I/O Interfacing & Bus Organization

The difficulty in transferring information among the components of a computer is largely dependent on the physical distances separating the components. We distinguish two major cases here: *intrasystem communication*, which occurs within a single computer system and involves information transfer over distances of no more than a meter or so; and *intersystem communication*, which involves communication over longer distances. Intrasystem communication is primarily implemented by means of groups of electrical conductors called buses, which allow parallel (word-by-word) transmission of data. Intersystem communication, on the other hand, is implemented by a variety of physical media, including electrical cables, optical fibers, and radio links. Serial (bit-by-bit) rather than parallel data transmission is used for communicating over longer distances. Serial communication links cost less than parallel, and are also more reliable and simpler to control.

A group of computers, user terminals, and other system components that are linked together over long distances (a kilometer or more) constitute a *computer network*.

A bus is a communication pathway connecting two or more devices. A key characteristic of a bus is that it is a shared transmission medium. Multiple devices connect to the bus, and a signal transmitted by any one device is available for reception by all other devices attached to the bus. If two devices transmit during the same time period, their signals will overlap and become garbled. Thus, only one device at a time can successfully transmit.

Typically, a bus consists of multiple communication pathways, or lines. Each line is capable of transmitting signals representing binary 1 and binary 0. Over time, a sequence of binary digits can be transmitted across a single line. Taken together, several lines of a bus can be used to transmit binary digits simultaneously (in parallel). For example, an 8-bit unit of data can be transmitted over eight bus lines.

Computer systems contain a number of different buses that provide pathways between components at various levels of the computer system hierarchy. A bus that connects major computer components (processor, memory, I/O) is called a *system bus*. The most common computer interconnection structures are based on the use of one or more system buses.

Bus Structure

A system bus consists, typically, of from about 50 to hundreds of separate lines. Each line is assigned a particular meaning or function. Although there are many different bus designs, on any bus the lines can be classified into three functional groups (Figure 3.16):

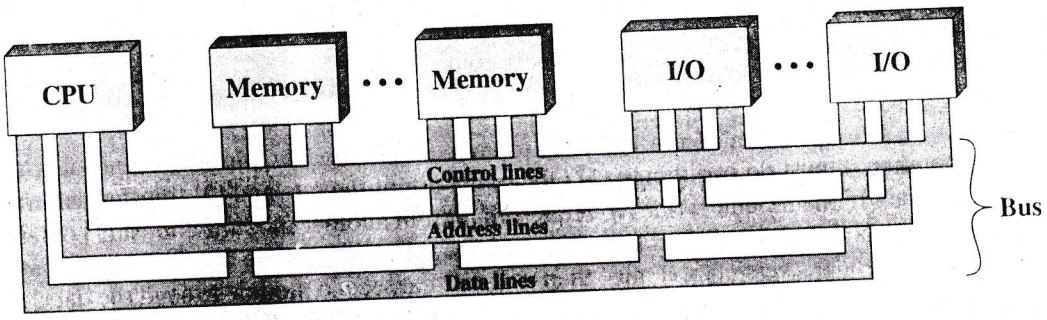


Figure 3.16 Bus Interconnection Scheme

data, address, and control lines. In addition, there may be power distribution lines that supply power to the attached modules.

The **data lines** provide a path for moving data between system modules. These lines, collectively, are called the *data bus*. The data bus may consist of from 32 to hundreds of separate lines, the number of lines being referred to as the *width* of the data bus. Because each line can carry only 1 bit at a time, the number of lines determines how many bits can be transferred at a time. The width of the data bus is a key factor in determining overall system performance. For example, if the data bus is 8 bits wide and each instruction is 16 bits long, then the processor must access the memory module twice during each instruction cycle.

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. Clearly, the width of the address bus determines the maximum possible memory capacity of the system. Furthermore, the address lines are generally also used to address I/O ports.

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. Control signals transmit both command and timing information between system modules. Timing signals indicate the validity of data and address information. Command signals specify operations to be performed. Typical control lines include

- **Memory write:** Causes data on the bus to be written into the addressed location
- **Memory read:** Causes data from the addressed location to be placed on the bus
- **I/O write:** Causes data on the bus to be output to the addressed I/O port
- **I/O read:** Causes data from the addressed I/O port to be placed on the bus

Elements of Bus Design

Although 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.

Table 3.2 Elements of Bus Design

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

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.

