Modes of Transfer

- **Binary information** received from an external device is usually stored in memory for later processing. Information transferred from the central computer into an external device originates in the memory unit.

- **The CPU merely** executes the I/O instructions and may accept the data temporarily, but the ultimate source or destination is the memory unit.

- **Data transfer** between the central computer and I/O devices may be handled in a variety of modes.

- **Some modes** use the CPU as an intermediate path; others transfer the data directly to and from the memory unit.

- **Data transfer** to and from peripherals may be handled in one of three possible modes:

```
1. Programmed I/O

2. Interrupt-initiated I/O

3. Direct memory access (DMA)
```

- **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. Transferring data under program control requires constant monitoring of the peripheral by the CPU.

  Once a data transfer is initiated, the CPU is required to monitor the interface to see when a transfer can again be made. It is up to the programmed instructions executed in the CPU to keep close tabs on everything that is taking place in the interface unit and the I/O device.

- **In the programmed** I/O method, the CPU stays in a program loop until the I/O unit indicates that it is ready for data transfer. This is a time-consuming process since it keeps the processor busy needlessly.

  It can be avoided by using an interrupt facility and special commands to inform the interface to issue an interrupt request signal when the data are available from the device. In the meantime the CPU can proceed to execute another program.

The interface meanwhile keeps monitoring the device. When the interface determines that the device is ready for data transfer, it generates an interrupt request to the computer. Upon detecting the external interrupt signal, the CPU momentarily stops the task it is processing, branches to a service program to process the I/O transfer, and then returns to the task it was originally performing.

- **Transfer of data under** programmed I/O is between CPU and peripheral

  In direct memory access (DMA), the interface transfers data into and out of the memory unit through the memory bus. The CPU initiates the transfer by supplying the interface with the starting address and the number of words needed to be transferred and then proceeds to execute other tasks.
  When the transfer is made, the DMA requests memory cycles through the memory bus

- **When the request** is granted by the memory controller, the DMA transfers the data directly into memory. The CPU merely delays its memory access operation to allow the direct memory I/O transfer.

  Since peripheral speed is usually slower than processor speed, I/O-memory transfers are infrequent compared to processor access to memory.

Example of Programmed I/O

In the programmed I/O method, the I/O device does not have direct access to memory.
A transfer from an I/O device to memory requires the execution of several instructions by the CPU, including an input instruction to transfer the data from the device to the CPU and a store instruction to transfer the data from the CPU to memory.

- **Other instructions** may be needed to verify that the data are available from the device and to count the numbers of words transferred.

  An example of data transfer from an I/O device through an interface into the CPU is shown in Fig. 10. The device transfers bytes of data one at a time as they are available.

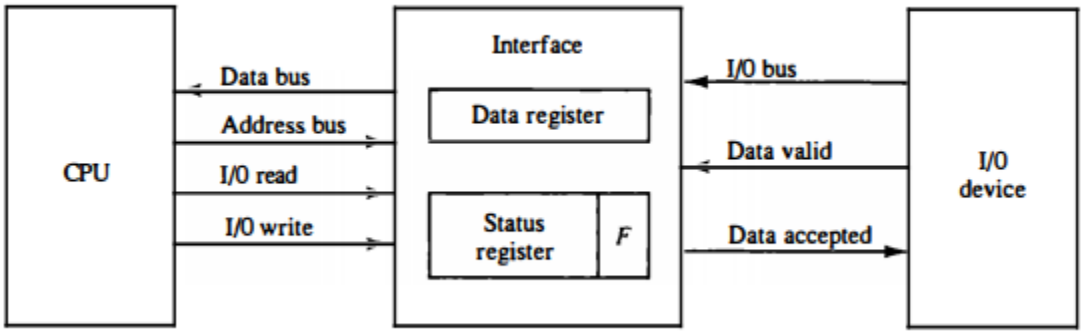
- **When a byte of data** is available, the device places it in the I/O bus and enables its data valid line.
- **The interface accepts** the byte into its data register and enables the data accepted line.
- **The interface sets** a bit in the status register that we will refer to as an F or "flag" bit. The 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 in Fig.
- **A program is written** for the computer to check the flag in the status register to determine if a byte has been placed in the data register by the I/O device.
- **This is done by reading** the status register into a CPU register and checking the value of the flag bit.
- **If the flag is equal** to 1, the CPU reads the data from 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.
- **Once the flag is cleared**, the interface disables the data accepted line and the device can then transfer the next data byte.

