

discuss multicore computers in Chapter 18.

Comparing Figures 17.7j and 17.7k, we see that a chip multiprocessor with the same instruction issue capability as an SMT cannot achieve the same degree of instruction-level parallelism. This is because the chip multiprocessor is not able to hide latencies by issuing instructions from other threads. On the other hand, the chip multiprocessor should outperform a superscalar processor with the same instruction issue capability, because the horizontal losses will be greater for the superscalar processor. In addition, it is possible to use multithreading within each of the cores on a chip multiprocessor, and this is done on some contemporary machines.

## 17.5 CLUSTERS

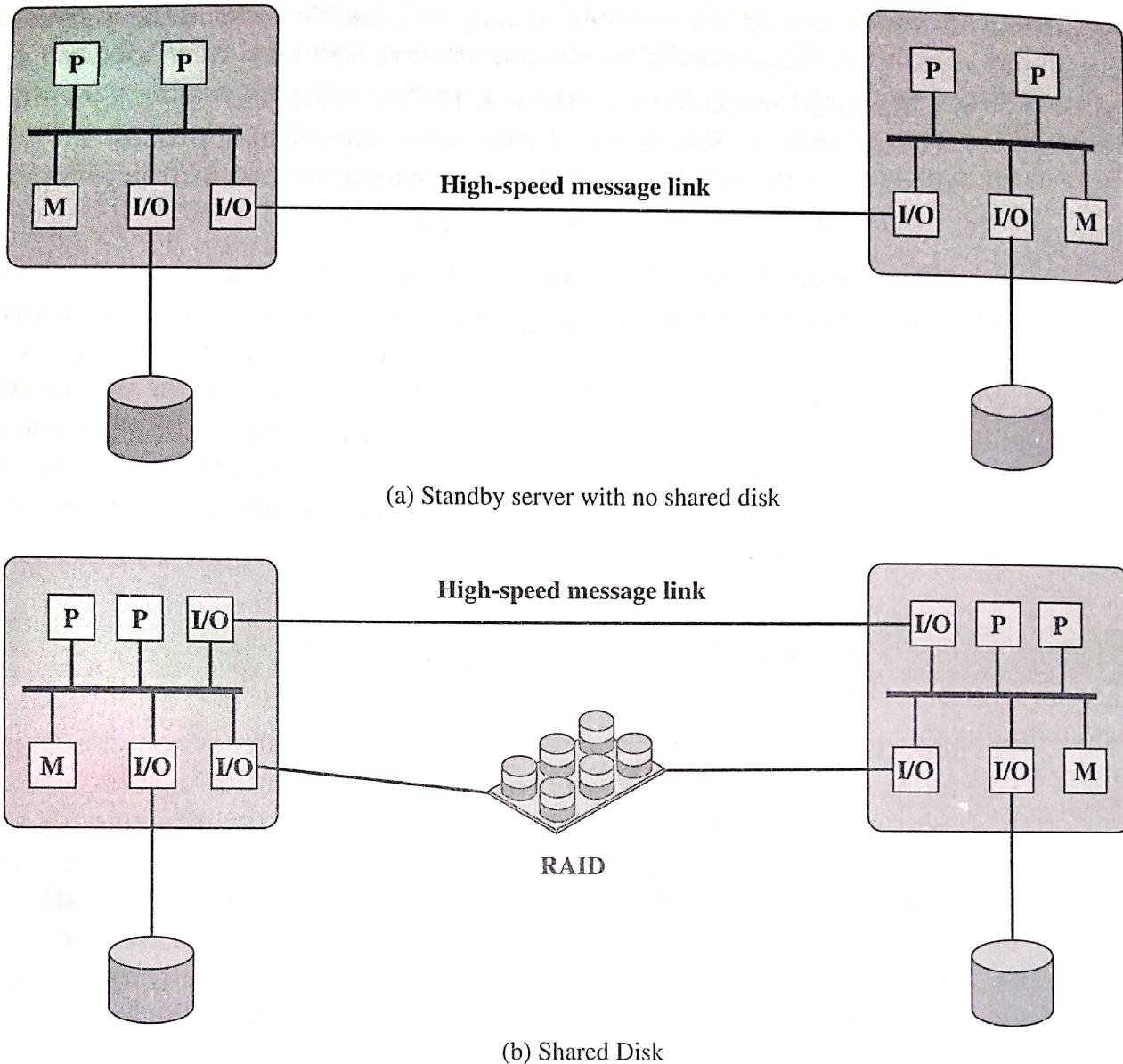
An important and relatively recent development computer system design is clustering. Clustering is an alternative to symmetric multiprocessing as an approach to providing high performance and high availability and is particularly attractive for server applications. We can define a cluster as a group of interconnected, whole computers working together as a unified computing resource that can create the illusion of being one machine. The term *whole computer* means a system that can run on its own, apart from the cluster; in the literature, each computer in a cluster is typically referred to as a *node*.

