CHAPTER THIRTEEN

Multiprocessors

IN THIS CHAPTER

13-1 Characteristics of Multiprocessors

A multiprocessor system is an interconnection of two or more CPUs with memory and input–output equipment. The term "processor" in *multiprocessor* can mean either a central processing unit (CPU) or an input–output processor (IOP). However, a system with a single CPU and one or more IOPs is usually not included in the definition of a multiprocessor system unless the IOP has computational facilities comparable to a CPU. As it is most commonly defined, a multiprocessor system implies the existence of multiple CPUs, although usually there will be one or more IOPs as well. As mentioned in Sec. 9-1, multiprocessors are classified as multiple instruction stream, multiple data stream (MIMD) systems.

MIMD

There are some similarities between multiprocessor and multicomputer systems since both support concurrent operations. However, there exists an important distinction between a system with multiple computers and a system with multiple processors. Computers are interconnected with each other by means of communication lines to form a *computer network*. The network consists of several autonomous computers that may or may not communicate with each other. A multiprocessor system is controlled by one operating system that provides interaction between processors and all the components of the system cooperate in the solution of a problem.

microprocessor

VLSI

Although some large-scale computers include two or more CPUs in their overall system, it is the emergence of the microprocessor that has been the major motivation for multiprocessor systems. The fact that microprocessors take very little physical space and are very inexpensive brings about the feasibility of interconnecting a large number of microprocessors into one composite system. Very-large-scale integrated circuit technology has reduced the cost of computer components to such a low level that the concept of applying multiple processors to meet system performance requirements has become an attractive design possibility.

Multiprocessing improves the reliability of the system so that a failure or error in one part has a limited effect on the rest of the system. If a fault causes one processor to fail, a second processor can be assigned to perform the functions of the disabled processor. The system as a whole can continue to function correctly with perhaps some loss in efficiency.

The benefit derived from a multiprocessor organization is an improved system performance. The system derives its high performance from the fact that computations can proceed in parallel in one of two ways.

1. Multiple independent jobs can be made to operate in parallel.
2. A single job can be partitioned into multiple parallel tasks.

An overall function can be partitioned into a number of tasks that each processor can handle individually. System tasks may be allocated to special-purpose processors whose design is optimized to perform certain types of processing efficiently. An example is a computer system where one processor performs the computations for an industrial process control while others monitor and control the various parameters, such as temperature and flow rate. Another example is a computer where one processor performs high-speed floating-point mathematical computations and another takes care of routine data-processing tasks.

Multiprocessing can improve performance by decomposing a program into parallel executable tasks. This can be achieved in one of two ways. The user can explicitly declare that certain tasks of the program be executed in parallel. This must be done prior to loading the program by specifying the parallel executable segments. Most multiprocessor manufacturers provide an operating system with programming language constructs suitable for specifying parallel processing. The other, more efficient way is to provide a compiler with multiprocessor software that can automatically detect parallelism in a user's program. The compiler checks for *data dependency* in the program. If a program depends on data generated in another part, the part yielding the needed data must be executed first. However, two parts of a program that do not use data generated by each can run concurrently. The parallelizing compiler checks the entire program to detect any possible data dependencies. These that have no data dependency are then considered for concurrent scheduling on different processors.

Multiprocessors are classified by the way their memory is organized. A multiprocessor system with common shared memory is classified as a *shared-memory* or *tightly coupled multiprocessor*. This does not preclude each processor from having its own local memory. In fact, most commercial tightly coupled multiprocessors provide a cache memory with each CPU. In addition, there is a global common memory that all CPUs can access. Information can therefore be shared among the CPUs by placing it in the common global memory.

tightly coupled

loosely coupled

An alternative model of microprocessor is the *distributed-memory* or *loosely coupled* system. Each processor element in a loosely coupled system has its own private local memory. The processors are tied together by a switching scheme designed to route information from one processor to another through a message-passing scheme. The processors relay program and data to other processors in packets. A packet consists of an address, the data content, and some error detection code. The packets are addressed to a specific processor or taken by the first available processor, depending on the communication system used. Loosely coupled systems are most efficient when the interaction between tasks is minimal, whereas tightly coupled systems can tolerate a higher degree of interaction between tasks.