- **A flowchart of the program** that must be written for the CPU is shown in Fig. . It is assumed that the device is sending a sequence of bytes that must be stored in memory.
- **The transfer of each byte requires three instructions:**

```
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.
```

- **Each byte** is read into a CPU register and then transferred to memory with a store instruction. A common I/O programming task is to transfer a block of words from an I/O device and store them in a memory buffer.

Figure 10 Data transfer from I/O device to CPU.



F = Flag bit

- **Each byte** is read into a CPU register and then transferred to memory with a store instruction. A common I/O programming task is to transfer a block of words from an I/O device and store them in a memory buffer.

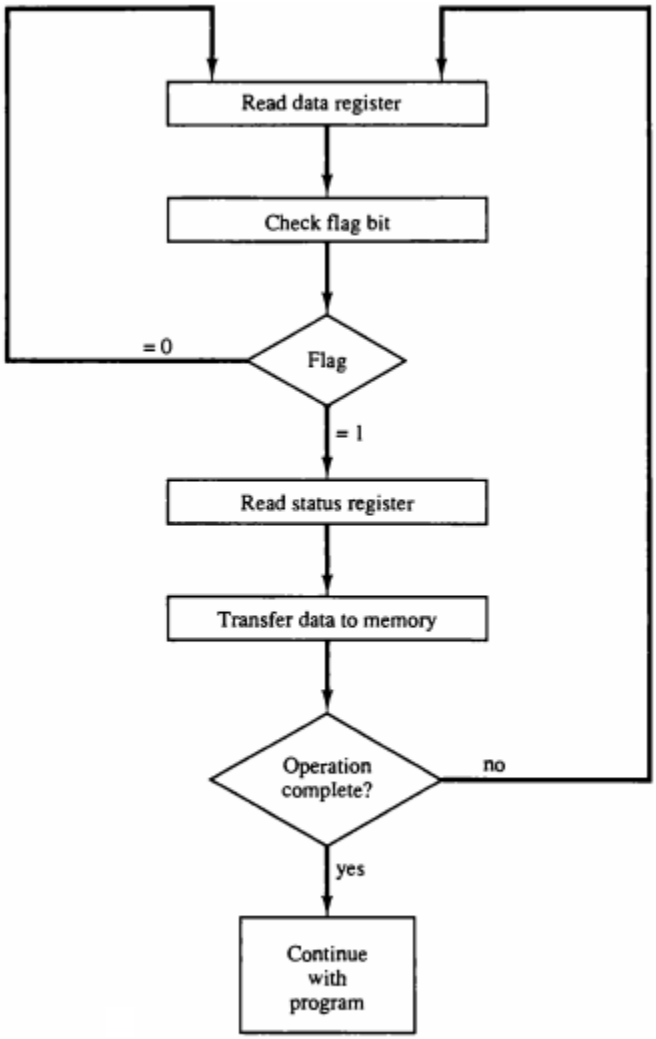


Figure 11 Flowchart for CPU program to input data.

- **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.
- **To see why this** is inefficient, consider a typical computer that can execute the two instructions that read the status register and check the flag in 1 μS.
- **Assume that** the input device transfers its data at an average rate of 100 bytes per second.
- **This is equivalent to one byte** every 10,000 μS.
- **This means** that the CPU will check the flag 10,000 times between each transfer.
- **The CPU is wasting** time while checking the flag instead of doing some other useful processing task.

