Modes of Transfer

Data transfer to and from peripherals may be handled in one of three possible modes:
A. Programmed I/O
B. Interrupt-initiated I/O
C. Direct memory access (DMA)

A) Programmed I/O

Programmed I/O operations are the result of I/O instructions written in the computer program. Each data item transfer is initiated by an instruction in the program. Usually, the transfer is to and from a CPU register and peripheral. Other instructions are needed to transfer the data to and from CPU and memory. Once a data transfer is initiated, the CPU is required to monitor the interface to see when a transfer can again be made.

If the speed of an I/O device is in the right range, neither too fast for the processor to read and write the signaling bits nor too slow for the processor to wait for its activity, this form of signaling may be sufficient.

An example of data transfer from an I/O device through an interface into the CPU is shown in Figure below.

![Diagram of data transfer from I/O to CPU]

The following operations will occur:

By device
When a byte of data is available, the device places it in the I/O bus and enables its data valid line.

By interface
Accepts the byte into its data register and enables the data accepted line. Sets F.

By device
Can now disable the data valid line, but it will not transfer another byte until the data accepted line is disabled by the interface. This is according to the handshaking procedure established.
By program
1. Read the status register.
2. Check the status of the flag bit and branch to step 1 if not set or to step 3 if set.
3. Read the data register.

The flag bit is then cleared to 0 by either the CPU or the interface, depending on how the interface circuits are designed.

By interface
Once the flag is cleared, the interface disables the data accepted line and the device can then transfer the next data byte.

Each byte is read into a CPU register and then transferred to memory with a store instruction.

The programmed I/O method is particularly useful in small low-speed computers or in systems that are dedicated to monitor a device continuously. The difference in information transfer rate between the CPU and the I/O device makes this type of transfer inefficient.

B) Interrupt-Initiated I/O
An alternative to the CPU constantly monitoring the flag is to let the interface inform the computer when it is ready to transfer data. This mode of transfer uses the interrupt facility. While the CPU is running a program it does not check the flag. However, when the flag is set, the computer is momentarily interrupted from proceeding with the current program and is informed of the fact that the flag has been set. The CPU deviates from what it is doing to take care of the input or output transfer. After the transfer is completed, the computer returns to the previous program to continue what it was doing before the interrupt.

For each interrupt there is service routine, service routine address must be known by CPU to branch to it. This is accomplished by two methods:

Nonvectored interrupt: the branch address is assigned to a fixed location in memory. Vectored interrupt: the source that interrupts supplies the branch information to the computer. This information is called the interrupt vector. In some computers the interrupt vector is the first address of the I/O service routine. In other computers the interrupt vector is an address that points to a location in memory where the beginning address of the I/O service routine is stored.

There is possibility that several sources will request service simultaneously; in this case the system must also decide which device to service first.

A priority interrupt is a system that establishes a priority over the various sources to determine which condition is to be serviced first when two or requests arrive simultaneously. The system may also determine which conditions are permitted to interrupt the computer while another interrupt is being serviced.

Establishing the priority of simultaneous interrupts can be done by software or hardware. A polling procedure is used to identify the highest-priority source by software means. In this method there is one common branch address for all interrupts. The program that takes care of interrupts begins at the branch address and polls the interrupt sources in sequence. The order in which they are tested determines the priority of each interrupt. The highest-priority source is tested first, and if its interrupt signal is on, control branches to a service routine for this source. Otherwise, the next-lower-priority source is tested, and so on. Thus the initial service routine for all interrupts consists of a program that tests the interrupt sources in sequence and branches to one of many possible service routines. The disadvantage of the software method is that if there are many interrupts, the time required to poll them can exceed the time available to service the I/O device. In this situation a hardware priority-interrupt unit can be used to speed up the operation.
A **hardware priority-interrupt unit** functions as an overall manager in an interrupt system environment. It accepts interrupt requests from many sources, determines which of the incoming requests has the highest priority, and issues an interrupt request to the computer based on this determination. To speed up the operation, each interrupt source has its own interrupt vector to access its own service routine directly. Thus no polling is required because all the decisions are established by the hardware priority-interrupt unit.

