration

Centralized Arbitration



Direct Memory Access (DMA)

Bus Arbitration

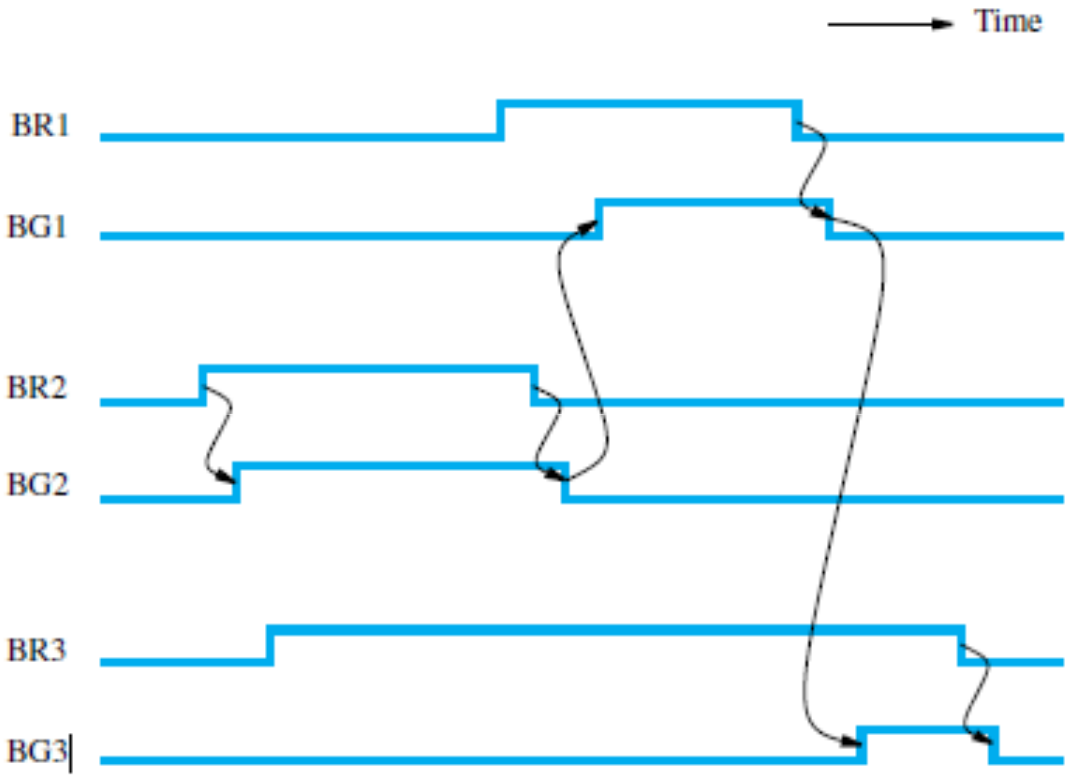
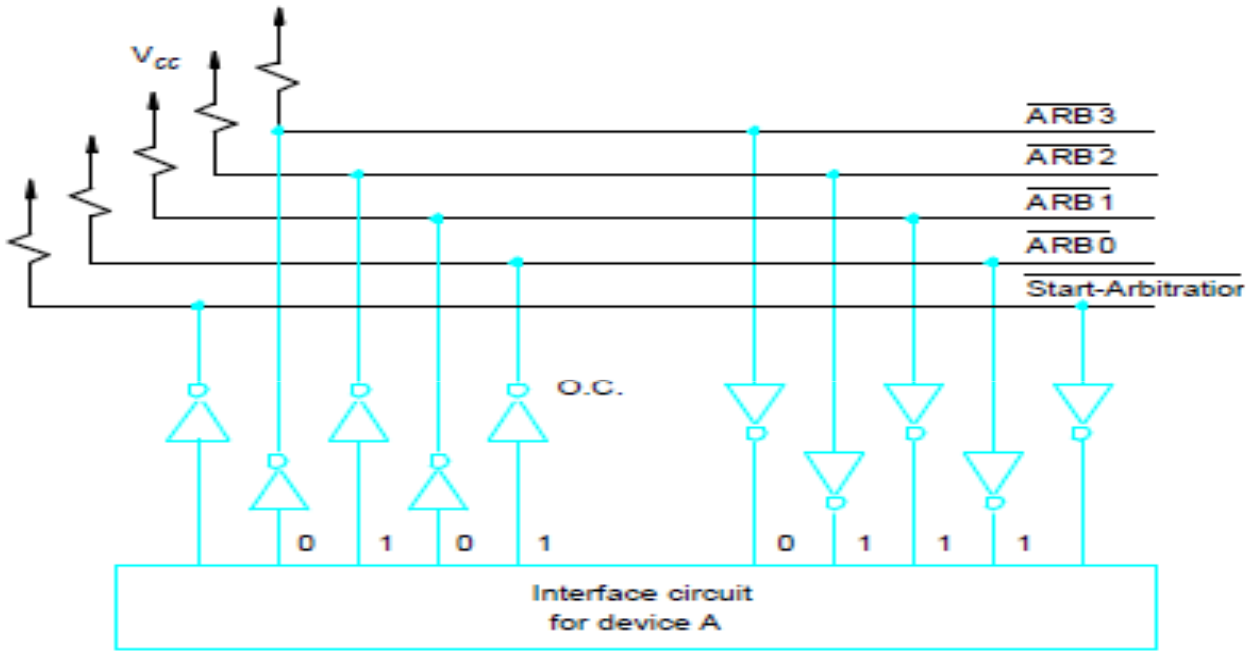


Figure Granting use of the bus based on priorities.

Direct Memory Access (DMA)

Bus Arbitration

Distributed Arbitration



- ID 5 (0101) and 6 (0110)
- seen by both device is 0111
- Compares pattern and ID5 change to 0100
- 6 wins

Buses

- The bus protocol - rules are implemented by control signals that indicate what and when actions are to be taken.
- **Master or initiator- Device which initiate data transfers**
- **Slave or target- Device addressed by master**

Buses- Synchronous Bus

- All devices derive timing information from a control line called the *bus clock*