An example of functional dedication is the use of separate dedicated address and data lines, which is common on many buses. However, it is not essential. For example, address and data information may be transmitted over the same set of lines using an Address Valid control line. At the beginning of a data transfer, the address is placed on the bus and the Address Valid line is activated. At this point, each module has a specified period of time to copy the address and determine if it is the addressed module. 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*.

The advantage of time multiplexing is the use of fewer lines, which saves space and, usually, cost. The disadvantage is that more complex circuitry is needed within each module. Also, there is a potential reduction in performance because certain events that share the same lines cannot take place in parallel.

Physical dedication refers to the use of multiple buses, each of which connects only a subset of modules. A typical example is the use of an I/O bus to interconnect all I/O modules; this bus is then connected to the main bus through some type of I/O adapter module. The potential advantage of physical dedication is high throughput, because there is less bus contention. A disadvantage is the increased size and cost of the system.

Method of Arbitration In all but the simplest systems, more than one module may need control of the bus. For example, an I/O module may need to read or write directly to memory, without sending the data to the processor. Because only one unit at a time can successfully transmit over the bus, some method of arbitration is needed. The various methods can be roughly classified as being either centralized or distributed. In a centralized scheme, a single hardware device, referred to as a *bus controller* or *arbiter*, is responsible for allocating time on the bus. The device may be a separate module or part of the processor. 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. With both methods of arbitration, the purpose is to designate one device, either the processor or an I/O module, as master. The master may then initiate a data transfer (e.g., read or write) with some other device, which acts as slave for this particular exchange.

Arbitration. The possibility exists of several master and/or slave units connected to a shared bus requesting access to it at the same time. A selection mechanism called bus arbitration is therefore required to decide among such competing requests. Following Thurber et al. [27] we distinguish three main arbitration schemes: daisy chaining, polling, and independent requesting. These methods differ in the number of control lines they require and in the speed with which the bus controller can respond to bus-access requests of different priorities. Some bus systems such as the UNIBUS combine several distinct arbitration techniques.

The *daisy-chaining* method is depicted in Fig. 6.17. Three control signals are involved in the arbitration process to which we assign the generic names BUS REQUEST, BUS GRANT, and BUS BUSY. All the bus devices are connected to a common BUS REQUEST line. When activated, it merely serves to indicate that one or more units are requesting use of the bus. The bus control unit responds to a BUS REQUEST signal only if BUS BUSY is inactive. This response takes the form of a signal placed on the BUS GRANT line. On receiving the BUS GRANT signal, a requesting unit enables its physical bus connections and activates BUS BUSY for the duration of its new bus activity.

The main distinguishing feature of the daisy-chaining technique is the manner in which the BUS GRANT signal is distributed. The BUS GRANT line is connected serially from unit to unit as shown in Fig. 6.17. When the first unit that is requesting access to the bus receives the BUS GRANT signal, it blocks further propagation of that signal, activates BUS BUSY, and begins to use the bus. When a nonrequesting unit receives the BUS GRANT signal, it forwards it to the next unit. Thus if two units are simultaneously requesting bus access, the one closest to the bus-control unit, i.e., the one that receives the BUS GRANT signal first, gains access to the bus. Selection priority is therefore completely determined by the order in which the units are linked together (chained) by the BUS GRANT lines.

Daisy-chaining thus requires very few control lines and employs a very simple arbitration algorithm. It can be used with an essentially unlimited number

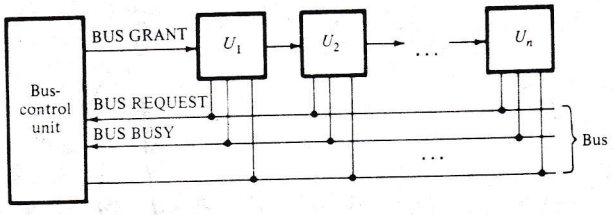


FIGURE 6.17
Bus arbitration using daisy-chaining.

of bus units. Since priority is wired in, the priority of each unit cannot be changed under program control. If it generates bus requests at a sufficiently high rate, a high-priority device like U_1 can lock out a low-priority device like U_n. A further difficulty with daisy-chaining is its susceptibility to failures involving the BUS GRANT line and its associated circuitry. If unit U_i is unable to propagate the BUS GRANT signal, then all $\{U_j\}$ where $j > i$ cannot gain access to the bus.