C) **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.

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 (1) 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. 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. 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.*
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** (contains an address to specify the desired location in memory, the address register is incremented after each word that is transferred to memory), a **word count register** (specifies the number of words that must be transferred), a set of **address lines**, and a **control register**. The address register and address lines are used for direct communication with the memory.

The position of the DMA controller among the other components in a computer system is illustrated in Figure (1). 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 (DMA select) and RS (register select) lines.

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 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

![Figure (2) DMA transfer in computer system](image-url)
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 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.

For each word that is transferred, the DMA increments its address register and decrements its word count register. If the word count does not reach zero, the DMA checks the request line coming from the peripheral. For a high-speed device, the line will be active, as soon as the previous transfer is completed. A second transfer is then initiated, and the process continues until the entire block is transferred. If the peripheral speed is slower, the DMA request line may come somewhat later. In this case the DMA disable the bus request line so that the CPU can continue to execute its program. When the peripheral requests a transfer, the DMA requests the buses again.

If the word count register reaches zero, the DMA stops any further transfer and removes its bus request. It also informs the CPU of the termination by means of an interrupt. When the CPU responds to the interrupt, it reads the content of the word count register. The zero value of this register indicates that all the words were transferred successfully.

DMA transfer is very useful in many applications. It is used for fast transfer of information between magnetic disks and memory.
Channel I/O

Programmed I/O transfers data one byte at a time. Interrupt-driven I/O can handle data one byte at a time or in small blocks, depending on the type of device participating in the I/O. Slower devices such as keyboards generate more interrupts per number of bytes transferred than disks or printers. DMA methods are all block-oriented, interrupting the CPU only after completion (or failure) of transferring a group of bytes. After the DMA signals the I/O completion, the CPU may give it the address of the next block of memory to be read from or written to. In the event of failure, the CPU is solely responsible for taking appropriate action. Thus, DMA I/O requires only a little less CPU participation than does interrupt-driven I/O. Such overhead is fine for small, single-user systems; however, it does not scale well to large, multi-user systems such as mainframe computers. Most mainframes use an intelligent type of DMA interface known as an I/O channel.

With channel I/O, one or more I/O processors control various I/O pathways called channel paths. Channel paths for "slow" devices such as terminals and printers can be combined (multiplexed), allowing management of several of these devices through only one controller. On IBM mainframes, a multiplexed channel path is called a multiplexer channel. Channels for disk drives and other "fast" devices are called selector channels. I/O channels are driven by small CPUs called I/O processors (IOPs), which are optimized for I/O. Unlike DMA circuits, IOPs have the ability to execute programs that include arithmetic-logic and branching instructions. Figure bellow shows a simplified channel I/O configuration.

IOPs execute programs that are placed in main system memory by the host processor. These programs, consisting of a series of channel command words (CCWs), include not only the actual transfer instructions, but also commands that control the I/O devices. These commands include such things as
various kinds of device initializations, printer page ejects, and tape rewind commands, to name a few. Once the I/O program has been placed in memory, the host issues a start subchannel command (SSCH), informing the IOP of the location in memory where the program can be found. After the IOP has completed its work, it places completion information in memory and sends an interrupt to the CPU. The CPU then obtains the completion information and takes action appropriate to the return codes.

The principal distinction between standalone DMA and channel I/O lies in the intelligence of the IOP. The IOP negotiates protocols, issues device commands, translates storage coding to memory coding, and can transfer entire files or groups of files independent of the host CPU. The host has only to create the program instructions for the I/O operation and tell the IOP where to find them.

Like standalone DMA, an IOP must steal memory cycles from the CPU. Unlike standalone DMA, channel I/O systems are equipped with separate I/O buses, which help to isolate the host from the I/O operation. When copying a file from disk to tape, for example, the IOP uses the system memory. Memory bus used only to fetch its instructions from main memory. The remainder of the transfer is affected using only the I/O bus. Owing to its intelligence and bus isolation, channel I/O is used in high-throughput transaction processing environments, where its cost and complexity can be justified.