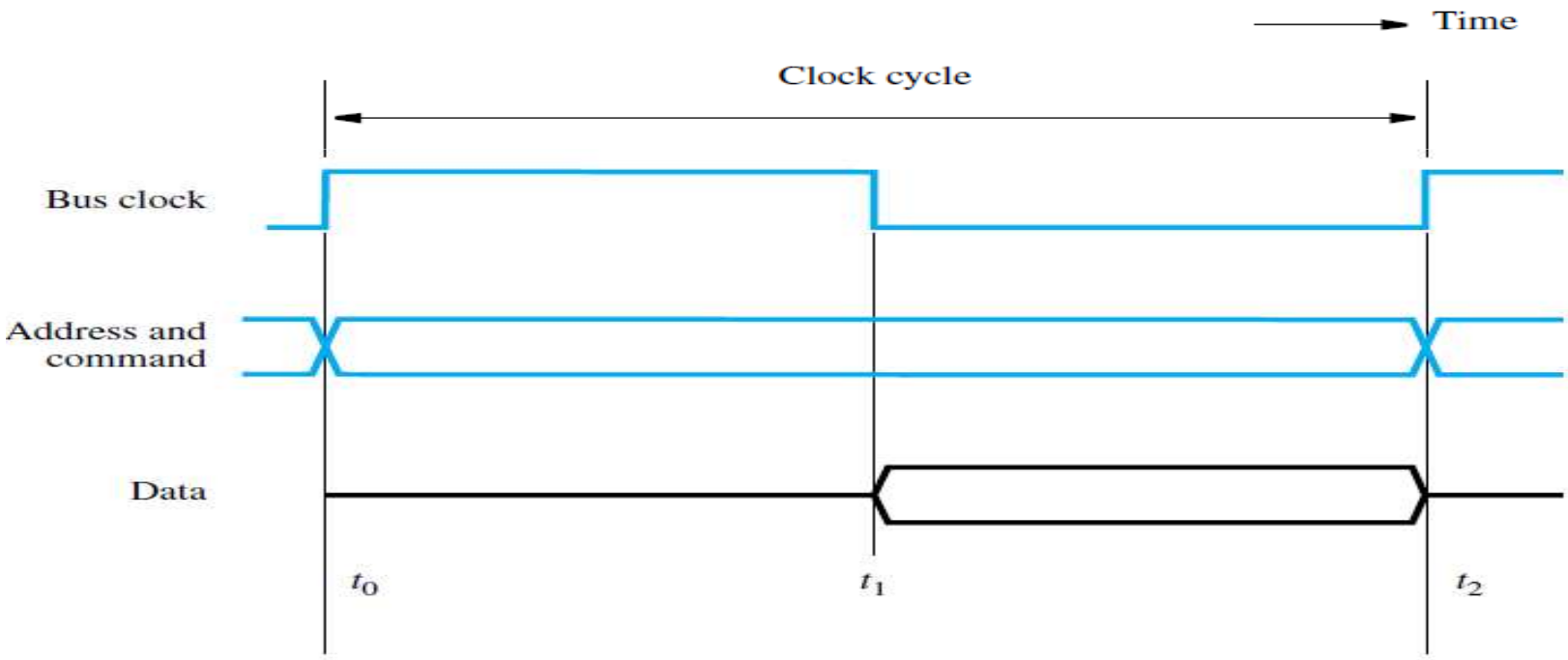


Figure 7.3 Timing of an input transfer on a synchronous bus.

Bus Synchronous Bus

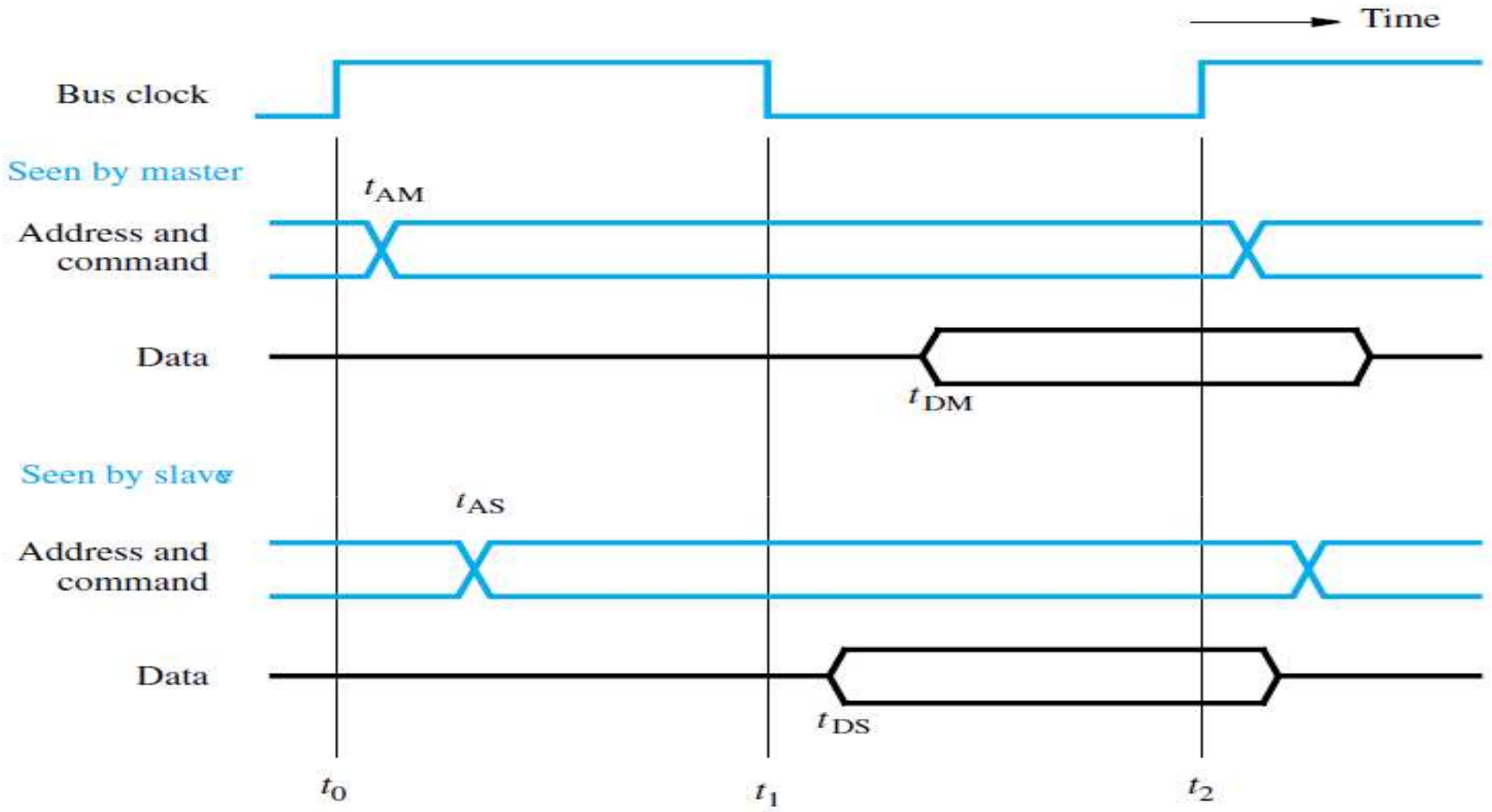


Figure 7.4 A detailed timing diagram for the input transfer of Figure 7.3.

The **exact times at** which signals change state are somewhat different from **idealized representation**, because of propagation delays on bus wires and in the circuits of the devices.