In a bus-control system that uses *polling*, the BUS GRANT line of the daisy-chain method is replaced by a set of lines called poll count lines which are connected directly to all units on the bus, as depicted in Fig. 6.18. As before, the units request access to the bus via a common BUS REQUEST line. In response to a signal on BUS REQUEST, the bus controller proceeds to generate a sequence of numbers on the poll count lines. These numbers, which may be thought of as unit addresses, are compared by each unit with a unique address assigned to that unit. When a requesting unit U_i finds that its address matches the number on the poll count lines, it activates BUS BUSY. The bus controller responds by terminating the polling process, and U_i connects to the bus.

Clearly, the priority of a bus unit is determined by the position of its address in the polling sequence. This sequence is normally programmable (the poll count lines are connected to a programmable register); hence selection priority can be altered under program control. A further advantage of polling over daisy-chaining is that a failure in one unit need not affect any of the other units. This flexibility is achieved at the cost of more control lines (k poll count lines instead of one BUS GRANT line). Also, the number of units that can share the bus is limited by the addressing capability of the poll count lines.

The third arbitration technique, *independent requesting*, uses separate BUS REQUEST and BUS GRANT lines for each unit sharing the bus. This approach, which is depicted in Fig. 6.19, provides the bus-control unit with immediate identification of all requesting units and enables it to respond very rapidly to requests for bus access. Priority is determined by the bus-control unit and may be programmable. The main drawback of bus control by independent requesting is the fact that $2n$ BUS REQUEST and BUS GRANT lines must be connected

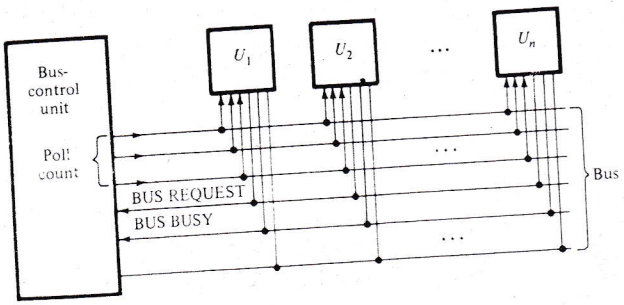


FIGURE 6.18
Bus arbitration using polling.

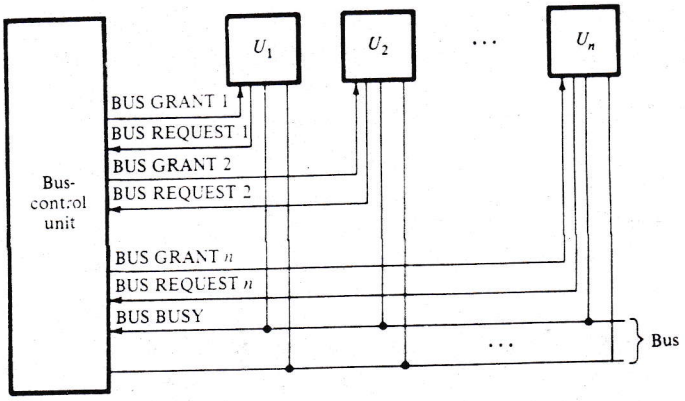


FIGURE 6.19
Bus arbitration using independent requesting.

to the bus-control unit in order to control n devices. In contrast, daisy-chaining requires two such lines, while polling requires approximately $\log_2 n$ lines.

Timing 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. A single 1–0 transmission is referred to as a *clock cycle* or *bus cycle* and defines a time slot. All other devices on the bus can read the clock line, and all events start at the beginning of a clock cycle.

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 example of Figure 3.20a, 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. The appropriate memory decodes the address and responds by placing the data on the data line. Once the data lines have stabilized, the memory module asserts the acknowledged line to signal the processor that the

data is available.

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. The width of the address bus has an impact on system capacity: The wider the address bus, the greater the range of locations that can be referenced.

Programmed I/O and Interrupt initiated I/O

Overview of Programmed I/O

When the processor is executing a program and encounters an instruction relating to I/O, it executes that instruction by issuing a command to the appropriate I/O module. With programmed I/O, the I/O module will perform the requested action and then set the appropriate bits in the I/O status register (Figure 7.3). The I/O module takes no further action to alert the processor. In particular, it does not interrupt the processor. Thus, it is the responsibility of the processor periodically to check the status of the I/O module until it finds that the operation is complete.

Interrupt Initiated I/O.

The problem with programmed I/O is that the processor has to wait a long time for the I/O module of concern to be ready for either reception or transmission of data.

