Computer Organization and Architecture

Referred Book

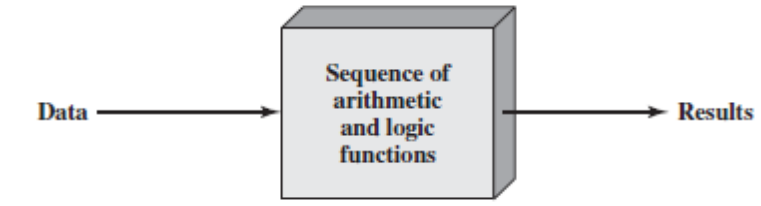
Computer_ Organization _ and_ Architecture - William Stallings

C h a p t e r **3 A Top-Level Views of Computer Function and Interconnection**

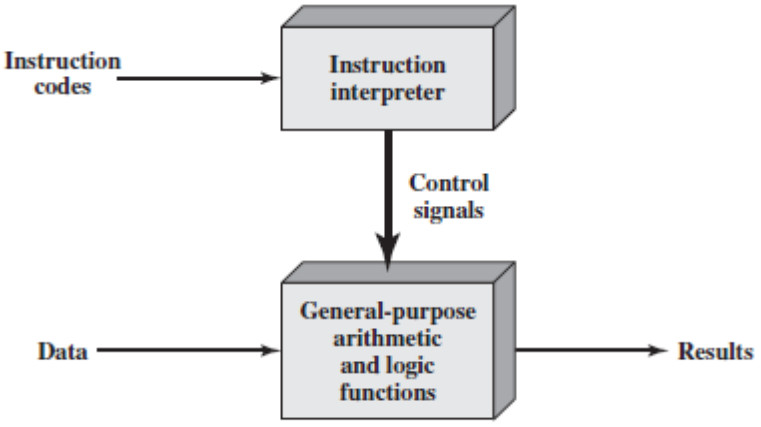
- **Hardware and software approaches**

This set of hardware will perform various functions on data depending on control signals applied to the hardware. In the original case of customized hardware, the system accepts data and produces results

(Figure 3.1a).With general-purpose hardware, the system accepts data and control signals and produces results. Thus, instead of rewiring the hardware for each new program, the programmer simply needs to supply a new set of control signals.



(a) Programming in hardware



(b) Programming in software

Figure 3.1 Hardware and Software Approaches

Figure 3.1b indicates two major components of the system: an instruction interpreter and a module of general-purpose arithmetic and logic functions. These two constitute the CPU. Several other components are needed to yield a functioning computer. Data and instructions must be put into the system. For this we need some sort of input module.

- **Basic Instruction Cycle:**

The processing required for a single instruction is called an *instruction cycle*. Using the simplified two-step description given, the instruction cycle is depicted.

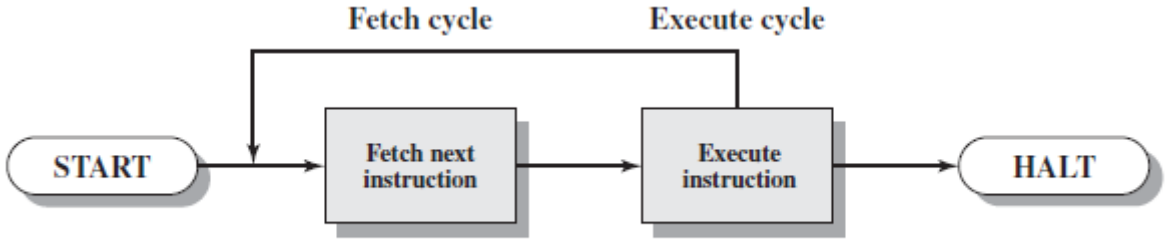


Figure 3.3 Basic Instruction Cycle

In Figure 3.3, the two steps are referred to as the *fetch cycle* and the *execute cycle*. Program execution halts only if the machine is turned off, some sort of unrecoverable error occurs, or a program instruction that halts the computer is encountered.