Buses Synchronous Bus
Multiple-Cycle Data Transfer

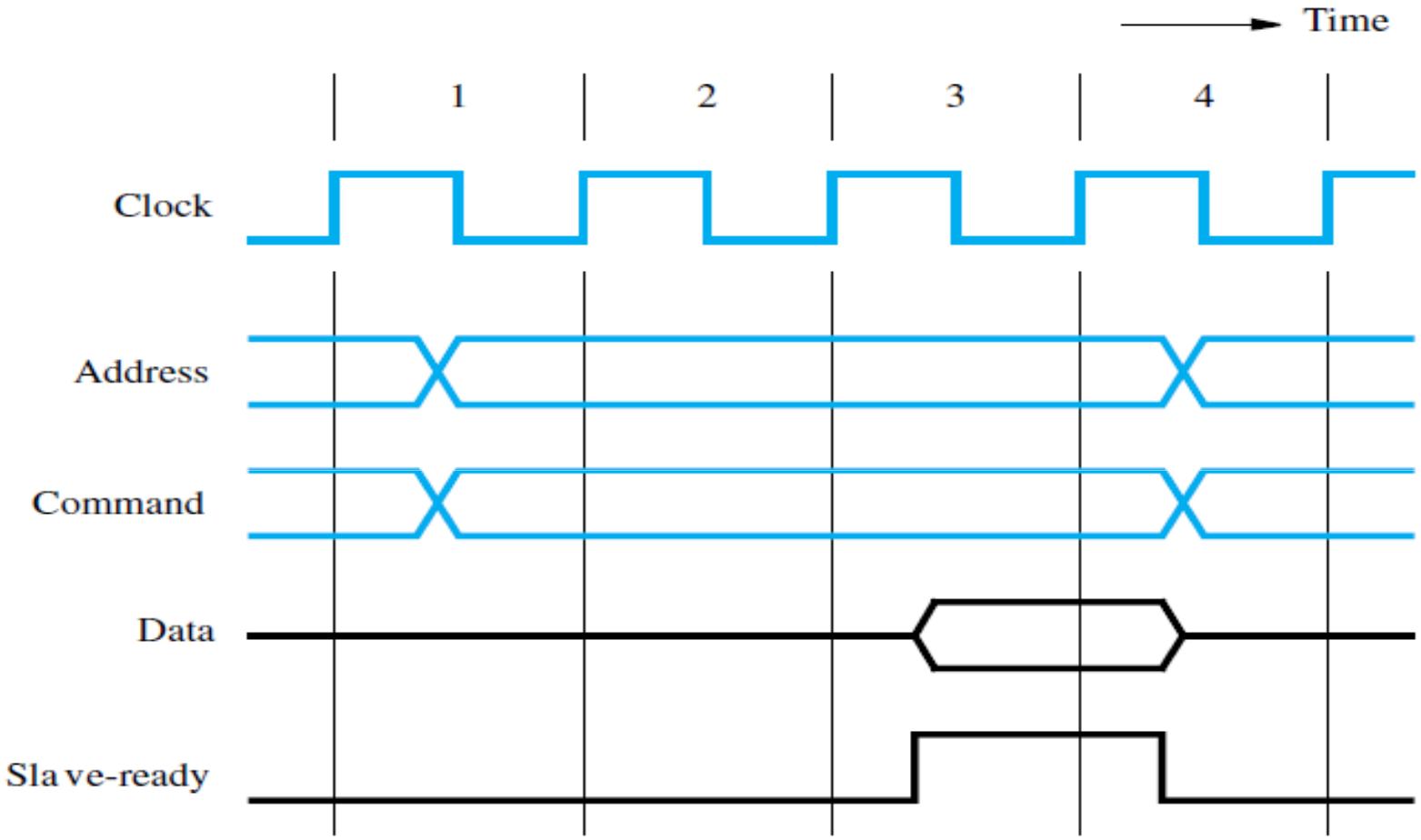


Figure 7.5 An input transfer using multiple clock cycles.

Single Cycle: This forces all devices to operate at the speed of the slowest device
Multi Cycle : incorporate control signals that represent a **response from the device**.

Buses Asynchronous Bus

- *Handshake* protocol between the master and the slave.
- A handshake is an exchange of command and response signals between the master and the slave.

Buses Asynchronous Bus

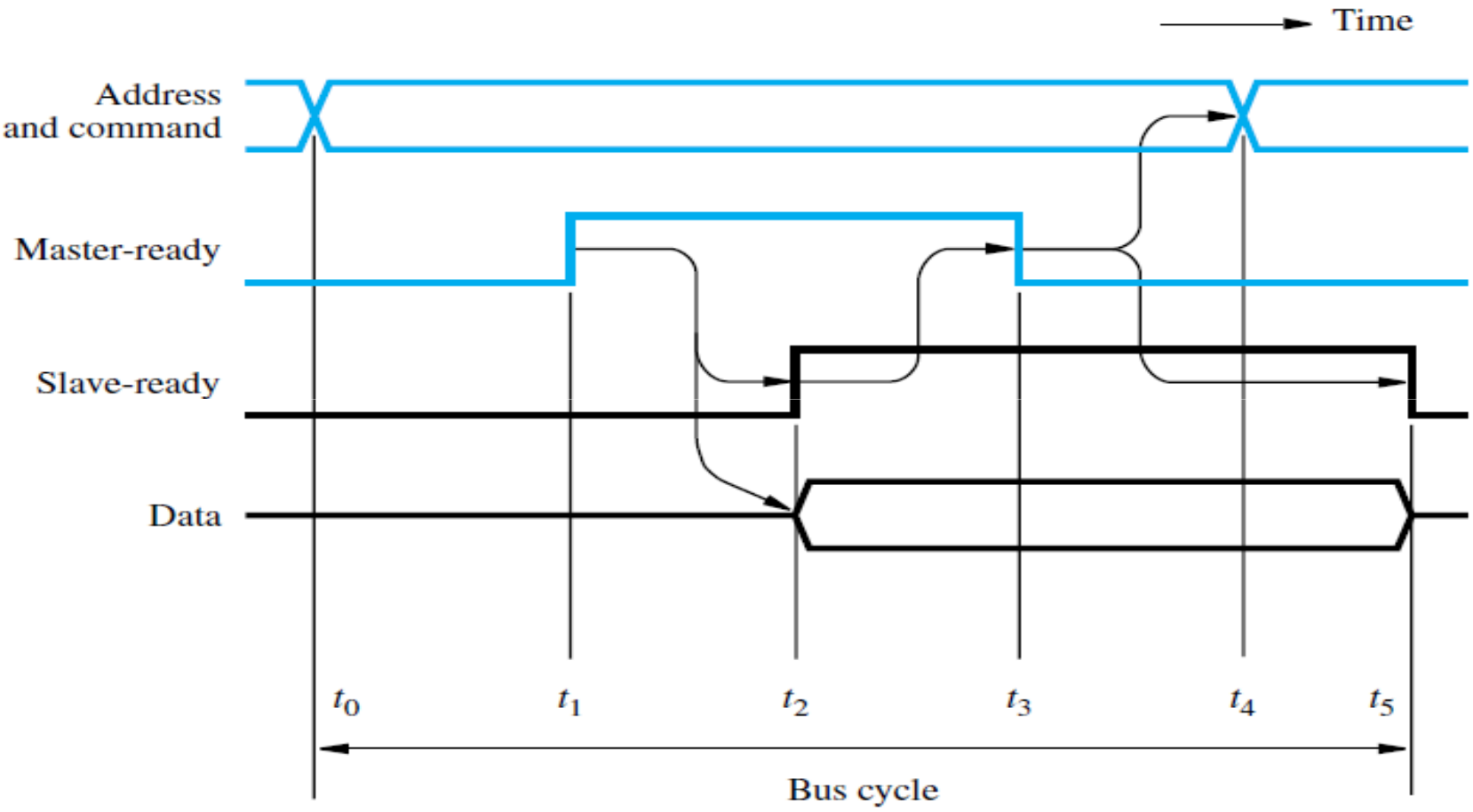


Figure 7.6 Handshake control of data transfer during an input operation.

