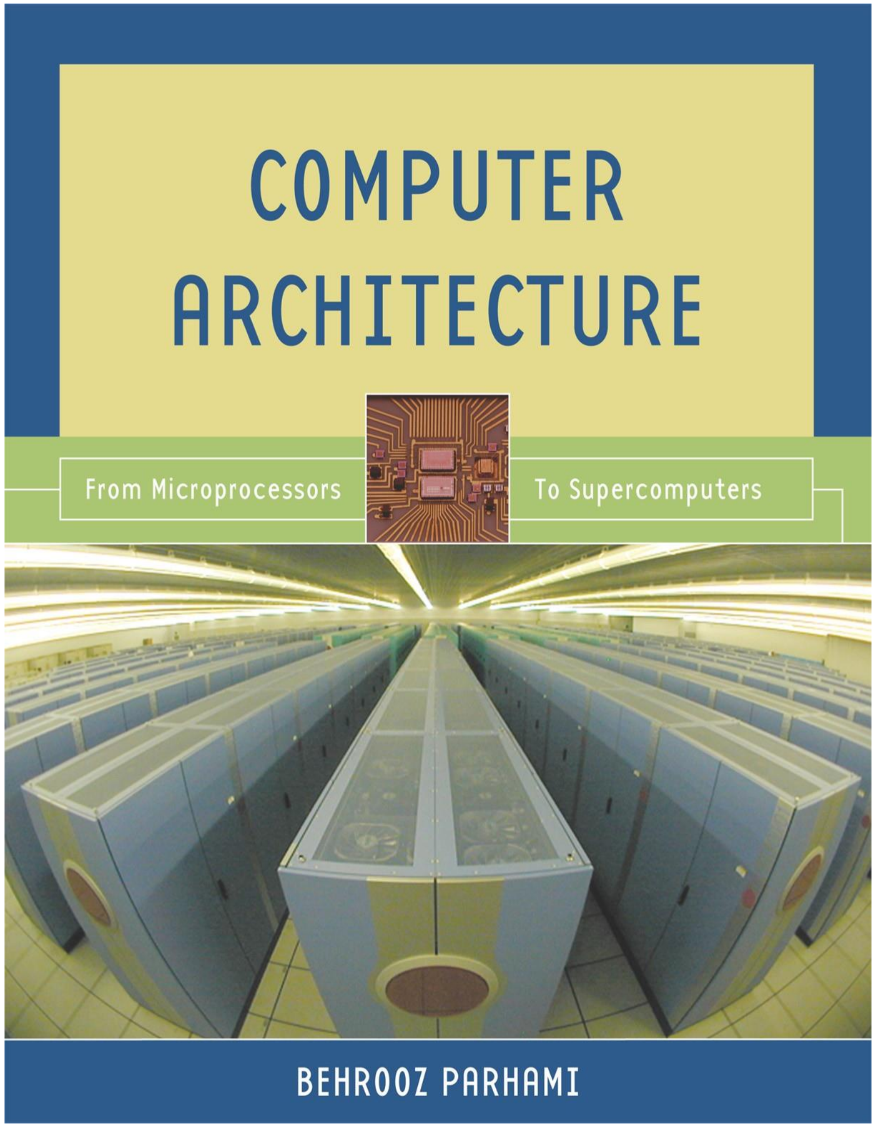


Part VI
Input/Output and Interfacing

	Parts	Chapters
	I. Background and Motivation	1. Combinational Digital Circuits 2. Digital Circuits with Memory 3. Computer System Technology 4. Computer Performance
	II. Instruction-Set Architecture	5. Instructions and Addressing 6. Procedures and Data 7. Assembly Language Programs 8. Instruction-Set Variations
C	III. The Arithmetic/Logic Unit	9. Number Representation 10. Adders and Simple ALUs 11. Multipliers and Dividers 12. Floating-Point Arithmetic
P U	IV. Data Path and Control	13. Instruction Execution Steps 14. Control Unit Synthesis 15. Pipelined Data Paths 16. Pipeline Performance Limits
	V. Memory System Design	17. Main Memory Concepts 18. Cache Memory Organization 19. Mass Memory Concepts 20. Virtual Memory and Paging
	VI. Input/Output and Interfacing	21. Input/Output Devices 22. Input/Ouput Programming 23. Buses, Links, and Interfacing 24. Context Switching and Interrupts
	VII. Advanced Architectures	25. Road to Higher Performance 26. Vector and Array Processing 27. Shared-Memory Multiprocessing 28. Distributed Multicomputing

About This Presentation

Edition	Released	Revised	Revised	Revised	Revised
First	July 2003	July 2004	July 2005	Mar. 2007	Mar. 2008
	Mar. 2009	Feb. 2011			

VI Input/Output and Interfacing

Effective computer design & use requires awareness of:

- I/O device types, technologies, and performance
- Interaction of I/O with memory and CPU
- Automatic data collection and device actuation

21 Input/Output Devices

Learn about input and output devices as categorized by:

- Type of data presentation or recording
- Data rate, which influences interaction with system

Topics in This Chapter	
21.1	Input/Output Devices and Controllers
21.2	Keyboard and Mouse
21.3	Visual Display Units
21.4	Hard-Copy Input/Output Devices
21.5	Other Input/Output Devices
21.6	Networking of Input/Output Devices

Section 21.1: Introduction

Section 21.3

Section 21.4

Section 21.2

Section 21.5: Other devices

Section 21.6: Networked I/O

21.1 Input/Output Devices and Controllers

Table 3.3 Some input, output, and two-way I/O devices.

Input type	**Prime examples**	**Other examples**	**Data rate (b/s)**	**Main uses**
Symbol	Keyboard, keypad	Music note, OCR	10s	Ubiquitous
Position	Mouse, touchpad	Stick, wheel, glove	100s	Ubiquitous
Identity	Barcode reader	Badge, fingerprint	100s	Sales, security
Sensory	Touch, motion, light	Scent, brain signal	100s	Control, security
Audio	Microphone	Phone, radio, tape	1000s	Ubiquitous
Image	Scanner, camera	Graphic tablet	1000s-10^6s	Photos, publishing
Video	Camcorder, DVD	VCR, TV cable	1000s-10^9s	Entertainment
Output type	**Prime examples**	**Other examples**	**Data rate (b/s)**	**Main uses**
Symbol	LCD line segments	LED, status light	10s	Ubiquitous
Position	Stepper motor	Robotic motion	100s	Ubiquitous
Warning	Buzzer, bell, siren	Flashing light	A few	Safety, security
Sensory	Braille text	Scent, brain stimulus	100s	Personal assistance
Audio	Speaker, audiotape	Voice synthesizer	1000s	Ubiquitous
Image	Monitor, printer	Plotter, microfilm	1000s	Ubiquitous
Video	Monitor, TV screen	Film/video recorder	1000s-10^9s	Entertainment
Two-way I/O	**Prime examples**	**Other examples**	**Data rate (b/s)**	**Main uses**
Mass storage	Hard/floppy disk	CD, tape, archive	10^6s	Ubiquitous
Network	Modem, fax, LAN	Cable, DSL, ATM	1000s-10^9s	Ubiquitous

Simple Organization for Input/Output

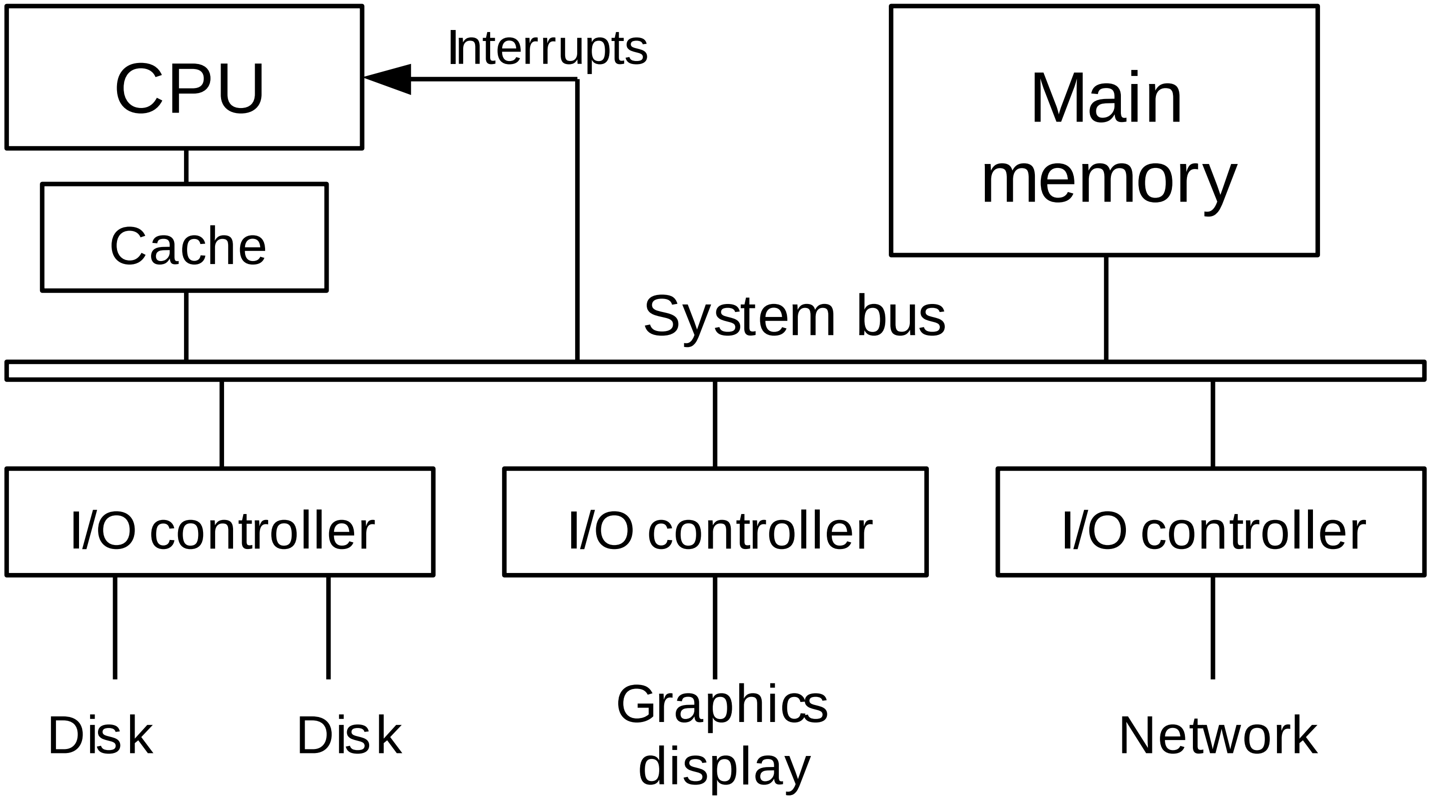


Figure 21.1 Input/output via a single common bus.

I/O Organization for Greater Performance

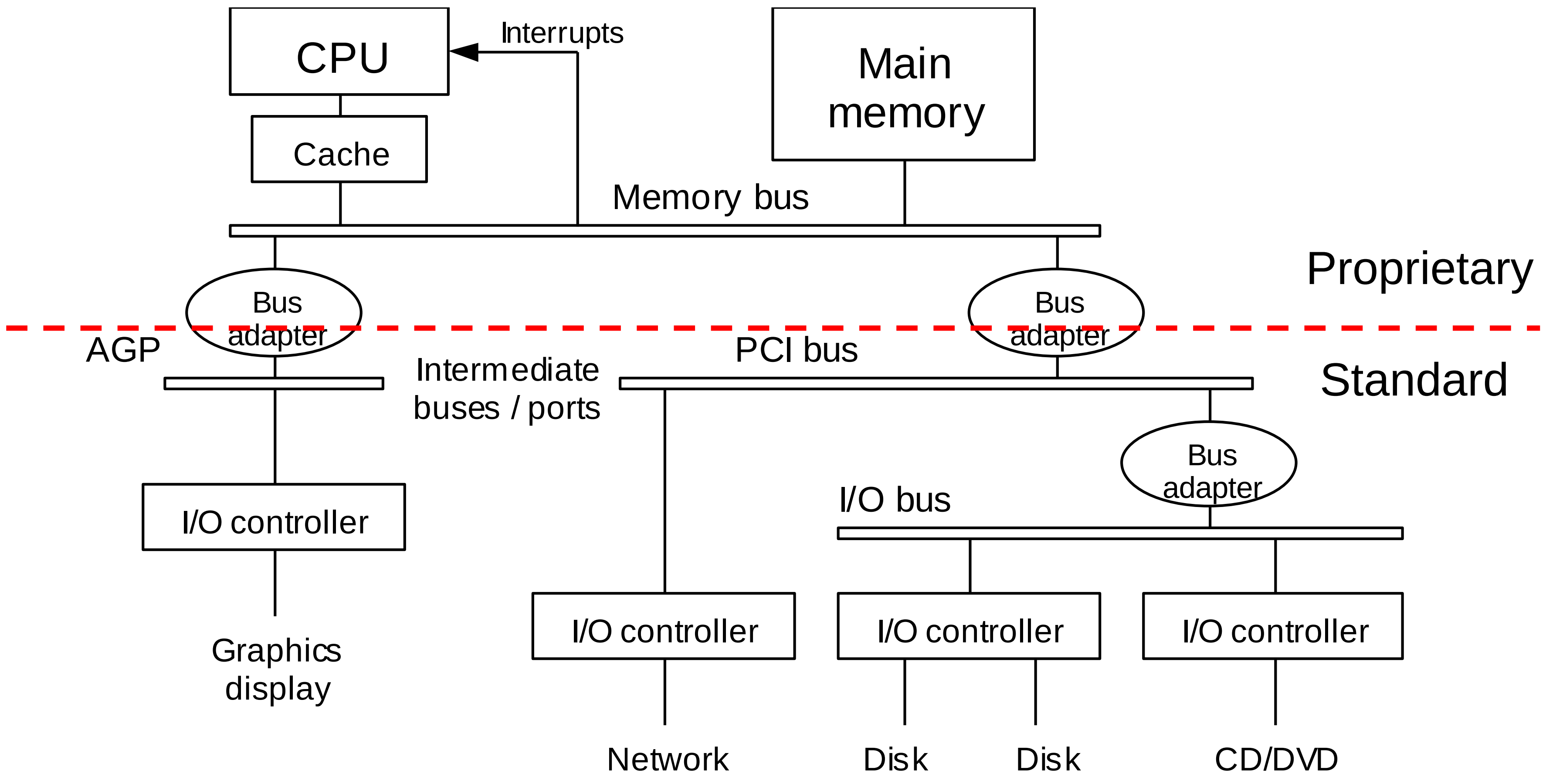
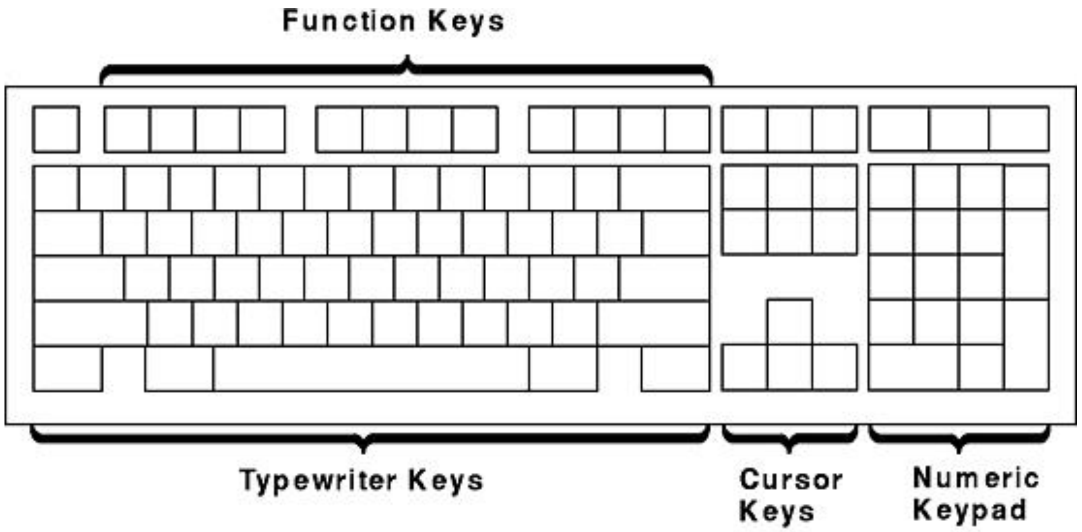
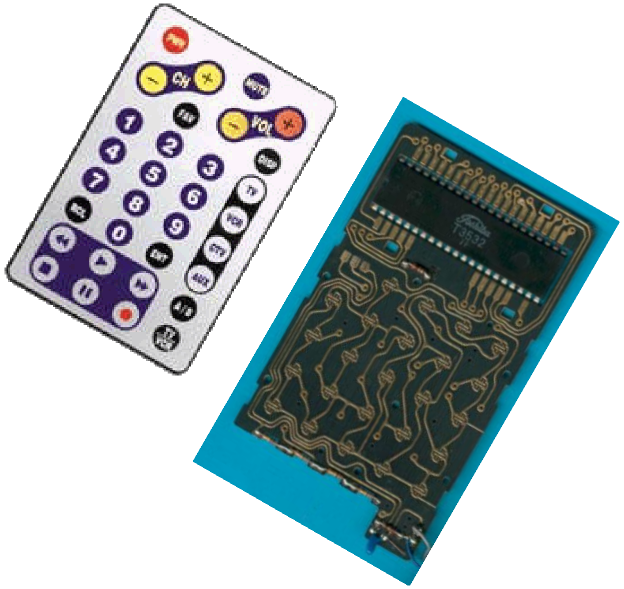


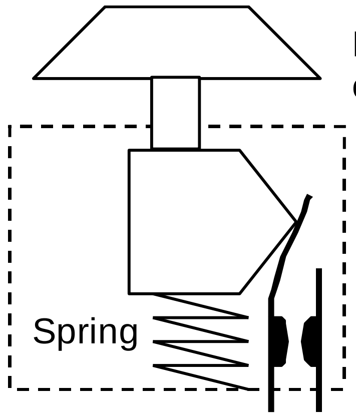
Figure 21.2 Input/output via intermediate and dedicated I/O buses (to be explained in Chapter 23).

21.2 Keyboard and Mouse

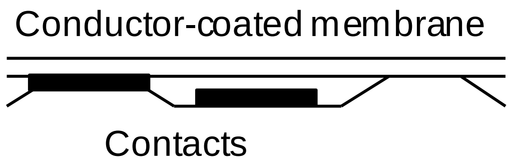




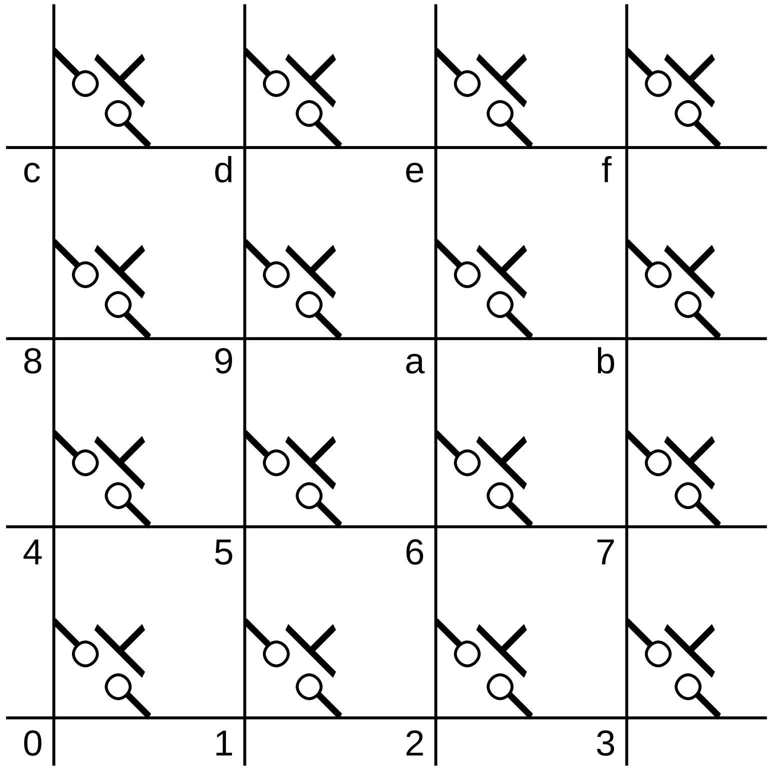
Keyboard Switches and Encoding



(a) Mechanical switch with a plunger



(b) Membrane switch



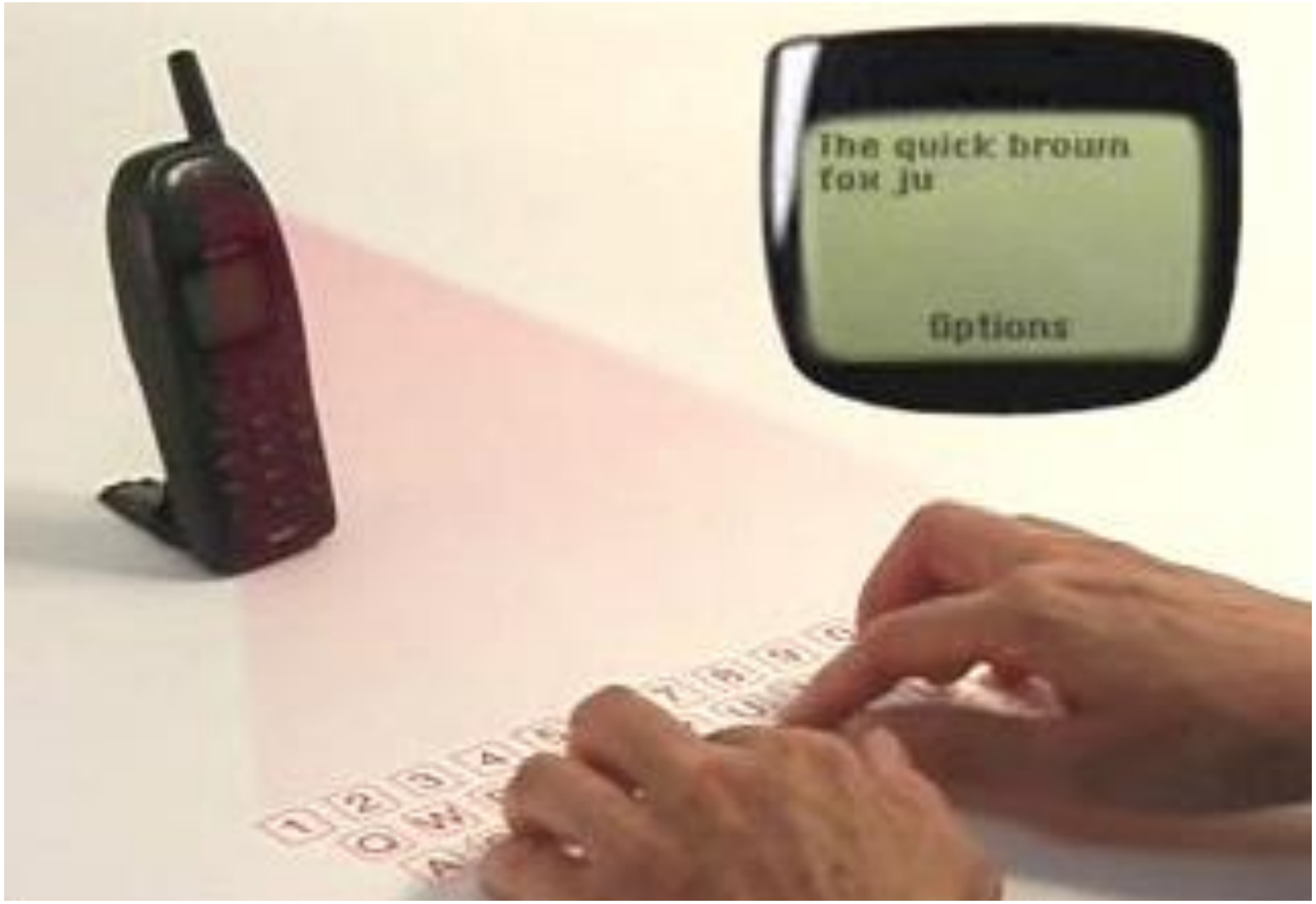
(c) Logical arrangement of keys

Figure 21.3 Two mechanical switch designs and the logical layout of a hex keypad.