[BREW97] lists four benefits that can be achieved with clustering. These can also be thought of as objectives or design requirements:

- **Absolute scalability:** It is possible to create large clusters that far surpass the power of even the largest standalone machines. A cluster can have tens, hundreds, or even thousands of machines, each of which is a multiprocessor.
- **Incremental scalability:** A cluster is configured in such a way that it is possible to add new systems to the cluster in small increments. Thus, a user can start out with a modest system and expand it as needs grow, without having to go through a major upgrade in which an existing small system is replaced with a larger system.
- **High availability:** Because each node in a cluster is a standalone computer, the failure of one node does not mean loss of service. In many products, fault tolerance is handled automatically in software.
- **Superior price/performance:** By using commodity building blocks, it is possible to put together a cluster with equal or greater computing power than a single large machine, at much lower cost.

### Cluster Configurations

In the literature, clusters are classified in a number of different ways. Perhaps the simplest classification is based on whether the computers in a cluster share access to the same disks. Figure 17.8a shows a two-node cluster in which the only interconnection

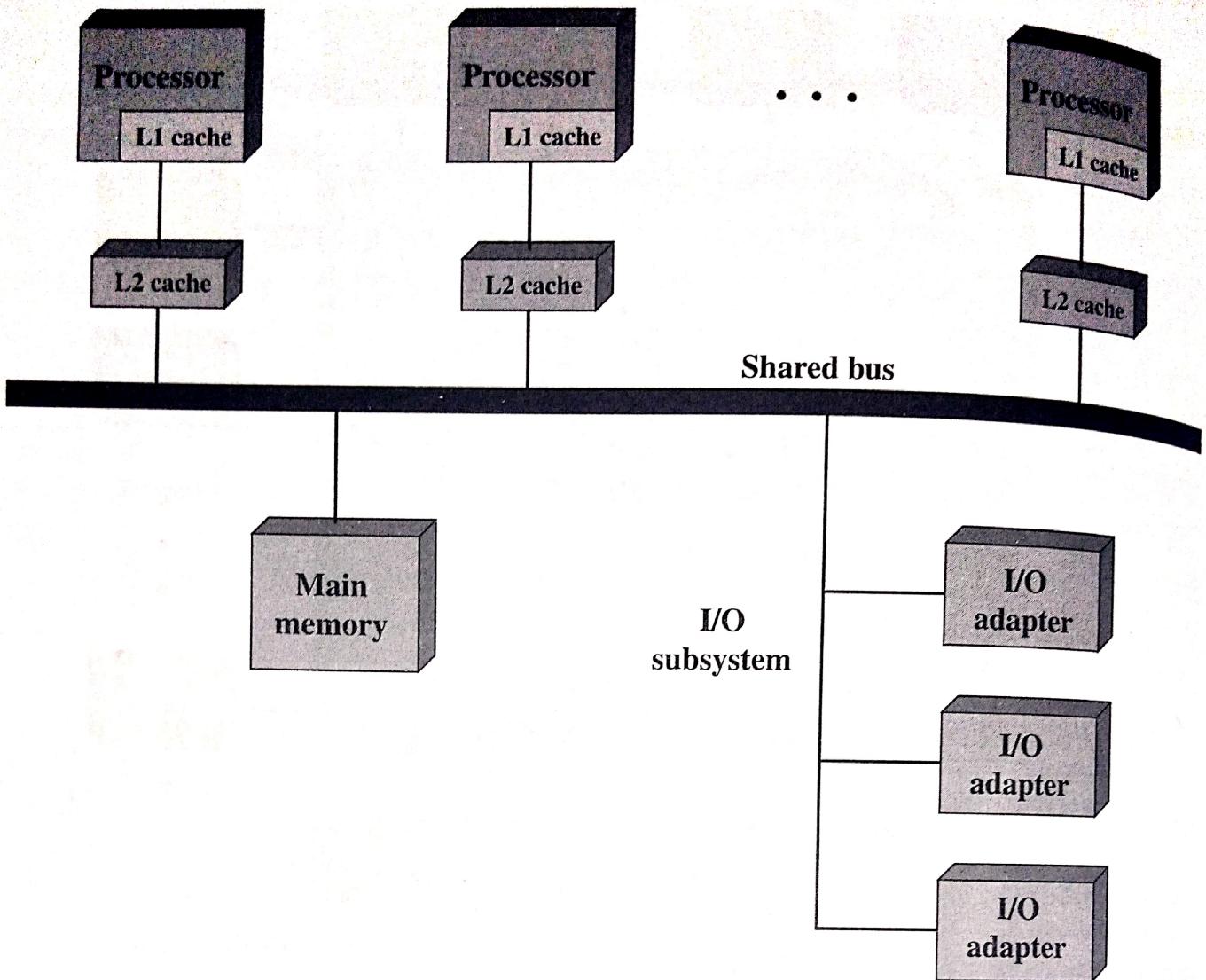


**Figure 17.8** Cluster Configurations

is by means of a high-speed link that can be used for message exchange to coordinate cluster activity. The link can be a LAN that is shared with other computers that are not part of the cluster or the link can be a dedicated interconnection facility. In the latter case, one or more of the computers in the cluster will have a link to a LAN or WAN so that there is a connection between the server cluster and remote client systems. Note that in the figure, each computer is depicted as being a multiprocessor. This is not necessary but does enhance both performance and availability.

In the simple classification depicted in Figure 17.8, the other alternative is a shared-disk cluster. In this case, there generally is still a message link between nodes. In addition, there is a disk subsystem that is directly linked to multiple computers within the cluster. In this figure, the common disk subsystem is a RAID system. The use of RAID or some similar redundant disk technology is common in clusters so that the high availability achieved by the presence of multiple computers is not compromised by a shared disk that is a single point of failure.