- **Interrupt**

an **interrupt** is a signal to the processor emitted by hardware or software indicating an event that needs immediate attention. An interrupt alerts the processor to a high-priority condition requiring the interruption of the current code the processor is executing.

Classes of Interrupts

Program

Generated by some condition that occurs as a result of an instruction execution, such as arithmetic overflow, division by zero, attempt to execute an illegal machine instruction, and reference outside a user's allowed memory space.

Timer

Here generated by a timer within the processor. This allows the operating system to perform certain functions on a regular basis.

I/O

Here generated by an I/O controller, to signal normal completion of an operation or to signal a variety of error conditions.

Hardware failure

Here generated by a failure, such as power failure or memory parity error.

- **Instruction Cycle State Diagram**

The figure is in the form of a state diagram. For any given instruction cycle, some states may be null and others may be visited more than once. The states can be described as follows:

• **Instruction address calculation (iac):** Determine the address of the next instruction to be executed. Usually, this involves adding a fixed number to the address of the previous instruction. For example, if each instruction is 16 bits long and memory is organized into 16-bit words, then add 1 to the previous address.

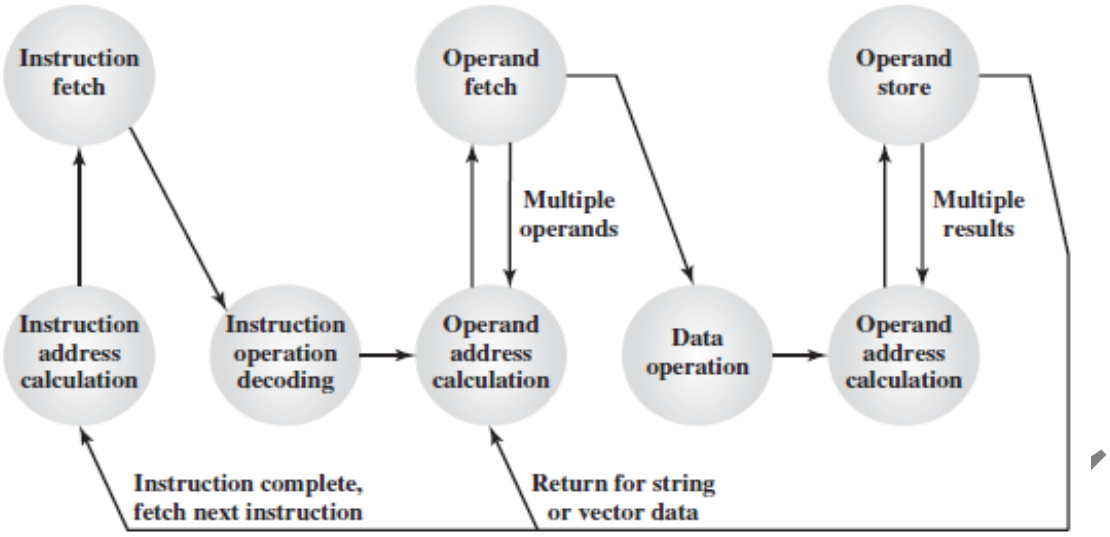


Figure 3.6 Instruction Cycle State Diagram

• **Instruction fetch (if):** Read instruction from its memory location into the processor.
• **Instruction operation decoding (iod):** Analyze instruction to determine type of operation to be performed and operand(s) to be used.
• **Operand address calculation (oac):** If the operation involves reference to an operand in memory or available via I/O, then determine the address of the operand.
• **Operand fetch (of):** Fetch the operand from memory or read it in from I/O.
• **Data operation (do):** Perform the operation indicated in the instruction.
• **Operand store (os):** Write the result into memory or out to I/O.