Projection Virtual Keyboard

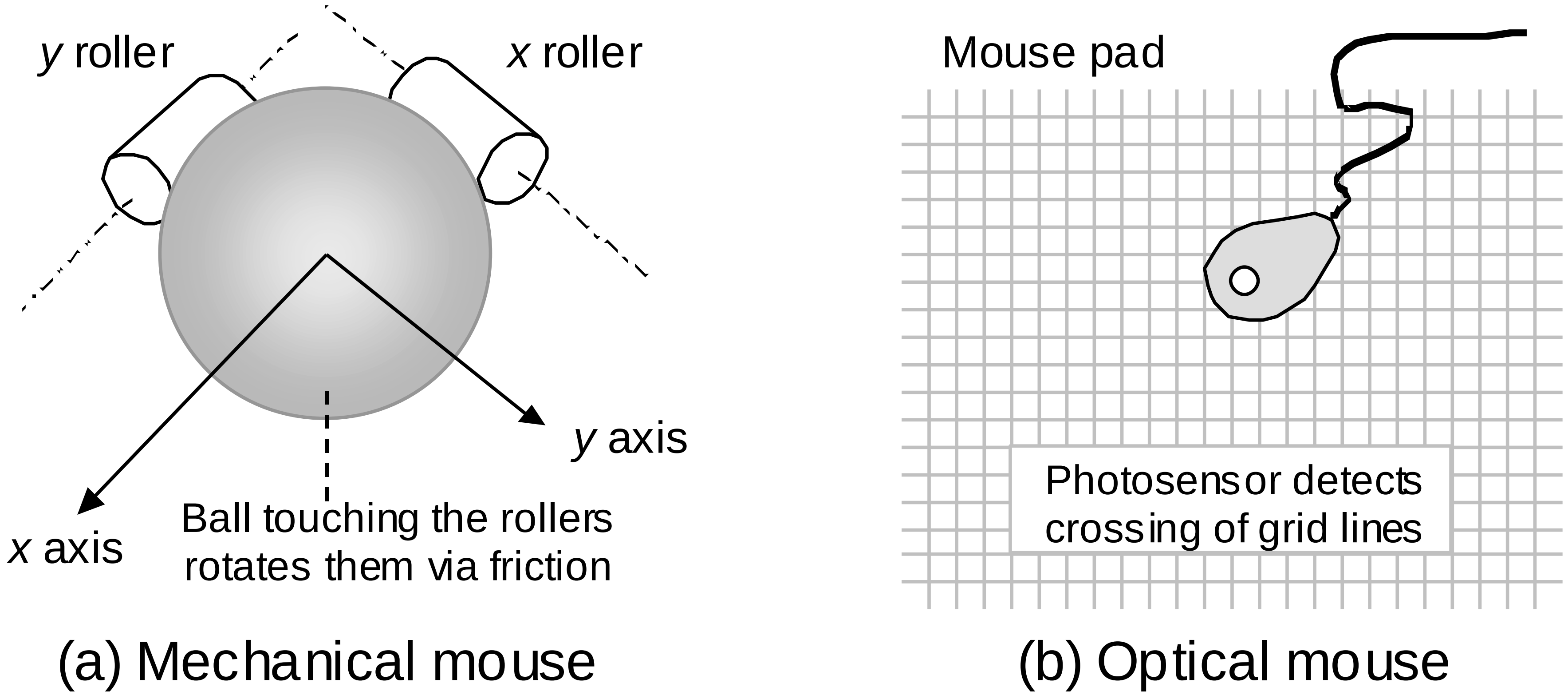
Hardware:
A tiny laser device projects the image of a full-size keyboard on any surface

Software:
Emulates a real keyboard, even clicking key sounds



Pointing Devices

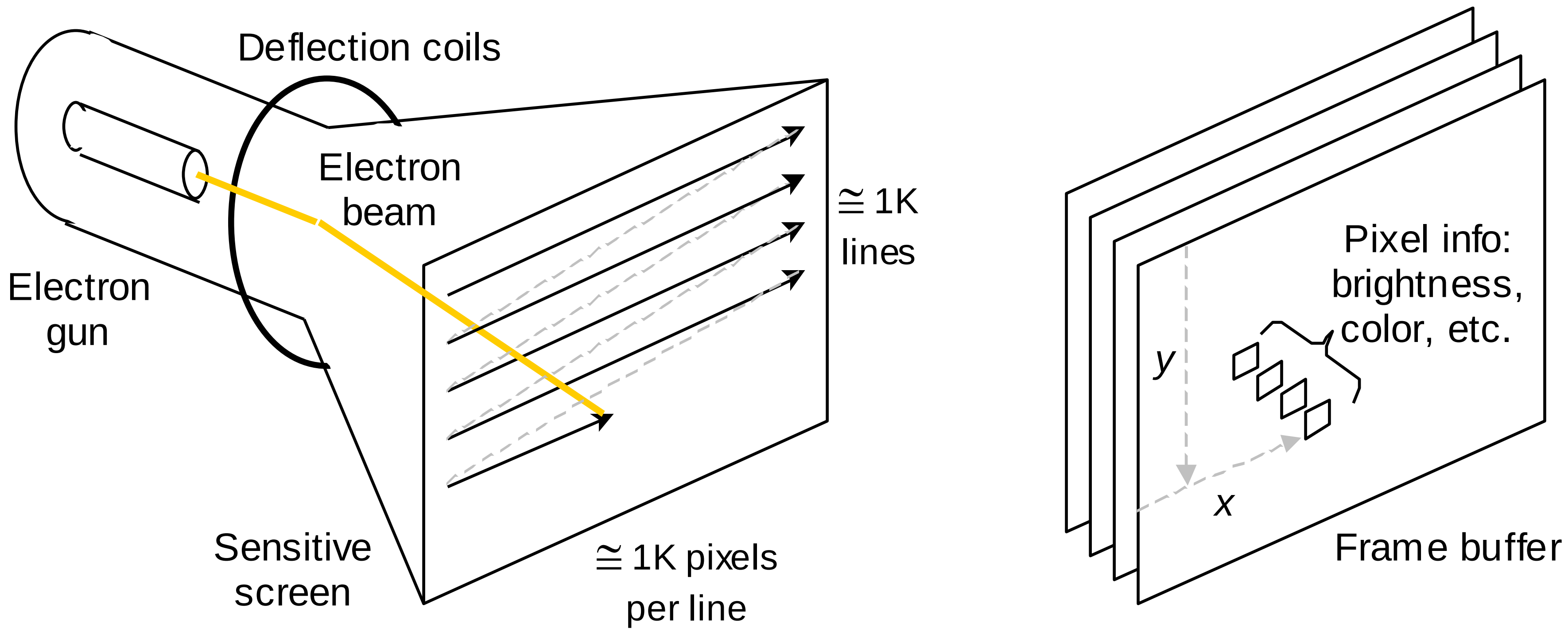
How a Mouse Works



y roller
x roller
y axis
x axis
Ball touching the rollers rotates them via friction
(a) Mechanical mouse
Mouse pad
Photosensor detects crossing of grid lines
(b) Optical mouse

Figure 21.4 Mechanical and simple optical mice.

21.3 Visual Display Units

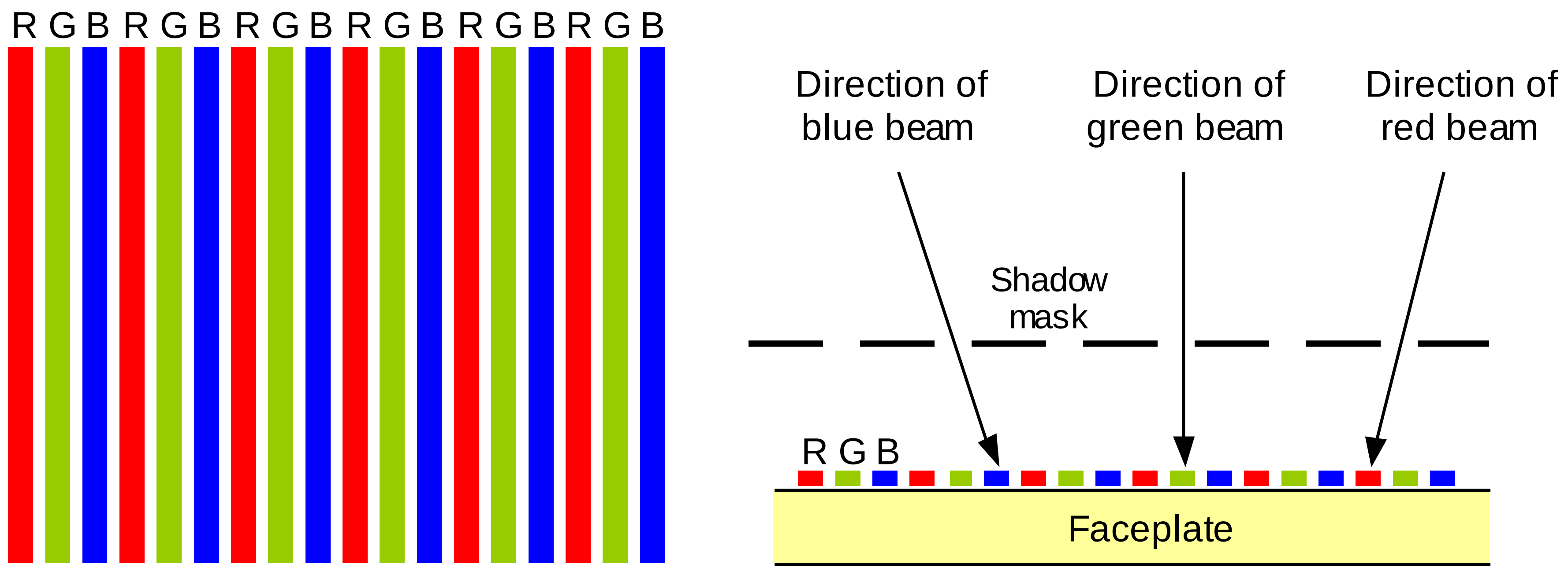


(a) Image formation on a CRT

(b) Data defining the image

Figure 21.5 CRT display unit and image storage in frame buffer.

How Color CRT Displays Work

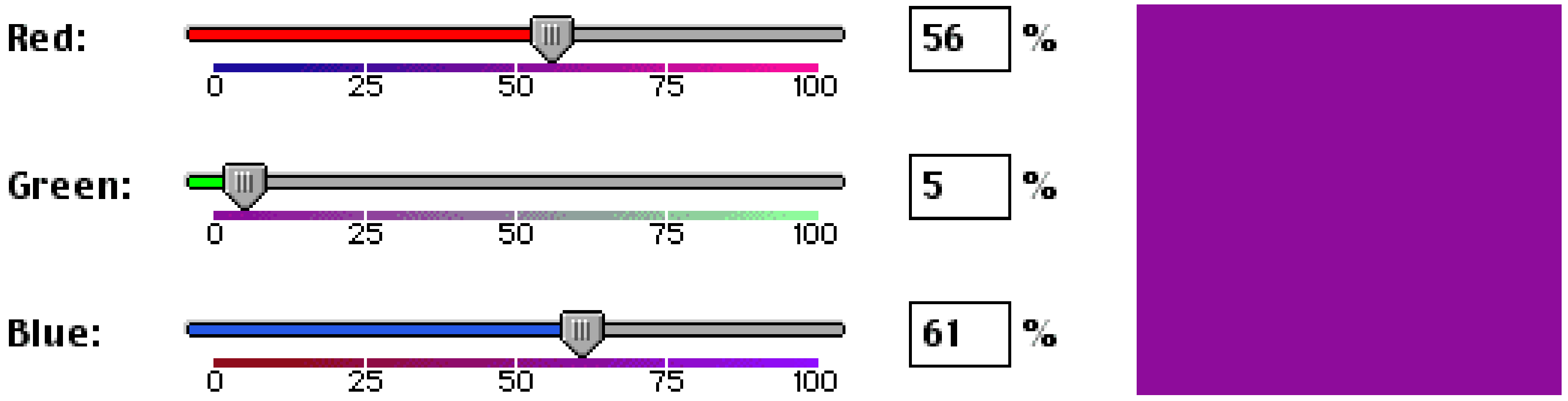


(a) The RGB color stripes

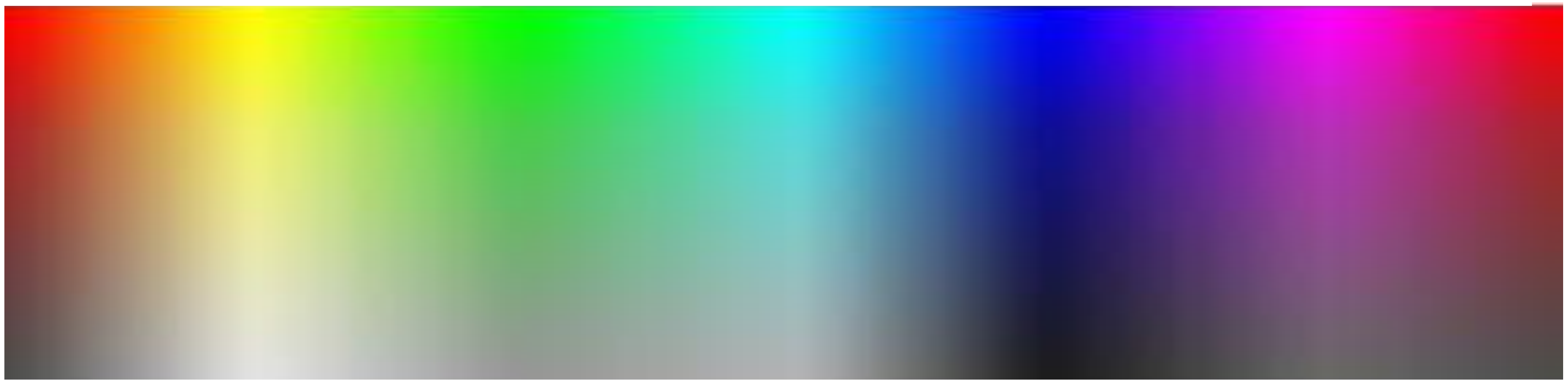
(b) Use of shadow mask

Figure 21.6 The RGB color scheme of modern CRT displays.

Encoding Colors in RGB Format



Besides hue, saturation is used to affect the color’s appearance (high saturation at the top, low saturation at the bottom)



Flat-Panel Displays

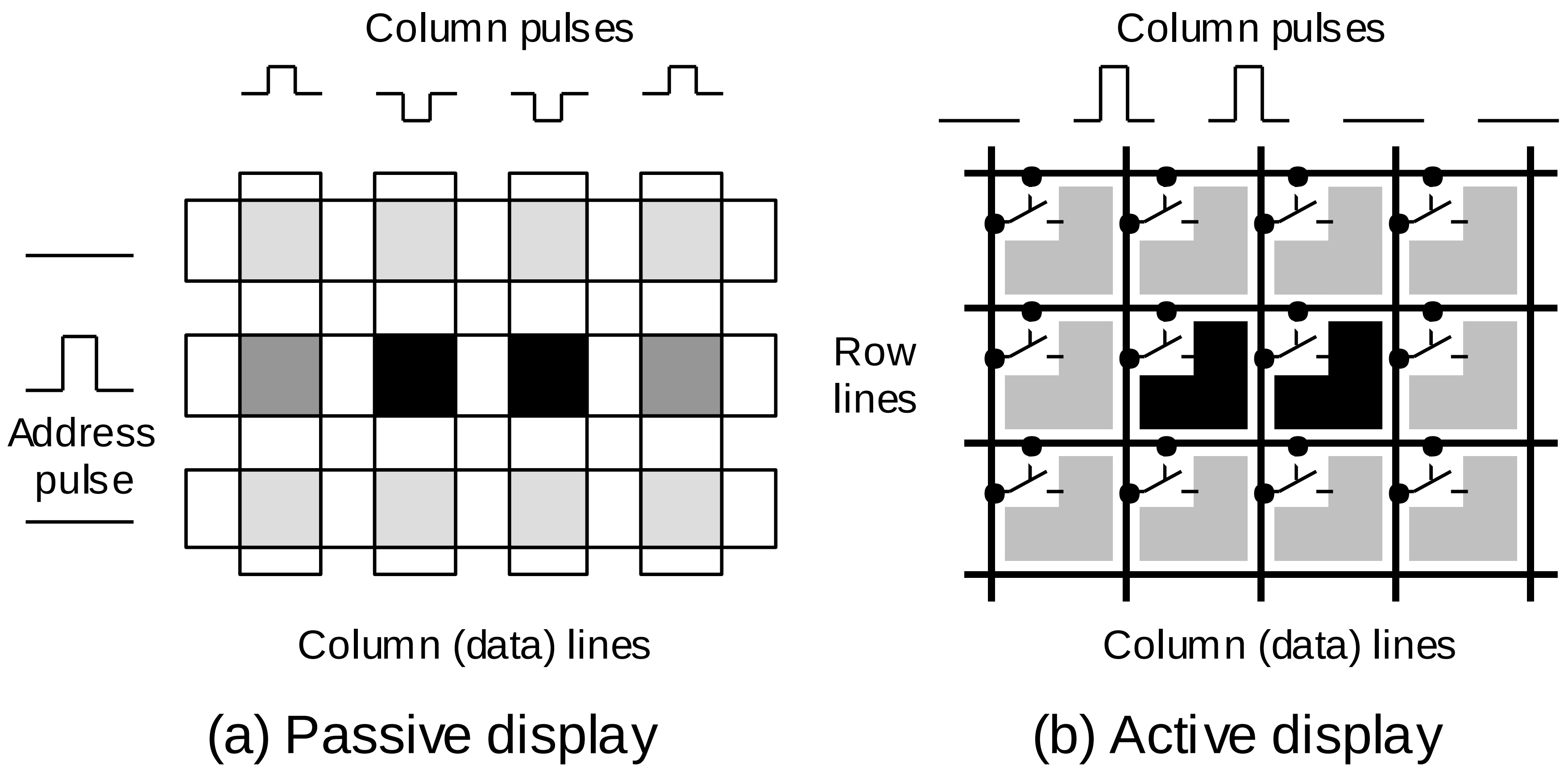


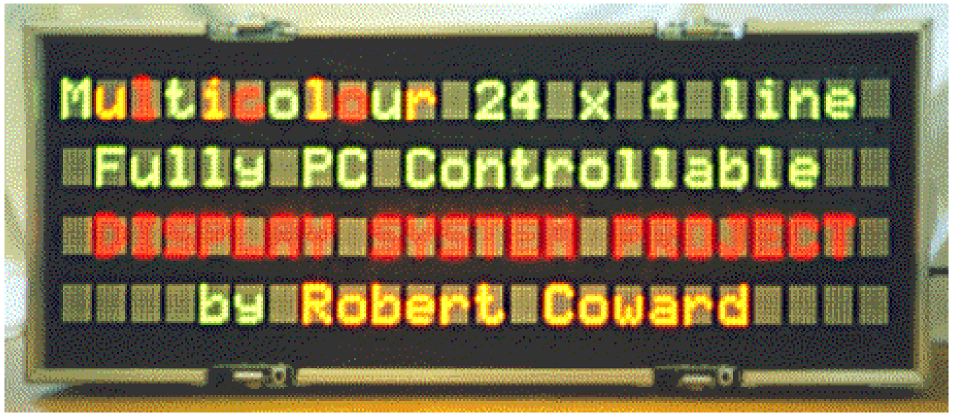
Figure 21.7 Passive and active LCD displays.

Flexible Display Devices

Paper-thin tablet-size display unit by E Ink

Sony organic light-emitting diode (OLED) flexible display

Other Display Technologies



21.4 Hard-Copy Input/Output Devices

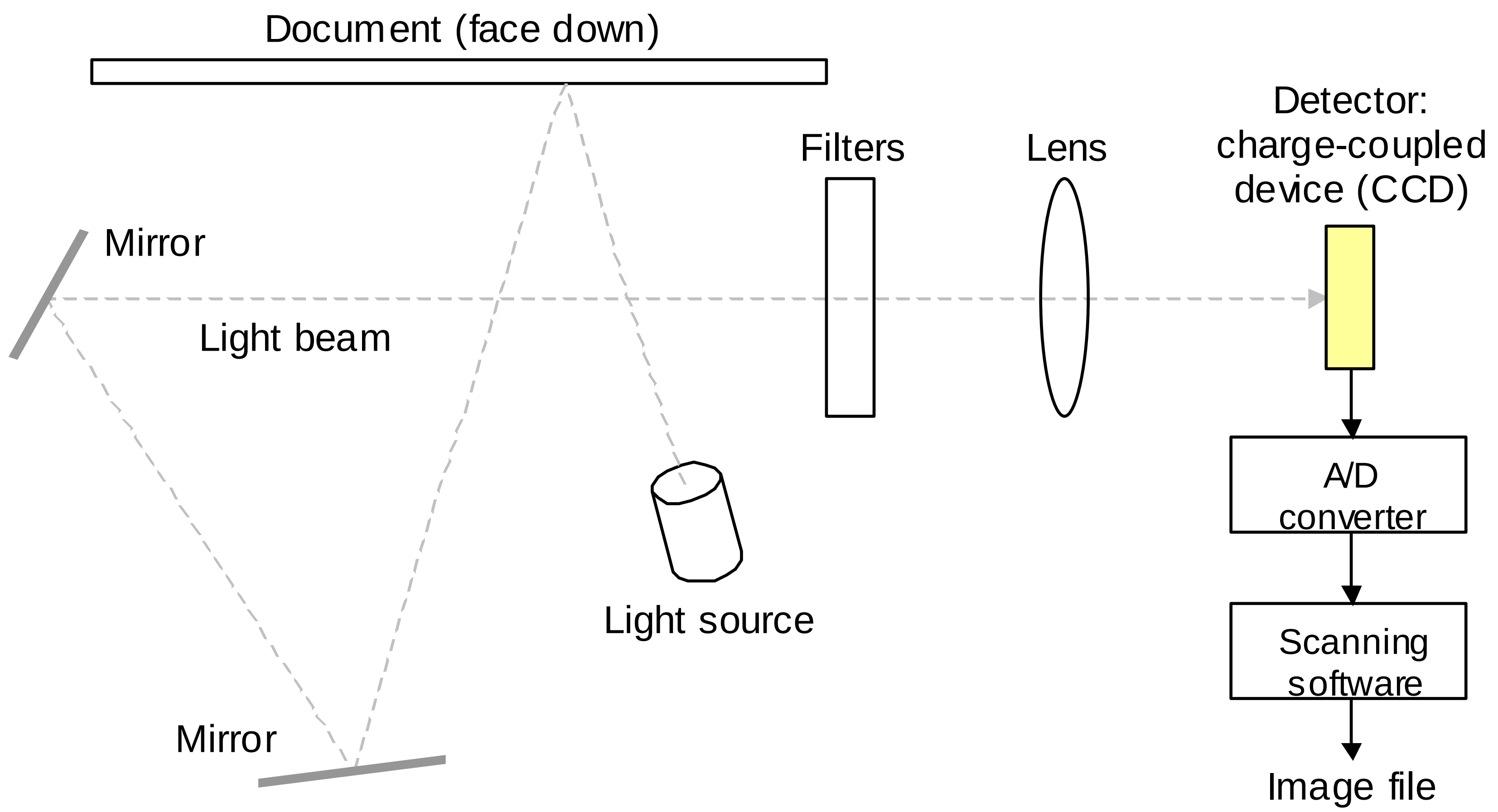


Figure 21.8 Scanning mechanism for hard-copy input.