Buses Asynchronous Bus

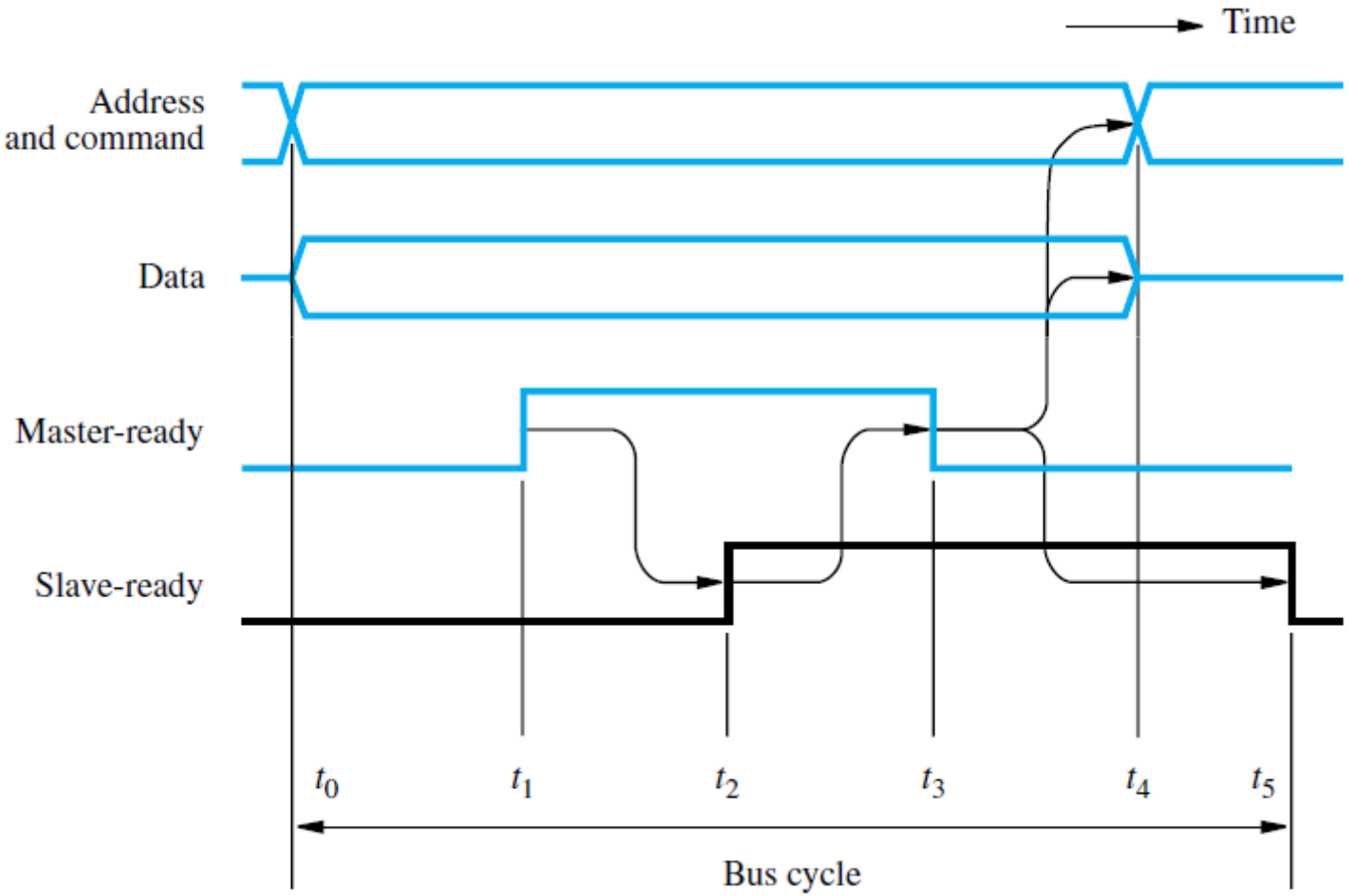


Figure 7.7 Handshake control of data transfer during an output operation.

Interface Circuits

Interface circuits-1

- Port : side of the interface which connects to the I/O device
  - Parallel port
  - Serial port

Parallel port

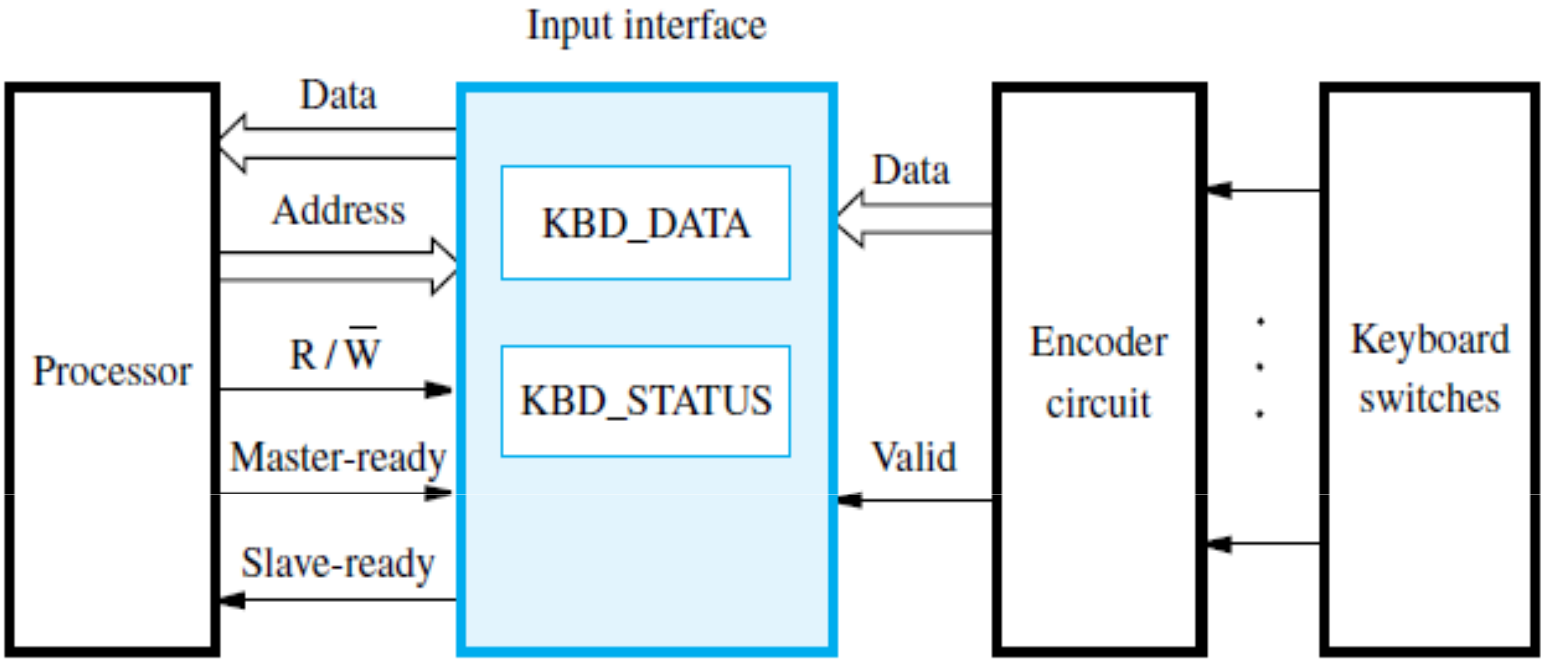


Figure Keyboard to processor connection.

Debouncing circuit which eliminates the effect of a key bounce (a single key stroke may appear as multiple events to a processor).

