

the programmer can use any bit configuration for the mask register. The interrupt status bit must be cleared so it can be set again when a higher-priority interrupt occurs. The contents of processor registers are saved because they may be needed by the program that has been interrupted after control returns to it. The interrupt enable *IEN* is then set to allow other (higher-priority) interrupts and the computer proceeds to service the interrupt request.

The final sequence in each interrupt service routine must have instructions to control the interrupt hardware in the following manner:

1. Clear interrupt enable bit *IEN*.
2. Restore contents of processor registers.
3. Clear the bit in the interrupt register belonging to the source that has been serviced.
4. Set lower-level priority bits in the mask register.
5. Restore return address into *PC* and set *IEN*.

The bit in the interrupt register belonging to the source of the interrupt must be cleared so that it will be available again for the source to interrupt. The lower-priority bits in the mask register (including the bit of the source being interrupted) are set so they can enable the interrupt. The return to the interrupted program is accomplished by restoring the return address to *PC*. Note that the hardware must be designed so that no interrupts occur while executing steps 2 through 5; otherwise, the return address may be lost and the information in the mask and processor registers may be ambiguous if an interrupt is acknowledged while executing the operations in these steps. For this reason *IEN* is initially cleared and then set after the return address is transferred into *PC*.

The initial and final operations listed above are referred to as *overhead* operations or *housekeeping* chores. They are not part of the service program proper but are essential for processing interrupts. All overhead operations can be implemented by software. This is done by inserting the proper instructions at the beginning and at the end of each service routine. Some of the overhead operations can be done automatically by the hardware. The contents of processor registers can be pushed into a stack by the hardware before branching to the service routine. Other initial and final operations can be assigned to the hardware. In this way, it is possible to reduce the time between receipt of an interrupt and the execution of the instructions that service the interrupt source.

11-6 Direct Memory Access (DMA)

The transfer of data between a fast storage device such as magnetic disk and memory is often limited by the speed of the CPU. Removing the CPU from the path and letting the peripheral device manage the memory buses directly

would improve the speed of transfer. This transfer technique is called direct memory access (DMA). During DMA transfer, the CPU is idle and has no control of the memory buses. A DMA controller takes over the buses to manage the transfer directly between the I/O device and memory.

bus request

bus grant

The CPU may be placed in an idle state in a variety of ways. One common method extensively used in microprocessors is to disable the buses through special control signals. Figure 11-16 shows two control signals in the CPU that facilitate the DMA transfer. The *bus request* (*BR*) input is used by the DMA controller to request the CPU to relinquish control of the buses. When this input is active, the CPU terminates the execution of the current instruction and places the address bus, the data bus, and the read and write lines into a high-impedance state. The high-impedance state behaves like an open circuit, which means that the output is disconnected and does not have a logic significance (see Sec. 4-3). The CPU activates the *bus grant* (*BG*) output to inform the external DMA that the buses are in the high-impedance state. The DMA that originated the bus request can now take control of the buses to conduct memory transfers without processor intervention. When the DMA terminates the transfer, it disables the bus request line. The CPU disables the bus grant, takes control of the buses, and returns to its normal operation.

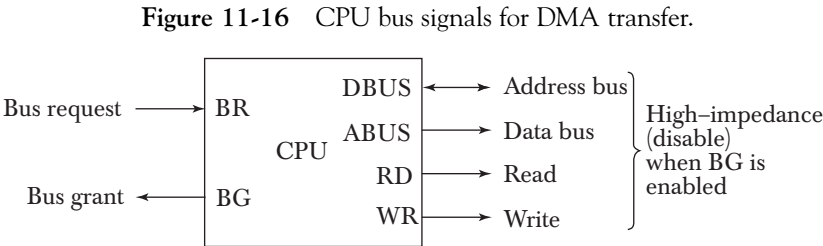
burst transfer

cycle stealing

When the DMA takes control of the bus system, it communicates directly with the memory. The transfer can be made in several ways. In DMA *burst transfer*, a block sequence consisting of a number of memory words is transferred in a continuous burst while the DMA controller is master of the memory buses. This mode of transfer is needed for fast devices such as magnetic disks, where data transmission cannot be stopped or slowed down until an entire block is transferred. An alternative technique called *cycle stealing* allows the DMA controller to transfer one data word at a time, after which it must return control of the buses to the CPU. The CPU merely delays its operation for one memory cycle to allow the direct memory I/O transfer to “steal” one memory cycle.