- **Instruction cycle state diagram with interrupts**

We described the processor's instruction cycle (Figure 3.9).To recall; an instruction cycle includes the following stages:
• **Fetch:** Read the next instruction from memory into the processor.
• **Execute:** Interpret the op-code and perform the indicated operation.
• **Interrupt:** If interrupts are enabled and an interrupt has occurred, save the current process state and service the interrupt.
The main line of activity consists of alternating instruction fetch and instruction execution activities. After an instruction is fetched, it is examined to determine if any indirect addressing is involved. If so, the required operands are fetched using indirect addressing. Following execution, an interrupt may be processed before the next instruction fetch.

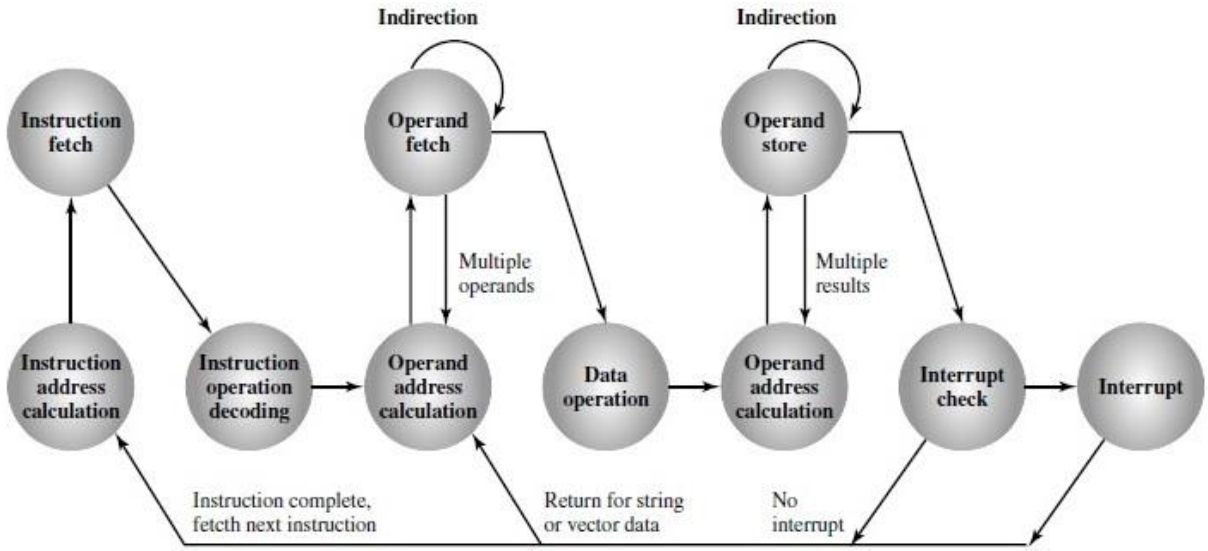


Figure 12.5 Instruction Cycle State Diagram

- **Computer Buses**

A **bus** is a communication system that transfers data between components inside a computer, or between computers. This expression covers all related hardware components (wire, optical fiber, etc.) and software, including communication protocols.

Although there are many different bus designs, on any bus the lines can be classified into three functional groups (Figure 3.16): data, address, and control lines.
The **data lines** provide a path for moving data among system modules. These lines, collectively, are called the *data bus*. The data bus may consist of 32, 64, 128, or even more separate lines, the number of lines being referred to as the *width* of the data bus.