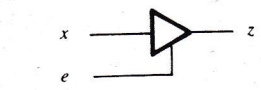
The processor, while waiting, must repeatedly interrogate the status of the I/O module. As a result, the level of the performance of the entire system is severely degraded.

An alternative is for the processor to issue an I/O command to a module and then go on to do some other useful work. The I/O module will then interrupt the processor to request service when it is ready to exchange data with the processor. The processor then executes the data transfer, as before, and then resumes its former processing.

Interfacing. The lines that constitute a communication bus can usually be divided into three functional groups: data lines, address lines, and control lines. The data lines are designed to transmit all bits of an n-bit word in parallel; they therefore consist of either two sets of n unidirectional lines or a single set of n bidirectional lines. The data-bus size n is usually a multiple of 8, with $n = 8$, 16, or 32 being common values. Address lines are used to identify a unit or part of a unit to be used in a data transfer and therefore to be given access to the bus. It is possible to use the data lines for transferring addresses as well as data; this is termed data/address multiplexing. This may be done to decrease the cost of the bus; or to decrease the number of external connections (pins) of the units served by the bus. Memory buses usually contain separate address lines, but IO buses usually do not (see Fig. 6.1*b*). This is because every word transfer to main memory must be accompanied by an address, whereas data transfers via an IO bus are usually in long blocks of words, which require only the address of the start of the block. Finally, the bus control lines are used to transfer timing signals and status information about the units in the system. They may also be used to indicate the type of information present on the data lines.

A significant contributor to the cost of a system bus is the number and type of logic circuits required to transfer signals to and from the bus. A bus line represents a logic path with potentially very large fan-in and fan-out. Consequently, interface circuits termed bus *drivers* and *receivers*, which are basically amplifier or buffer circuits, may be needed to transfer signals to the bus and from the bus, respectively.

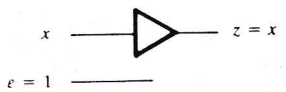
A special logic circuit technology called *tristate* logic is often used in bus design. It is characterized by the presence of three signal values 0, 1, and Z, where the third value Z is termed the *high-impedance* state. While 0 and 1 typically correspond to two electrical voltage levels, e.g., 0 and 5 V, Z represents the state of a line that is electrically disconnected from all voltage sources, i.e., an open-circuited line. Figures 6.9*a* and *b* define a basic tristate buffer which can serve as a bus-line driver. The inputs x and e are ordinary binary signals that assume only the values 0 and 1; the output z, however, can assume the values 0,



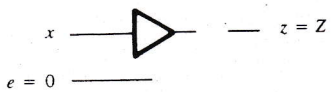
(*a*)

Inputs		Output
x	e	z
0	1	0
1	1	1
0	0	Z
1	0	Z

(*b*)



(*c*)



(*d*)

FIGURE 6.9
Tristate buffer: (*a*) logic symbol; (*b*) truth table; (*c*) equivalent circuit when enabled; (*d*) equivalent circuit when disabled.

1, and Z. The tristate buffer (and every other tristate device) has a special input line e called *output enable*, which when set to 0 disables the output line z by placing it in the high-impedance state Z. When $e = 1$, the circuit becomes an ordinary noninverting buffer with $z = x$. Figures 6.9*b* and *c* show equivalent circuits corresponding to the buffer in the enabled and disabled states.

Tristate logic circuits have two major advantages in the design of shared buses.

1. They greatly reduce fan-in and fan-out constraints on bus lines.
2. They facilitate bidirectional signal transmission over a bus line by allowing the same bus connection to serve as both an input and an output port.

These issues are illustrated by Fig. 6.10, which shows the use of tristate logic for interfacing two units U_1 and U_2 to a set of bidirectional bus lines. If $e_1 = 1$ and $e_2 = 0$, then U_1 drives the bus lines in question, and information is transferred over the bus from U_1 to U_2, in effect making $x_{2,i} = z_{1,i}$ for all i. Conversely, if $e_1 = 0$ and $e_2 = 1$, then U_2 drives the bus, and information is transferred in the opposite direction from U_2 to U_1, making $x_{1,i} = z_{2,i}$ for all i. If $e_1 = e_2 = 0$, then the outputs of both U_1 and U_2 are logically disconnected from the bus, and impose only a minuscule electrical load on the bus. The condition $e_1 = e_2 = 1$ is invalid, since it applies two different signals to each bus line making the resultant bus state indeterminate. Thus proper operation of the bus requires that at most one driver connected to each bus line be enabled at any time. Wired logic circuits of the kind illustrated by Fig. 2.2 and implemented by open-collector bipolar transistor circuits can also be used to drive a common bus line z in a similar manner to a tristate driver.

