by the cost of manufacture and to maximize performance as measured by the speed of operation. There are some other performance- and cost-related constraints to satisfy such as high reliability, low power consumption, and compatibility with existing systems. These multiple objectives interact in poorly understood ways that depend on the complexity and novelty of the design.

Despite careful attention to detail and the assistance of CAD tools, the initial versions of a new system often fail to meet some design objective, sometimes in subtle and hard-to-detect ways. This failure can be attributed to incomplete specifications for the design (some mode of behavior was overlooked), errors made by human designers or their CAD tools (which are also ultimately due to human error), and unanticipated interactions between structure, performance, and cost. For example, increasing a system's speed to a desired level can make the cost unacceptably high.

The complexity of computer systems is such that the design problem must be broken down into smaller, easier tasks involving various classes of components. These smaller problems can then be solved independently by different designers or design teams. Each major design step is often implemented via the multistep or iterative process depicted by a flowchart in Figure 2.6. An initial design is created, perhaps in ad hoc fashion, by adapting an existing design of a similar system. The result is then evaluated to see if it meets the relevant design objectives. If not, the design is revised and the result reevaluated. Many iterations through the redesign and evaluation steps of Figure 2.6 may be necessary to obtain a satisfactory design.

Computer-aided design. The emergence of powerful and inexpensive desktop computers with good graphics interfaces provides designers with a range of programs to support their design tasks. CAD tools are used to automate, at least in

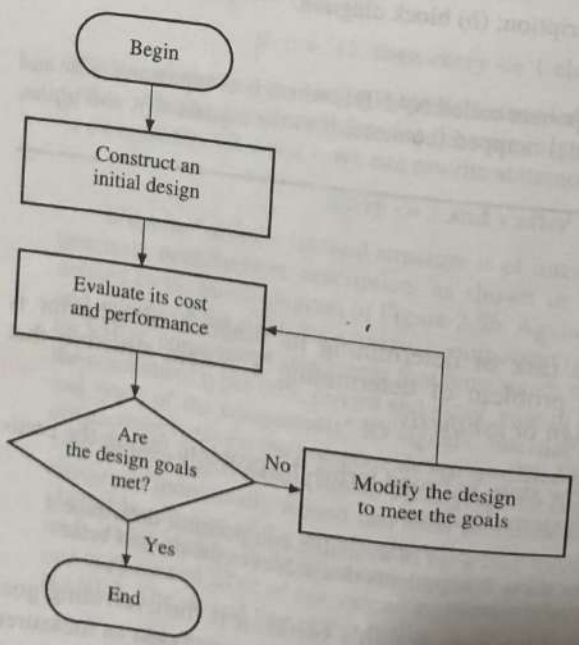


Figure 2.6
Flowchart of an iterative design process.

part, the more tedious design and evaluation steps and contribute in three important ways to the overall design process.

- CAD *editors* or *translators* convert design data into forms such as HDL descriptions or schematic diagrams, which humans, computers, or both can efficiently process.
- *Simulators* create computer models of a new design, which can mimic the design's behavior and help designers determine how well the design meets various performance and cost goals.
- *Synthesizers* automate the design process itself by deriving structures that implement all or part of some design step.

Editing is the easiest of these three tasks, and synthesis the most difficult. Some synthesis methods incorporate exact or *optimal* algorithms which, even if easy to program into CAD tools, often demand excessive amounts of computing resources. Many synthesis approaches are therefore based on trial-and-error methods and experience with earlier designs. These computationally efficient but inexact methods are called *heuristics* and form the basis of most practical CAD tools.

Design levels. The design of a complex system such as a computer is carried out at several levels of abstraction. Three such levels are generally recognized in computer design, although they are referred to by various different names in the literature:

- The *processor* level, also called the *architecture, behavior,* or *system* level.
- The *register* level, also called the *register-transfer level* (*RTL*).
- The *gate* level, also called the *logic* level.