A clearer picture of the range of cluster options can be gained by looking at functional alternatives. Table 17.2 provides a useful classification along functional



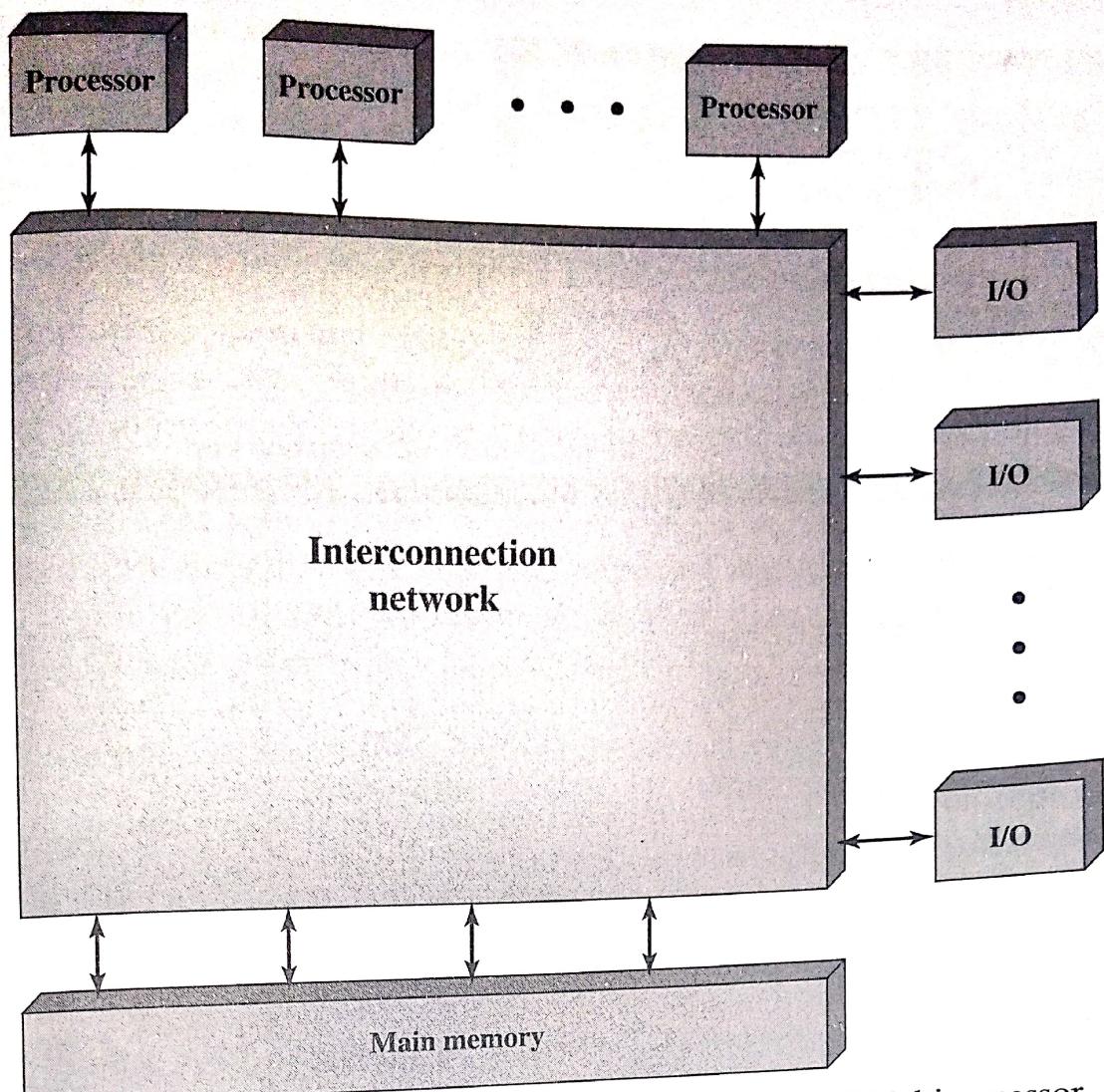
**Figure 17.5** Symmetric Multiprocessor Organization

The bus organization has several attractive features:

- **Simplicity:** This is the simplest approach to multiprocessor organization. The physical interface and the addressing, arbitration, and time-sharing logic of each processor remain the same as in a single-processor system.
- **Flexibility:** It is generally easy to expand the system by attaching more processors to the bus.
- **Reliability:** The bus is essentially a passive medium, and the failure of any attached device should not cause failure of the whole system.

The main drawback to the bus organization is performance. All memory references pass through the common bus. Thus, the bus cycle time limits the speed of the system. To improve performance, it is desirable to equip each processor with a cache memory. This should reduce the number of bus accesses dramatically. Typically, workstation and PC SMPs have two levels of cache, with the L1 cache internal (same chip as the processor) and the L2 cache either internal or external. Some processors now employ a L3 cache as well.

The use of caches introduces some new design considerations. Because each local cache contains an image of a portion of memory, if a word is altered in one



**Figure 17.4** Generic Block Diagram of a Tightly Coupled Multiprocessor

so that multiple simultaneous accesses to separate blocks of memory are possible. In some configurations, each processor may also have its own private main memory and I/O channels in addition to the shared resources.