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.
- **The CPU responds** to the interrupt signal by storing the return address from the program counter into a memory stack and then control branches to a service routine that processes the required I/O transfer.
- **The way that the processor chooses** the branch address of the service routine varies from one unit to another.
- **In principle**, there are two methods for accomplishing this.
- **One is called vectored**interrupt and the other, nonvectored interrupt. In a non vectored interrupt, the branch address is assigned to a fixed location in memory.
- **In a 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.

Priority Interrupt

- **Data transfer** between the CPU and an I/O device is initiated by the CPU. However, the CPU cannot start the transfer unless the device is ready to communicate with the CPU.

- **The readiness** of the device can be determined from an interrupt signal. The CPU responds to the interrupt request by storing the return address from PC into a memory stack and then the program branches to a service routine that processes the required transfer.

- **Some processors** also push the current PSW (program status word) onto the stack and load a new PSW for the service routine.

- **In a typical application** a number of I/O devices are attached to the computer, with each device being able to originate an interrupt request. The first task of the interrupt system is to identify the source of the interrupt.

- **There is also the 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 more requests arrive simultaneously.

- **The system may also** determine which conditions are permitted to interrupt the computer while another interrupt is being serviced.

- **Higher-priority interrupt** levels are assigned to requests which, if delayed or interrupted, could have serious consequences.

- **Devices with high speed** transfers such as magnetic disks are given high priority, and slow devices such as keyboards receive low priority.

- **When two devices** interrupt the computer at the same time, the computer services the device, with the higher priority first.

- **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 wJ;tich 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 particular** service routine reached belongs to the highest-priority device among all devices that interrupted the computer.

- **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.

- **The hardware priority function** can be established by either a serial or a parallel connection of interrupt lines. The serial connection is also known as the daisychaining method.

Daisy-Chaining Priority

- **The daisy-chaining** method of establishing priority consists of a serial connection of all devices that request an interrupt.

- **The device** with the highest priority is placed in the first position, followed by lower-priority devices up to the device with the lowest priority, which is placed last in the chain.

- **This method** of connection between three devices and the CPU is shown in Fig. 12. The interrupt request line is common to all devices and forms a wired logic connection.

- **If any device** has its interrupt signal in the low-level state, the interrupt line goes to the low-level state and enables the interrupt input in the CPU.

- **When no interrupts** are pending, the interrupt line stays in the high-level state and no interrupts are recognized by the CPU.

- **This is equivalent** to a negativelogic OR operation. The CPU responds to an interrupt request by enabling the interrupt acknowledge line. This signal is received by device 1 at its PI (priority in) input.

- **The acknowledge** signal passes on to the next device through the PO (priority out) output only if device 1 is not requesting an interrupt.

- **If device 1 has a pending interrupt**, it blocks the acknowledge signal from the next device by placing a 0 in the PO output.
- **It then proceeds** to insert its own interrupt vector address (VAD) into the data bus for the CPU to use during the interrupt cycle.
- **A device** with a 0 in its PI input generates a 0 in its PO output to inform the next-lower-priority device that the acknowledge signal has been blocked.
- **A device** that is requesting an interrupt and has a 1 in its PI input will intercept the acknowledge signal by placing a 0 in its PO output.
- **If the device** does not have pending interrupts, it transmits the acknowledge signal to the next device by placing a 1 in its PO output.
- **If the device** does not have pending interrupts, it transmits the acknowledge signal to the next device by placing a 1 in its PO output.