As Figure 2.7 indicates we are naming each level for a key component treated as primitive or indivisible at that level of abstraction. The processor level corresponds to a user's or manager's view of a computer. The register level is approximately the level of detail seen by a programmer. The gate level is primarily the concern of the hardware designer. These three design levels also correspond roughly to the major subdivisions of integrated-circuit technology into VLSI, MSI, and SSI components. The boundaries between the levels are far from clear-cut, and it is common to encounter descriptions that mix components from more than one level.

Level	Components	IC density	Information units	Time units
Gate	Logic gates, flip-flops.	SSI	Bits	10^{-12} to 10^{-9} s
Register	Registers, counters, combinational circuits, small sequential circuits.	MSI	Words	10^{-9} to 10^{-6} s
Processor	CPUs, memories, IO devices.	VLSI	Blocks of words	10^{-3} to 10^{3} s

Figure 2.7
The major computer design levels.

A few basic component types from each design level are listed in Figure 2.7. The logic gates recognized as primitive at the gate level include AND, OR, NAND, NOR, and NOT gates. Consequently, the EXCLUSIVE-OR circuit of Figure 2.2 is an example of a gate-level circuit composed of five gates. The component marked XOR in Figure 2.5*b* performs the EXCLUSIVE-OR function and so can be thought of as a more abstract or higher-level view of the circuit of Figure 2.2, in which all internal structure has been abstracted away. Similarly, the half-adder block of Figure 2.4*b* represents a higher-level view of the three-component circuit of Figure 2.5*b*. We consider a half adder to be a register-level component. We might regard the circuit of Figure 2.5*b* as being at the register level also, but because NAND is another gate type and XOR is sometimes treated as a gate, this circuit can also be viewed as gate level.

Figure 2.7 indicates some further differences between the design levels. The units of information being processed increase in complexity as one goes from the gate to the processor level. At the gate level individual bits (0s and 1s) are processed. At the register level information is organized into multibit words or vectors, usually of a small number of standard types. Such words represent numbers, instructions, and the like. At the processor level the units of information are blocks of words, for example, a program or a data set. Another important difference lies in the time required for an elementary operation; successive levels can differ by several orders of magnitude in this parameter. At the gate level the time required to switch the output of a gate between 0 and 1 (the gate delay) serves as the time unit and typically is a nanosecond (ns) or less. A clock cycle of, say, 10 ns, is a commonly used unit of time at the register level. The time unit at the processor level might be a program's execution time, a quantity that can vary widely.

System hierarchy. It is customary to refer to a design level as high or low; the more complex the components, the higher the level. In this book we are primarily concerned with the two highest levels listed in Figure 2.7, the processor and register levels, which embrace what is generally regarded as computer architecture. The ordering of the levels suggested by the terms high and low is, in fact, quite strong. A component in any level L_i is equivalent to a (sub) system of components taken from the level L_{i-1} beneath it. This relationship is illustrated in Figure 2.8. Formally speaking, there is a one-to-one mapping h_i between components in L_i and disjoint subsystems in level L_{i-1}; a system with levels of this type is called a *hierarchical system.* Thus in Figure 2.8 the subsystem composed of blocks 1, 3, and 4 in the low-level description maps onto block *A* in the high-level description. Figures 2.4*b* and 2.5*b* show two hierarchical descriptions of a half-adder circuit.

Complex systems, both natural and artificial, tend to have a well-defined hierarchical organization. A profound explanation of this phenomenon has been given by Herbert A. Simon [Simon 1962]. The components of a hierarchical system at each level are self-contained and stable entities. The evolution of systems from simple to complex organizations is greatly helped by the existence of stable intermediate structures. Hierarchical organization also has important implications in the design of computer systems. It is perhaps most natural to proceed from higher to lower design levels because this sequence corresponds to a progression of successively greater levels of detail. Thus if a complex system is to be designed using small-scale ICs or a single IC composed of standard cells, the design process might consist of the following three steps.