13-2 Interconnection Structures

The components that form a multiprocessor system are CPUs, IOPs connected to input–output devices, and a memory unit that may be partitioned into a number of separate modules. The interconnection between the components can have different physical configurations, depending on the number of transfer paths that are available between the processors and memory in a shared memory system or among the processing elements in a loosely coupled system. There are several physical forms available for establishing an interconnection network. Some of these schemes are presented in this section:

1. Time-shared common bus
2. Multiport memory
3. Crossbar switch
4. Multistage switching network
5. Hypercube system

Time-Shared Common Bus

A common-bus multiprocessor system consists of a number of processors connected through a common path to a memory unit. A time-shared common bus for five processors is shown in Fig. 13-1. Only one processor can communicate with the memory or another processor at any given time. Transfer

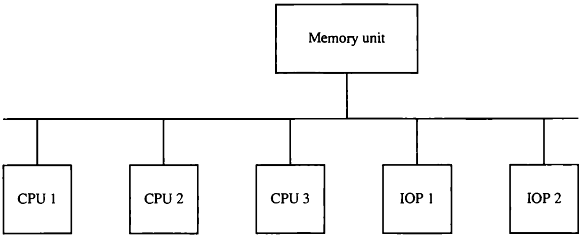


Figure 13-1 Time-shared common bus organization.

operations are conducted by the processor that is in control of the bus at the time. Any other processor wishing to initiate a transfer must first determine the availability status of the bus, and only after the bus becomes available can the processor address the destination unit to initiate the transfer. A command is issued to inform the destination unit what operation is to be performed. The receiving unit recognizes its address in the bus and responds to the control signals from the sender, after which the transfer is initiated. The system may exhibit transfer conflicts since one common bus is shared by all processors. These conflicts must be resolved by incorporating a bus controller that establishes priorities among the requesting units.

A single common-bus system is restricted to one transfer at a time. This means that when one processor is communicating with the memory, all other processors are either busy with internal operations or must be idle waiting for the bus. As a consequence, the total overall transfer rate within the system is limited by the speed of the single path. The processors in the system can be kept busy more often through the implementation of two or more independent buses to permit multiple simultaneous bus transfers. However, this increases the system cost and complexity.

A more economical implementation of a dual bus structure is depicted in Fig. 13-2. Here we have a number of local buses each connected to its own local memory and to one or more processors. Each local bus may be connected to a CPU, an IOP, or any combination of processors. A system bus controller links each local bus to a common system bus. The I/O devices connected to the local IOP, as well as the local memory, are available to the local processor. The memory connected to the common system bus is shared by all processors. If an IOP is connected directly to the system bus, the I/O devices attached to it may be made available to all processors. Only one processor can communicate

shared memory

with the shared memory and other common resources through the system bus at any given time. The other processors are kept busy communicating with their local memory and I/O devices. Part of the local memory may be designed

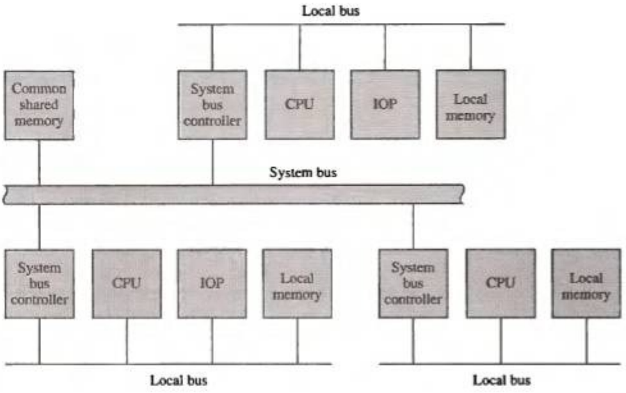


Figure 13-2 System bus structure for multiprocessors.

as a cache memory attached to the CPU (see Sec. 12-6). In this way, the average access time of the local memory can be made to approach the cycle time of the CPU to which it is attached.

Multiport Memory

A multiport memory system employs separate buses between each memory module and each CPU. This is shown in Fig. 13-3 for four CPUs and four memory modules (MMs). Each processor bus is connected to each memory module. A processor bus consists of the address, data, and control lines required to communicate with memory. The memory module is said to have four ports and each port accommodates one of the buses. The module must have internal control logic to determine which port will have access to memory at any given time. Memory access conflicts are resolved by assigning fixed priorities to each memory port. The priority for memory access associated with each processor may be established by the physical port position that its bus occupies in each module. Thus CPU 1 will have priority over CPU 2, CPU 2 will have priority over CPU 3, and CPU 4 will have the lowest priority.