Character Formation by Dot Matrices

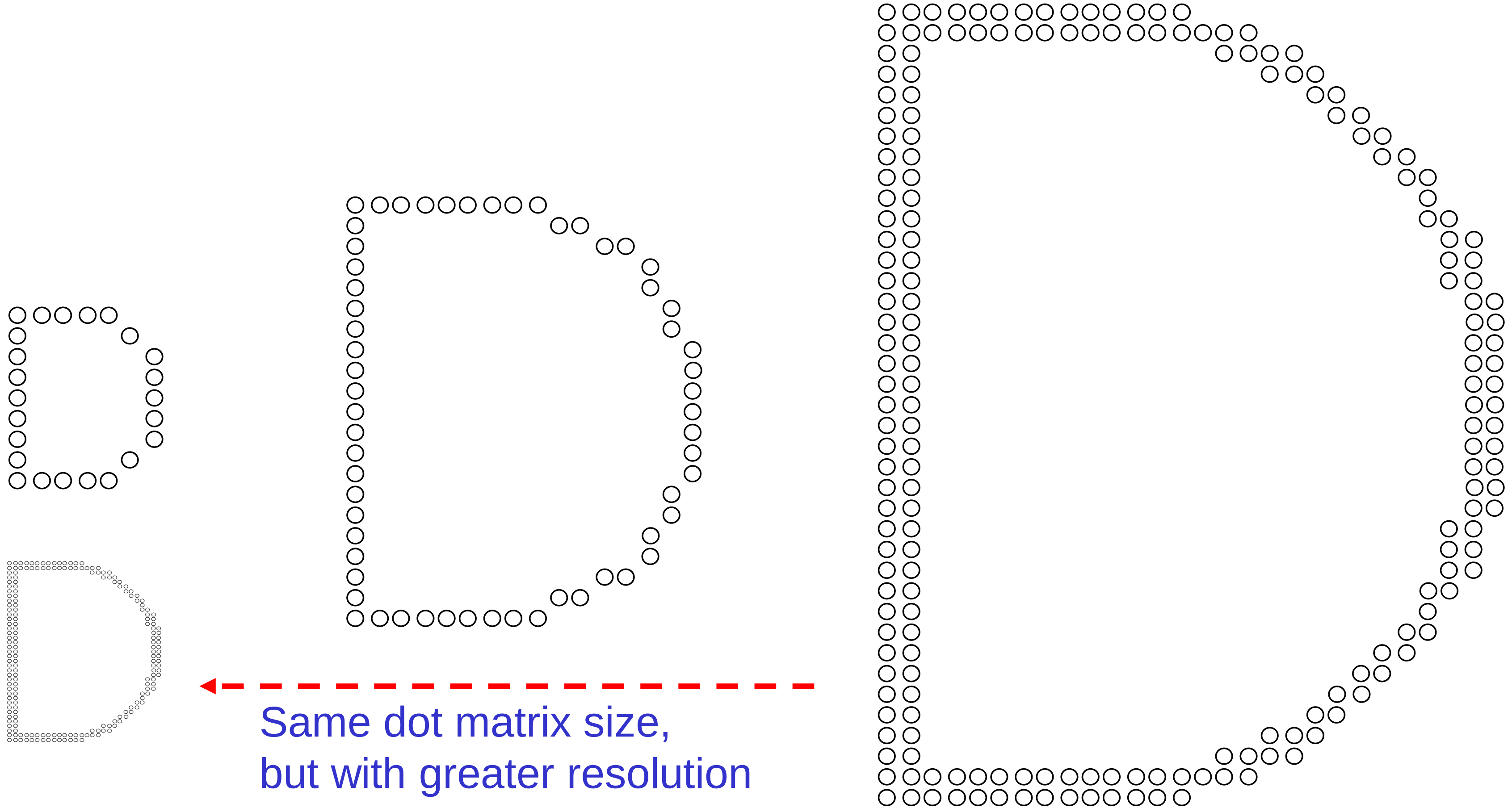
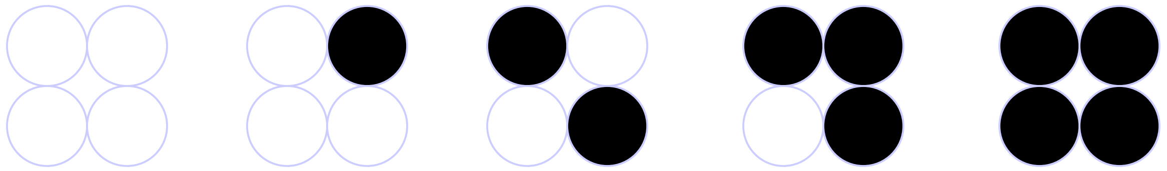


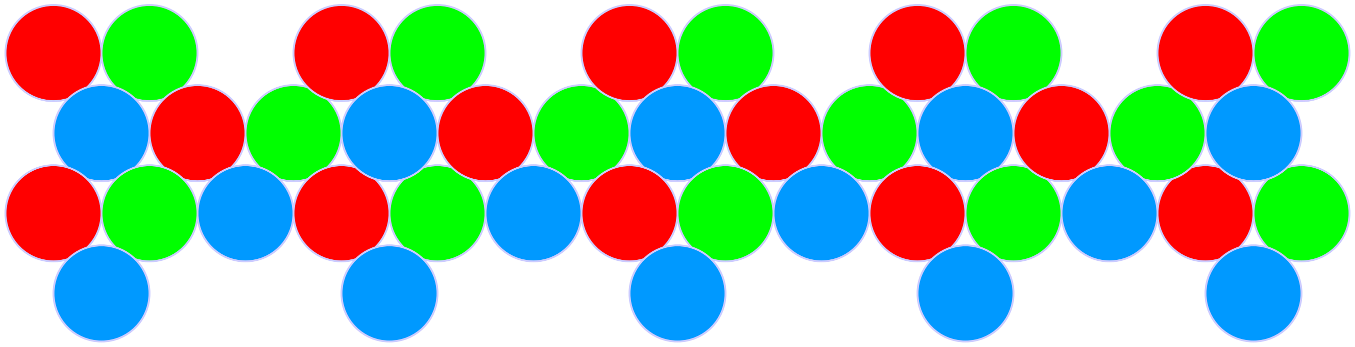
Figure 21.9   Forming the letter “D” via dot matrices of varying sizes.

Simulating Intensity Levels via Dithering

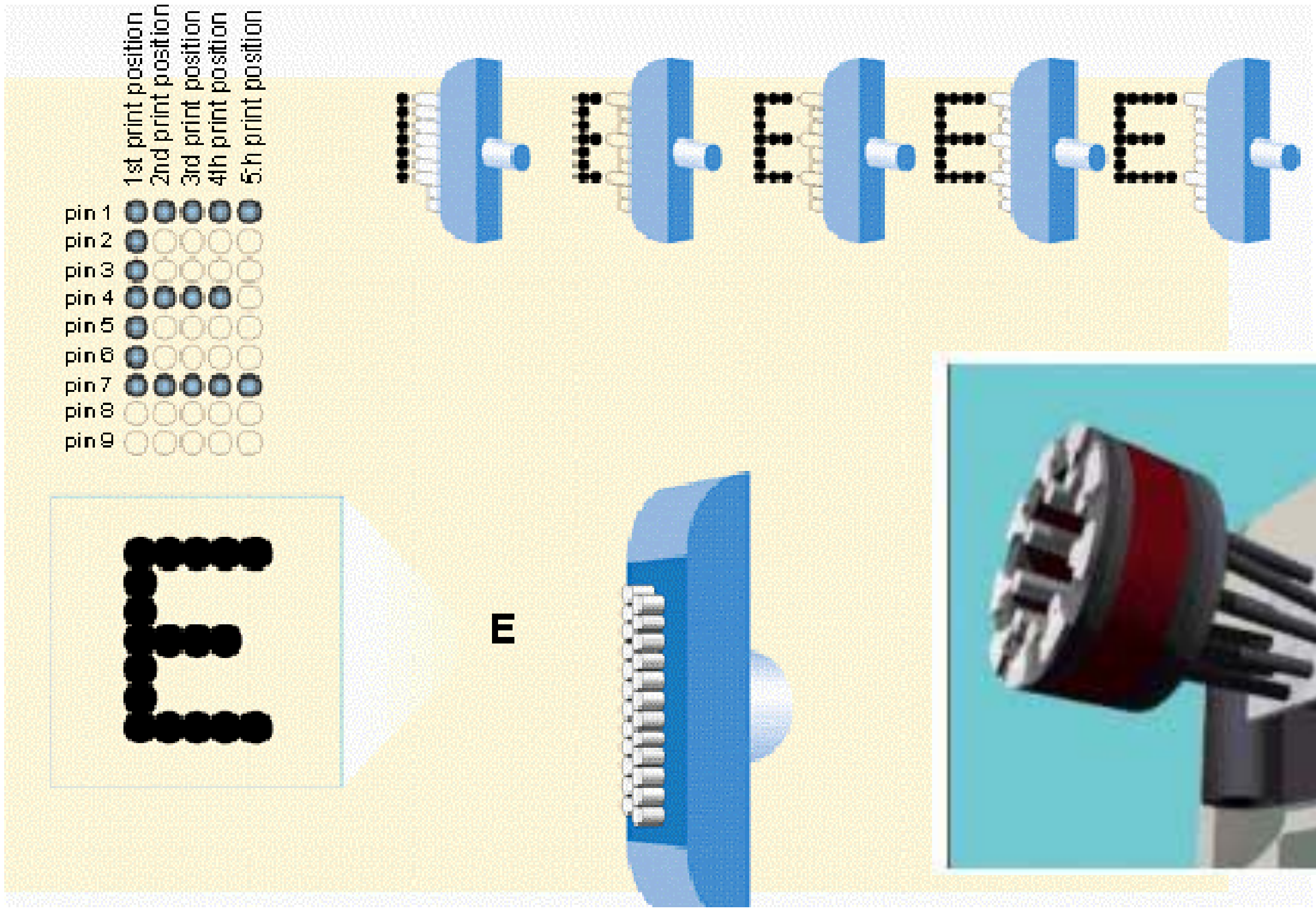


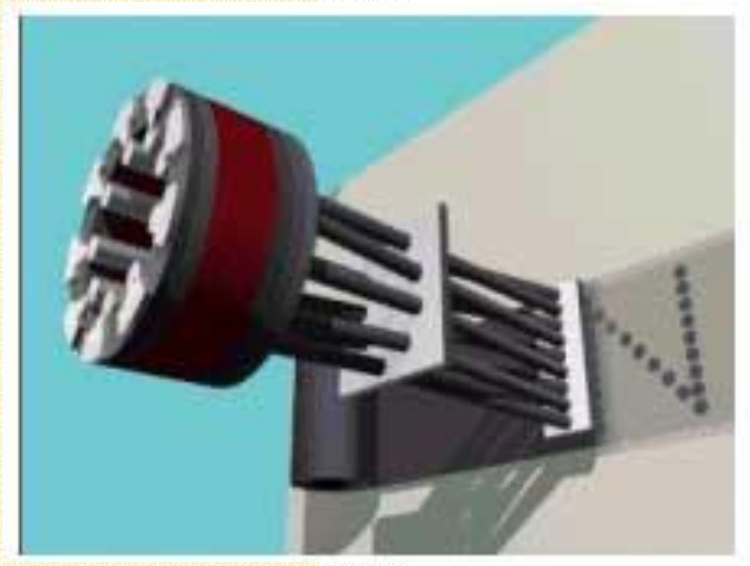
Forming five gray levels on a device that supports only black and white (e.g., ink-jet or laser printer)

Using the dithering patterns above on each of three colors forms 5 × 5 × 5 = 125 different colors

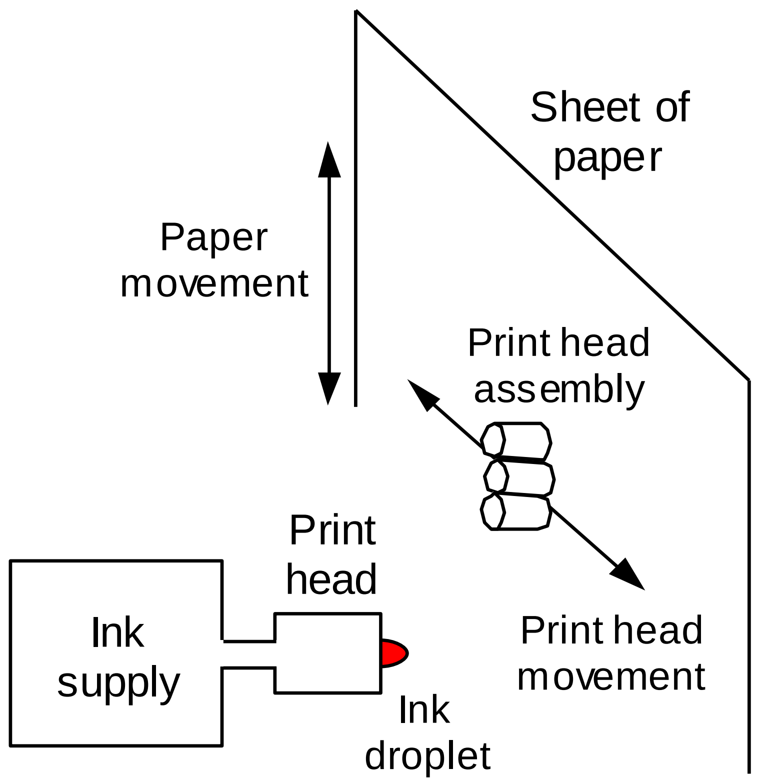


Simple Dot-Matrix Printer Mechanism

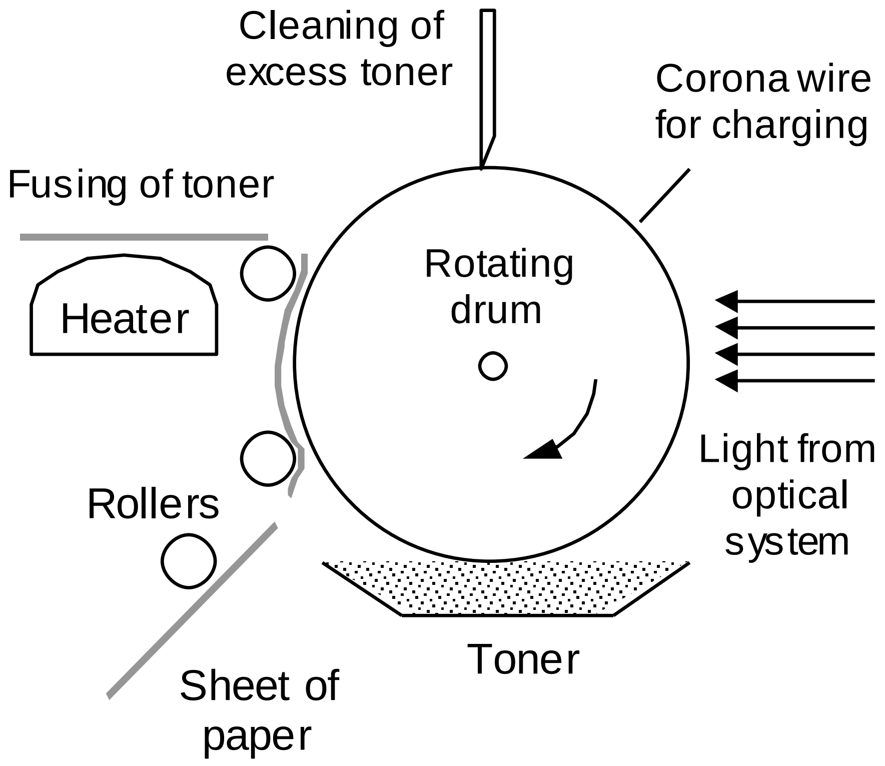




Common Hard-Copy Output Devices



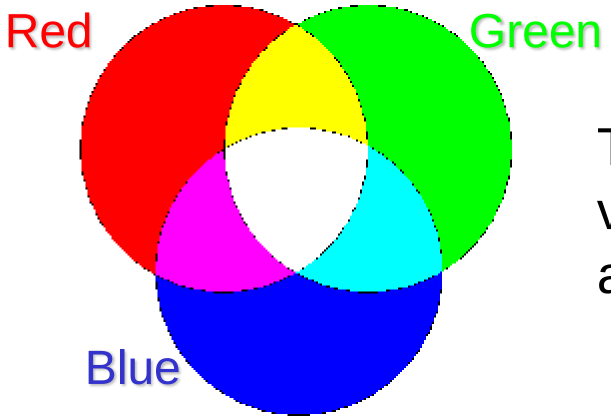
(a) Ink jet printing



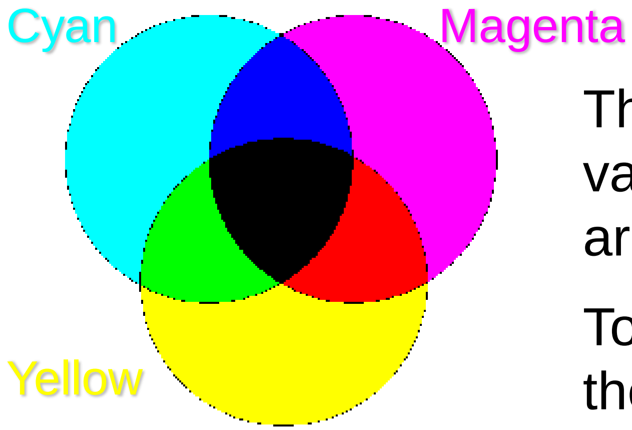
(b) Laser printing

Figure 21.10 Ink-jet and laser printers.

How Color Printers Work



The RGB scheme of color monitors is additive: various amounts of the three primary colors are added to form a desired color

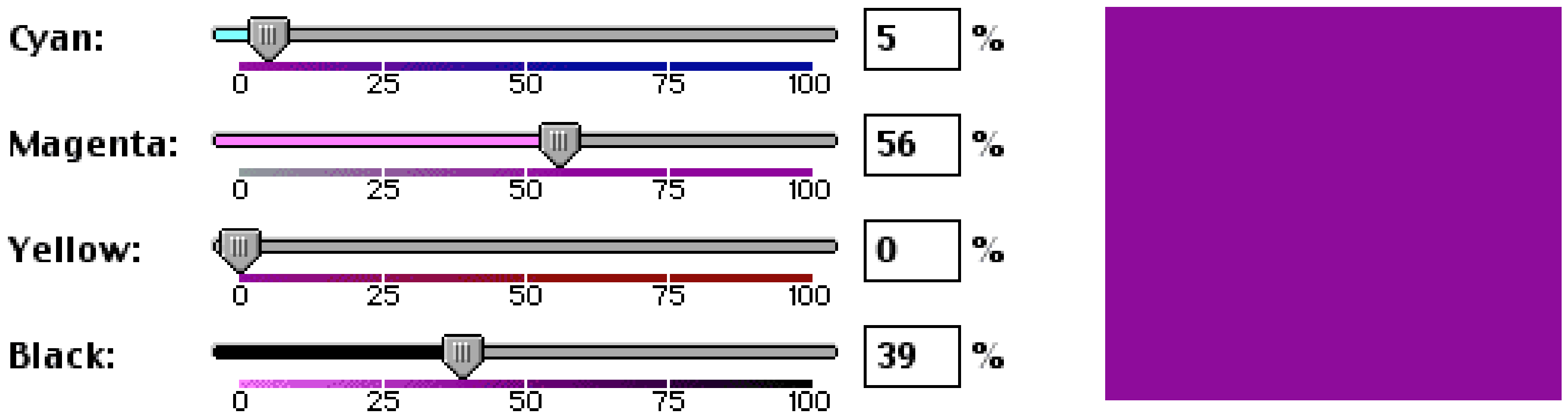


Absence of green

The CMY scheme of color printers is subtractive: various amounts of the three primary colors are removed from white to form a desired color

To produce a more satisfactory shade of black, the CMYK scheme is often used (K = black)

The CMYK Printing Process



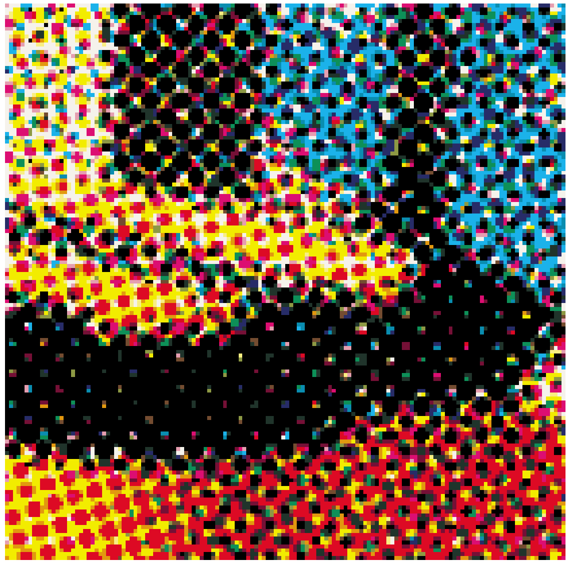
Cyan: 5 %

Magenta: 56 %

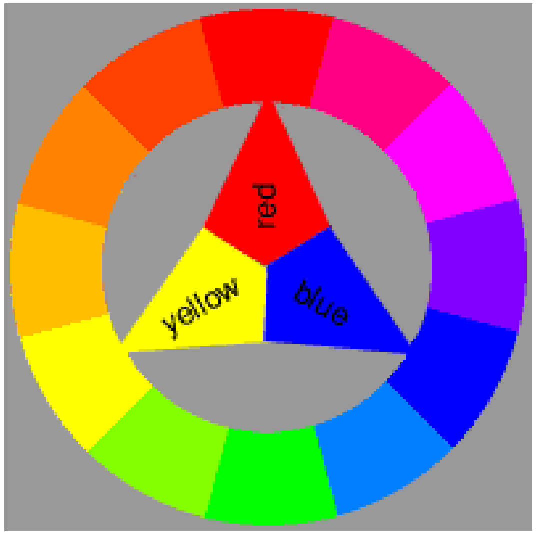
Yellow: 0 %

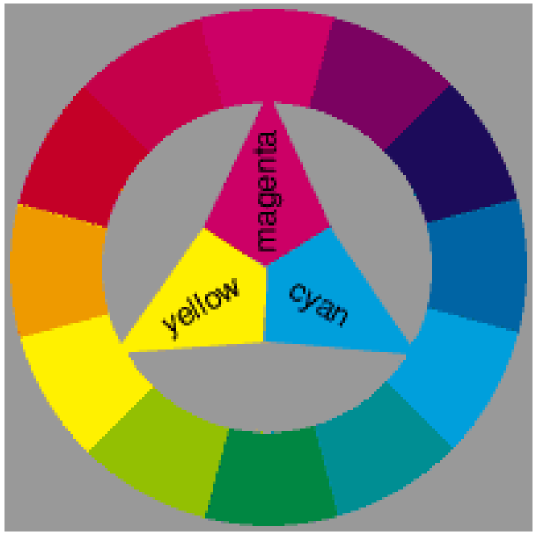
Black: 39 %

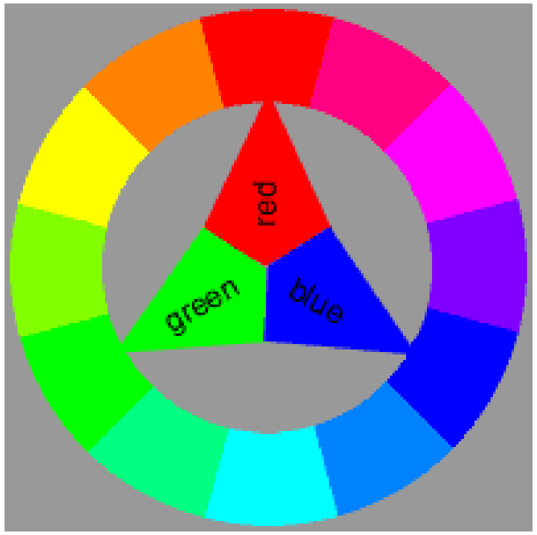
Illusion of full color created with CMYK dots