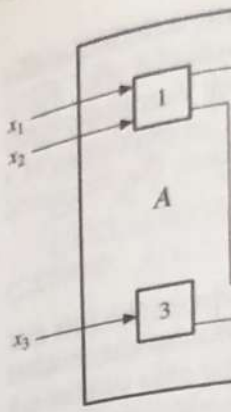


Figure 2.8
Two descriptions of

1. Specify the pro
2. Specify the reg
3. Specify the gat

This design appro
and software desi
ICs or standard c

As might be
ferent. Only in th
(Boolean algebra
puter design, but
on the designers'
at the register an
register level. We
gate-level design
Somenzi 1996],

2.1.3 The Gate

Gate-level (logic
sible values are r
are logic gates, v
which are bit-sto

Combination
Boolean function
variables onto th

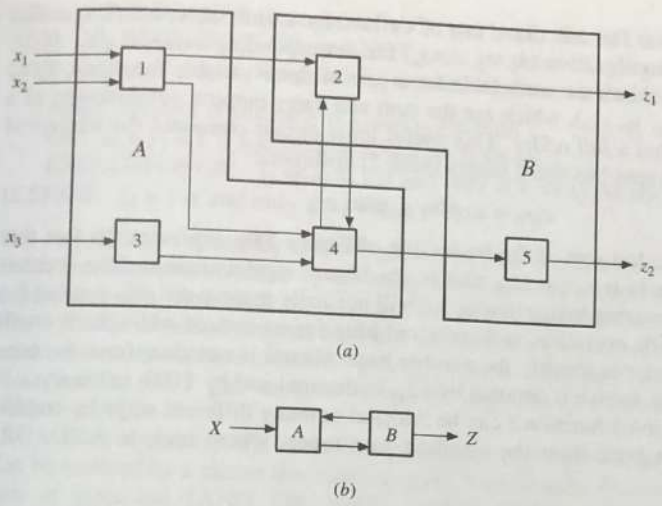


Figure 2.8
Two descriptions of a hierarchical system: (*a*) low level; (*b*) high level.

1. Specify the processor-level structure of the system.
2. Specify the register-level structure of each component type identified in step 1.
3. Specify the gate-level structure of each component type identified in step 2.

This design approach is termed *top down;* it is extensively used in both hardware and software design. If the foregoing system is to be designed using medium-scale ICs or standard cells, then the third step, gate-level design, is no longer needed.

As might be expected, the design problems arising at each level are quite different. Only in the case of gate-level design is there a substantial theoretical basis (Boolean algebra). The register and processor levels are of most interest in computer design, but unfortunately, design at these levels is largely an art that depends on the designers' skill and experience. In the following sections we examine design at the register and processor levels in detail, beginning with the better-understood register level. We assume that the reader is familiar with binary numbers and with gate-level design concepts [Armstrong and Gray 1993; Hayes 1993; Hachtel and Somenzi 1996], which we review in the next section.

2.1.3 The Gate Level

Gate-level (logic) design is concerned with processing binary variables whose possible values are restricted to the bits (binary digits) 0 and 1. The design components are logic gates, which are simple, memoryless processing elements, and flip-flops, which are bit-storage devices.

Combinational logic. *A combinational function,* also referred to as a *logic* or a *Boolean* function, is a mapping from the set of 2^n input combinations of n binary variables onto the output values 0 and 1. Such a function is denoted by $z(x_1, x_2, \ldots,$

Minimizing the number of gates in a sequential circuit is difficult because it is affected by the flip-flop types, the state assignment, and, of course, the way in which the combinational subcircuit C is designed. Other design techniques exist to simplify the design process at the expense of using more logic elements. It is impractical to deal with complete binary descriptions like state tables if they contain more than, say, a dozen states. Consequently, large, sequential circuits are designed by heuristic techniques whose implementations use reasonable but nonminimal amounts of hardware [Hayes 1993; Hachtel and Somenzi 1996]. These circuits are often best designed at the more abstract register level rather than the gate level.

2.2 THE REGISTER LEVEL

At the register or register-transfer level, related information bits are grouped into ordered sets called *words* or *vectors*. The primitive components are small combinational or sequential circuits intended to process or store words.