The most common organization for personal computers, workstations, and servers is the time-shared bus. The time-shared bus is the simplest mechanism for constructing a multiprocessor system (Figure 17.5). The structure and interfaces are basically the same as for a single-processor system that uses a bus interconnection. The bus consists of control, address, and data lines. To facilitate DMA transfers from I/O subsystems to processors, the following features are provided:

- **Addressing:** It must be possible to distinguish modules on the bus to determine the source and destination of data.
- **Arbitration:** Any I/O module can temporarily function as “master.” A mechanism is provided to arbitrate competing requests for bus control, using some sort of priority scheme.
- **Time-sharing:** When one module is controlling the bus, other modules are locked out and must, if necessary, suspend operation until bus access is achieved.

These uniprocessor features are directly usable in an SMP organization. In this latter case, there are now multiple processors as well as multiple I/O processors all attempting to gain access to one or more memory modules via the bus.

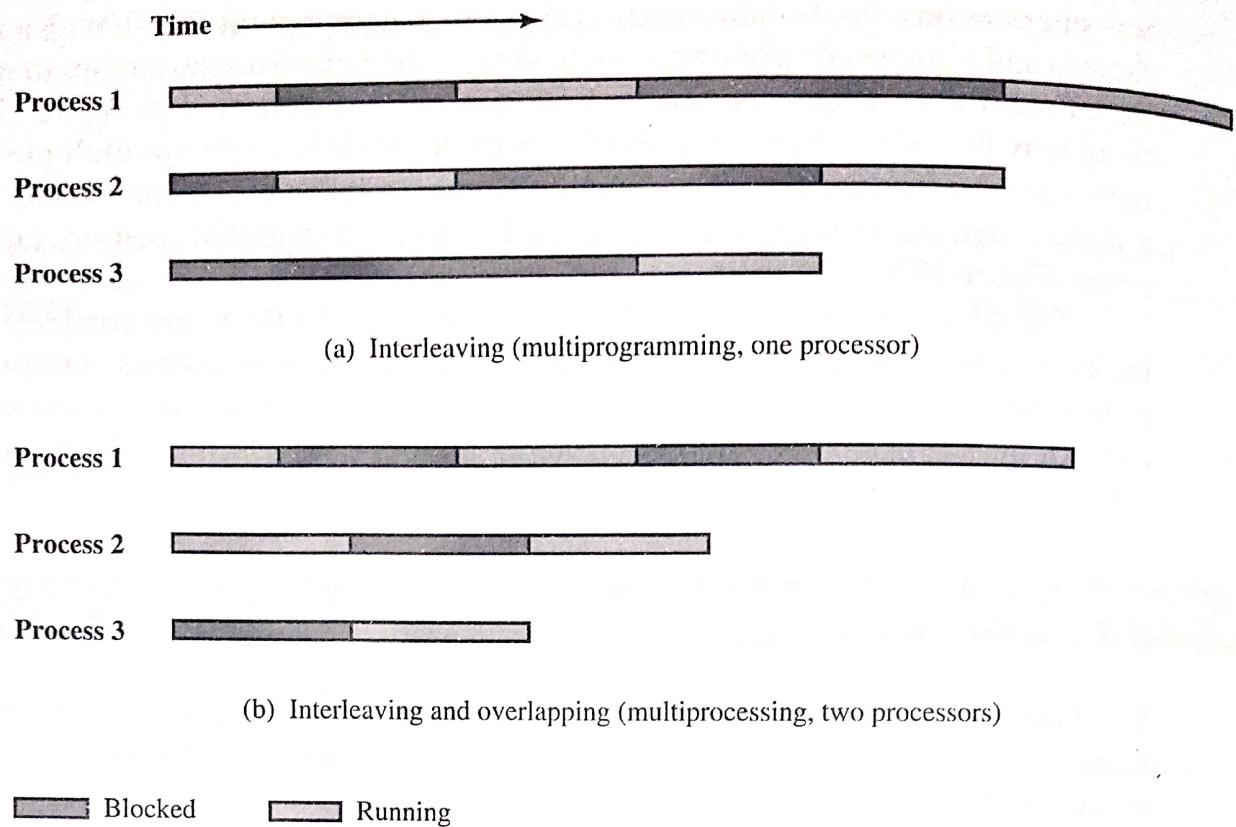


Figure 17.3 Multiprogramming and Multiprocessing

- **Availability:** In a symmetric multiprocessor, because all processors can perform the same functions, the failure of a single processor does not halt the machine. Instead, the system can continue to function at reduced performance.
- **Incremental growth:** A user can enhance the performance of a system by adding an additional processor.
- **Scaling:** Vendors can offer a range of products with different price and performance characteristics based on the number of processors configured in the system.

It is important to note that these are potential, rather than guaranteed, benefits. The operating system must provide tools and functions to exploit the parallelism in an SMP system.

An attractive feature of an SMP is that the existence of multiple processors is transparent to the user. The operating system takes care of scheduling of threads or processes on individual processors and of synchronization among processors.