Input Interface Circuit

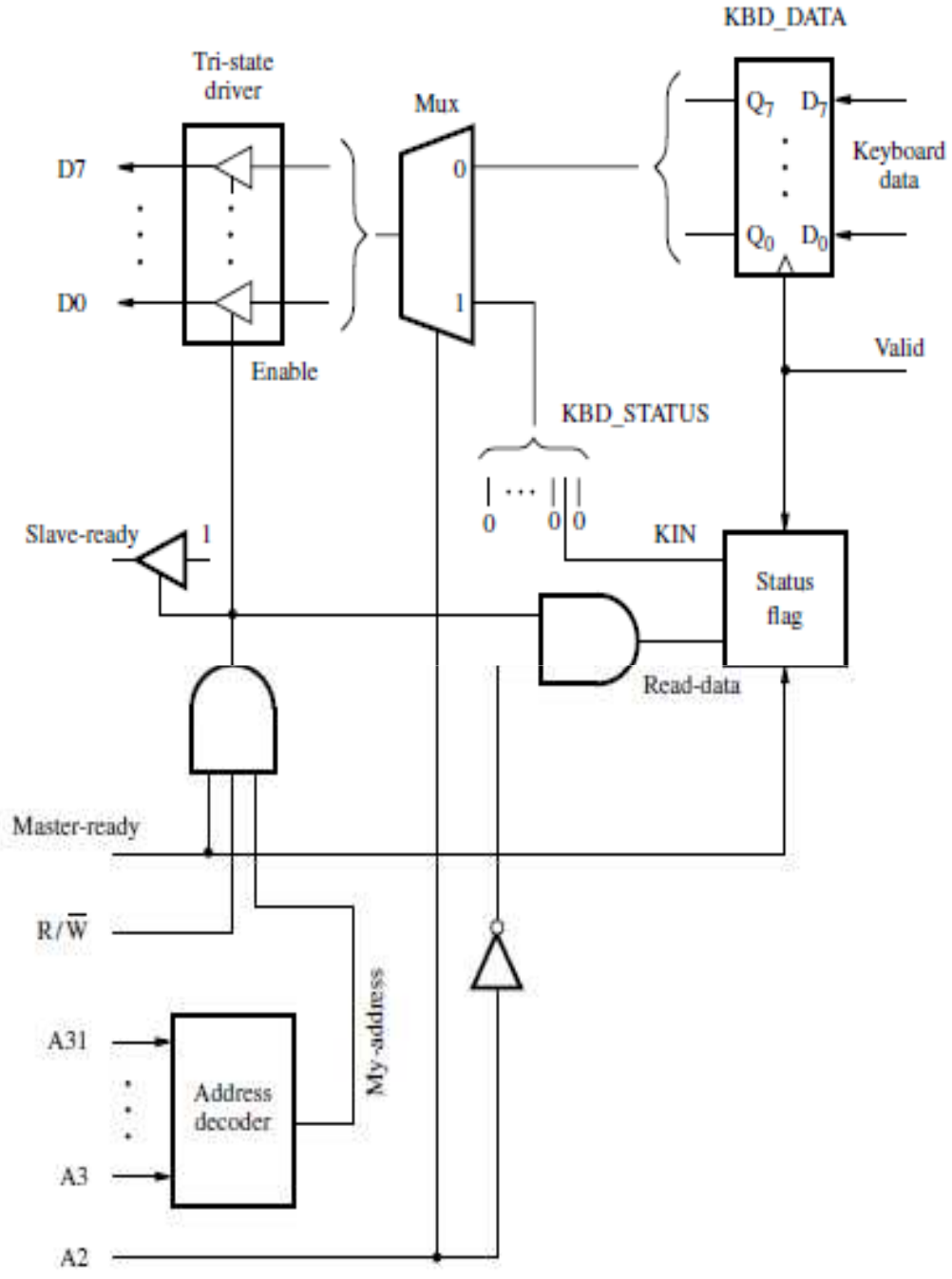


Figure An input interface circuit.

Input Interface Circuit

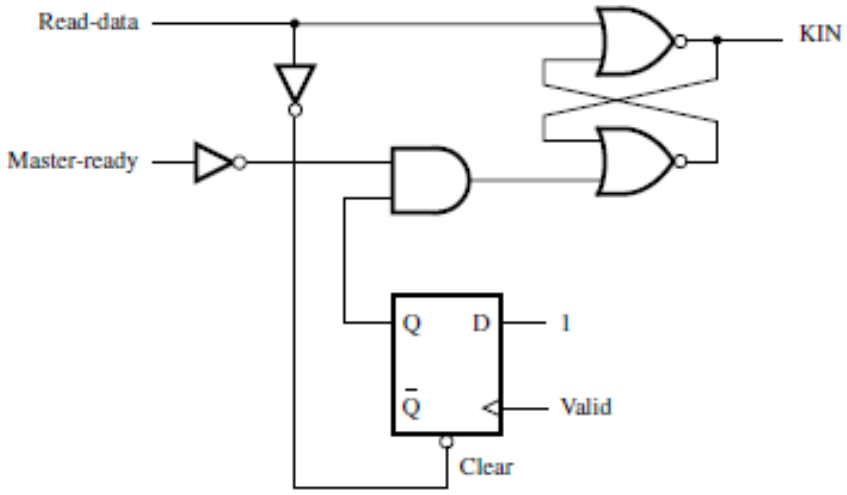
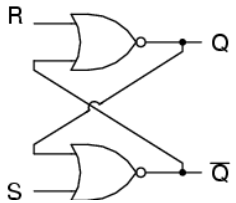


Figure Circuit for the status flag block



S	R	Q	$\overline{Q}$
0	0	latch	latch
0	1	0	1
1	0	1	0
1	1	0	0

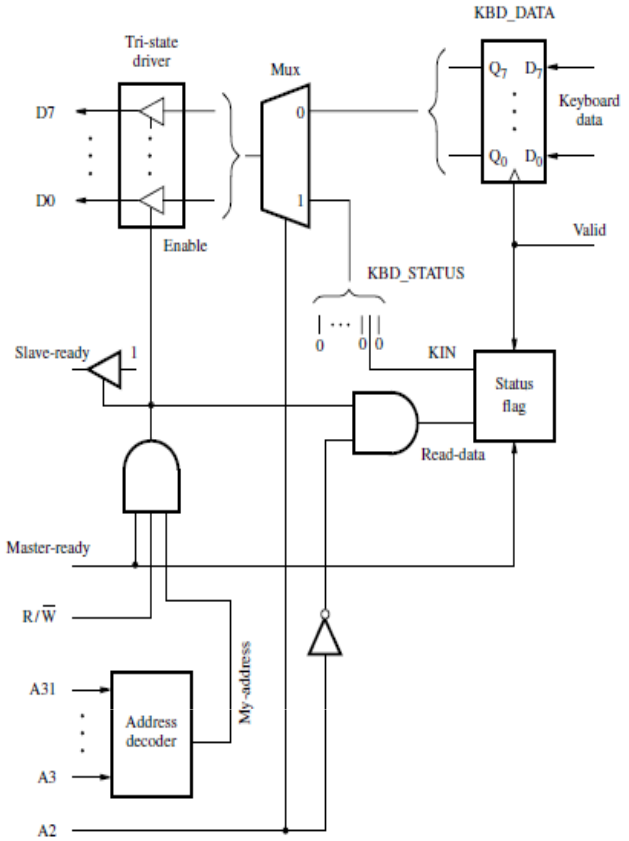


Figure An input interface circuit.

Valid =1, master slave=0 then Kin =1

Read-data =1 reset latch and ff , indicating that KBD_DATA is being read.