2.2.1 Register-Level Components

Register-level circuits are composed of word-oriented devices, the more important of which are listed in Figure 2.15. The key sequential component, which gives this level of abstraction its name, is a (parallel) *register*, a storage device for words. Other common sequential elements are shift registers and counters. A number of standard combinational components exist, ranging from general-purpose devices, such as word gates, to more specialized circuits, such as decoders and adders.

Type	Component	Functions
Combinational	Word gates.	Logical (Boolean) operations.
	Multiplexers.	Data routing; general combinational functions.
	Decoders and encoders.	Code checking and conversion.
	Adders.	Addition and subtraction.
	Arithmetic-logic units.	Numerical and logical operations.
	Programmable logic devices.	General combinational functions.
Sequential	(Parallel) registers.	Information storage.
	Shift registers.	Information storage; serial-parallel conversion.
	Counters.	Control/timing signal generation.
	Programmable logic devices.	General sequential functions.

Figure 2.15
The major component types at the register level.

Register-level components are linked to form circuits by means of word-carrying groups of lines, referred to as *buses*.

Types. The component types of Figure 2.15 are generally useful in register-level design; they are available as MSI parts in various IC series and as standard cells in VLSI design libraries. However, they cannot be identified a priori based on some property analogous to the functional completeness of gate-level operations. For example, we will show that multiplexers can realize any combinational function. This completeness property is incidental to the main application of multiplexers, which is signal selection or path switching.

There are no universally accepted graphic symbols for register-level components. They are usually represented in circuit diagrams by blocks containing an abbreviated description of their behavior, as in Figure 2.16. A single signal line in a diagram can represent a bus transmitting $m > 1$ bits of information in parallel; m is indicated explicitly by placing a slash (/) in the line and writing m next to it (see Figure 2.16). A components's IO lines are often separated into data and control lines. An m-bit bus may be given a name that identifies the bus's role, for example, the type of data transmitted over a data bus. A control line's name indicates the operation determined by the line in its *active*, *enabled*, or *asserted* state. Unless otherwise indicated, the active state of a bus occurs when its lines assume the logical 1 value. A small circle representing inversion is placed at an input or output port of a block to indicate that the corresponding lines are active in the 0 state and inactive in the 1 state. Alternatively, the name of a signal whose active value is 0 includes an overbar.

The input control lines associated with a multifunction block fall into two broad categories: *select* lines, which specify one of several possible operations that the unit is to perform, and *enable* lines, which specify the time or condition for a selected operation to be performed. Thus in Figure 2.16, to perform some operation F_1, first set the select line F to a bit pattern denoting F_1 and then activate the edge-triggered enable line E by applying a 0-to-1 edge signal. Enable lines are often connected to clock sources. The output control signals, if any, indicate when or how the unit completes its processing. Figure 2.16 indicates termination by $\bar{S} = 0$. The arrowheads are omitted when we can infer signal direction from the circuit structure or signal names.

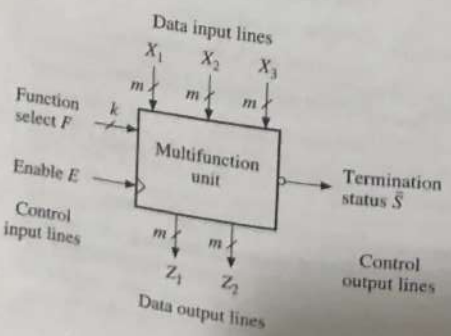


Figure 2.16
Generic block representation of a register-level component.

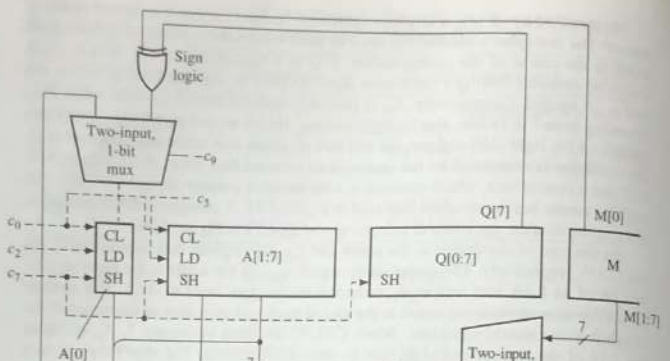


Figure 2.44
Implementation of some control points of *multiplier8*.