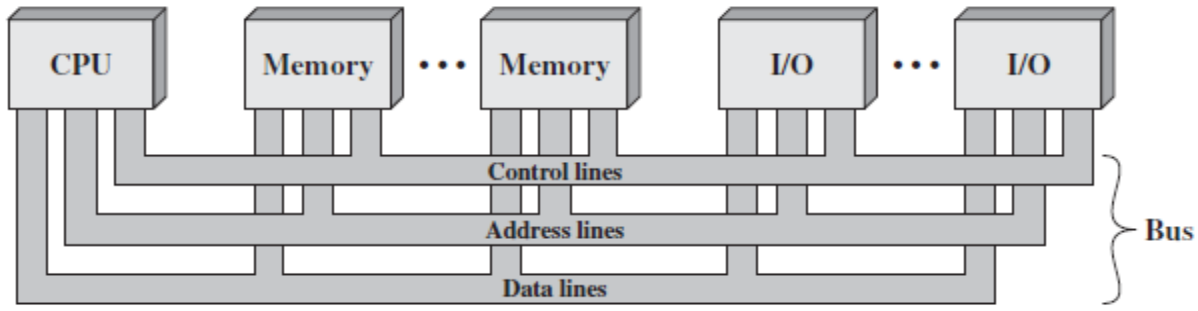
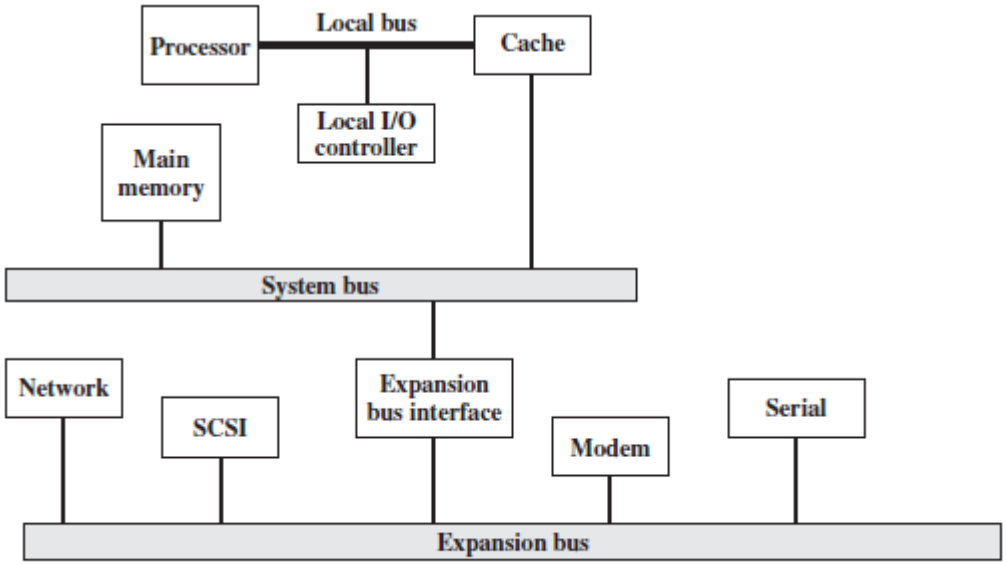


Figure 3.16 Bus Interconnection Scheme

The **address lines** are used to designate the source or destination of the data on the data bus. For example, if the processor wishes to read a word (8, 16, or 32 bits) of data from memory, it puts the address of the desired word on the address lines.
The **control lines** are used to control the access to and the use of the data and address lines. Because the data and address lines are shared by all components, there must be a means of controlling their use.

- **Traditional Bus Architecture**

Figure 3.18a shows some typical examples of I/O devices that might be attached to the expansion bus. Network connections include local area networks (LANs) such as a 10-Mbps Ethernet and connections to wide area networks (WANs) such as a packet-switching network.