The advantage of the multiport memory organization is the high transfer rate that can be achieved because of the multiple paths between processors and memory. The disadvantage is that it requires expensive memory control logic and a large number of cables and connectors. As a consequence, this intercon-

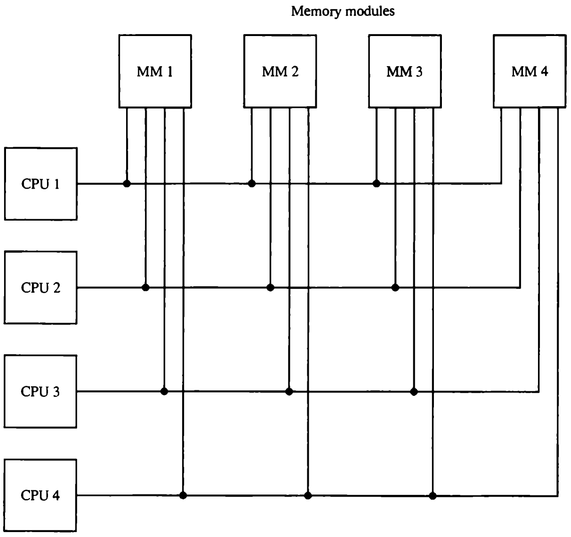


Figure 13-3 Multiport memory organization.

nection structure is usually appropriate for systems with a small number of processors.

Crossbar Switch

The crossbar switch organization consists of a number of crosspoints that are placed at intersections between processor buses and memory module paths. Figure 13-4 shows a crossbar switch interconnection between four CPUs and four memory modules. The small square in each crosspoint is a switch that determines the path from a processor to a memory module. Each switch point has control logic to set up the transfer path between a processor and memory. It examines the address that is placed in the bus to determine whether its particular module is being addressed. It also resolves multiple requests for access to the same memory module on a predetermined priority basis.

Figure 13-5 shows the functional design of a crossbar switch connected to one memory module. The circuit consists of multiplexers that select the data,

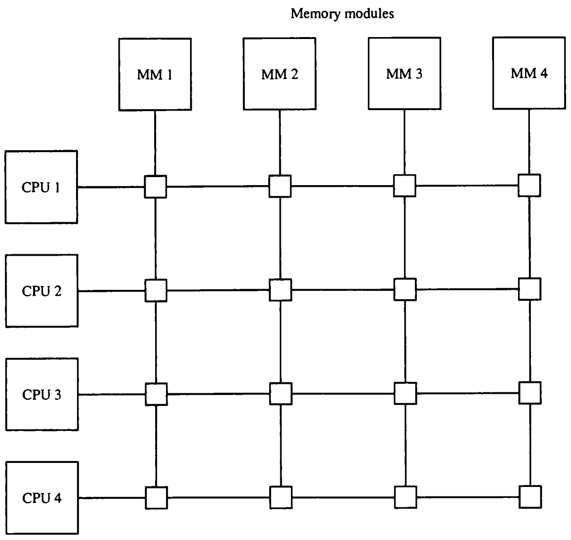
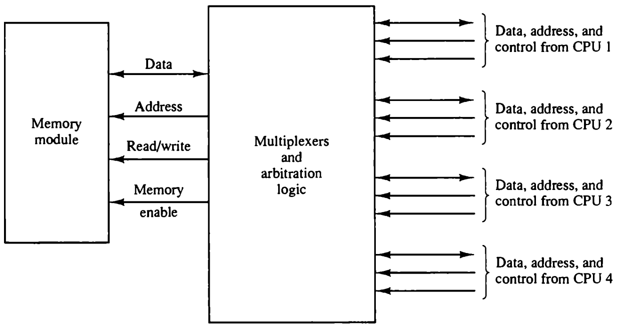


Figure 13-4 Crossbar switch.

Figure 13-5 Block diagram of crossbar switch.