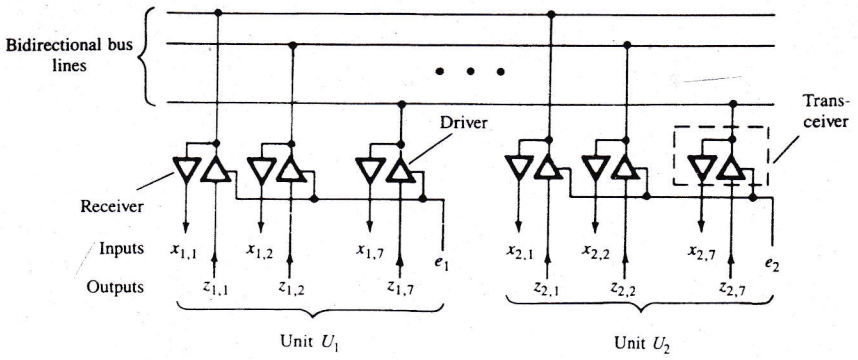


FIGURE 6.10
Use of tristate logic for bus interfacing.

Direct Memory Access (DMA)

The programmed IO method discussed in the preceding section has two main drawbacks.

1. IO transfer rates are limited by the speed with which the CPU can test and service an IO device.
2. The time that the CPU spends testing IO device status and executing IO data transfers can often be better spent on other processing tasks.

The influence of the CPU on IO transfer rates is twofold. First, a delay may occur while an IO device that requires service waits to be tested by the CPU. If there are many IO devices in the system, each device may be tested relatively infrequently. Second, programmed IO transmits data through the CPU rather than allowing it to be passed directly from main memory to the IO device, and vice versa.

DMA and interrupt circuits are used to increase the speed of IO operations and eliminate most of the role played by the CPU in such operations. In each case special control lines to which we assign the generic names DMA REQUEST and INTERRUPT REQUEST go from the IO devices to the CPU. Signals on these lines cause the CPU to suspend its current activities at an appropriate breakpoint and attend to the DMA or interrupt request. Thus the need for the CPU to execute routines that determine IO device status is eliminated. DMA further allows IO data transfers to take place without the execution of IO instructions by the CPU.

A DMA request by an IO device requires the CPU only to yield control of the main-memory (system) bus to the requesting device. The CPU can yield control at the end of any transactions involving the use of this bus. Figure 6.41 shows a typical sequence of CPU actions during an instruction cycle. The instruction cycle is divided into a number of CPU cycles, several of which require use of the system bus. A common technique is to allow the machine to respond to a DMA request at the end of any CPU cycle. Thus during the instruction cycle of Fig. 6.41, there are five points in time (breakpoints) when the CPU can respond to a DMA request. When such a request is received by the CPU, it waits until the

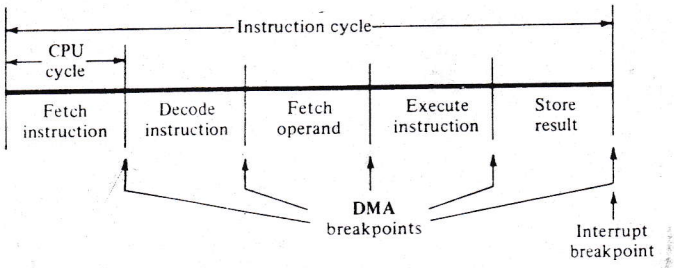


FIGURE 6.41
DMA and interrupt breakpoints during an instruction cycle.

next breakpoint, releases the system bus, and signals the requesting IO device by activating a DMA ACKNOWLEDGE control line.

Interrupts are requested and acknowledged in much the same way as DMA requests. However, an interrupt is not a request for bus control; rather, it asks the CPU to begin executing an interrupt service program. The interrupt program may perform a variety of tasks, such as initiating an IO data transfer, responding to an error encountered by the IO device, etc. The CPU transfers control to this program in essentially the same way it transfers control to a subroutine. The CPU responds to interrupts only between instruction cycles, as indicated in Fig. 6.41.