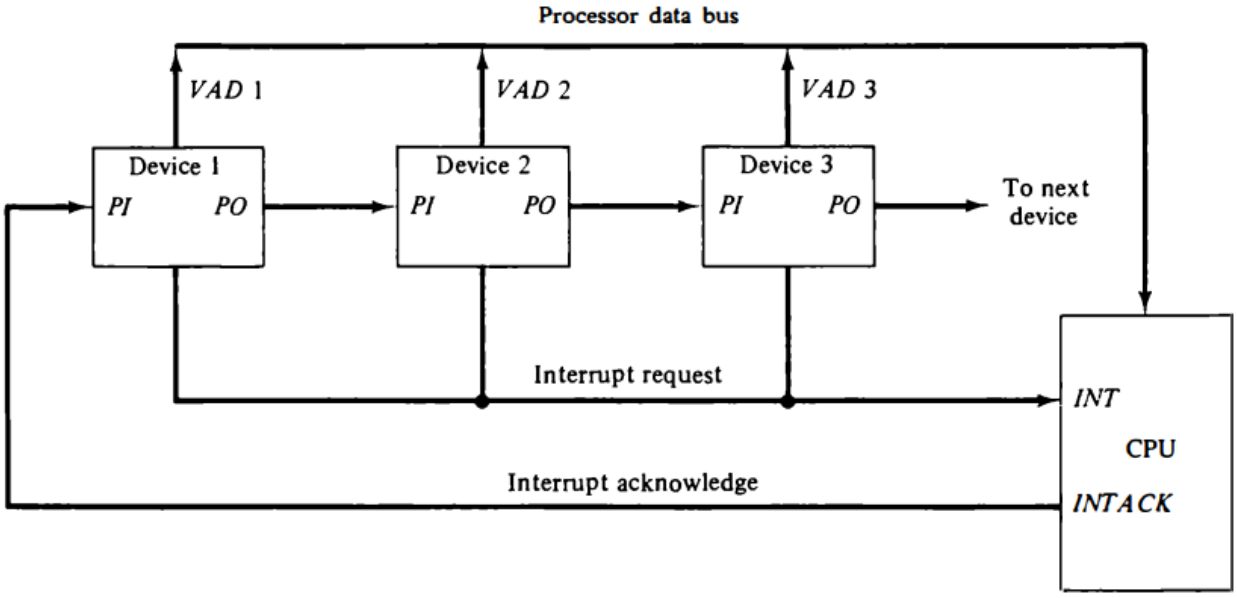


Figure 12 Daisy-chain priority interrupt.

- **Thus the device** with PI = 1 and PO = 0 is the one with the highest priority that is requesting an interrupt, and this device places its VAD on the data bus. **The daisy chain** arrangement gives the highest priority to the device that receives the interrupt acknowledge signal from the CPU.
- **The farther** the device is from the first position, the lower is its priority.
- **Figure 13** shows the internal logic that must be included within each device when connected in the daisy-chaining scheme. The device sets its RF flip-flop when it wants to interrupt the CPU.
- **The output of the RF flip-flop** goes through an open-collector inverter, a circuit that provides the wired logic for the common interrupt line.
- **If PI = 0, both PO** and the enable line to VAD are equal to 0, irrespective of the value of RF. If PI = 1 and RF = 0, then PO = 1 and the vector address is disabled.
- **This condition** passes the acknowledge signal to the next device through PO.