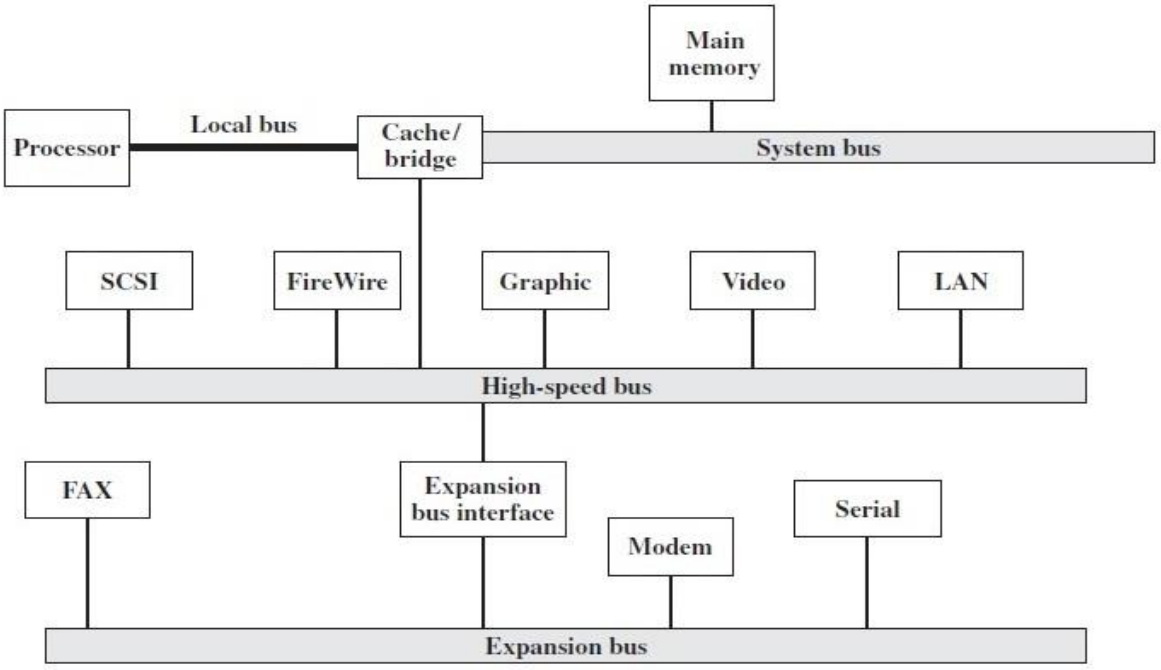
(a) Traditional bus architecture

SCSI (small computer system interface) is itself a type of bus used to support local disk drives and other peripherals. A serial port could be used to support a printer or scanner. This traditional bus architecture is reasonably efficient but begins to break down as higher and higher performance is seen in the I/O devices.

- **High performance bus structure**

Figure 3.18b shows a typical realization of this approach .Again, there is a local bus that connects the processor to a cache controller, which is in turn connected to a system bus that supports main memory. The cache controller is integrated into a bridge, or buffering device, that connects to the high-speed bus.

This bus supports connections to high-speed LANs, such as Fast Ethernet at 100 Mbps, video and graphics workstation controllers, as well as interface controllers to local peripheral buses, including SCSI and FireWire.



(b) High-performance architecture

Figure 3.18 Example Bus Configurations

The latter is a high-speed bus arrangement specifically designed to support high-capacity I/O devices. Lower-speed devices are still supported off an expansion bus, with an interface buffering traffic between the expansion bus and the high-speed bus.

- **Elements of BUS design**

A variety of different bus implementations exist, there are a few basic parameters or design elements that serve to classify and differentiate buses. Table 3.2 lists key elements.

BUS TYPES Bus lines can be separated into two generic types: dedicated and multiplexed.
A dedicated bus line is permanently assigned either to one function or to a physical subset of computer components.
The address is then removed from the bus, and the same bus connections are used for the subsequent read or write data transfer. This method of using the same lines for multiple purposes is known as *time multiplexing.*

METHOD OF ARBITRATION In all but the simplest systems, more than one module may need control of the bus.
In a **centralized scheme**, a single hardware device, referred to as a *bus controller* or *arbiter,* is responsible for allocating time on the bus.
In a **distributed scheme**, there is no central controller. Rather, each module contains access control logic and the modules act together to share the bus.

METHOD OF ARBITRATION In all but the simplest systems, more than one module may need control of the bus.

In a **centralized scheme**, a single hardware device, referred to as a *bus controller* or *arbiter*, is responsible for allocating time on the bus.

Table 3.2 Elements of Bus Design

Type	**Bus Width**
Dedicated	Address
Multiplexed	Data
Method of Arbitration	**Data Transfer Type**
Centralized	Read
Distributed	Write
Timing	Read-modify-write
Synchronous	Read-after-write
Asynchronous	Block

In a **distributed scheme**, there is no central controller. Rather, each module contains access control logic and the modules act together to share the bus.