Color Wheels







Artist's color wheel, used for mixing paint

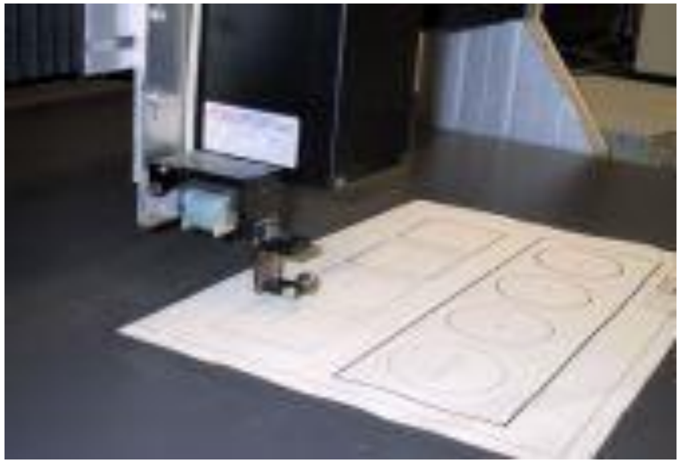
Subtractive color wheel, used in printing (CMYK)

Additive color wheel, used for projection

Primary colors appear at center and equally spaced around the perimeter
Secondary colors are midway between primary colors
Tertiary colors are between primary and secondary colors

Source of this and several other slides on color: http://www.devx.com/projectcool/Article/19954/0/
(see also color theory tutorial: http://graphics.kodak.com/documents/Introducing%20Color%20Theory.pdf)

21.5 Other Input/Output Devices



Sensors and Actuators

Collecting info about the environment and other conditions

- Light sensors (photocells)
- Temperature sensors (contact and noncontact types)
- Pressure sensors

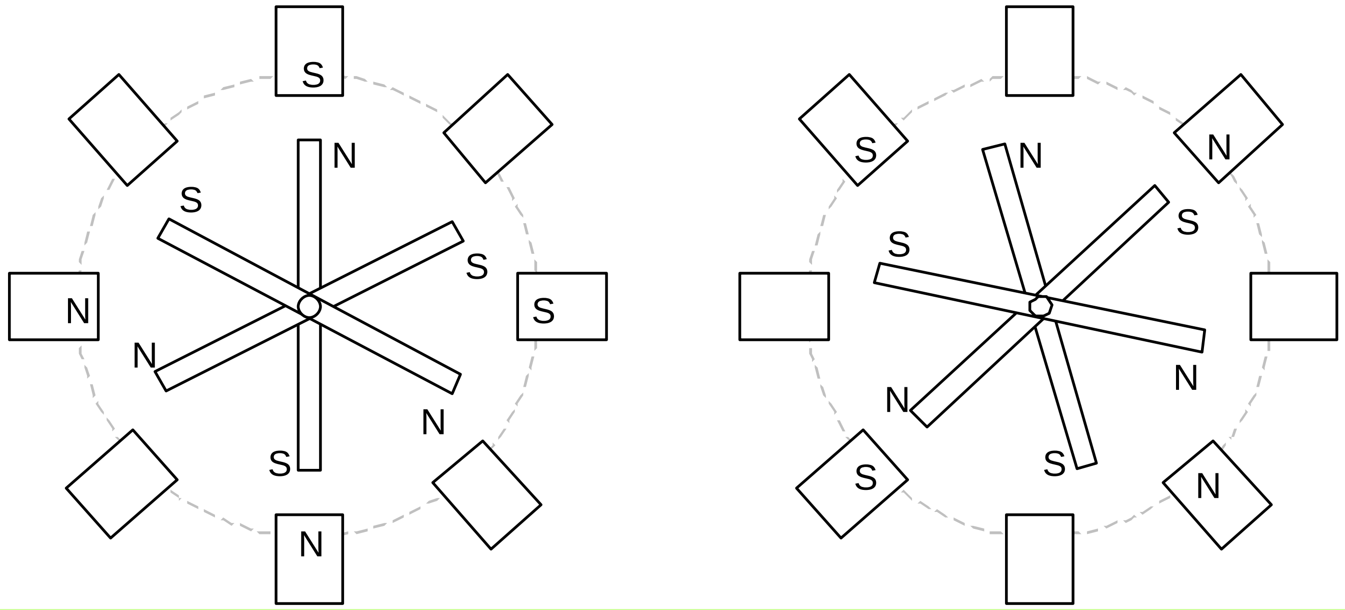
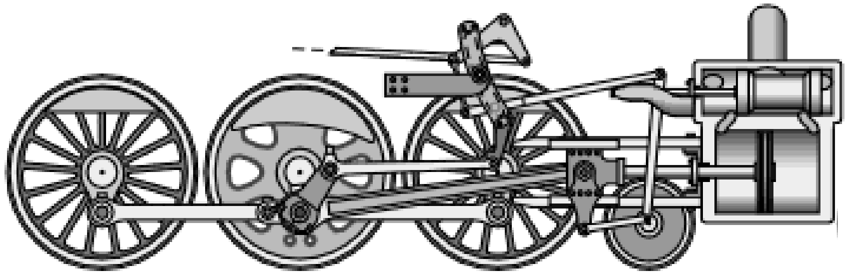


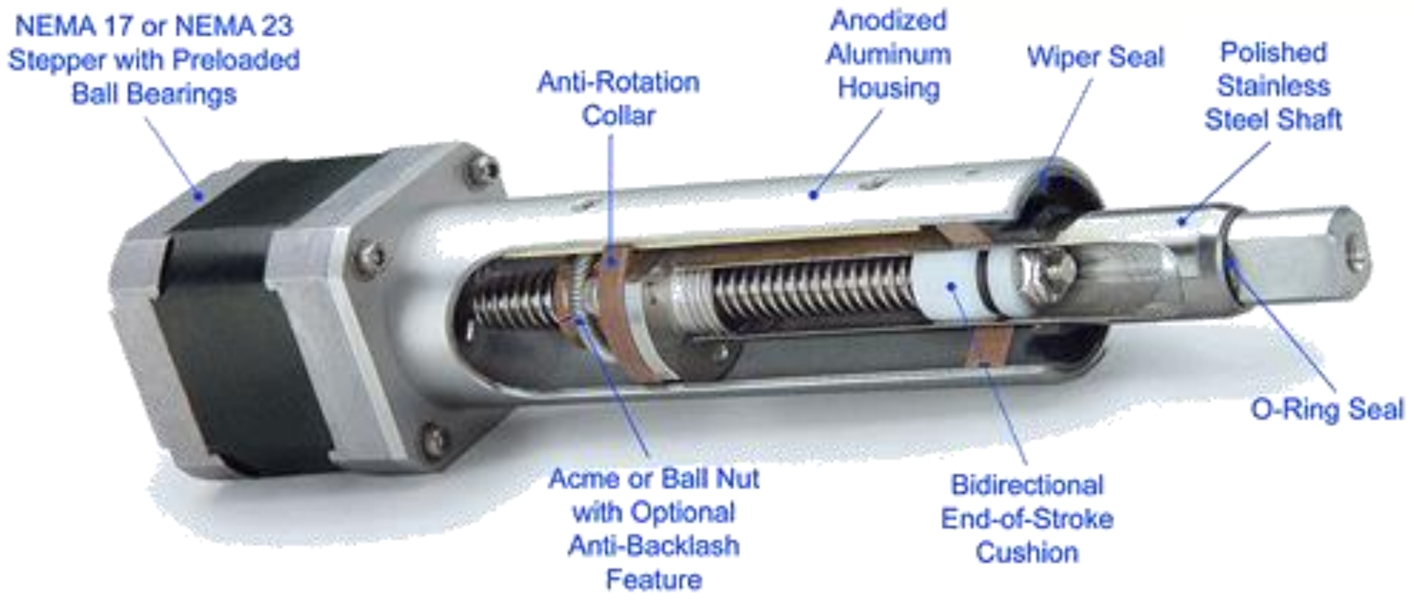
Figure 21.11 Stepper motor principles of operation.

Converting Circular Motion to Linear Motion

Locotive



Screw


NEMA 17 or NEMA 23 Stepper with Preloaded Ball Bearings
Anti-Rotation Collar
Anodized Aluminum Housing
Wiper Seal
Polished Stainless Steel Shaft
O-Ring Seal
Acme or Ball Nut with Optional Anti-Backlash Feature
Bidirectional End-of-Stroke Cushion

21.6 Networking of Input/Output Devices

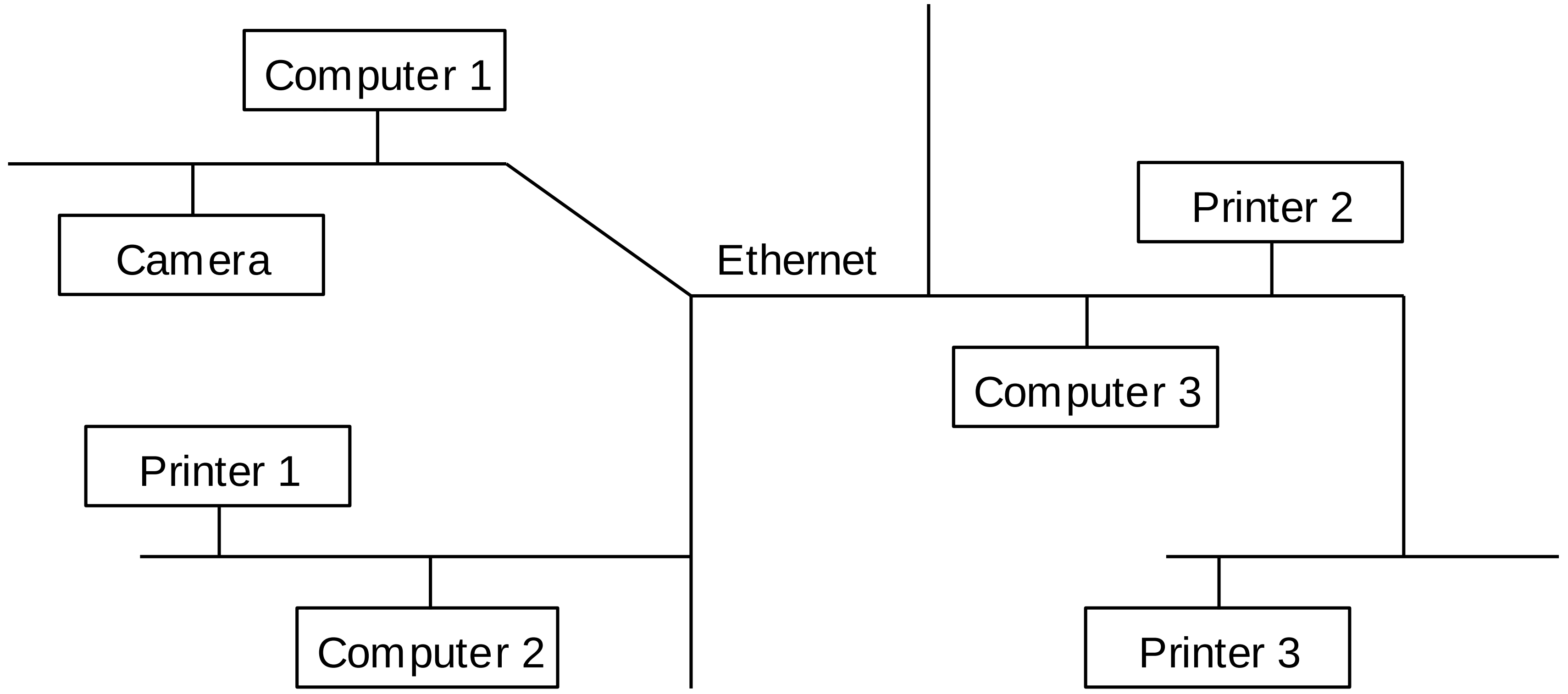


Figure 21.12 With network-enabled peripherals, I/O is done via file transfers.

Input/Output in Control and Embedded Systems

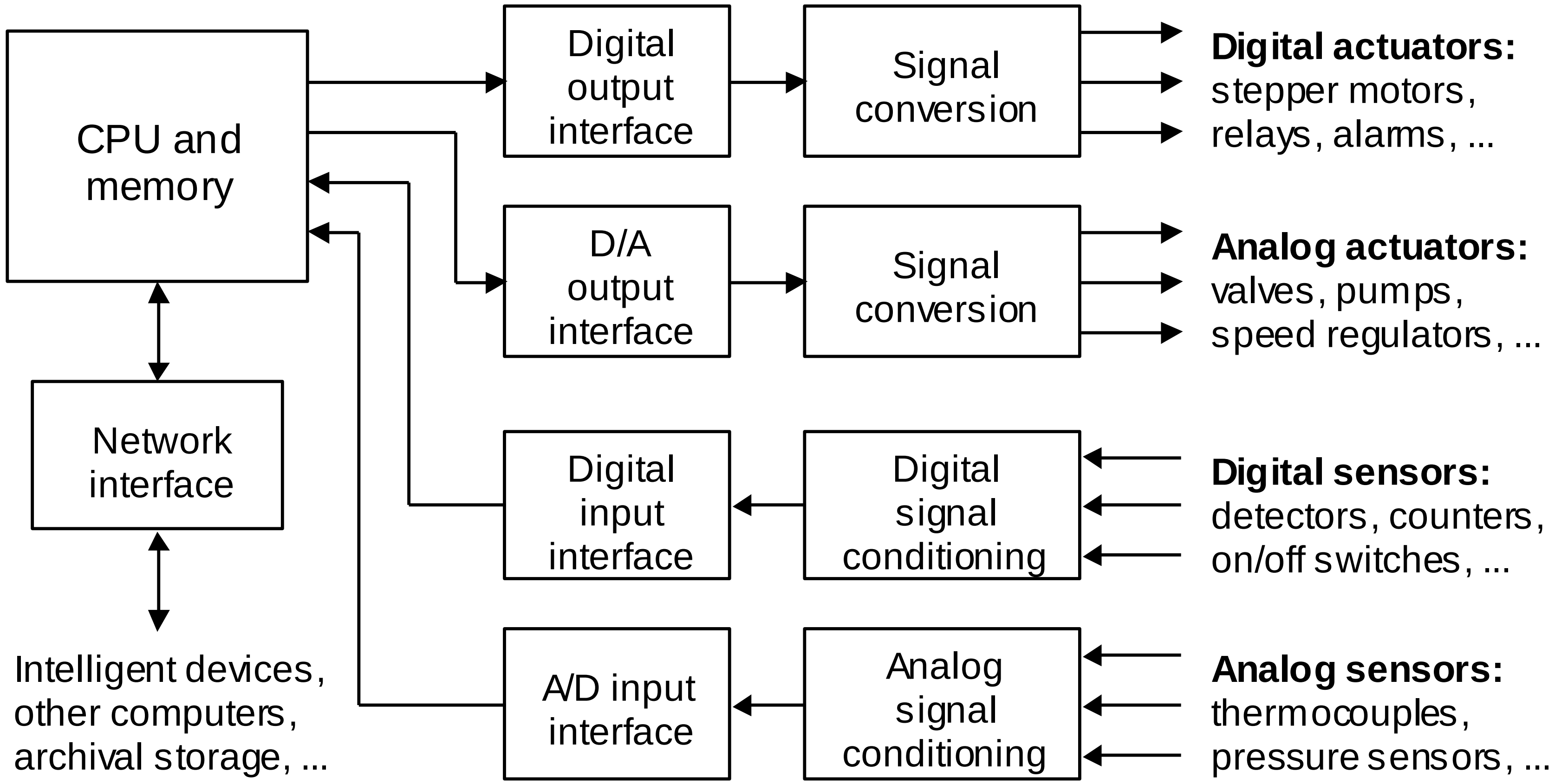

CPU and memory
Digital output interface
Signal conversion
Digital actuators: stepper motors, relays, alarms, ...
D/A output interface
Signal conversion
Analog actuators: valves, pumps, speed regulators, ...
Network interface
Intelligent devices, other computers, archival storage, ...
Digital input interface
Digital signal conditioning
Digital sensors: detectors, counters, on/off switches, ...
A/D input interface
Analog signal conditioning
Analog sensors: thermocouples, pressure sensors, ...

Figure 21.13 The structure of a closed-loop computer-based control system.

22 Input/Output Programming

Like everything else, I/O is controlled by machine instructions

- I/O addressing (memory-mapped) and performance
- Scheduled vs demand-based I/O: polling vs interrupts

Topics in This Chapter
22.1 I/O Performance and Benchmarks
22.2 Input/Output Addressing
22.3 Scheduled I/O: Polling
22.4 Demand-Based I/O: Interrupts
22.5 I/O Data Transfer and DMA
22.6 Improving I/O Performance

22.1 I/O Performance and Benchmarks

Example 22.1: The I/O wall

An industrial control application spent 90% of its time on CPU operations when it was originally developed in the early 1980s. Since then, the CPU component has been upgraded every 5 years, but the I/O components have remained the same. Assuming that CPU performance improved tenfold with each upgrade, derive the fraction of time spent on I/O over the life of the system.

Solution

Apply Amdahl's law with 90% of the task speeded up by factors of 10, 100, 1000, and 10000 over a 20-year period. In the course of these upgrades the running time has been reduced from the original 1 to 0.1 + 0.9/10 = 0.19, 0.109, 0.1009, and 0.10009, making the fraction of time spent on input/output 52.6, 91.7, 99.1, and 99.9%, respectively. The last couple of CPU upgrades did not really help.

Types of Input/Output Benchmark

Supercomputer I/O benchmarks

Reading large volumes of input data
Writing many snapshots for checkpointing
Saving a relatively small set of results
I/O data throughput, in MB/s, is important

Transaction processing I/O benchmarks

Huge database, but each transaction fairly small
A handful (2-10) of disk accesses per transaction
I/O rate (disk accesses per second) is important

File system I/O benchmarks

File creation, directory management, indexing, . . .
Benchmarks are usually domain-specific

22.2 Input/Output Addressing

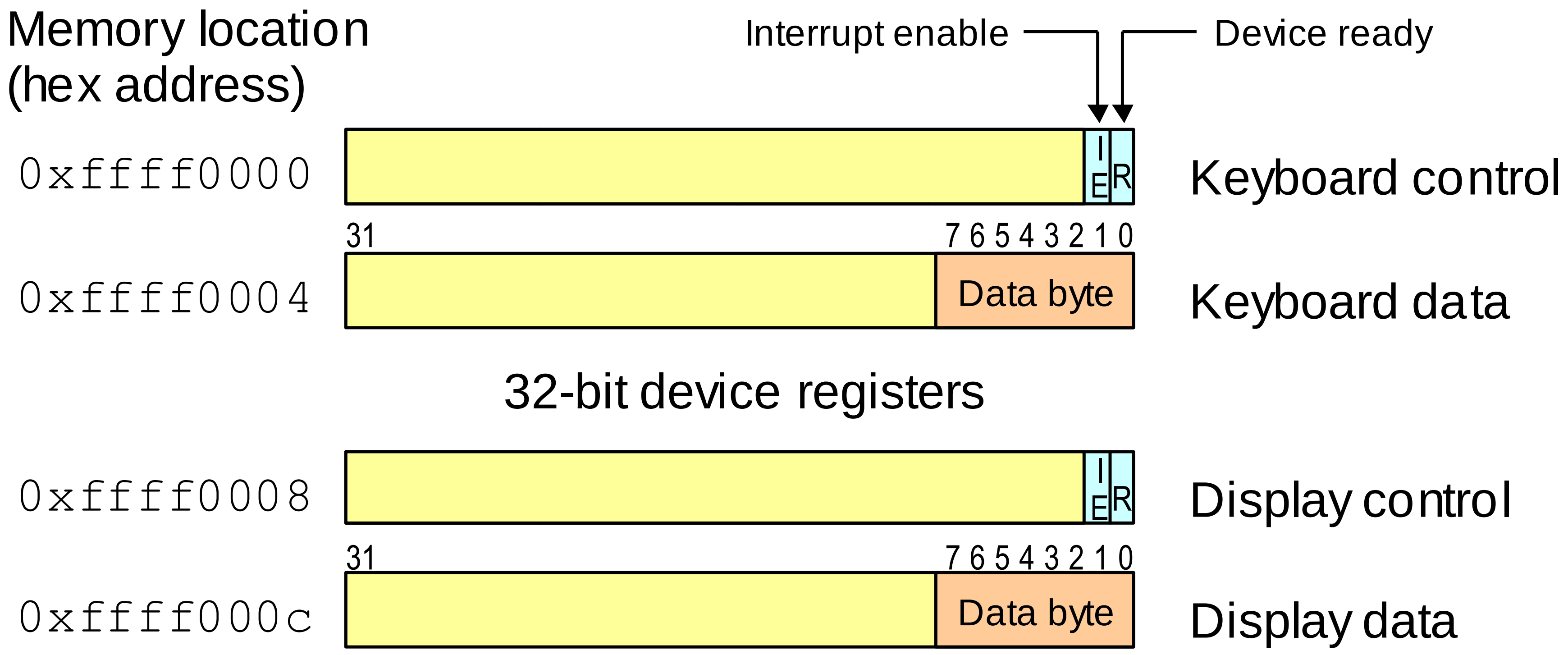


Figure 22.1 Control and data registers for keyboard and display unit in MiniMIPS.