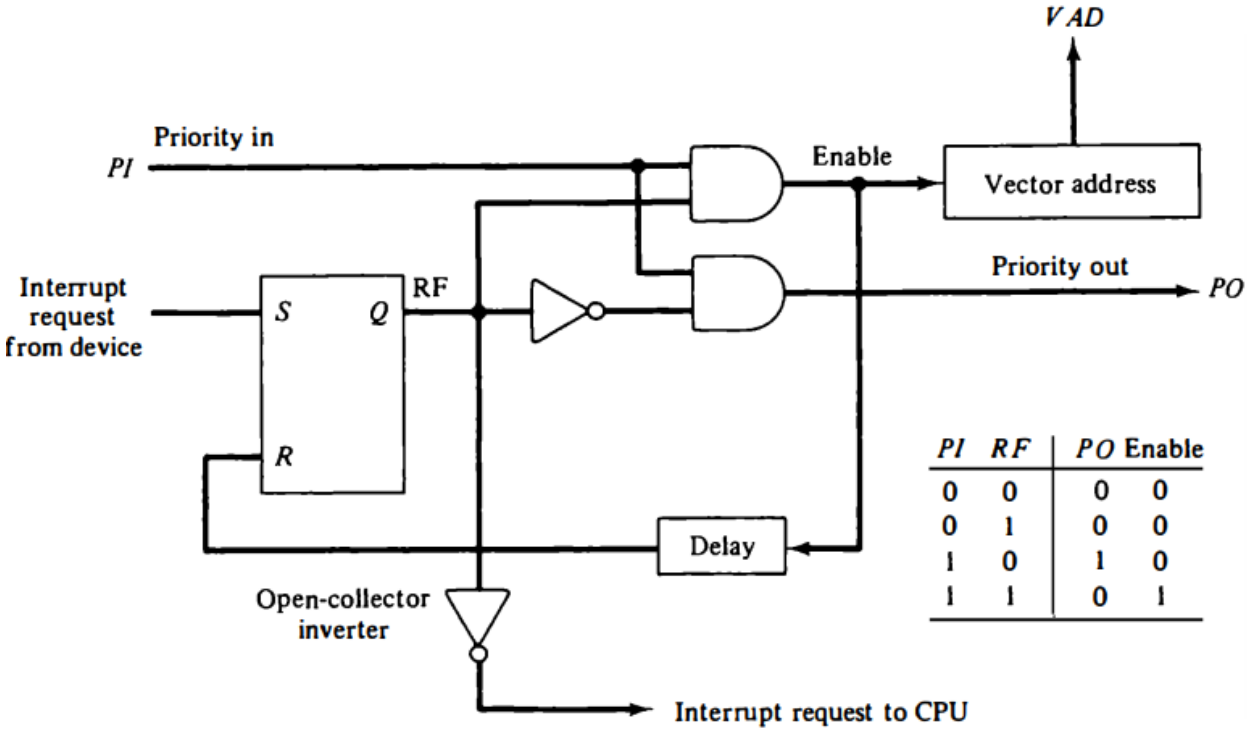
  The device is active when PI = 1 and RF = 1. This condition places a 0 in PO and enables the vector address for the data bus.
- **It is assumed** that each device has its own distinct vector address. The RF flip-flop is reset after a sufficient delay to ensure that the CPU has received the vector address.

Parallel Priority Interrupt

- **The parallel priority** interrupt method uses a register whose bits are set separately by the interrupt signal from each device.

- **Priority** is established according to the position of the bits in the register.

- **In addition** to the interrupt register, the circuit may include a mask register whose purpose is to control the status of each interrupt request.



- **Figure 13** One stage of the daisy-chain priority arrangement.

- **The mask register** can be programmed to disable lower-priority interrupts while a higher-priority device is being serviced.
- **It can also provide** a facility that allows a high-priority device to interrupt the CPU while a lower-priority device is being serviced.
- **The priority logic** for a system of four interrupt sources is shown in Fig. 14.
- **It consists** of an interrupt register whose individual bits are set by external conditions and cleared by program instructions.
- **The magnetic disk,** being a high-speed device, is given the highest priority. The printer has the next priority, followed by a character reader and a keyboard.
- **The mask register** has the same number of bits as the interrupt register. By means of program instructions, it is possible to set or reset any bit in the mask register.

- **Each interrupt bit** and its corresponding mask bit are applied to an AND gate to produce the four inputs to a priority encoder.
- **In this way** an interrupt is recognized only if its corresponding mask bit is set to 1 by the program. The priority encoder generates two bits of the vector address, which is transferred to the CPU.
- **Another output** from the encoder sets an interrupt status flip-flop IST when an interrupt that is not masked occurs. The interrupt enable flip-flop IEN can be set or cleared by the program to provide an overall control over the interrupt system.
- **The outputs** of IST ANDed with IEN provide a common interrupt signal for the CPU.
- **The interrupt** acknowledge INTACK signal from the CPU enables the bus buffers in the output register and a vector address VAD is placed into the data bus.