## Organization

Figure 17.4 depicts in general terms the organization of a multiprocessor system. There are two or more processors. Each processor is self-contained, including a control unit, ALU, registers, and, typically, one or more levels of cache. Each processor has access to a shared main memory and the I/O devices through some form of interconnection mechanism. The processors can communicate with each other through memory (messages and status information left in common data areas). It may also be possible for processors to exchange signals directly. The memory is often organized

unit operates on a single data stream (DS) from a memory unit (MU). With an SIMD, there is still a single control unit, now feeding a single instruction stream to multiple PUs. Each PU may have its own dedicated memory (illustrated in Figure 17.2b), or there may be a shared memory. Finally, with the MIMD, there are multiple control units, each feeding a separate instruction stream to its own PU. The MIMD may be a shared-memory multiprocessor (Figure 17.2c) or a distributed-memory multicomputer (Figure 17.2d).

The design issues relating to SMPs, clusters, and NUMAs are complex, involving issues relating to physical organization, interconnection structures, interprocessor communication, operating system design, and application software techniques. Our concern here is primarily with organization, although we touch briefly on operating system design issues.

## 17.2 SYMMETRIC MULTIPROCESSORS

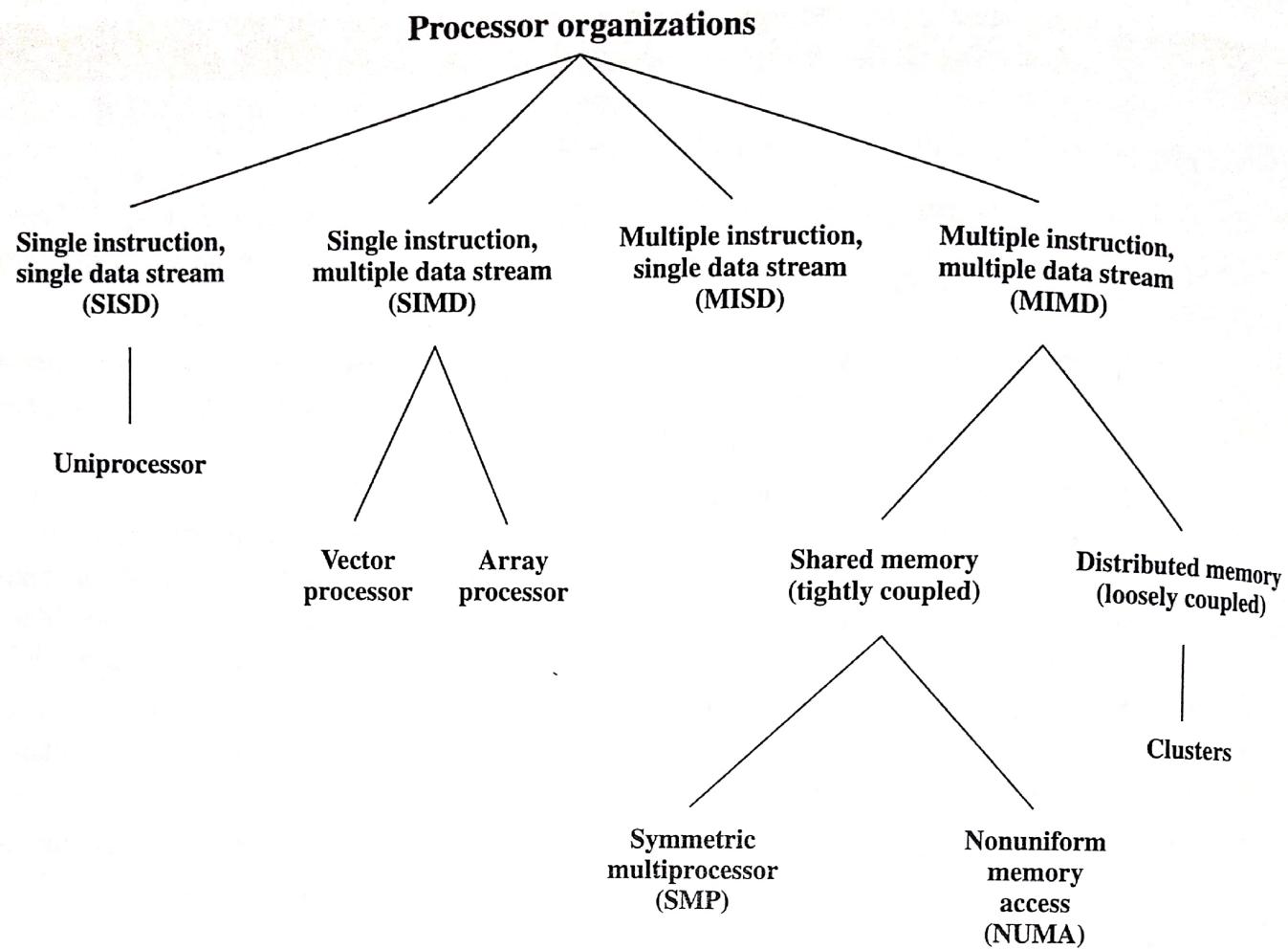
Until fairly recently, virtually all single-user personal computers and most workstations contained a single general-purpose microprocessor. As demands for performance increase and as the cost of microprocessors continues to drop, vendors have introduced systems with an SMP organization. The term *SMP* refers to a computer hardware architecture and also to the operating system behavior that reflects that architecture. An SMP can be defined as a standalone computer system with the following characteristics:

1. There are two or more similar processors of comparable capability.
2. These processors share the same main memory and I/O facilities and are interconnected by a bus or other internal connection scheme, such that memory access time is approximately the same for each processor.
3. All processors share access to I/O devices, either through the same channels or through different channels that provide paths to the same device.
4. All processors can perform the same functions (hence the term *symmetric*).
5. The system is controlled by an integrated operating system that provides interaction between processors and their programs at the job, task, file, and data element levels.