Memory module
Data
Address
Read/write
Memory enable
Multiplexers and arbitration logic
Data, address, and control from CPU 1
Data, address, and control from CPU 2
Data, address, and control from CPU 3
Data, address, and control from CPU 4

address, and control from one CPU for communication with the memc module. Priority levels are established by the arbitration logic to select one CI when two or more CPUs attempt to access the same memory. The multiplex are controlled with the binary code that is generated by a priority encoc within the arbitration logic.

A crossbar switch organization supports simultaneous transfers from memory modules because there is a separate path associated with each mc ule. However, the hardware required to implement the switch can becoı quite large and complex.

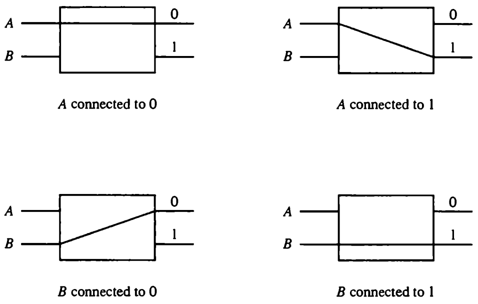
Multistage Switching Network

interchange switch

The basic component of a multistage network is a two-input, two-outp interchange switch. As shown in Fig. 13-6, the 2 × 2 switch has two inpu labeled *A* and *B*, and two outputs, labeled 0 and 1. There are control sign (not shown) associated with the switch that establish the interconnecti between the input and output terminals. The switch has the capability connecting input *A* to either of the outputs. Terminal *B* of the switch beha in a similar fashion. The switch also has the capability to arbitrate betwe conflicting requests. If inputs *A* and *B* both request the same output termin only one of them will be connected; the other will be blocked.

Using the 2 × 2 switch as a building block, it is possible to build multistage network to control the communication between a number of sour and destinations. To see how this is done, consider the binary tree shown Fig. 13-7. The two processors P_1 and P_2 are connected through switches to eig memory modules marked in binary from 000 through 111. The path fron source to a destination is determined from the binary bits of the destinati

Figure 13-6 Operation of a 2 × 2 interchange switch.



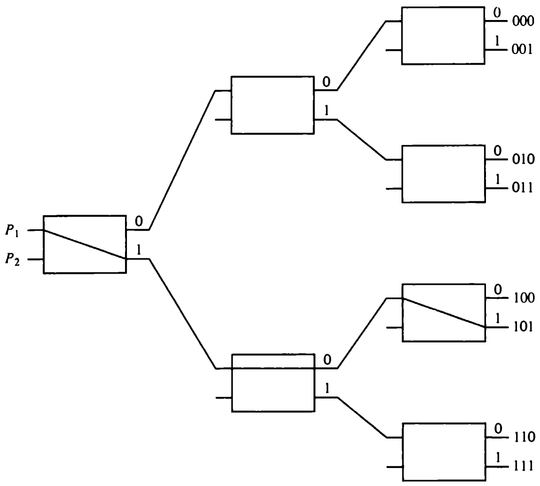


Figure 13-7 Binary tree with 2 × 2 switches.

number. The first bit of the destination number determines the switch output in the first level. The second bit specifies the output of the switch in the second level, and the third bit specifies the output of the switch in the third level. For example, to connect P_1 to memory 101, it is necessary to form a path from P_1 to output 1 in the first-level switch, output 0 in the second-level switch, and output 1 in the third-level switch. It is clear that either P_1 or P_2 can be connected to any one of the eight memories. Certain request patterns, however, cannot be satisfied simultaneously. For example, if P_1 is connected to one of the destinations 000 through 011, P_2 can be connected to only one of the destinations 100 through 111.

omega network

Many different topologies have been proposed for multistage switching networks to control processor–memory communication in a tightly coupled multiprocessor system or to control the communication between the processing elements in a loosely coupled system. One such topology is the omega switching network shown in Fig. 13-8. In this configuration, there is exactly one path from each source to any particular destination. Some request patterns, however, cannot be connected simultaneously. For example, any two sources cannot be connected simultaneously to destinations 000 and 001.