TIMING refers to the way in which events are coordinated on the bus. Buses use either synchronous timing or asynchronous timing.
With **synchronous timing**, the occurrence of events on the bus is determined by a clock. The bus includes a clock line upon which a clock transmits a regular sequence of alternating 1s and 0s of equal duration.
With **asynchronous timing**, the occurrence of one event on a bus follows and depends on the occurrence of a previous event. In the simple read the processor places address and status signals on the bus. After pausing for these signals to stabilize, it issues a read command, indicating the presence of valid address and control signals.

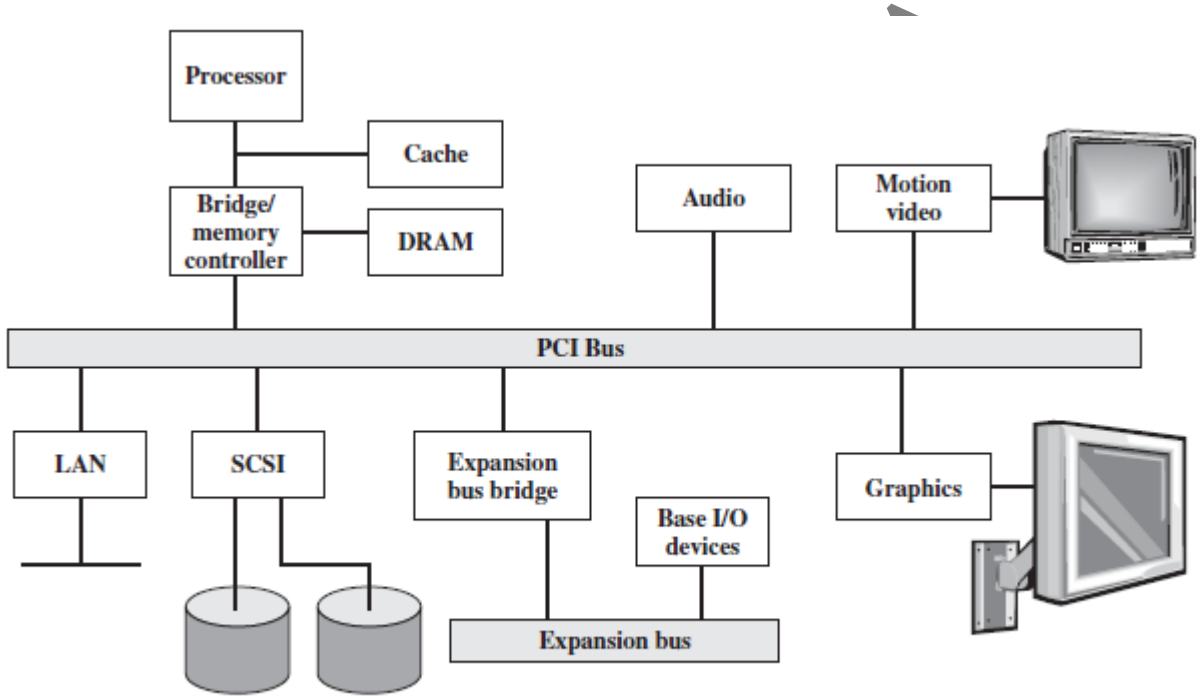
BUS WIDTH We have already **addressed** the concept of bus width. The width of the **data** bus has an impact on system performance: The wider the data bus, the greater the number of bits transferred at one time.

DATA TRANSFER TYPE Finally, a bus supports various data transfer types. All buses support both **write** (master to slave) and **read** (slave to master) transfers.
A read–modify–write operation is simply a read followed immediately by a write to the same address.
Read-after-write is an indivisible operation consisting of a write followed immediately by a read from the same address.
Some bus systems also support a **block** data transfer. In this case, one address cycle is followed by n data cycles.