Direct memory access (DMA). The essential elements of a DMA system are shown in Fig. 6.42. The IO device is connected to the system bus via a special interface circuit, a *DMA controller*, which contains a data buffer register IODR as in the programmed IO case; but in addition there is an address register IOAR and a data count register DC. These registers allow the DMA controller to transfer data to or from a contiguous region of main memory. IOAR is used to store the address of the next word to be transferred. It is automatically incremented after each word transfer. The data count register DC stores the number of words that remain to be transferred. It is automatically decremented after each transfer and tested for zero. When the data count reaches zero, the DMA transfer halts. The controller is normally provided with an interrupt capability, in which case it sends an interrupt to the CPU to signal the end of the data transfer. The logic necessary to control DMA can easily be placed in a single integrated circuit. DMA controllers are available that can supervise DMA transfers involving several IO devices, each with a different priority of access to the system bus.

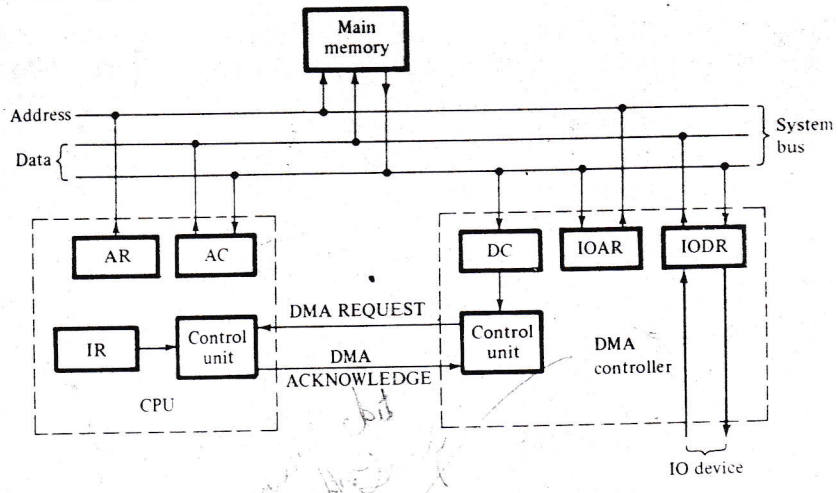


FIGURE 6.42
Circuitry required for direct memory access (DMA).

Data can be transferred in several different ways under DMA control. In a *DMA block transfer* a data-word sequence of arbitrary length is transferred in a single continuous burst while the DMA controller is master of the system bus. This DMA mode is needed by secondary memory devices like magnetic-disk drives where data transmission cannot be stopped or slowed down without loss of data, and block transfers are the norm. Block DMA supports the maximum IO data-transmission rates, but it may require the CPU to remain inactive for relatively long periods. An alternative DMA technique termed *cycle stealing* allows the DMA controller to use the system bus to transfer one, or perhaps several, data words, after which it must return control of the bus to the CPU. This means that long blocks of IO data are transferred by a sequence of DMA bus transactions interspersed with CPU bus transactions. Cycle stealing reduces the maximum IO transfer rate, but it also reduces the interference by the DMA controller in the CPU's activities. It is possible to eliminate this interference completely by designing the DMA interface so that bus cycles are stolen only when the CPU is not actually using the system bus; this is *transparent DMA*. Thus varying degrees of overlap between CPU and DMA operations are possible to accommodate the many different data-transfer characteristics of IO devices.

DMA transfers proceed as follows for the system depicted in Fig. 6.42.

1. The CPU executes two IO instructions, which load the DMA registers IOAR and DC with their initial values. IOAR should contain the base address of the main-memory region to be used in the data transfer. DC should contain the number of words to be transferred to or from that region.
2. When the DMA controller is ready to transmit or receive data, it activates the DMA REQUEST line to the CPU. The CPU waits for the next DMA breakpoint. It then relinquishes control of the data and address lines and activates DMA ACKNOWLEDGE. Note that DMA REQUEST and DMA ACKNOWLEDGE are essentially BUS REQUEST and BUS GRANT lines for the system bus. Simultaneous DMA requests from several DMA controllers can be resolved by using one of the bus priority control techniques discussed earlier.
3. The DMA controller now transfers data directly to or from main memory. After a word is transferred, IOAR and DC are incremented and decremented, respectively.
4. If DC is not decremented to zero but the IO device is not ready to send or receive the next batch of data, the DMA controller returns control to the CPU by releasing the system bus and deactivating the DMA REQUEST line. The CPU responds by deactivating DMA ACKNOWLEDGE and resuming normal operation.
5. If DC is decremented to zero, the DMA controller again relinquishes control of the system bus. It may also send an interrupt signal to the CPU. The CPU responds by halting the IO device or by initiating a new DMA transfer.