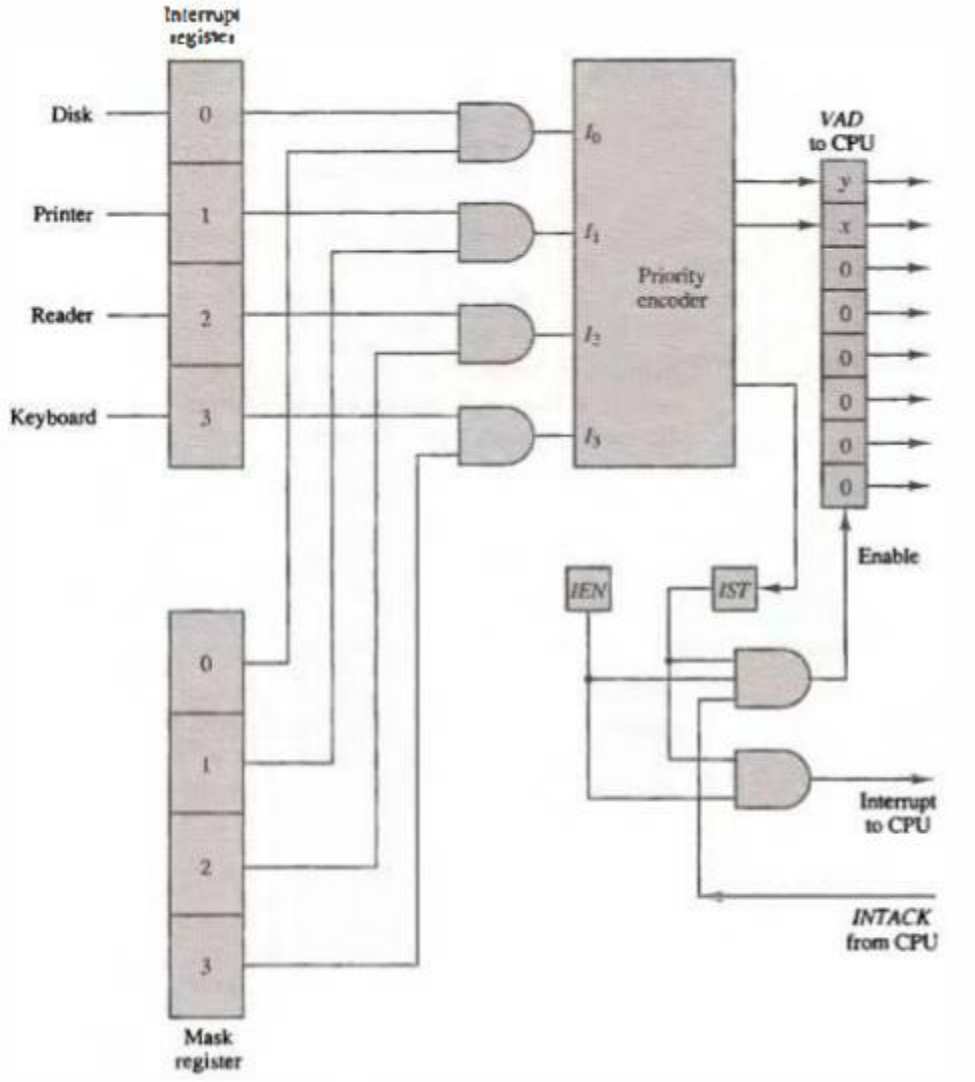


Figure 14 Priority interrupt hardware.