Hardware for I/O Addressing

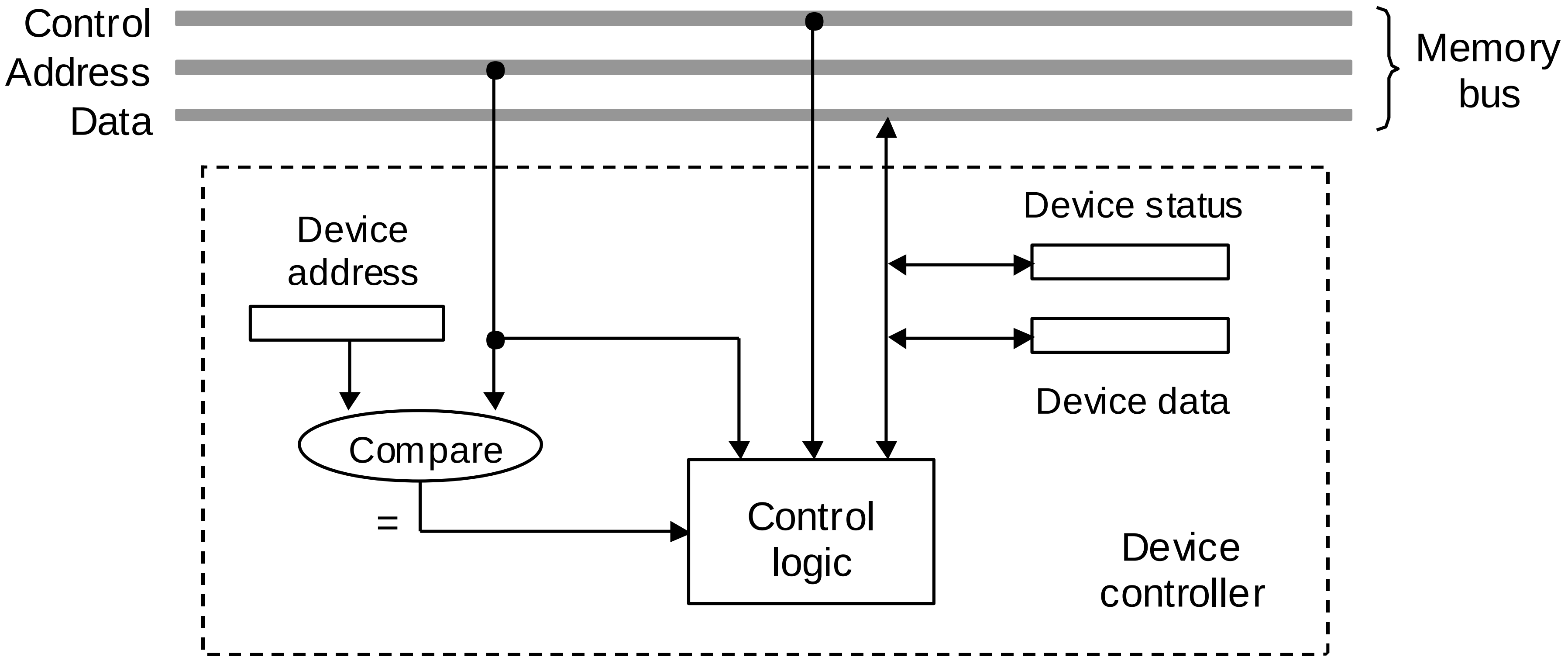
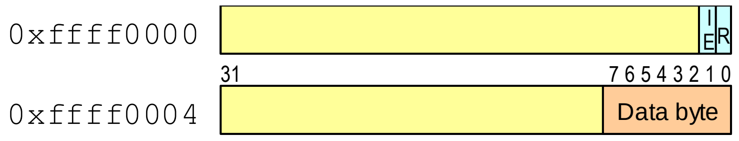


Figure 22.2 Addressing logic for an I/O device controller.

Data Input from Keyboard

Example 22.2

Write a sequence of MiniMIPS assembly language instructions to make the program wait until the keyboard has a symbol to transmit and then read the symbol into register `$v0`.



Solution

The program must continually examine the keyboard control register, ending its "busy wait" when the R bit has been asserted.

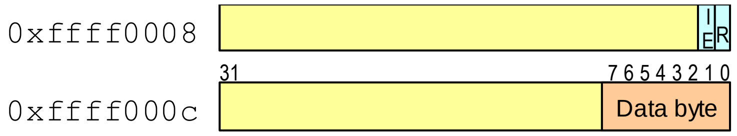
```
      lui  $t0,0xffff       # put 0xffff0000 in $t0
idle: lw   $t1,0($t0)       # get keyboard's control word
      andi $t1,$t1,0x0001   # isolate the LSB (R bit)
      beq  $t1,$zero,idle   # if not ready (R = 0), wait
      lw   $v0,4($t0)       # retrieve data from keyboard
```

This type of input is appropriate only if the computer is waiting for a critical input and cannot continue in the absence of such input.

Data Output to Display Unit

Example 22.3

Write a sequence of MiniMIPS assembly language instructions to make the program wait until the display unit is ready to accept a new symbol and then write the symbol from `$a0` to the display unit.



Solution

The program must continually examine the display unit's control register, ending its "busy wait" when the R bit has been asserted.

```
      lui   $t0,0xffff        # put 0xffff0000 in $t0
idle: lw    $t1,8($t0)        # get display's control word
      andi  $t1,$t1,0x0001    # isolate the LSB (R bit)
      beq   $t1,$zero,idle    # if not ready (R = 0), wait
      sw    $a0,12($t0)       # supply data to display unit
```

This type of output is appropriate only if we can afford to have the CPU dedicated to data transmission to the display unit.

22.3 Scheduled I/O: Polling

Examples 22.4, 22.5, 22.6

What fraction of a 1 GHz CPU's time is spent polling the following devices if each polling action takes 800 clock cycles?

Keyboard must be interrogated at least 10 times per second
Floppy sends data 4 bytes at a time at a rate of 50 KB/s
Hard drive sends data 4 bytes at a time at a rate of 3 MB/s

Solution

For keyboard, divide the number of cycles needed for 10 interrogations by the total number of cycles available in 1 second:
$(10 \times 800)/10^9 \cong 0.001\%$

The floppy disk must be interrogated 50K/4 = 12.5K times per sec
$(12.5\text{K} \times 800)/10^9 \cong 1\%$

The hard disk must be interrogated 3M/4 = 750K times per sec
$(750\text{K} \times 800)/10^9 \cong 60\%$

22.4 Demand-Based I/O: Interrupts

Example 22.7

Consider the disk in Example 22.6 (transferring 4 B chunks of data at 3 MB/s when active). Assume that the disk is active 5% of the time. The overhead of interrupting the CPU and performing the transfer is 1200 clock cycles. What fraction of a 1 GHz CPU's time is spent attending to the hard disk drive?

Solution

When active, the hard disk produces 750K interrupts per second

$0.05 \times (750\text{K} \times 1200)/10^9 \cong 4.5\%$ (compare with 60% for polling)

Note that even though the overhead of interrupting the CPU is higher than that of polling, because the disk is usually idle, demand-based I/O leads to better performance.

Interrupt Handling

Upon detecting an interrupt signal, provided the particular interrupt or interrupt class is not masked, the CPU acknowledges the interrupt (so that the device can deassert its request signal) and begins executing an interrupt service routine.

1. Save the CPU state and call the interrupt service routine.
2. Disable all interrupts.
3. Save minimal information about the interrupt on the stack.
4. Enable interrupts (or at least higher priority ones).
5. Identify cause of interrupt and attend to the underlying request.
6. Restore CPU state to what existed before the last interrupt.
7. Return from interrupt service routine.

The capability to handle nested interrupts is important in dealing with multiple high-speed I/O devices.

22.5 I/O Data Transfer and DMA

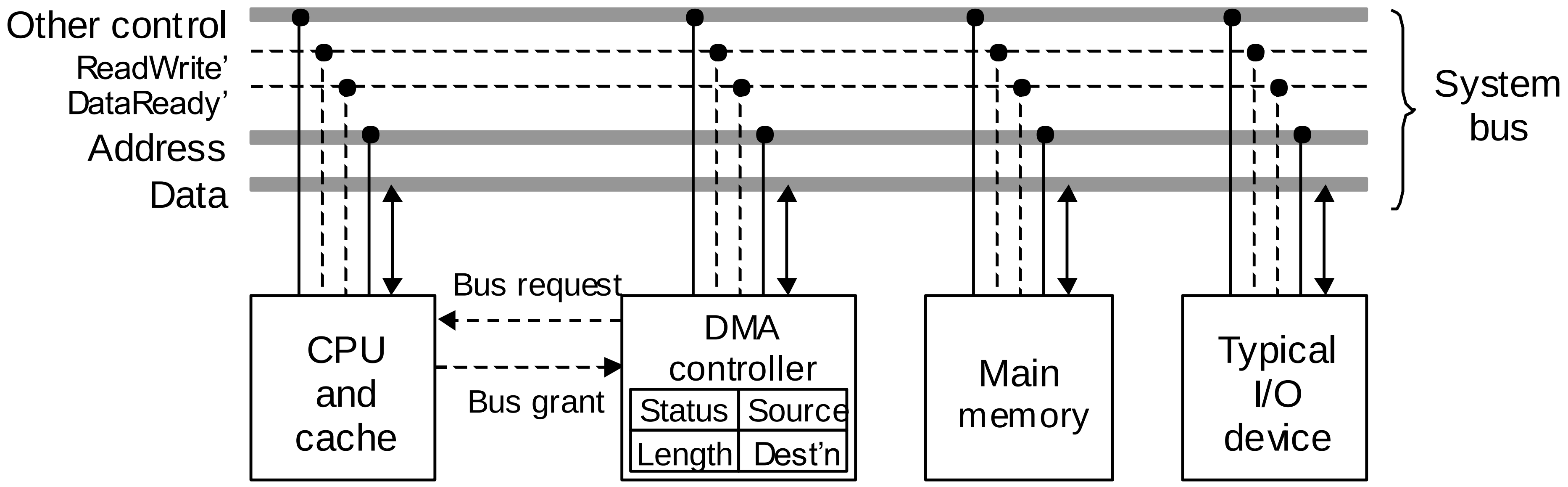
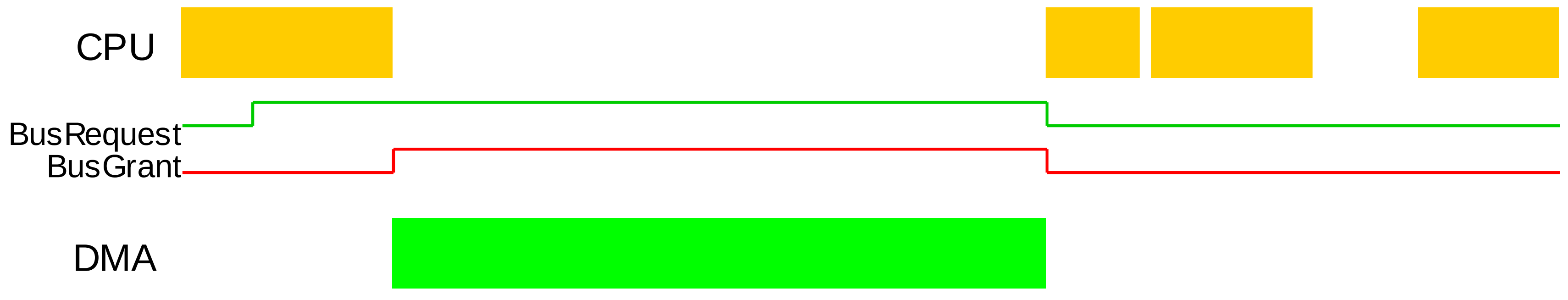
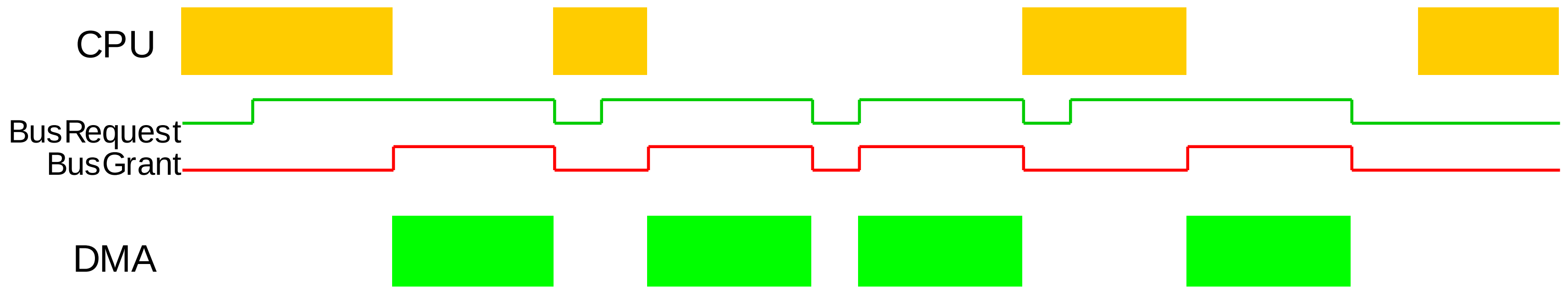


Figure 22.3 DMA controller shares the system or memory bus with the CPU.

DMA Operation



(a) DMA transfer in one continuous burst



(b) DMA transfer in several shorter bursts

Figure 22.4 DMA operation and the associated transfers of bus control.

22.6 Improving I/O Performance

Example 22.9: Effective I/O bandwidth from disk

Consider a hard disk drive with 512 B sectors, average access latency of 10 ms, and peak throughput of 10 MB/s. Plot the variation of the effective I/O bandwidth as the unit of data transfer (block) varies in size from 1 sector (0.5 KB) to 1024 sectors (500 KB).

Solution

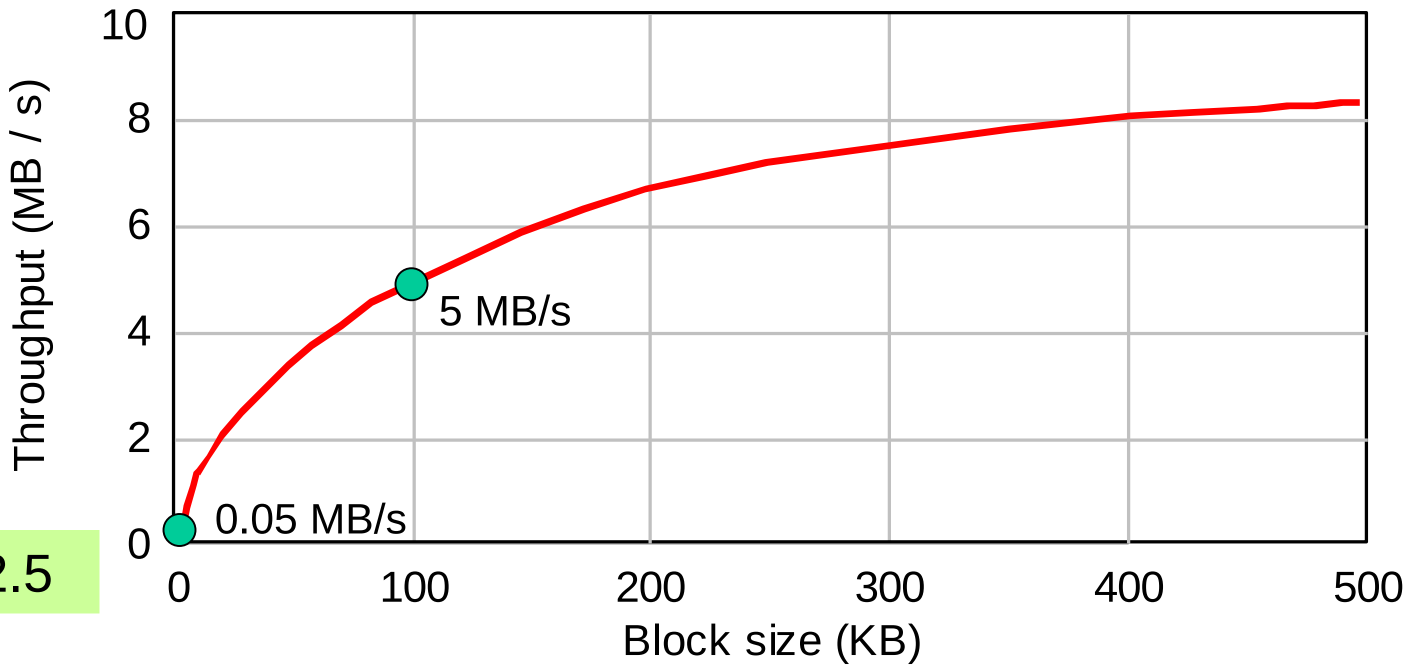


Figure 22.5

Computing the Effective Throughput

Elaboration on Example 22.9: Effective I/O bandwidth from disk

Total access time for x bytes = 10 ms + xfer time = $(0.01 + 10^{-7}x)$ s

Effective access time per byte = $(0.01 + 10^{-7}x)/x$ s/B

Effective transfer rate = $x/(0.01 + 10^{-7}x)$ B/s

For x = 100 KB: Effective transfer rate = $10^5/(0.01 + 10^{-2}) = 5\times10^6$ B/s

Average access latency = 10 ms

Peak throughput = 10 MB/s

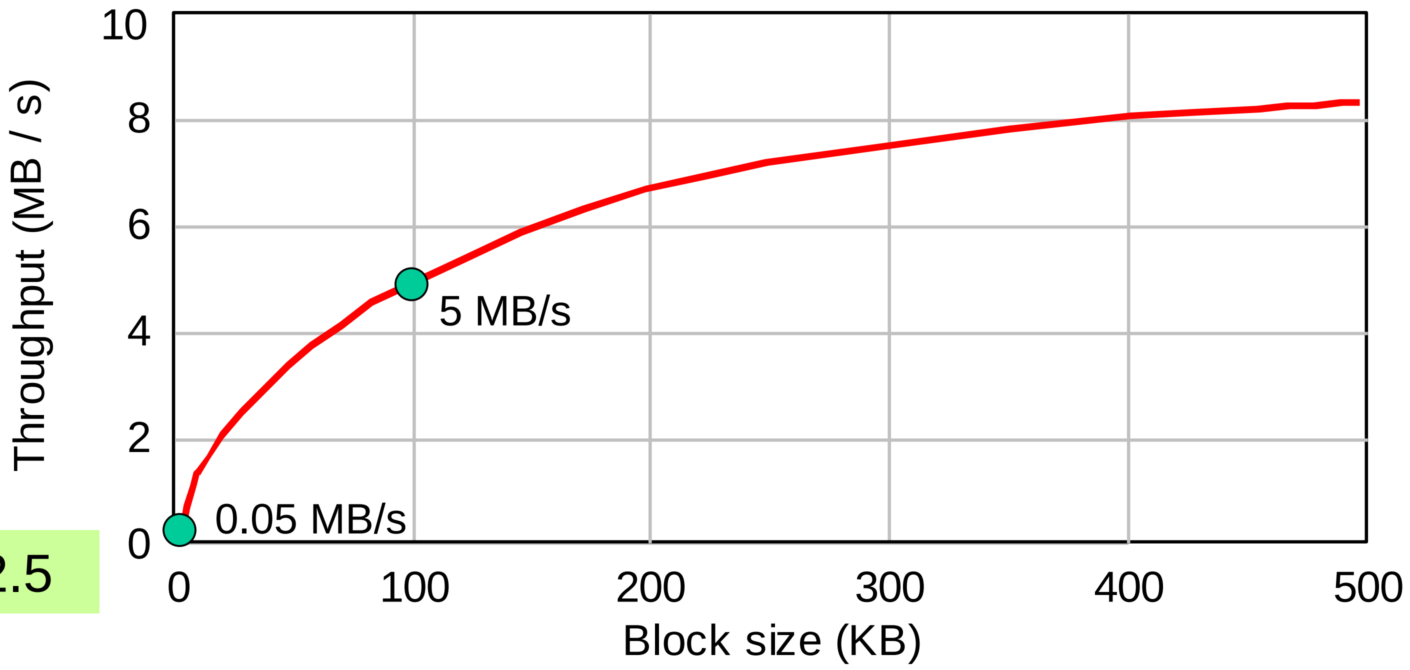


Figure 22.5

Distributed Input/Output

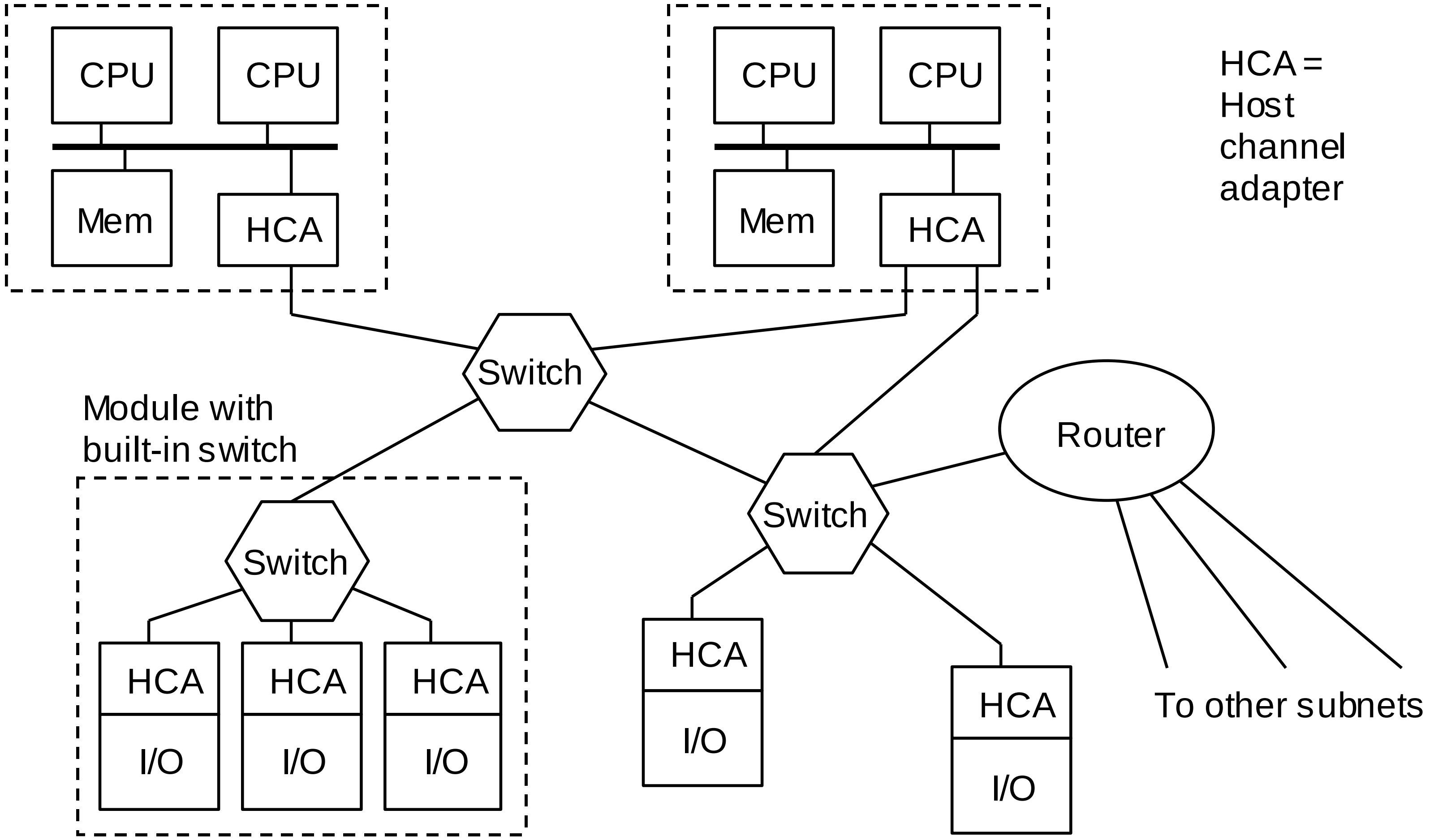


Figure 22.6 Example configuration for the Infiniband distributed I/O.