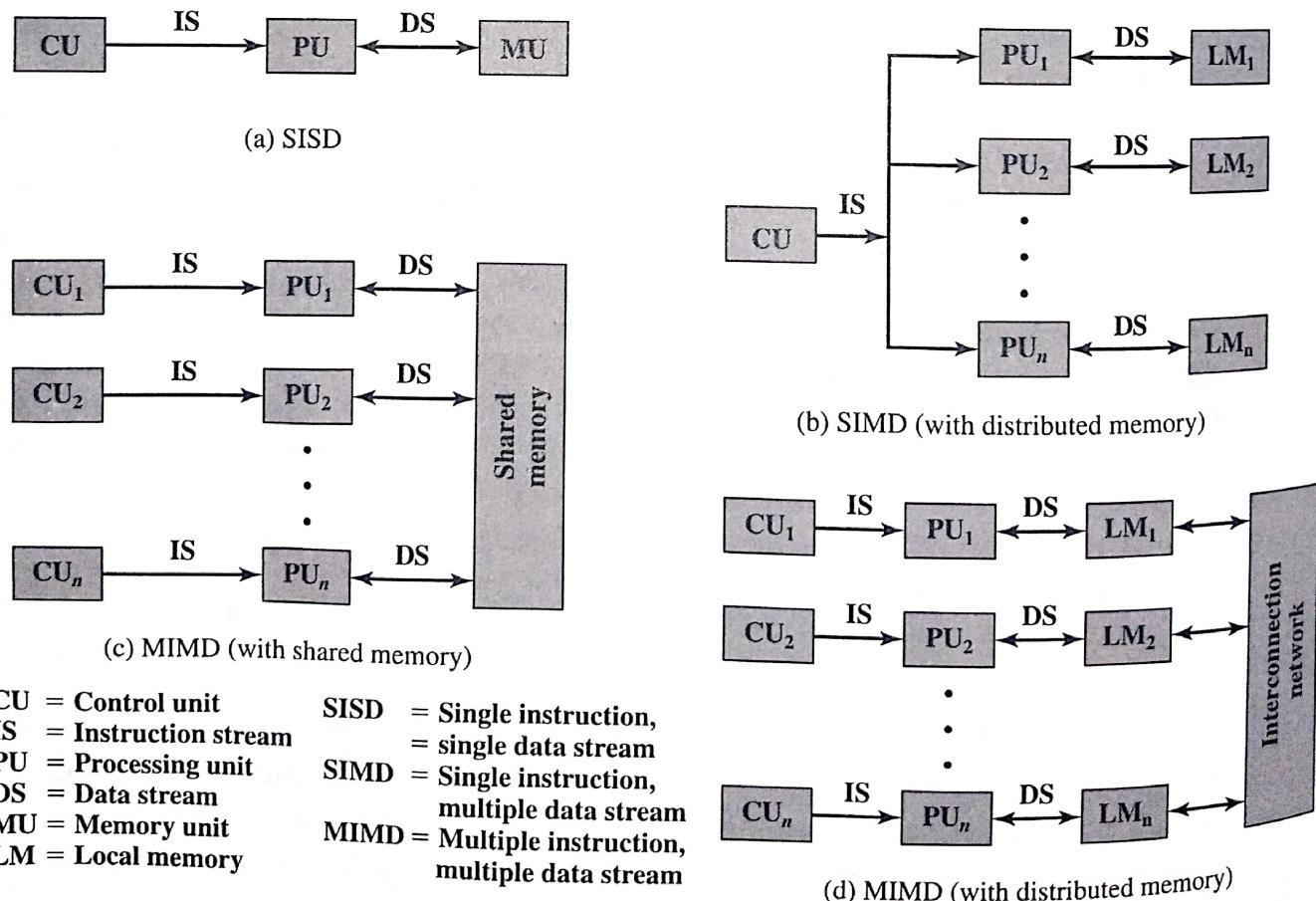
Points 1 to 4 should be self-explanatory. Point 5 illustrates one of the contrasts with a loosely coupled multiprocessing system, such as a cluster. In the latter, the physical unit of interaction is usually a message or complete file. In an SMP, individual data elements can constitute the level of interaction, and there can be a high degree of cooperation between processes.

The operating system of an SMP schedules processes or threads across all of the processors. An SMP organization has a number of potential advantages over a uniprocessor organization, including the following:

- **Performance:** If the work to be done by a computer can be organized so that some portions of the work can be done in parallel, then a system with multiple processors will yield greater performance than one with a single processor of the same type (Figure 17.3).