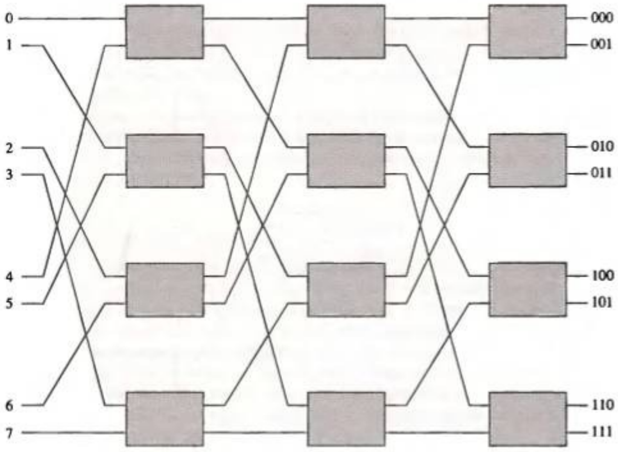


Figure 13-8 8 × 8 omega switching network.

A particular request is initiated in the switching network by the source, which sends a 3-bit pattern representing the destination number. As the binary pattern moves through the network, each level examines a different bit to determine the 2 × 2 switch setting. Level 1 inspects the most significant bit, level 2 inspects the middle bit, and level 3 inspects the least significant bit. When the request arrives on either input of the 2 × 2 switch, it is routed to the upper output if the specified bit is 0 or to the lower output if the bit is 1.

In a tightly coupled multiprocessor system, the source is a processor and the destination is a memory module. The first pass through the network sets up the path. Succeeding passes are used to transfer the address into memory and then transfer the data in either direction, depending on whether the request is a read or a write. In a loosely coupled multiprocessor system, both the source and destination are processing elements. After the path is established, the source processor transfers a message to the destination processor.

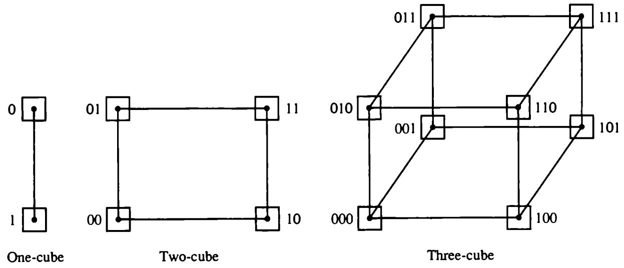
Hypercube Interconnection

The hypercube or binary n-cube multiprocessor structure is a loosely coupled system composed of $N = 2^n$ processors interconnected in an n-dimensional binary cube. Each processor forms a *node* of the cube. Although it is customary to refer to each node as having a processor, in effect it contains not only a CPU but also local memory and I/O interface. Each processor has direct communication paths to n other neighbor processors. These paths correspond to the *edges* of the cube. There are 2^n distinct n-bit binary addresses that can be assigned to the processors. Each processor address differs from that of each of its n neighbors by exactly one bit position.

Figure 13-9 shows the hypercube structure for $n = 1, 2$, and 3. A one-cube structure has $n = 1$ and $2^n = 2$. It contains two processors interconnected by a single path. A two-cube structure has $n = 2$ and $2^n = 4$. It contains four nodes interconnected as a square. A three-cube structure has eight nodes interconnected as a cube. An n-cube structure has 2^n nodes with a processor residing in each node. Each node is assigned a binary address in such a way that the addresses of two neighbors differ in exactly one bit position. For example, the three neighbors of the node with address 100 in a three-cube structure are 000, 110, and 101. Each of these binary numbers differs from address 100 by one bit value.

Routing messages through an n-cube structure may take from one to n links from a source node to a destination node. For example, in a three-cube structure, node 000 can communicate directly with node 001. It must cross at least two links to communicate with 011 (from 000 to 001 to 011 or from 000 to 010 to 011). It is necessary to go through at least three links to communicate from node 000 to node 111. A routing procedure can be developed by computing the exclusive-OR of the source node address with the destination node address. The resulting binary value will have 1 bits corresponding to the axes on which the two nodes differ. The message is then sent along any one of the axes. For example, in a three-cube structure, a message at 010 going to 001 produces an exclusive-OR of the two addresses equal to 011. The message can be sent along the second axis to 000 and then through the third axis to 001.

Figure 13-9 Hypercube structures for $n = 1,2,3$.


0
1
One-cube
01
11
00
10
Two-cube
011
111
010
110
001
101
000
100
Three-cube