23 Buses, Links, and Interfacing

Shared links or buses are common in modern computers:

- Fewer wires and pins, greater flexibility & expandability
- Require dealing with arbitration and synchronization

Topics in This Chapter
23.1 Intra- and Intersystem Links
23.2 Buses and Their Appeal
23.3 Bus Communication Protocols
23.4 Bus Arbitration and Performance
23.5 Basics of Interfacing
23.6 Interfacing Standards

23.1 Intra- and Intersystem Links

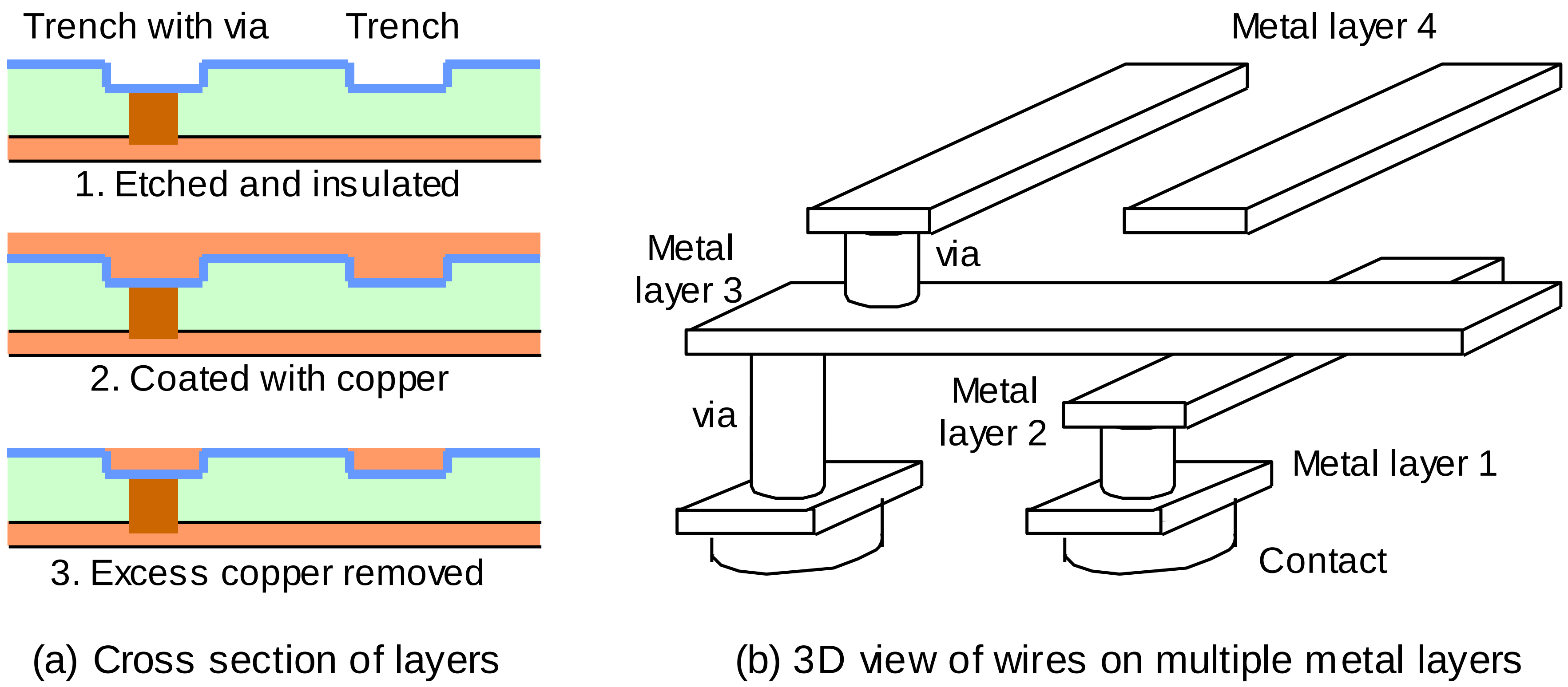
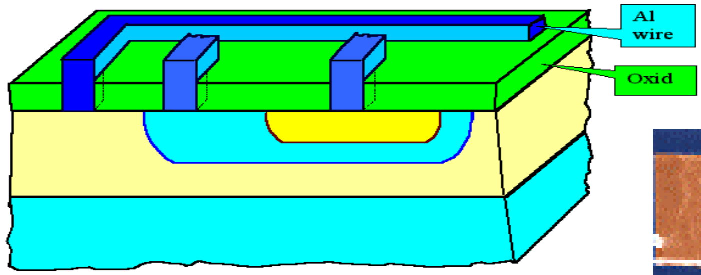


Figure 23.1 Multiple metal layers provide intrasystem connectivity on microchips or printed-circuit boards.

Multiple Metal Layers on a Chip or PC Board

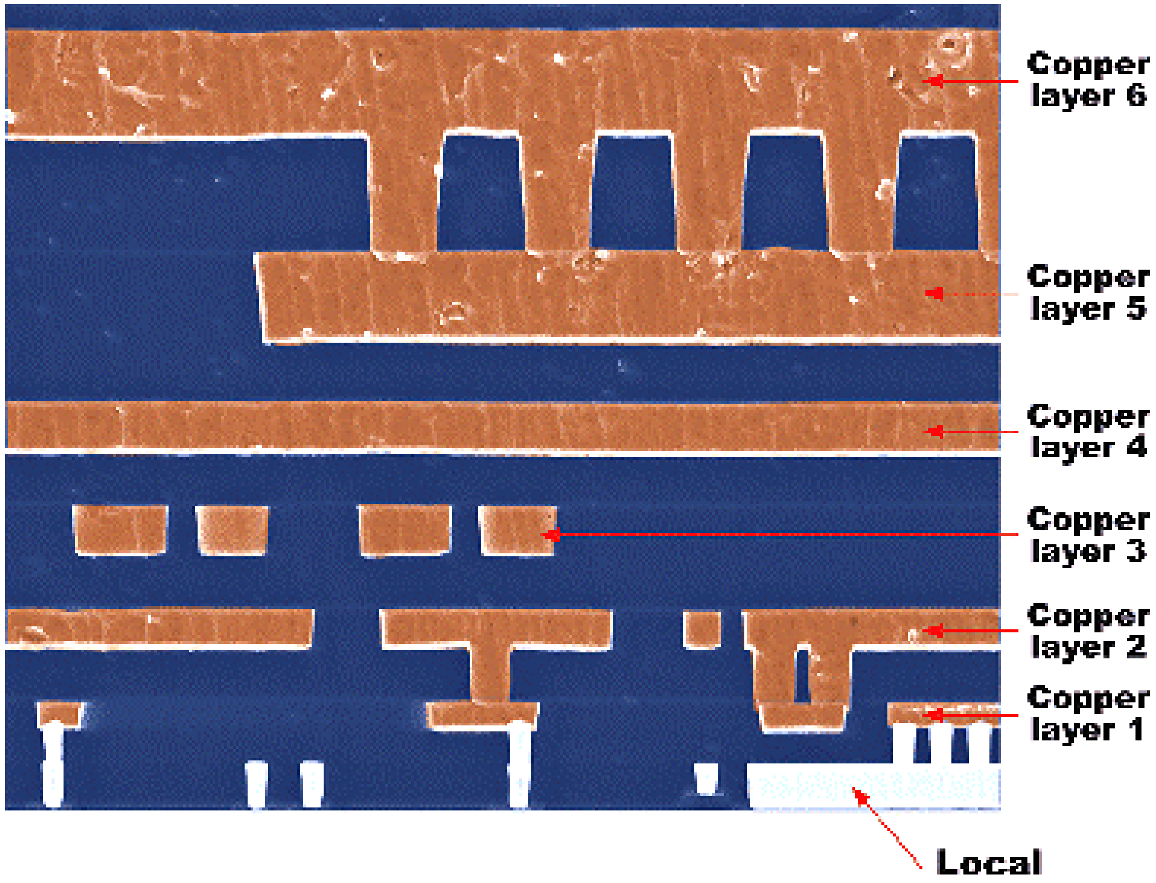


Active elements and their connectors

Modern chips have 8-9 metal layers

Upper layers carry longer wires as well as those that need more power

Cross section of metal layers



Intersystem Links

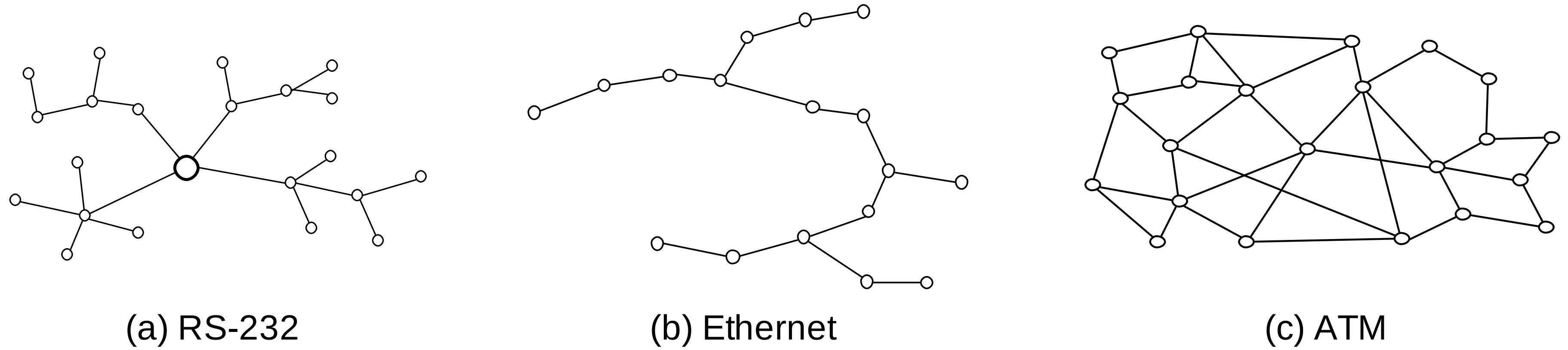


Figure 23.2 Example intersystem connectivity schemes.

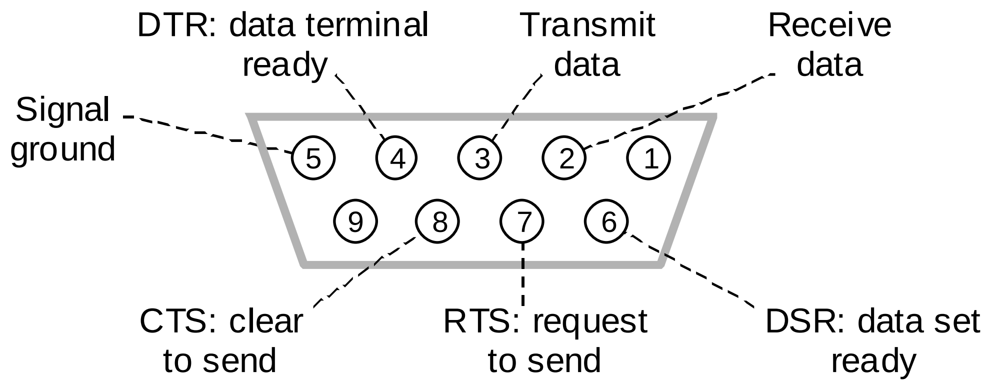


Figure 23.3 RS-232 serial interface 9-pin connector.

Intersystem Communication Media

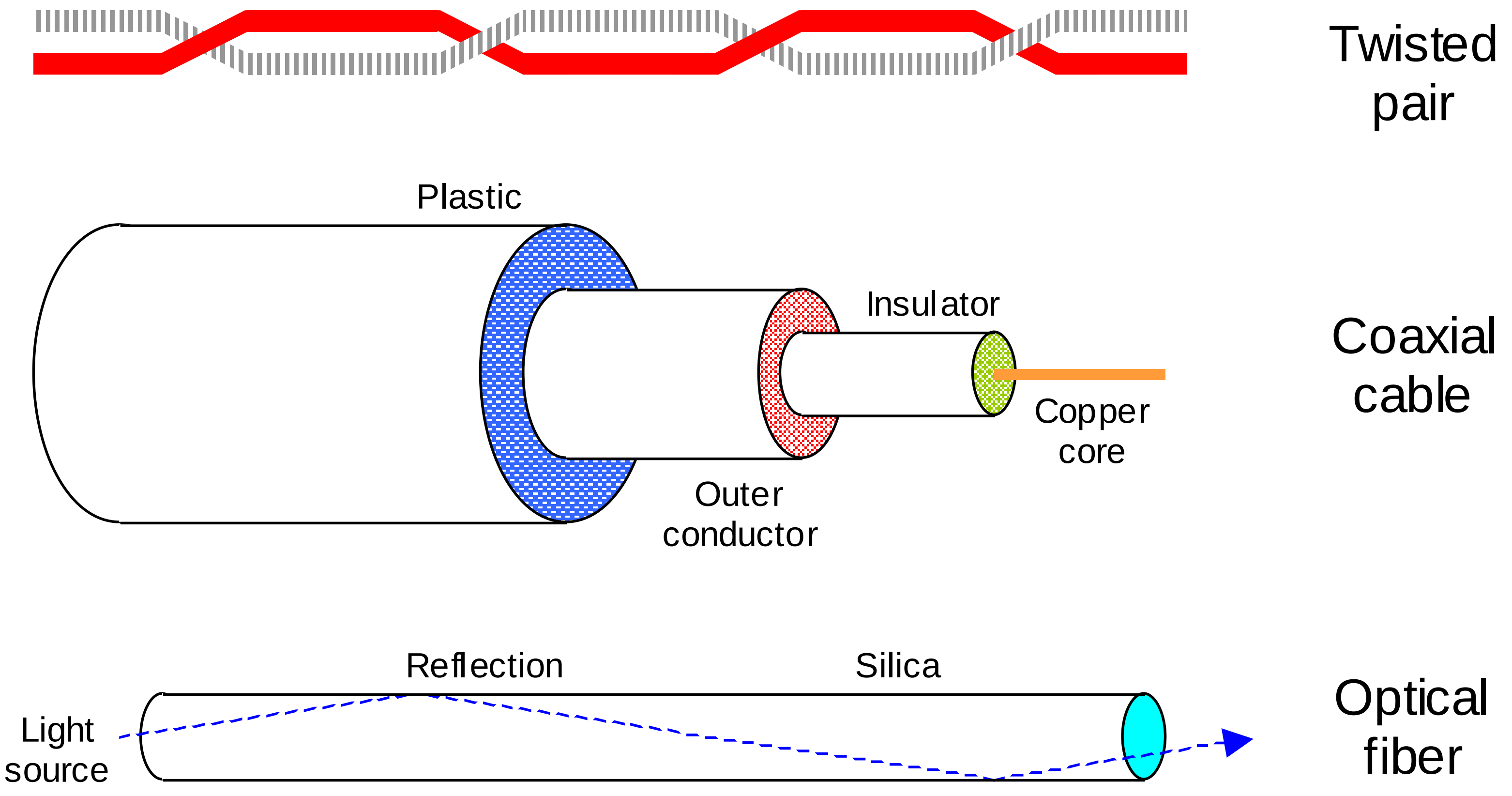


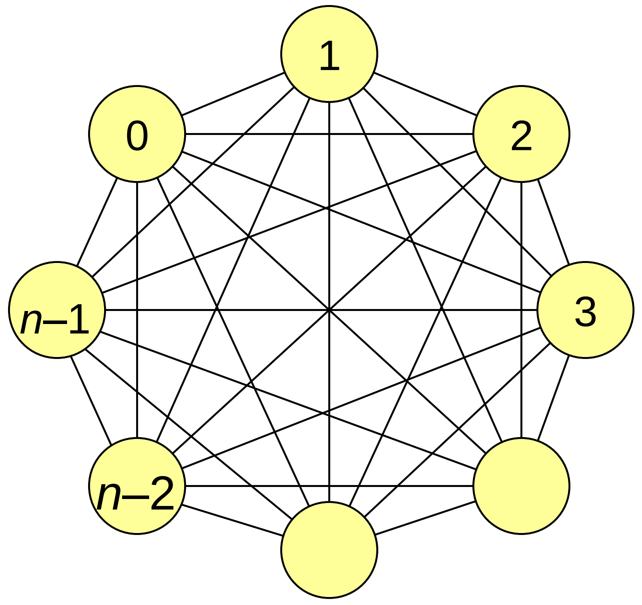
Figure 23.4 Commonly used communication media for intersystem connections.

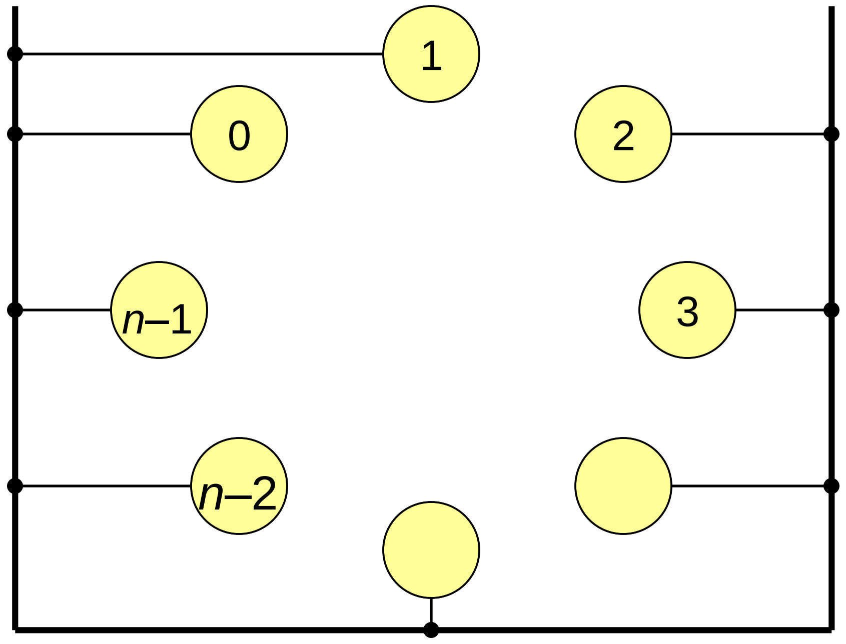
Comparing Intersystem Links

Table 23.1 Summary of three interconnection schemes.

Interconnection properties	RS-232	Ethernet	ATM
Maximum segment length (m)	10s	100s	1000s
Maximum network span (m)	10s	100s	Unlimited
Bit rate (Mb/s)	Up to 0.02	10/100/1000	155-2500
Unit of transmission (B)	1	100s	53
Typical end-to-end latency (ms)	< 1	10s-100s	100s
Typical application domain	Input/Output	LAN	Backbone
Transceiver complexity or cost	Low	Low	High

23.2 Buses and Their Appeal





Point-to-point connections between *n* units require *n*(*n* – 1) channels, or *n*(*n* – 1)/2 bidirectional links; that is, O(n^2) links

Bus connectivity requires only one input and one output port per unit, or O(*n*) links in all

Bus Components and Types

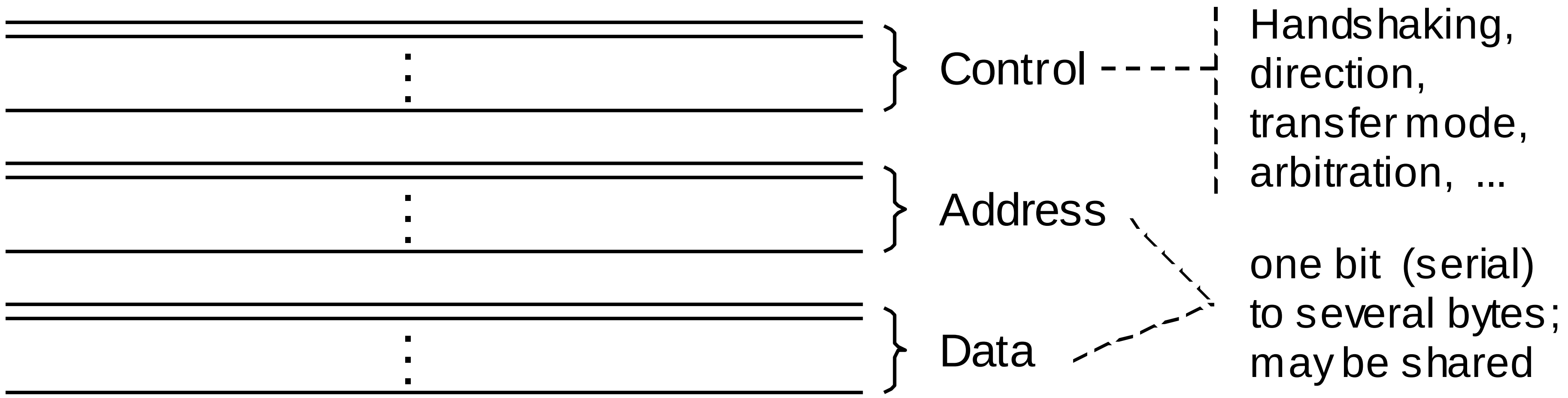

Control
Handshaking, direction, transfer mode, arbitration, ...
Address
Data
one bit (serial) to several bytes; may be shared

Figure 23.5 The three sets of lines found in a bus.

A typical computer may use a dozen or so different buses:

1. Legacy Buses: PC bus, ISA, RS-232, parallel port
2. Standard buses: PCI, SCSI, USB, Ethernet
3. Proprietary buses: for specific devices and max performance

23.3 Bus Communication Protocols

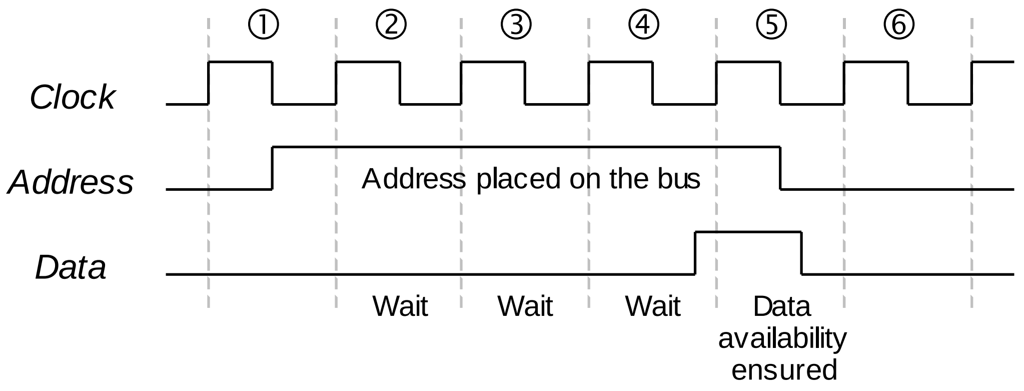


Figure 23.6 Synchronous bus with fixed-latency devices.