Parallel port

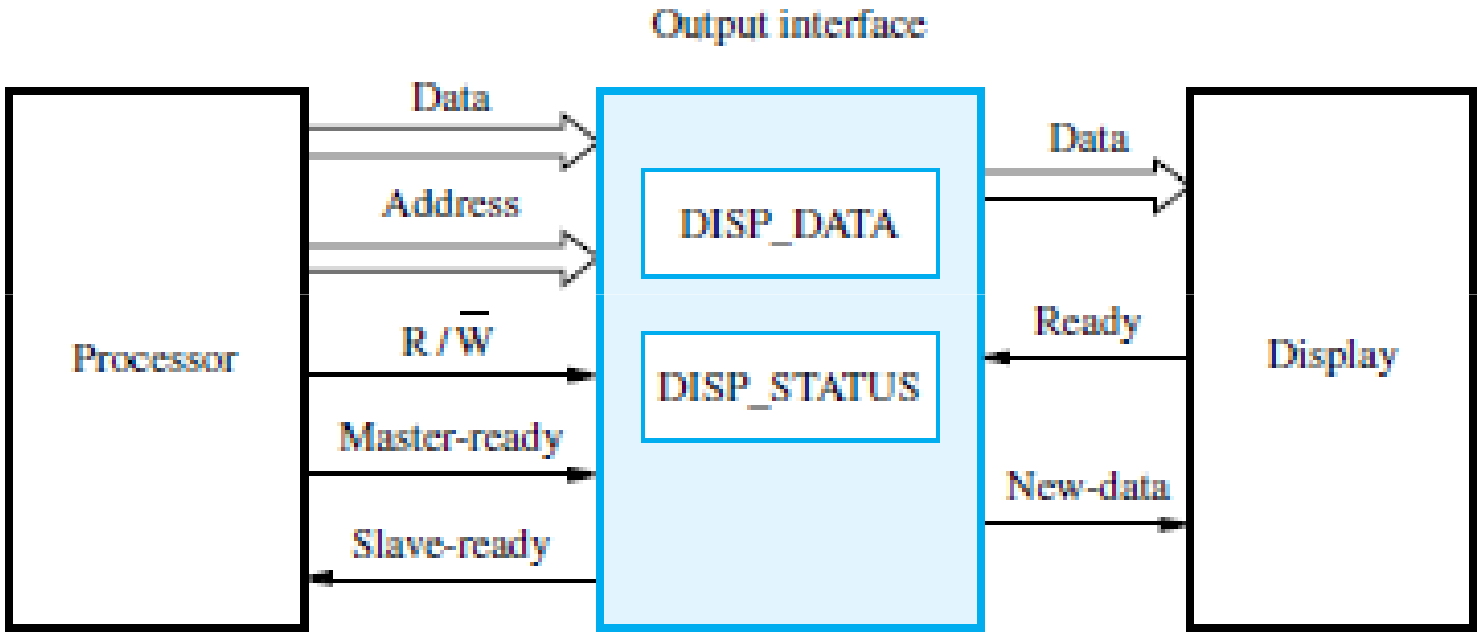


Figure 7 Display to processor connection.

Parallel port

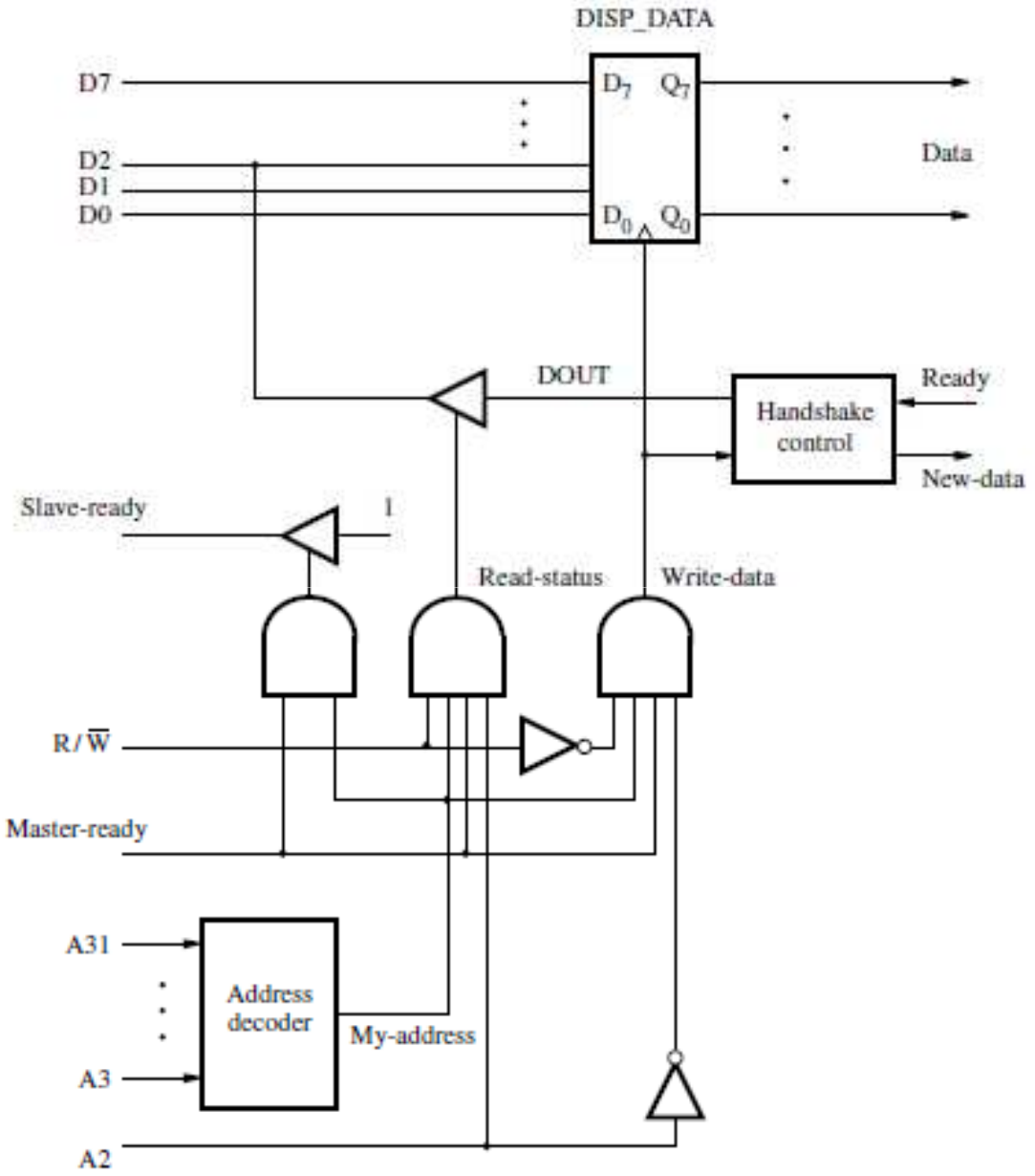


Figure An output interface circuit.