DMA Controller

The DMA controller needs the usual circuits of an interface to communicate with the CPU and I/O device. In addition, it needs an address register, a word count register, and a set of address lines. The address register and address lines

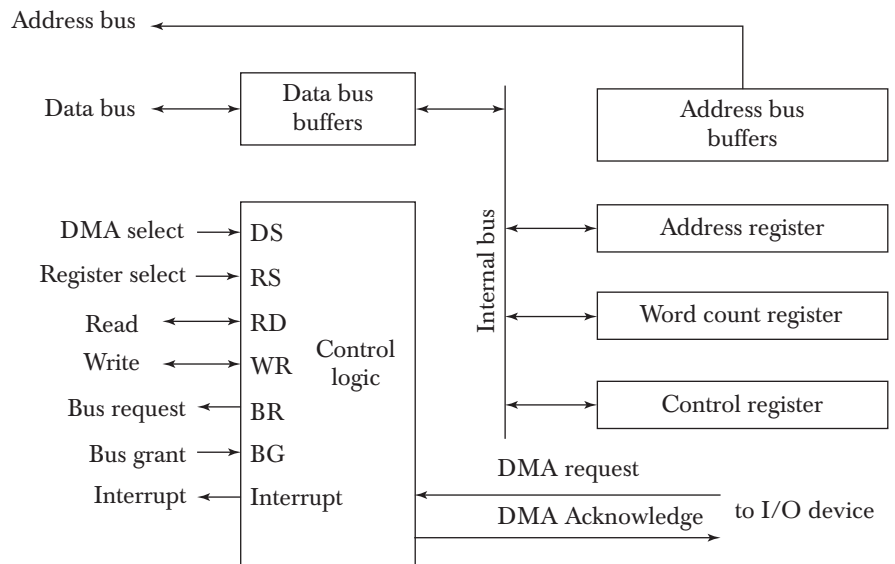


are used for direct communication with the memory. The word count register specifies the number of words that must be transferred. The data transfer may be done directly between the device and memory under control of the DMA.

Figure 11-17 shows the block diagram of a typical DMA controller. The unit communicates with the CPU via the data bus and control lines. The registers in the DMA are selected by the CPU through the address bus by enabling the *DS* (DMA select) and *RS* (register select) inputs. The *RD* (read) and *WR* (write) inputs are bidirectional. When the *BG* (bus grant) input is 0, the CPU can communicate with the DMA registers through the data bus to read from or write to the DMA registers. When *BG* = 1, the CPU has relinquished the buses and the DMA can communicate directly with the memory by specifying an address in the address bus and activating the *RD* or *WR* control. The DMA communicates with the external peripheral through the request and acknowledge lines by using a prescribed handshaking procedure.

The DMA controller has three registers: an address register, a word count register, and a control register. The address register contains an address to specify the desired location in memory. The address bits go through bus buffers into the address bus. The address register is incremented after each word that is transferred to memory. The word count register holds the number of words to be transferred. This register is decremented by one after each word transfer and internally tested for zero. The control register specifies the mode of transfer. All registers in the DMA appear to the CPU as I/O interface registers. Thus the CPU can read from or write into the DMA registers under program control via the data bus.

Figure 11-17 Block diagram of DMA controller.



The DMA is first initialized by the CPU. After that, the DMA starts and continues to transfer data between memory and peripheral unit until an entire block is transferred. The initialization process is essentially a program consisting of I/O instructions that include the address for selecting particular DMA registers. The CPU initializes the DMA by sending the following information through the data bus:

1. The starting address of the memory block where data are available (for read) or where data are to be stored (for write)
2. The word count, which is the number of words in the memory block
3. Control to specify the mode of transfer such as read or write
4. A control to start the DMA transfer

The starting address is stored in the address register. The word count is stored in the word count register, and the control information in the control register. Once the DMA is initialized, the CPU stops communicating with the DMA unless it receives an interrupt signal or if it wants to check how many words have been transferred.

DMA Transfer

The position of the DMA controller among the other components in a computer system is illustrated in Fig. 11-18. The CPU communicates with the DMA through the address and data buses as with any interface unit. The DMA has its own address, which activates the *DS* and *RS* lines. The CPU initializes the DMA through the data bus. Once the DMA receives the start control command, it can start the transfer between the peripheral device and the memory.

When the peripheral device sends a DMA request, the DMA controller activates the *BR* line, informing the CPU to relinquish the buses. The CPU responds with its *BG* line, informing the DMA that its buses are disabled. The DMA then puts the current value of its address register into the address bus, initiates the *RD* or *WR* signal, and sends a DMA acknowledge to the peripheral device. Note that the *RD* and *WR* lines in the DMA controller are bidirectional. The direction of transfer depends on the status of the *BG* line. When $BG = 0$, the *RD* and *WR* are input lines allowing the CPU to communicate with the internal DMA registers. When $BG = 1$, the *RD* and *WR* are output lines from the DMA controller to the random-access memory to specify the read or write operation for the data.

When the peripheral device receives a DMA acknowledge, it puts a word in the data bus (for write) or receives a word from the data bus (for read). Thus the DMA controls the read or write operations and supplies the address for the memory. The peripheral unit can then communicate with memory through the data bus for direct transfer between the two units while the CPU is momentarily disabled.