signals for the multiplier. In some cases several control signals implement a particular operation. For instance, the add operation employs c_6 to select the adder's right input operand, c_9 to select c_{OUT} for loading into $A[0]$, and c_2 and c_5 to actually load the 8-bit sum into $A[0{:}7]$. The number of distinguished control signals will vary with the details of the logic used to implement the control unit. Figure 2.44 shows a straightforward implementation of the control logic associated with the accumulator and adder subcircuits using the control signals defined in Figure 2.43.

2.3 THE PROCESSOR LEVEL

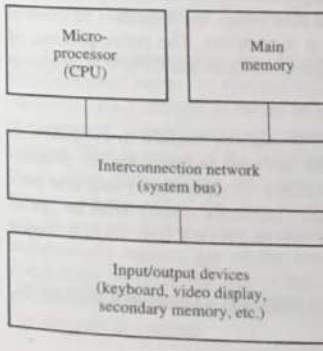
The processor or system level is the highest in the computer design hierarchy. It is concerned with the storage and processing of blocks of information such as programs and data files. The components at this level are complex, usually sequential, circuits that are based on VLSI technology. Processor-level design is very much a heuristic process, as there is little design theory at this level of abstraction.

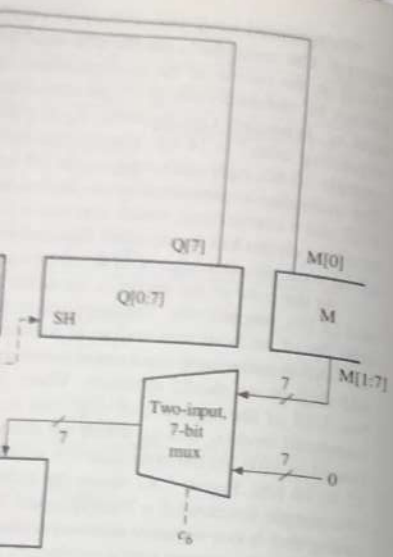
2.3.1 Processor-Level Components

The component types recognized at the
processors, memories, IO devices, and
In this section we give only a brief sur
level components; they are examined
later chapters.

Central processing unit. We define
tion-set processor that has overall res
execution in a computer system. The qu
from other, more specialized processo
functions are restricted. An instruction
that it operates on word-organized i
obtains from an external memory that
sor. Most contemporary CPUs are mi
implementation is a single VLSI chip.

Figure 2.46 shows the essential in
level. The CPU contains the logic nec
and is divided into datapath and contr
ates the addresses of instructions and
ticular system a cache memory is inter
CPU. The cache is a fast buffer memo
system's address space; it is often pl
the CPU. Each memory request genera
If the required information is not curr
directed to M and the cache is autom
instructions from the cache or M and
needed for their execution. The CPU
circuits that execute most instruction
data storage. The CPU manages a sy
link among the CPU-cache subsystem





lier8.

everal control signals implement a particular
employs c_6 to select the adder's right input
A[0], and c_2 and c_5 to actually load the 8-bit
ed control signals will vary with the details
unit. Figure 2.44 shows a straightforward
ted with the accumulator and adder subcir-
gure 2.43.

n the computer design hierarchy. It is
f blocks of information such as pro-
evel are complex, usually sequential,
rocessor-level design is very much a
y at this level of abstraction.

2.3.1 Processor-Level Components

The component types recognized at the processor level fall into four main groups: processors, memories, IO devices, and interconnection networks; see Figure 2.45. In this section we give only a brief summary of the characteristics of processor-level components; they are examined individually and in much greater depth in later chapters.

Central processing unit. We define a CPU to be a general-purpose, instruction-set processor that has overall responsibility for program interpretation and execution in a computer system. The qualifier *general-purpose* distinguishes CPUs from other, more specialized processors, such as IO processors (IOPs), whose functions are restricted. An instruction-set processor is characterized by the fact that it operates on word-organized instructions and data, which the processor obtains from an external memory that also stores results computed by the processor. Most contemporary CPUs are microprocessors, implying that their physical implementation is a single VLSI chip.