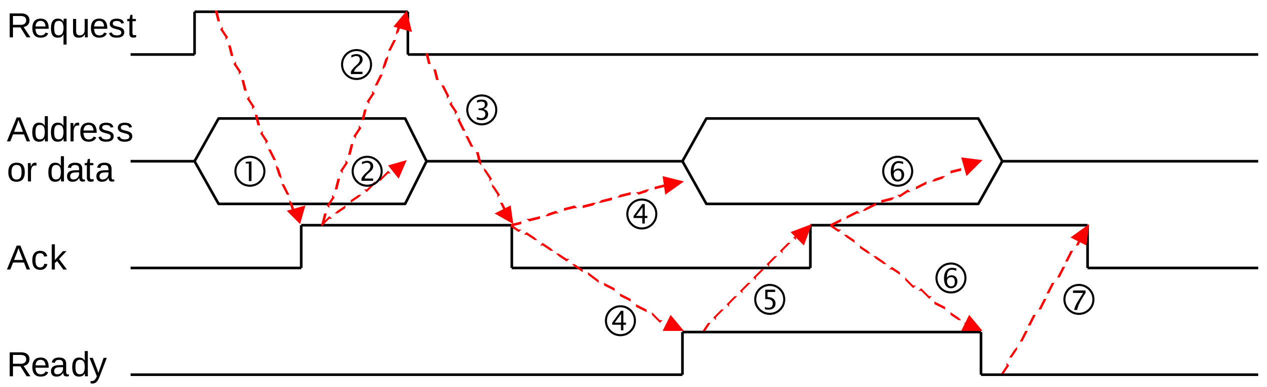


Figure 23.7 Handshaking on an asynchronous bus for an input operation (e.g., reading from memory).

Example Bus Operation

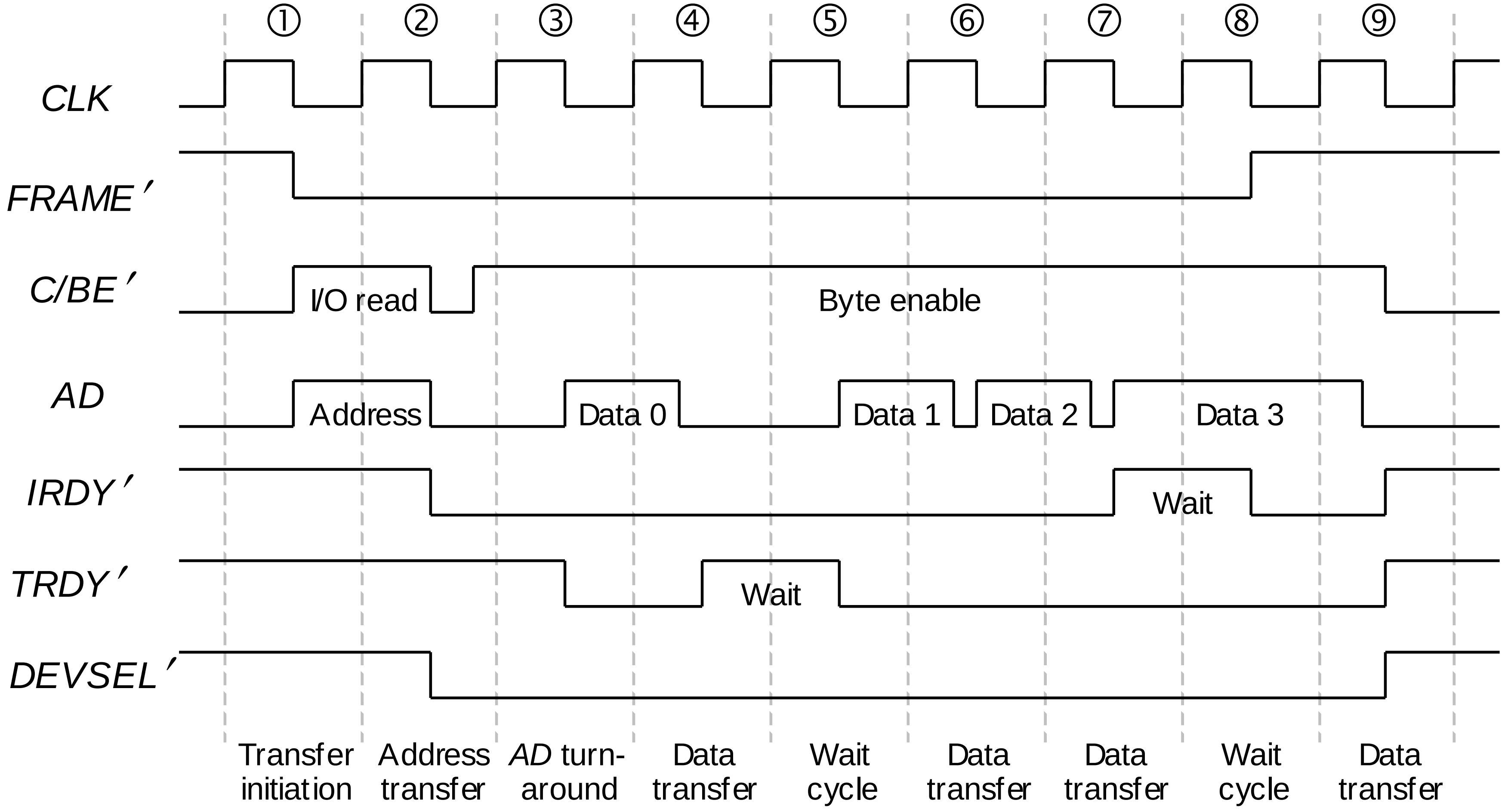


Figure 23.8 I/O read operation via PCI bus.

23.4 Bus Arbitration and Performance

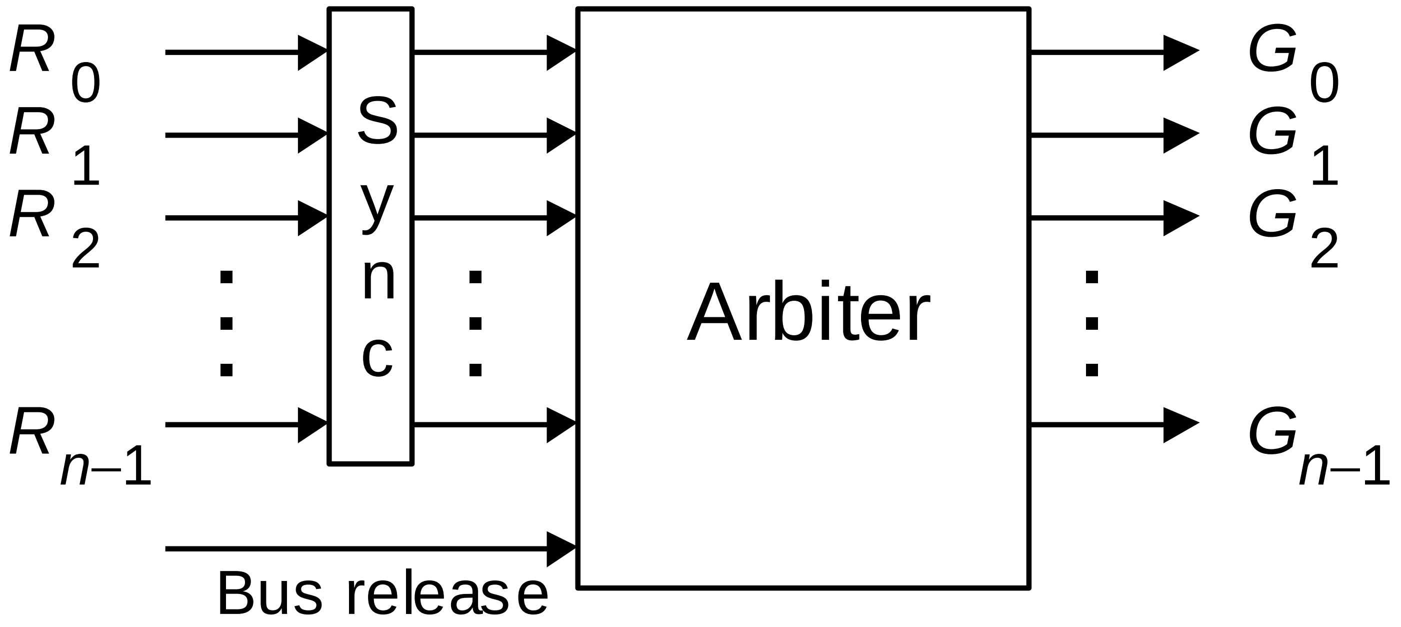
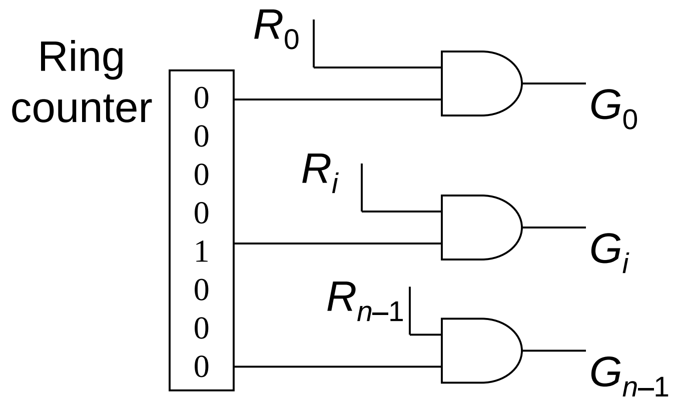


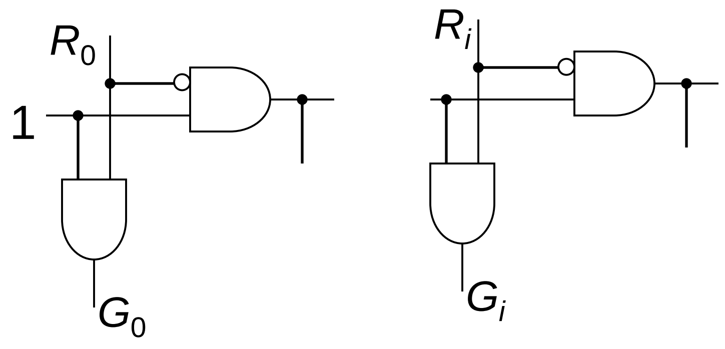
Figure 23.9 General structure of a centralized bus arbiter.

Some Simple Bus Arbiters

Round robin



Fixed-priority



Starvation avoidance

With fixed priorities, low-priority units may never get to use the bus (they could "starve")

Combining priority with service guarantee is desirable

Rotating priority

Idea: Order the units circularly, rather than linearly, and allow the highest-priority status to rotate among the units (combine a ring counter with a priority circuit)

Daisy Chaining

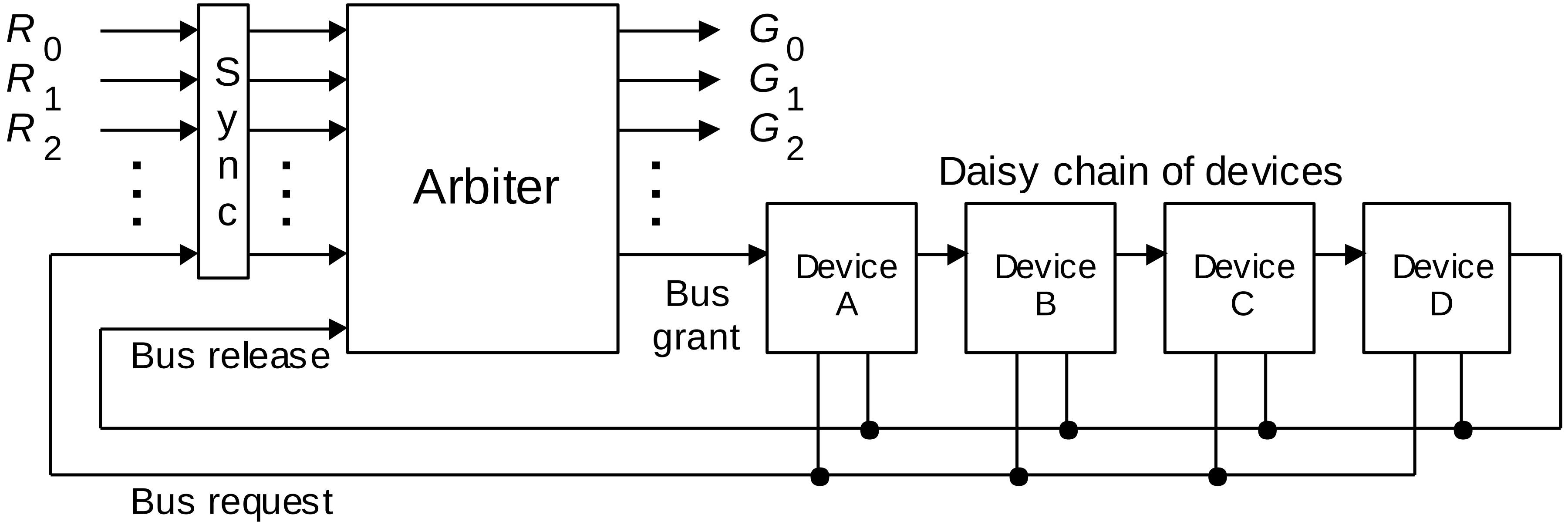


Figure 23.9 Daisy chaining allows a small centralized arbiter to service a large number of devices that use a shared resource.

23.5 Basics of Interfacing

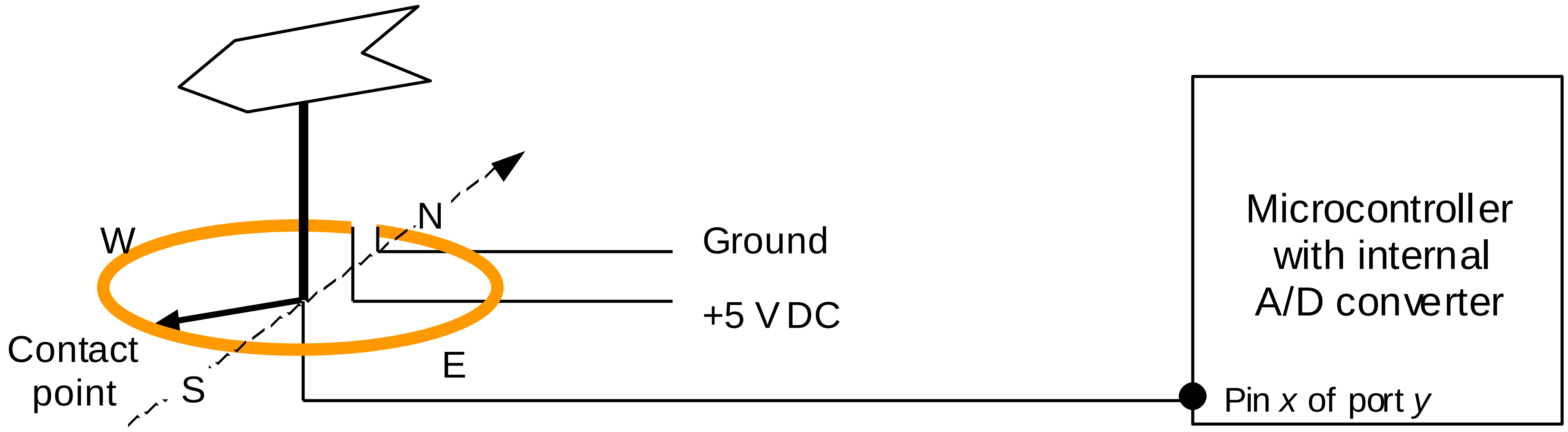


Figure 23.11 Wind vane supplying an output voltage in the range 0-5 V depending on wind direction.

23.6 Interfacing Standards

Table 23.2 Summary of four standard interface buses.

Attributes ↓ Name →	PCI	SCSI	FireWire	USB
Type of bus	Backplane	Parallel I/O	Serial I/O	Serial I/O
Standard designation	PCI	ANSI X3.131	IEEE 1394	USB 2.0
Typical application domain	System	Fast I/O	Fast I/O	Low-cost I/O
Bus width (data bits)	32-64	8-32	2	1
Peak bandwidth (MB/s)	133-512	5-40	12.5-50	0.2-15
Maximum number of devices	1024*	7-31#	63	127$
Maximum span (m)	< 1	3-25	4.5-72$	5-30$
Arbitration method	Centralized	Self-select	Distributed	Daisy chain
Transceiver complexity or cost	High	Medium	Medium	Low

Notes: * 32 per bus segment; # One less than bus width; $ With hubs (repeaters)

Standard Connectors

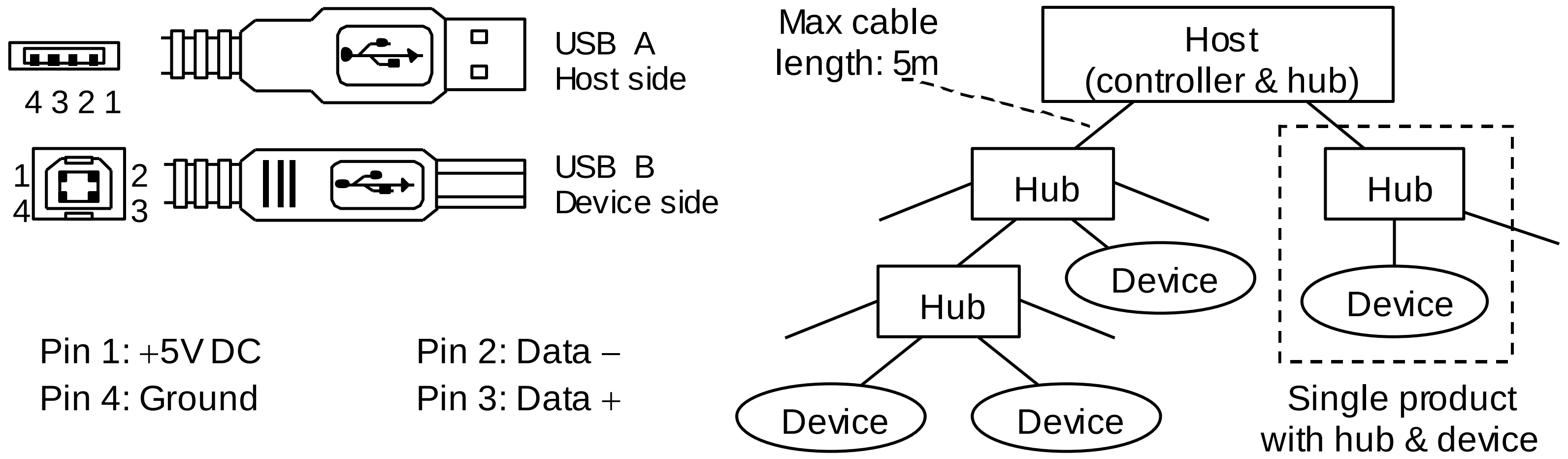
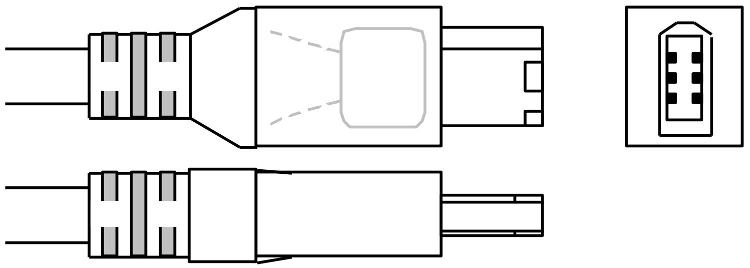


Figure 23.12 USB connectors and connectivity structure .



Pin 1: 8-40V DC, 1.5 A
Pin 2: Ground
Pin 3: Twisted pair B –
Pin 4: Twisted pair B +
Pin 5: Twisted pair A –
Pin 6: Twisted pair A +
Shell: Outer shield

Figure 23.13 IEEE 1394 (FireWire) connector. The same connector is used at both ends.

24 Context Switching and Interrupts

OS initiates I/O transfers and awaits notification via interrupts

- When an interrupt is detected, the CPU switches context
- Context switch can also be used between users/threads

Topics in This Chapter	
24.1	System Calls for I/O
24.2	Interrupts, Exceptions, and Traps
24.3	Simple Interrupt Handling
24.4	Nested Interrupts
24.5	Types of Context Switching
24.6	Threads and Multithreading

24.1 System Calls for I/O

Why the user must be isolated from details of I/O operations

Protection: User must be barred from accessing some disk areas

Convenience: No need to learn details of each device's operation

Efficiency: Most users incapable of finding the best I/O scheme

I/O abstraction: grouping of I/O devices into a small number of generic types so as to make the I/O device-independent

Character stream I/O: get(•), put(•) – e.g., keyboard, printer

Block I/O: seek(•), read(•), write(•) – e.g., disk

Network Sockets: create socket, connect, send/receive packet

Clocks or timers: set up timer (get notified via an interrupt)

24.2 Interrupts, Exceptions, and Traps

Interrupt	Both general term for any diversion and the I/O type
Exception	Caused by an illegal operation (often unpredictable)
Trap	AKA “software interrupt” (preplanned and not rare)

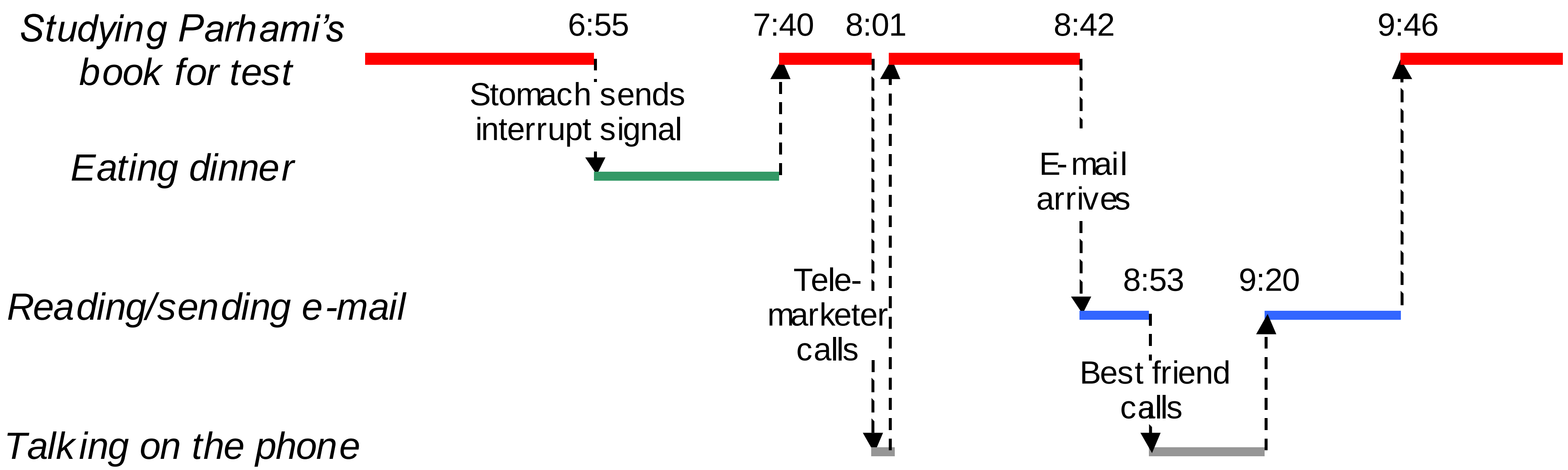


Figure 24.1 The notions of interrupts and nested interrupts.