**Figure 17.1** A Taxonomy of Parallel Processor Architectures



**Figure 17.2** Alternative Computer Organizations

## 17.1 MULTIPLE PROCESSOR ORGANIZATIONS

### Types of Parallel Processor Systems

A taxonomy first introduced by Flynn [FLYN72] is still the most common way of categorizing systems with parallel processing capability. Flynn proposed the following categories of computer systems:

- **Single instruction, single data (SISD) stream:** A single processor executes a single instruction stream to operate on data stored in a single memory. Uniprocessors fall into this category.
- **Single instruction, multiple data (SIMD) stream:** A single machine instruction controls the simultaneous execution of a number of processing elements on a lockstep basis. Each processing element has an associated data memory, so that instructions are executed on different sets of data by different processors. Vector and array processors fall into this category, and are discussed in Section 18.7.
- **Multiple instruction, single data (MISD) stream:** A sequence of data is transmitted to a set of processors, each of which executes a different instruction sequence. This structure is not commercially implemented.
- **Multiple instruction, multiple data (MIMD) stream:** A set of processors simultaneously execute different instruction sequences on different data sets. SMPs, clusters, and NUMA systems fit into this category.

With the MIMD organization, the processors are general purpose; each is able to process all of the instructions necessary to perform the appropriate data transformation. MIMDs can be further subdivided by the means in which the processors communicate (Figure 17.1). If the processors share a common memory, then each processor accesses programs and data stored in the shared memory, and processors communicate with each other via that memory. The most common form of such system is known as a **symmetric multiprocessor (SMP)**, which we examine in Section 17.2. In an SMP, multiple processors share a single memory or pool of memory by means of a shared bus or other interconnection mechanism; a distinguishing feature is that the memory access time to any region of memory is approximately the same for each processor. A more recent development is the **nonuniform memory access (NUMA)** organization, which is described in Section 17.5. As the name suggests, the memory access time to different regions of memory may differ for a NUMA processor.

A collection of independent uniprocessors or SMPs may be interconnected to form a **cluster**. Communication among the computers is either via fixed paths or via some network facility.

### Parallel Organizations

Figure 17.2 illustrates the general organization of the taxonomy of Figure 17.1. Figure 17.2a shows the structure of an SISD. There is some sort of control unit (CU) that provides an instruction stream (IS) to a processing unit (PU). The processing

## 13.1 ADDRESSING MODES

The address field or fields in a typical instruction format are relatively small. We would like to be able to reference a large range of locations in main memory or, for some systems, virtual memory. To achieve this objective, a variety of addressing techniques has been employed. They all involve some trade-off between address range and/or addressing flexibility, on the one hand, and the number of memory references in the instruction and/or the complexity of address calculation, on the other. In this section, we examine the most common addressing techniques, or modes:

- Immediate
- Direct
- Indirect
- Register
- Register indirect
- Displacement
- Stack

These modes are illustrated in Figure 13.1. In this section, we use the following notation:

A = contents of an address field in the instruction

R = contents of an address field in the instruction that refers to a register

EA = actual (effective) address of the location containing the referenced operand