Figure 2.46 shows the essential internal organization of a CPU at the register level. The CPU contains the logic needed to execute its particular instruction set and is divided into datapath and control units. The control part (the I-unit) generates the addresses of instructions and data stored in external memory. In this particular system a cache memory is interposed between the main memory M and the CPU. The cache is a fast buffer memory designed to hold an active portion of the system's address space; it is often placed, wholly or in part, on the same IC as the CPU. Each memory request generated by the CPU is first directed to the cache. If the required information is not currently assigned to the cache, the request is redirected to M and the cache is automatically updated from M. The I-unit fetches instructions from the cache or M and decodes them to derive the control signals needed for their execution. The CPU's datapath (E-unit) has the arithmetic-logic circuits that execute most instructions; it also has a set of registers for temporary data storage. The CPU manages a system bus, which is the main communication link among the CPU-cache subsystem, main memory, and the IO devices.

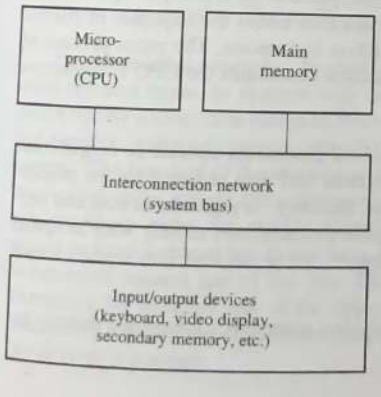


Figure 2.45
Major components of a computer system.

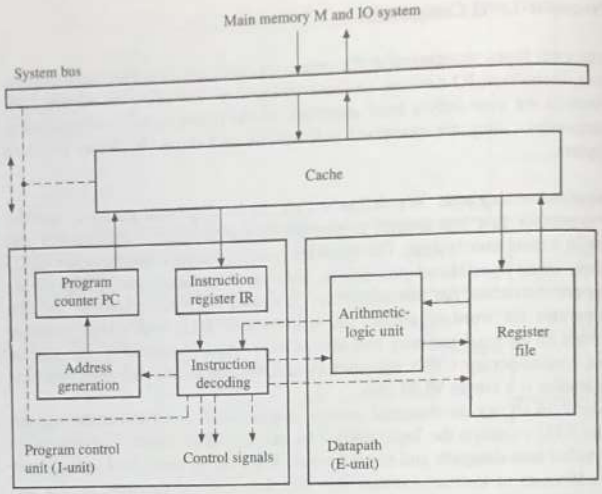


Figure 2.46
Internal organization of a CPU and cache memory.

The CPU is a synchronous sequential circuit whose clock period is the computer's basic unit of time. In one clock cycle the CPU can perform a register-transfer operation, such as fetching an instruction word from M via the system bus and loading it into the instruction register IR. This operation can be expressed formally by

$$IR := M(PC);$$

where PC is the program counter the CPU uses to hold the expected address of the next instruction word. Once in the I-unit, an instruction is decoded to determine the actions needed for its execution; for example, perform an arithmetic operation on data words stored in CPU registers. The I-unit then issues the sequence of control signals that enables execution of the instruction in question. The entire process of fetching, decoding, and executing an instruction constitutes the CPU's *instruction cycle*.

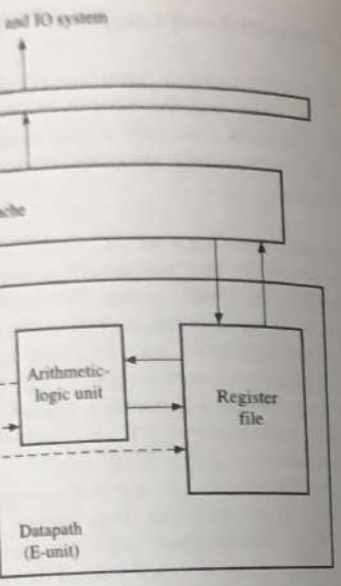
Memories. CPUs and other instruction-set processors operate in conjunction with external memories that store the programs and data required by the processors. Numerous memory technologies exist, and they vary greatly in cost and performance. The cost of a memory device generally increases rapidly with its speed of operation. The memory part of a computer can be divided into several major subsystems:

1. Main memory M, consisting of relatively fast storage ICs connected directly to, and controlled by, the CPU.
2. Secondary memory, consisting of less
storage capacity. These devices often
much slower than M. They are gener
CPU and form part of the computer's I
3. Many computers have a third type of
tioned between the CPU and main m
reduce the average time taken by the
or all of the cache may be integrated o

Main memory M is a word-organi
(RAM). The term *random access* stems f
location in M is the same. Random acce
memory access times vary with the loca
ries are slower and less expensive than
use some form of serial access. Because
access mode, the manner in which the st
memories is more complex than the si
Caches also use random access or an ev
associative or content addressing. Mer
stored information are covered in Chapt

IO devices. Input-output devices a
nicates with the outside world. A prim
transducers, that is, to convert inform
another. Unlike processors, IO devic
meaning of the data on which they a
within a computer system in the form
devices transform other forms of info
Figure 2.47 lists some widely used I
involve. Many of these devices use
speed of operation is slow compared
Although the CPU can take direct c
immediate control of a special-purpo
flow of information between the IO
systems is considered in Chapter 7.

Interconnection networks. Proce
word-oriented buses. In systems wit
controlled by a subsystem called an *i*
ing network, communications contro
context. The function of the intercon
munication paths among the compo
reasons, these paths are usually sh
access and use a shared bus at any t
components request use of the bus
contention by selecting one of the
connecting it to the bus. The interco
ing devices in a queue.



2. Secondary memory, consisting of less expensive devices that have very high storage capacity. These devices often involve mechanical motion and so are much slower than M. They are generally connected indirectly (via M) to the CPU and form part of the computer's IO system.
3. Many computers have a third type of memory called a cache, which is positioned between the CPU and main memory. The cache is intended to further reduce the average time taken by the CPU to access the memory system. Some or all of the cache may be integrated on the same IC chip as the CPU itself.

Main memory M is a word-organized addressable *random-access memory* (RAM). The term *random access* stems from the fact that the access time for every location in M is the same. Random access is contrasted with *serial access*, where memory access times vary with the location being accessed. Serial access memories are slower and less expensive than RAMs; most secondary-memory devices use some form of serial access. Because of their lower operating speeds and serial-access mode, the manner in which the stored information is organized in secondary memories is more complex than the simple word organization of main memory. Caches also use random access or an even faster memory-accessing method called associative or content addressing. Memory technologies and the organization of stored information are covered in Chapter 6.

IO devices. Input-output devices are the means by which a computer communicates with the outside world. A primary function of IO devices is to act as data transducers, that is, to convert information from one physical representation to another. Unlike processors, IO devices do not alter the information content or meaning of the data on which they act. Since data is transferred and processed within a computer system in the form of digital electrical signals, input (output) devices transform other forms of information to (from) digital electrical signals. Figure 2.47 lists some widely used IO devices and the information media they involve. Many of these devices use electromechanical technologies; hence their speed of operation is slow compared with processor and main-memory speeds. Although the CPU can take direct control of an IO device it is often under the immediate control of a special-purpose processor or control unit that directs the flow of information between the IO device and main memory. The design of IO systems is considered in Chapter 7.

Interconnection networks. Processor-level components communicate by word-oriented buses. In systems with many components, communication may be controlled by a subsystem called an *interconnection network;* terms such as *switching network, communications controller,* and *bus controller* are also used in this context. The function of the interconnection network is to establish dynamic communication paths among the components via the buses under its control. For cost reasons, these paths are usually shared. Only two communicating devices can access and use a shared bus at any time, so contention results when several system components request use of the bus. The interconnection network resolves such contention by selecting one of the requesting devices on some priority basis and connecting it to the bus. The interconnection network may place the other requesting devices in a queue.