24.3 Simple Interrupt Handling

Acknowledge the interrupt by asserting the IntAck signal
Notify the CPU's next-address logic that an interrupt is pending
Set the interrupt mask so that no new interrupt is accepted

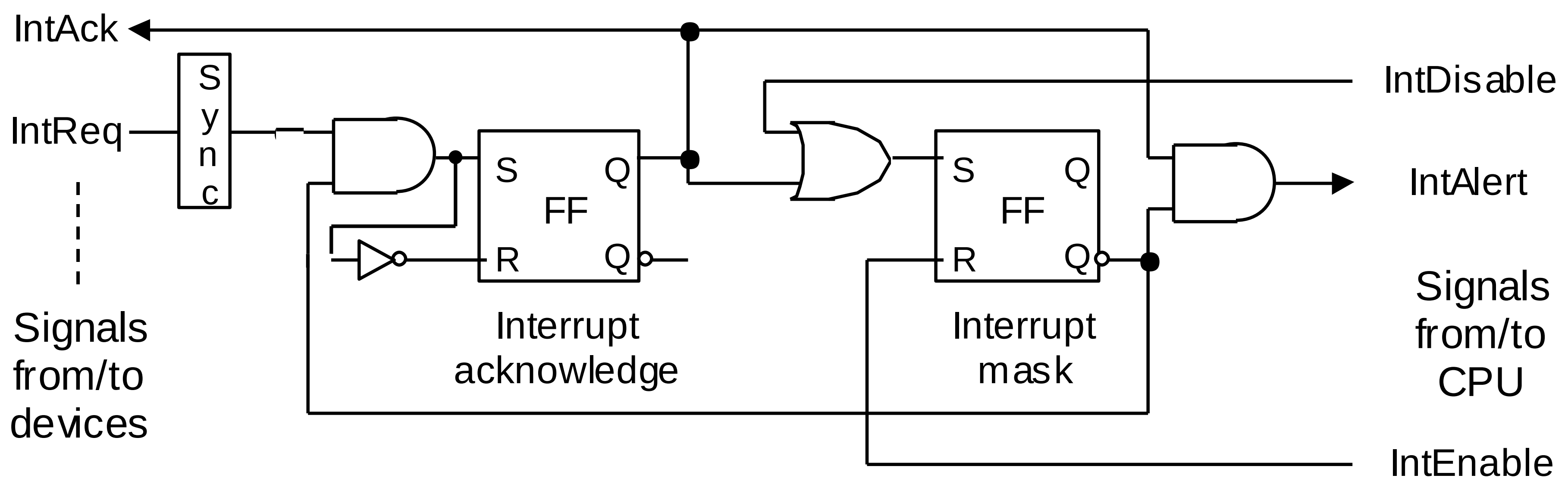


Figure 24.2 Simple interrupt logic for the single-cycle MicroMIPS.

Interrupt Timing

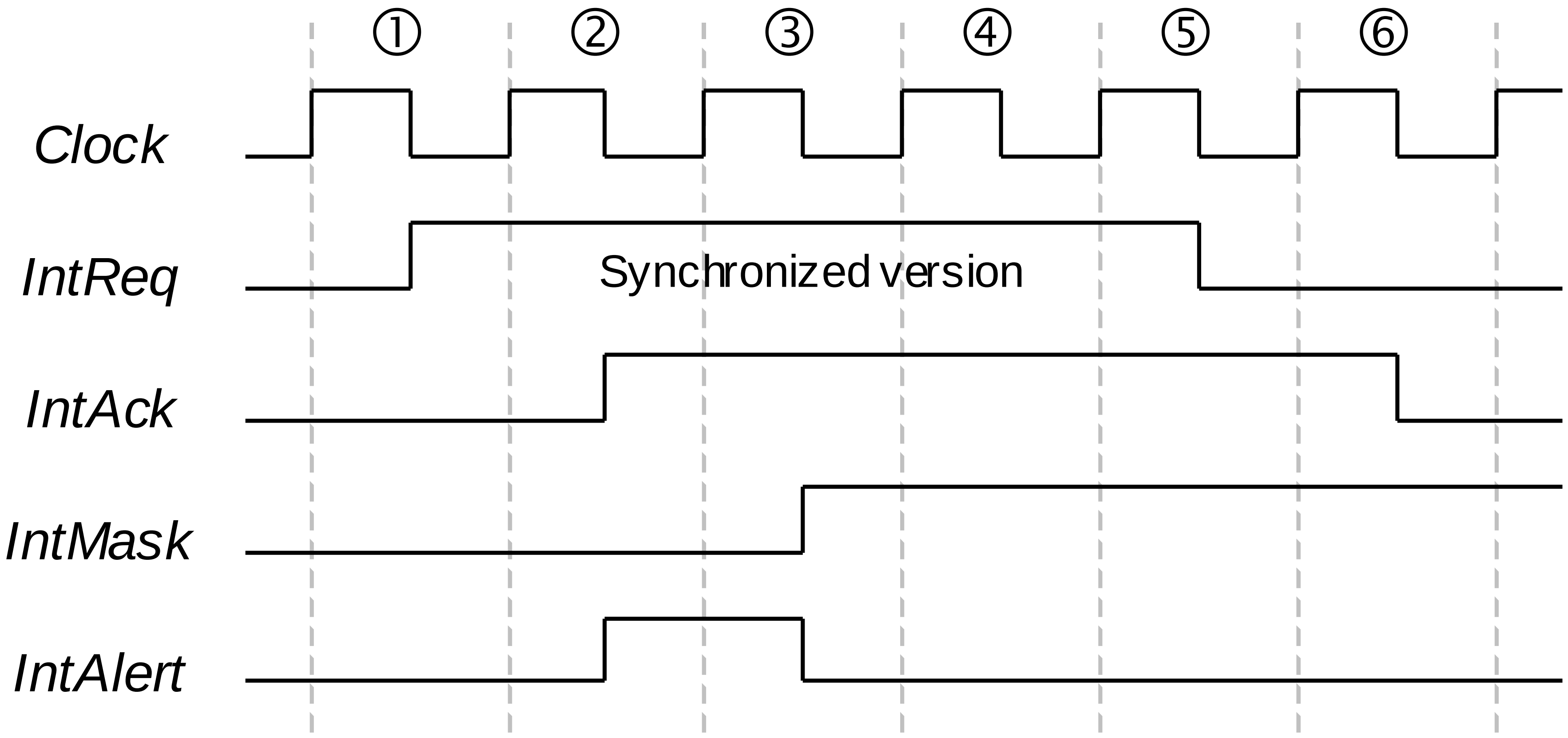


Figure 24.3 Timing of interrupt request and acknowledge signals.

Next-Address Logic with Interrupts Added

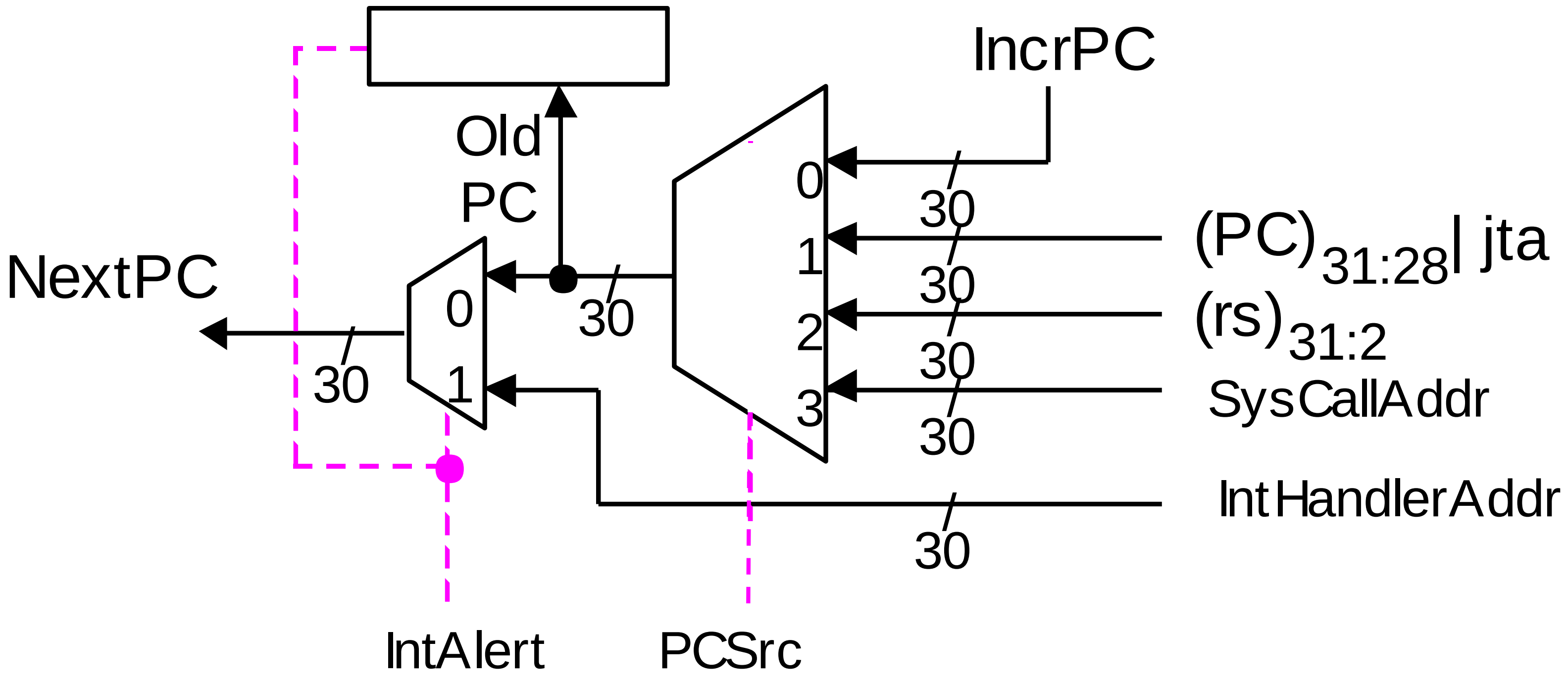


Figure 24.4 Part of the next-address logic for single-cycle MicroMIPS, with an interrupt capability added (compare with the lower left part of Figure 13.4).

24.4 Nested Interrupts

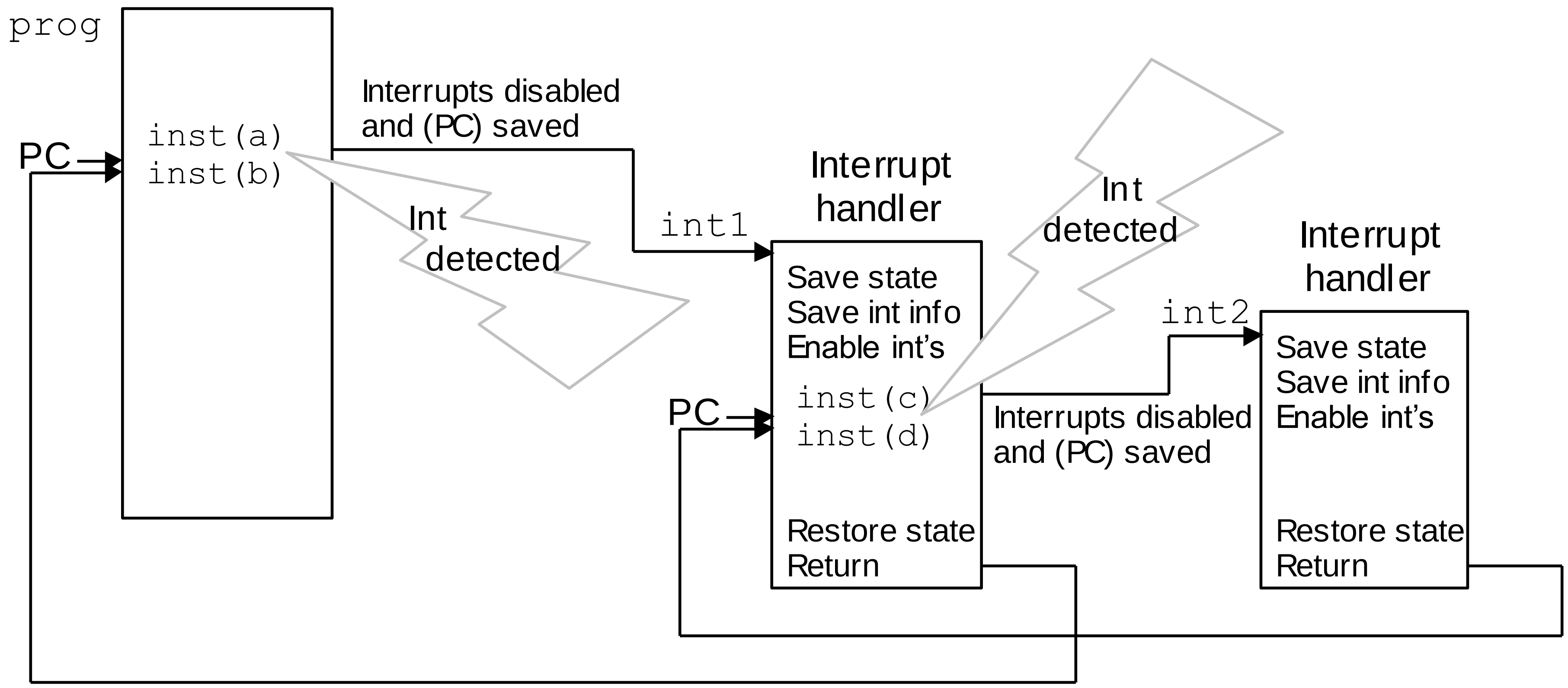
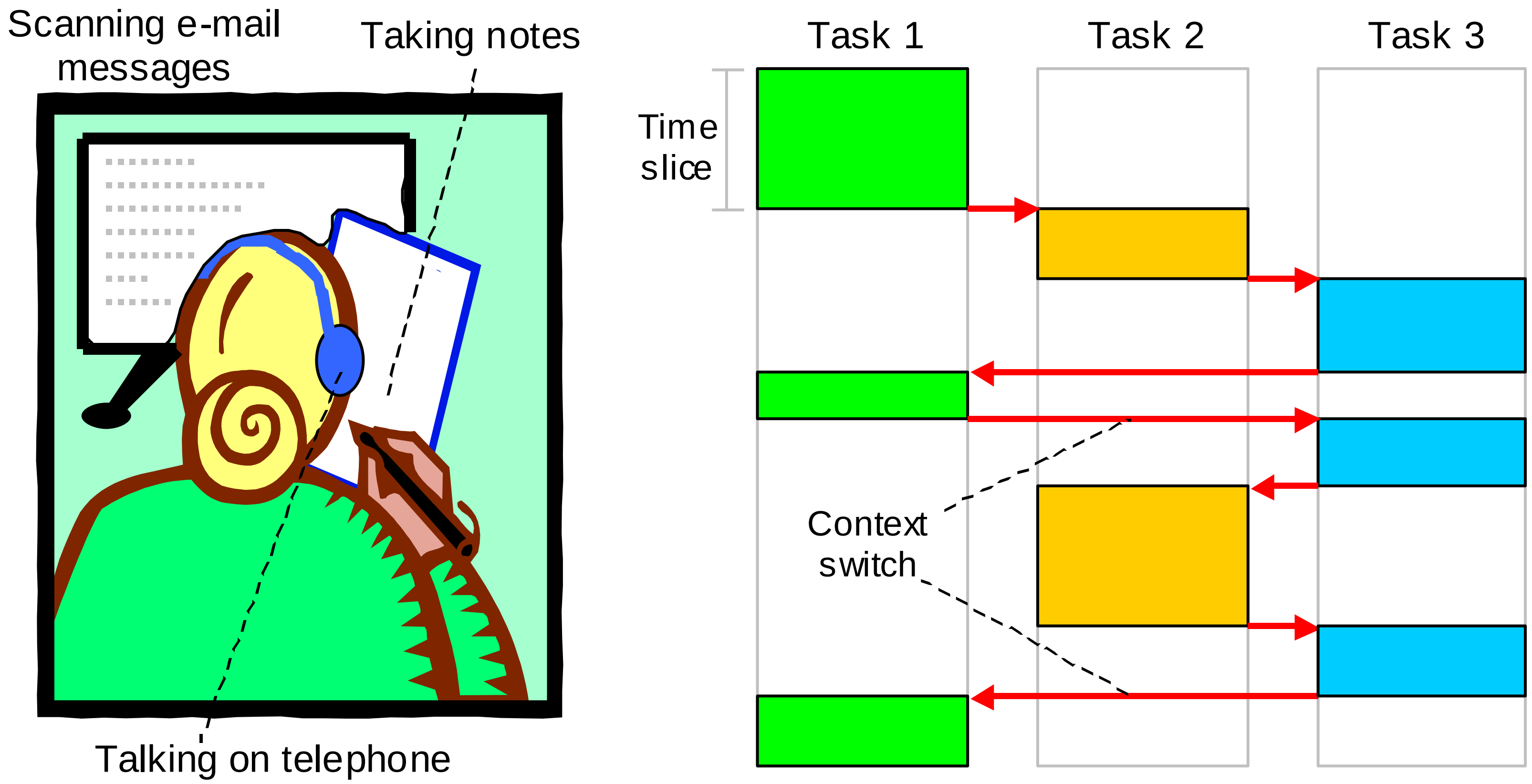


Figure 24.6 Example of nested interrupts.

24.5 Types of Context Switching

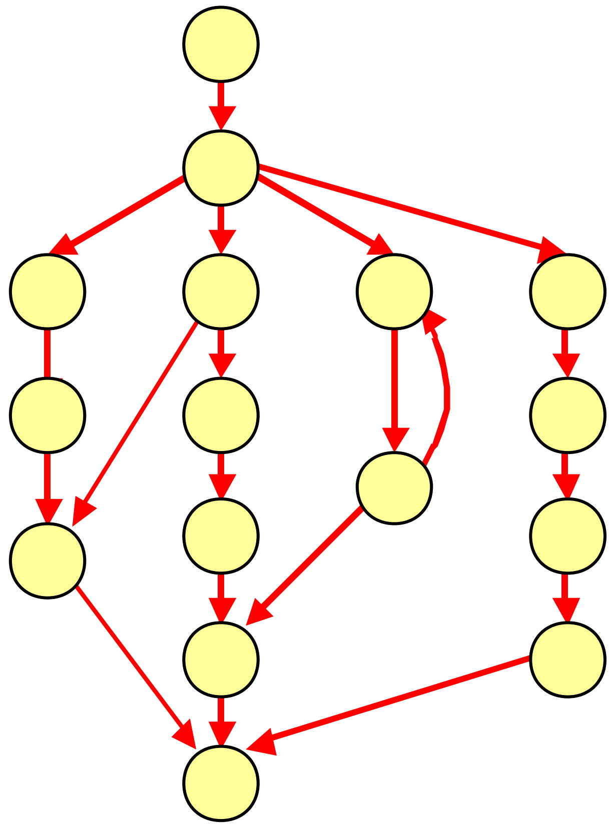


(a) Human multitasking

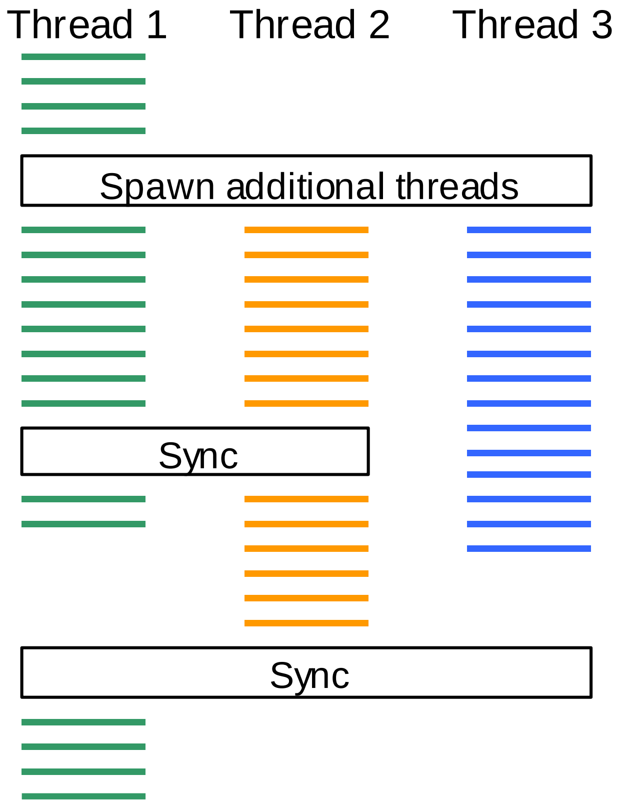
(b) Computer multitasking

Figure 24.7 Multitasking in humans and computers.

24.6 Threads and Multithreading



(a) Task graph of a program



(b) Thread structure of a task

Figure 24.8 A program divided into tasks (subcomputations) or threads.

Multithreaded Processors

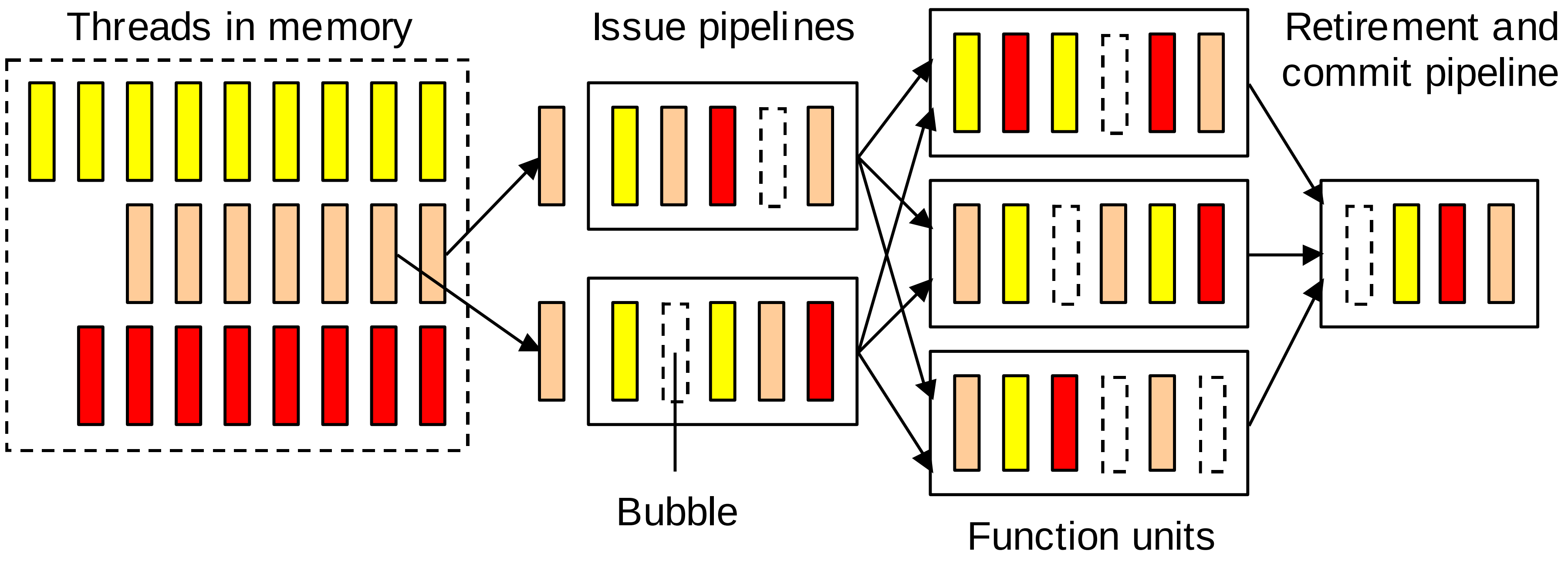


Figure 24.9 Instructions from multiple threads as they make their way through a processor's execution pipeline.