(X) = contents of memory location X or register X

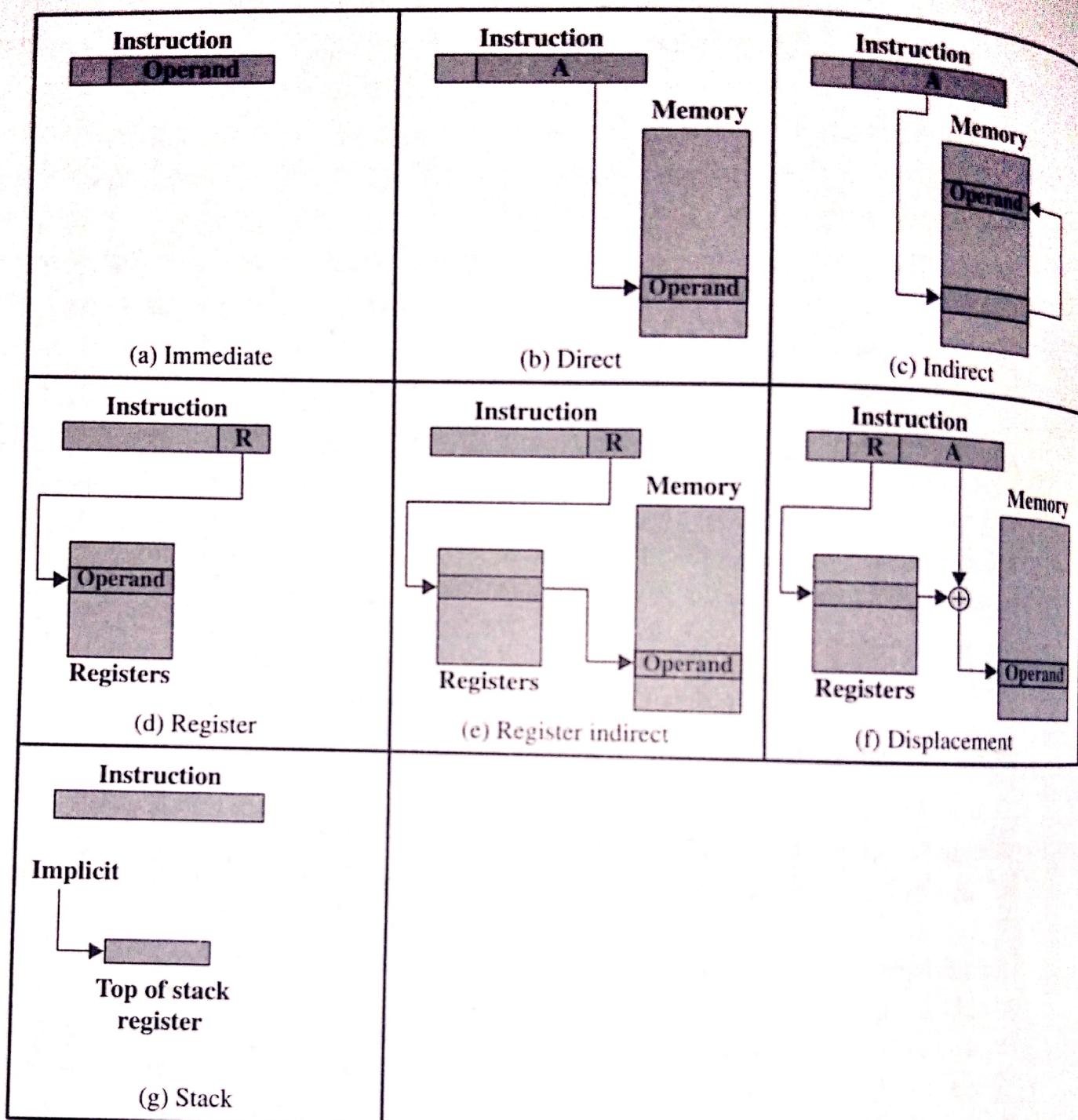


Figure 13.1 Addressing Modes

Table 13.1 indicates the address calculation performed for each addressing mode.

Before beginning this discussion, two comments need to be made. First, virtually all computer architectures provide more than one of these addressing modes. The question arises as to how the processor can determine which address mode is being used in a particular instruction. Several approaches are taken. Often, different opcodes will use different addressing modes. In other cases, more bits in the

**Table 13.1** Basic Addressing Modes

Mode	Algorithm	Principal Advantage	Principal Disadvantage
Immediate	Operand = A	No memory reference	Limited operand magnitude
Direct	EA = A	Simple	Limited address space
Indirect	EA = (A)	Large address space	Multiple memory references
Register	EA = R	No memory reference	Limited address space
Register indirect	EA = (R)	Large address space	Extra memory reference
Displacement	EA = A + (R)	Flexibility	Complexity
Stack	EA = top of stack	No memory reference	Limited applicability

### Immediate Addressing

The simplest form of addressing is **immediate addressing**, in which the operand value is present in the instruction

$$\text{Operand} = A$$

This mode can be used to define and use constants or set initial values of variables. Typically, the number will be stored in two's complement form; the leftmost bit of the operand field is used as a sign bit. When the operand is loaded into a data register, the sign bit is extended to the left to the full data **word** size. In some cases, the immediate binary value is interpreted as an unsigned nonnegative integer.

The advantage of immediate addressing is that no memory reference other than the instruction fetch is required to obtain the operand, thus saving one memory or cache cycle in the instruction cycle. The disadvantage is that the size of the number is restricted to the size of the address field, which, in most instruction sets, is small compared with the word length.

### Direct Addressing

A very simple form of addressing is direct addressing, in which the address field contains the effective address of the operand:

$$EA = A$$

The technique was common in earlier generations of computers but is not common on contemporary architectures. It requires only one memory reference and no special calculation. The obvious limitation is that it provides only a limited address space.

### Indirect Addressing

With direct addressing, the length of the address field is usually less than the word length, thus limiting the address range. One solution is to have the address field refer to the address of a word in memory, which in turn contains a full-length address of the operand. This is known as **indirect addressing**:

$$EA = (A)$$

for two's complement multiplication were to be implemented in software. The location labeled A in the flowchart (Figure 10.12) is referenced many times and should be implemented in a register rather than a main memory location.

It is up to the programmer or compiler to decide which values should remain in registers and which should be stored in main memory. Most modern processors employ multiple general-purpose registers, placing a burden for efficient execution on the assembly-language programmer (e.g., compiler writer).

## Register Indirect Addressing

Just as register addressing is analogous to direct addressing, **register indirect addressing** is analogous to indirect addressing. In both cases, the only difference is whether the address field refers to a memory location or a register. Thus, for register indirect address,

$$EA = (R)$$

The advantages and limitations of register indirect addressing are basically the same as for indirect addressing. In both cases, the address space limitation (limited range of addresses) of the address field is overcome by having that field refer to a word-length location containing an address. In addition, register indirect addressing uses one less memory reference than indirect addressing.

## Displacement Addressing

A very powerful mode of addressing combines the capabilities of direct addressing and register indirect addressing. It is known by a variety of names depending on the context of its use, but the basic mechanism is the same. We will refer to this as **displacement addressing**:

$$EA = A + (R)$$

Displacement addressing requires that the instruction have two address fields, at least one of which is explicit. The value contained in one address field (value = A) is used directly. The other address field, or an implicit reference based on opcode, refers to a register whose contents are added to A to produce the effective address.

We will describe three of the most common uses of displacement addressing:

- Relative addressing
- Base-register addressing
- Indexing