Parallel port

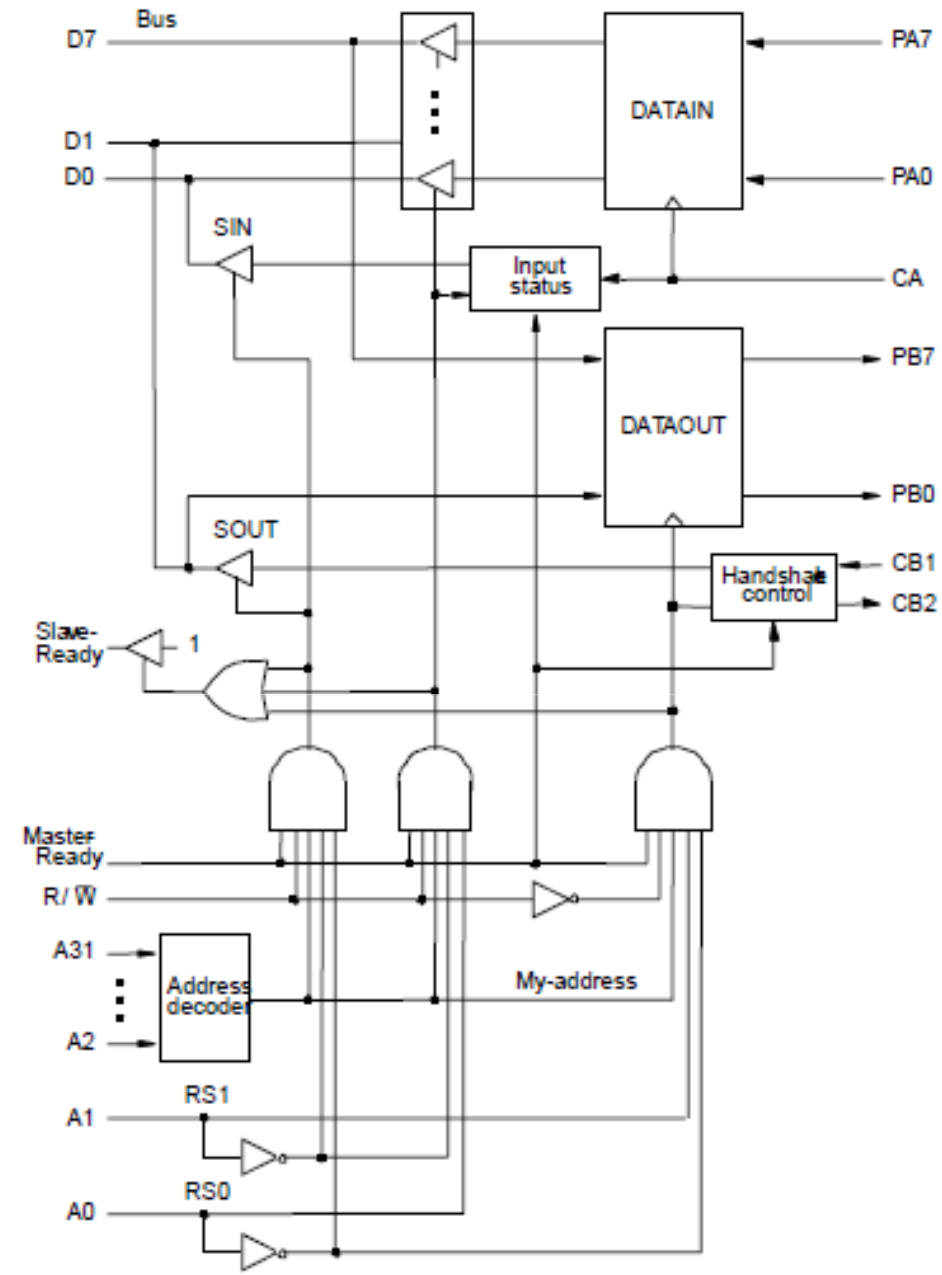


Figure Combined input/output interface circuit.

Serial port

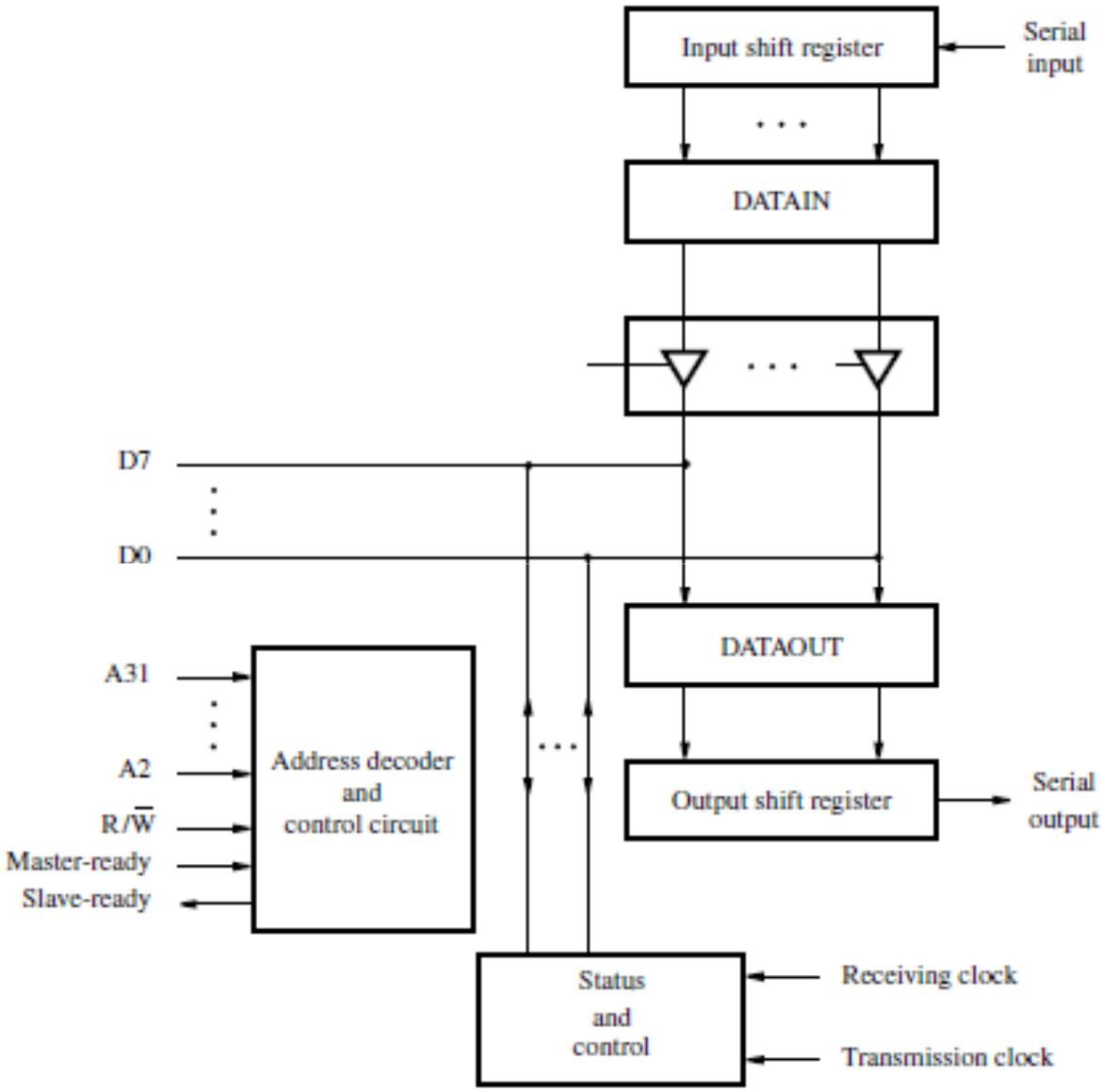


Figure A serial interface.

Serial port

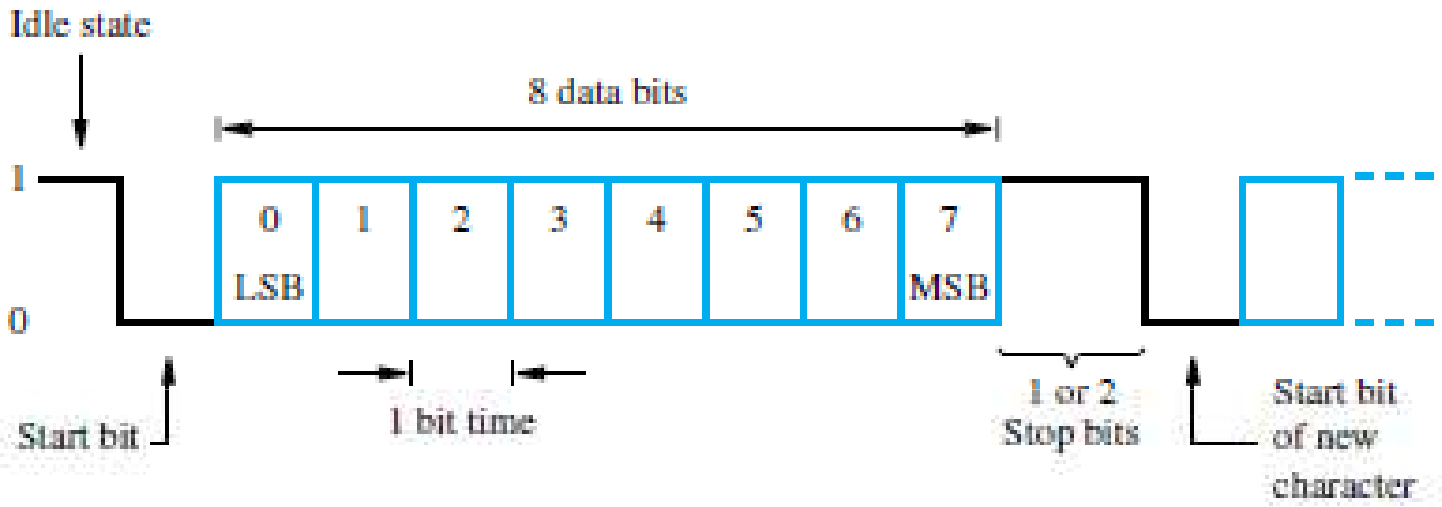


Figure Asynchronous serial character transmission.