PCI Bus
The peripheral component interconnect (PCI) is a popular high-bandwidth, processor-independent bus that can function as a peripheral bus. Compared with other common bus specifications, PCI delivers better system performance for high-speed I/O subsystems (e.g., graphic display adapters, network interface controllers, disk controllers, and so on).

- **PCI Bus typical Desktop system**

Figure 3.22a shows a typical use of PCI in a single-processor system. A combined DRAM controller and bridge to the PCI bus provides tight coupling with the processor and the ability to deliver data at high speeds.
The bridge acts as a data buffer so that the speed of the PCI bus may differ from that of the processor's I/O capability.

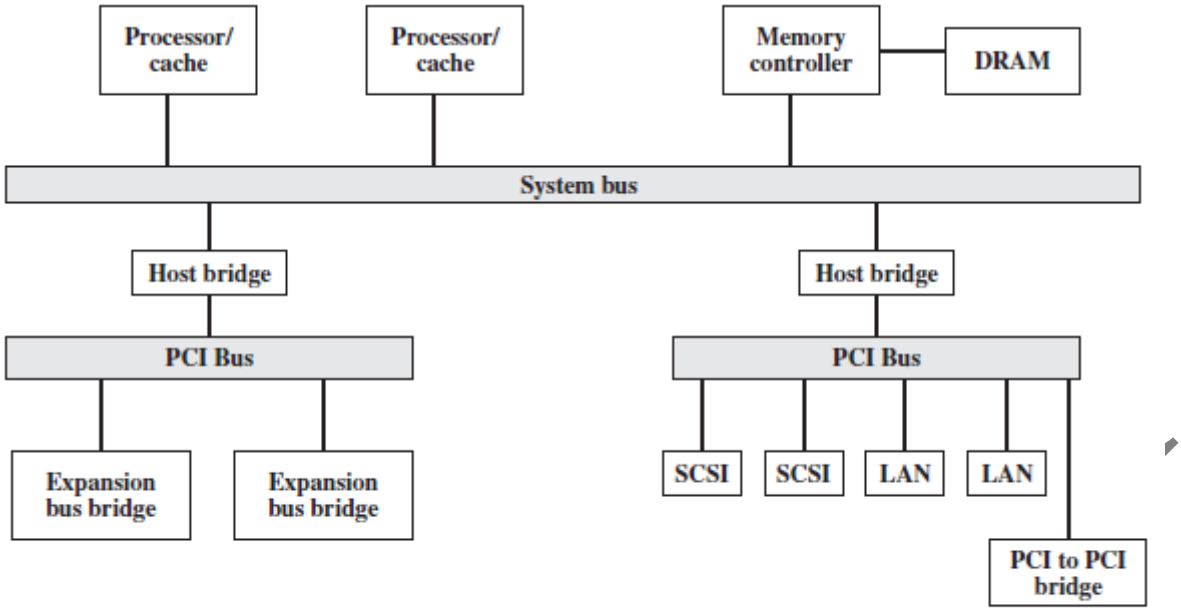


(a) Typical desktop system

- **PCI Bus typical Server system**

In a multiprocessor system (Figure 3.22b), one or more PCI configurations may be connected by bridges to the processor's system bus. The system bus supports only the processor/cache units, main memory, and the PCI bridges.

Again, the use of bridges keeps the PCI independent of the processor speed yet provides the ability to receive and deliver data rapidly.



(b) Typical server system

Figure 3.22 Example PCI Configurations

- **PCI Commands**

Bus activity occurs in the form of transactions between an initiator, or master, and a target. When a bus master acquires control of the bus, it determines the type of transaction that will occur next. The commands are as follows:

- Interrupt Acknowledge
- Special Cycle
- I/O Read
- I/O Write
- Memory Read
- Memory Read Line
- Memory Read Multiple
- Memory Write
- Memory Write and Invalidate
- Configuration Read
- Configuration Write
- Dual address Cycle