Standard I/O interfaces

- Three widely used bus standards:
  - PCI (Peripheral Component Interconnect)
  - SCSI (Small Computer System Interface)
  - USB (Universal Serial Bus)

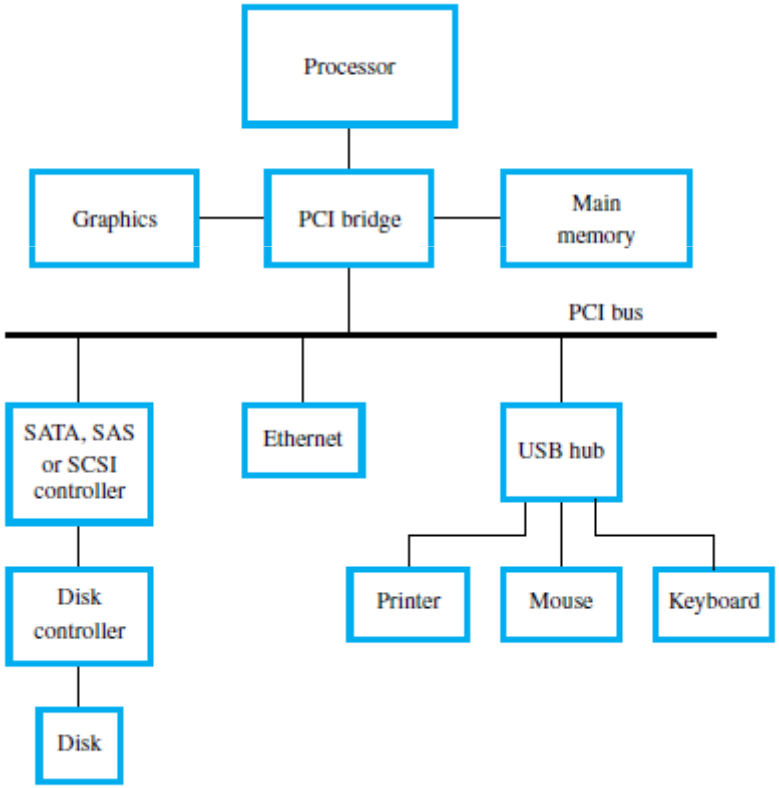
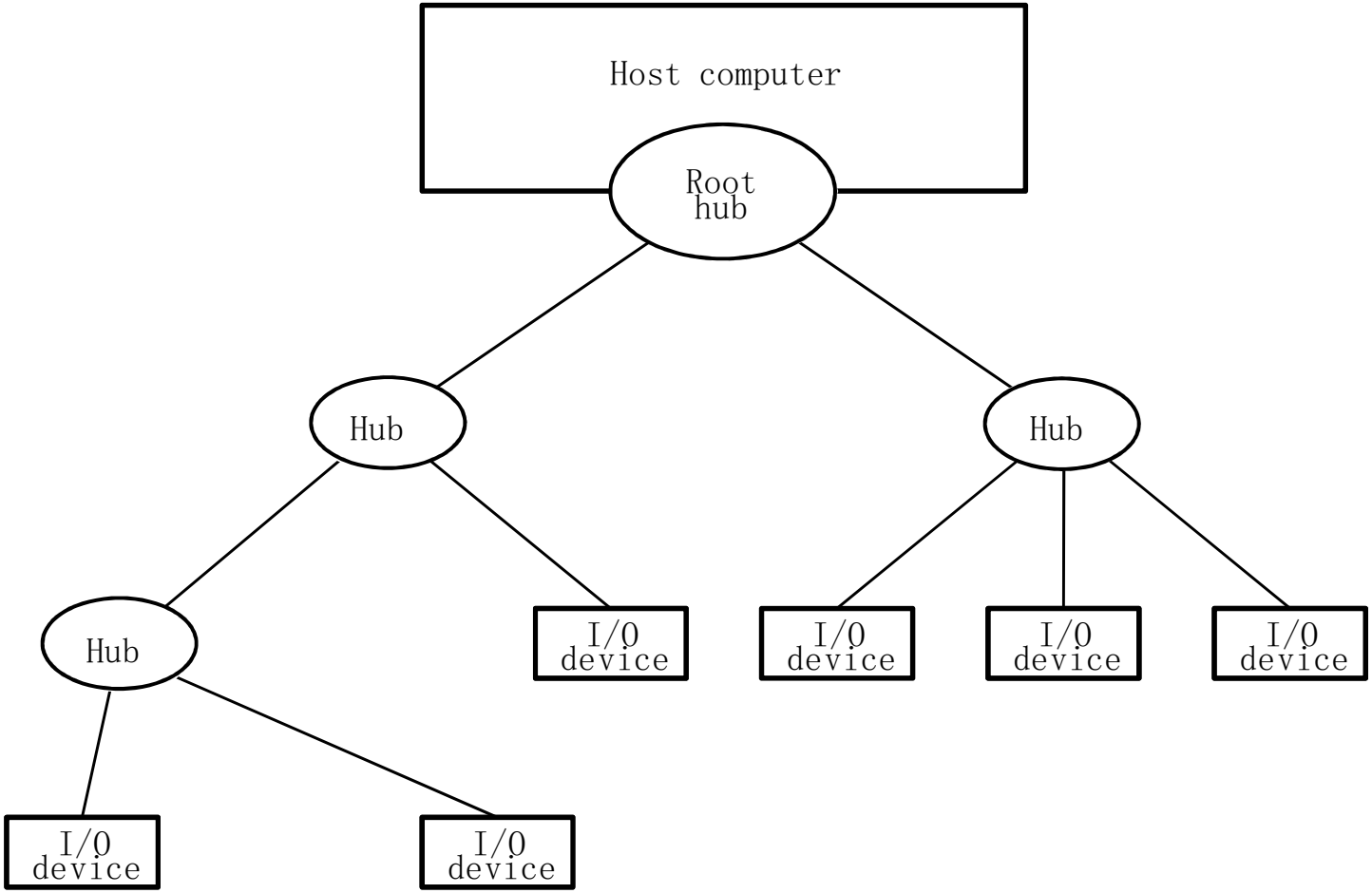


Figure Use of a PCI bus in a computer system.

USB

- Universal Serial Bus (USB) widely used interconnection standard
- Many devices are available with a USB connector: mice, memory keys, disk drives, printers, cameras
  - They vary in speed, volume, and timing constraints
- Simple and low cost
- Speed
  - Low-speed(1.5 Mb/s)
  - Full-speed(12 Mb/s)
  - High-speed(480 Mb/s)
- Plug-and-play -hot-pluggable

Universal Serial Bus tree structure



Universal Serial Bus tree structure

- Tree structure accommodate a large number of devices that can be added or removed at any time
- Each node - a hub - intermediate control point between the host and the I/O devices
- Root hub connects the entire tree to the host computer
- The leaves of the tree are the I/O devices being served
  - Host communicate with the I/O devices
  - No device can communicate among themself
  - I/O device communicate only to the host
  - serial transmission format
  - hub or an I/O device, is assigned a 7-bit address. This address is local to the USB tree
  - operates strictly on the basis of polling. A device may send a message only in response to a poll message from the host processor. Hence, no two devices can send messages at the same time.
    - Host polls each hub to collect status information and learn about new devices that may have been added or disconnected.
    - New device - information is read in a special memory in the device's USB interface. It assigns the device a unique USB address and writes that address in one of the device's interface registers.
    - Supports isochronous data – data need to be transferred at precisely timed regular intervals.
- four wires, of which two carry power, +5 V and Ground,and two carry data.
- low speed - uses *single-ended transmission*
- High-Speed – uses *differential signaling*