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

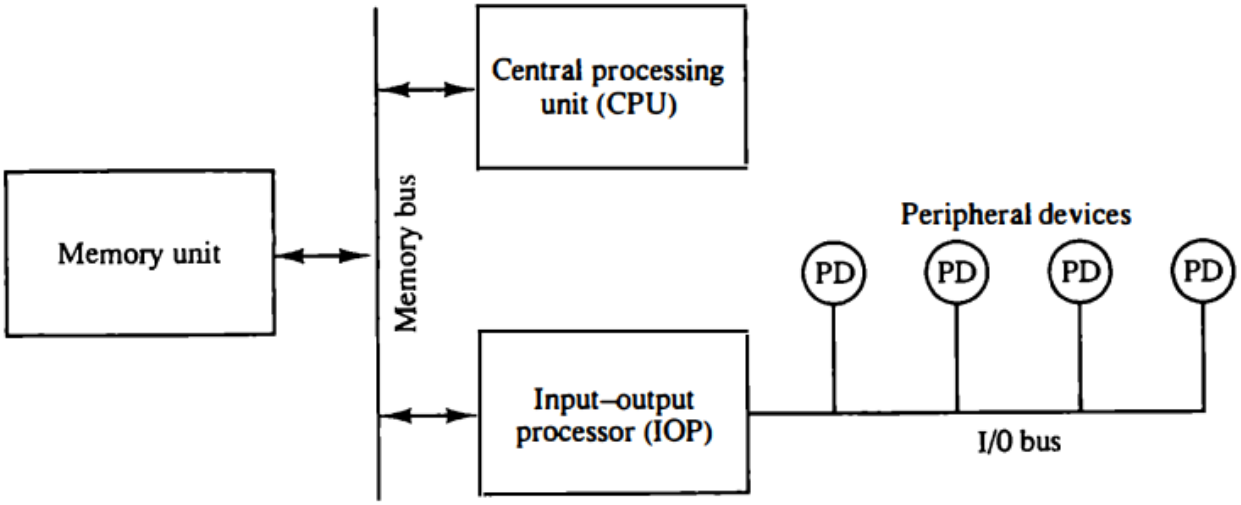


Figure 19 Block diagram of a computer with I/O processor.

- **The data formats** of peripheral devices differ from memory and CPU data formats. The IOP must structure data words from many different sources.
- **For example,** it may be necessary to take four bytes from an input device and pack them into one 32-bit word before the transfer to memory.
- **Data are gathered** in the IOP at the device rate and bit capacity while the CPU is executing its own program.

- **After the input** data are assembled into a memory word, they are transferred from IOP directly into memory by "stealing" one memory cycle from the CPU.
- **Similarly**, an output word transferred from memory to the lOP is directed from the IOP to the output device at the device rate and bit capacity.
- **The communication** between the IOP and the devices attached to it is similar to the program control method of transfer. Communication with the memory is similar to the direct memory access method.
- **The way by which the CPU and IOP** communicate depends on the level of sophistication included in the system.

In very-large-scale computers, each processor is independent of all others and any one processor can initiate an operation. In most computer systems, the CPU is the master while the IOP is a slave processor.

- **The CPU is assigned** the task of initiating all operations, but 110 instructions are executed in the IOP.
- **CPU instructions** provide operations to start an 110 transfer and also to test 110 status conditions needed for making decisions on various 110 activities.
- **The IOP**, in turn, typically asks for CPU attention by means of an interrupt.
- **It also responds** to CPU requests by placing a status word in a prescribed location in memory to be examined later by a CPU program.
- **When an 110 operation** is desired, the CPU informs the IOP where to find the 110 program and then leaves the transfer details to the IOP.
- **Instructions** that are read from memory by an IOP are sometimes called commands, to distinguish them from instructions that are read by the CPU. Otherwise, an instruction and a command have similar functions.
- **Commands** are prepared by experienced programmers and are stored in memory. The command words constitute the program for the IOP.
- **The CPU** informs the IOP where to find the commands in memory when it is time to execute the 110 program.

CPU-IOP Communication

- **The communication** between CPU and IOP may take different forms, depending on the particular computer considered.

- **In most cases** the memory unit acts as a message center where each processor leaves information for the other.

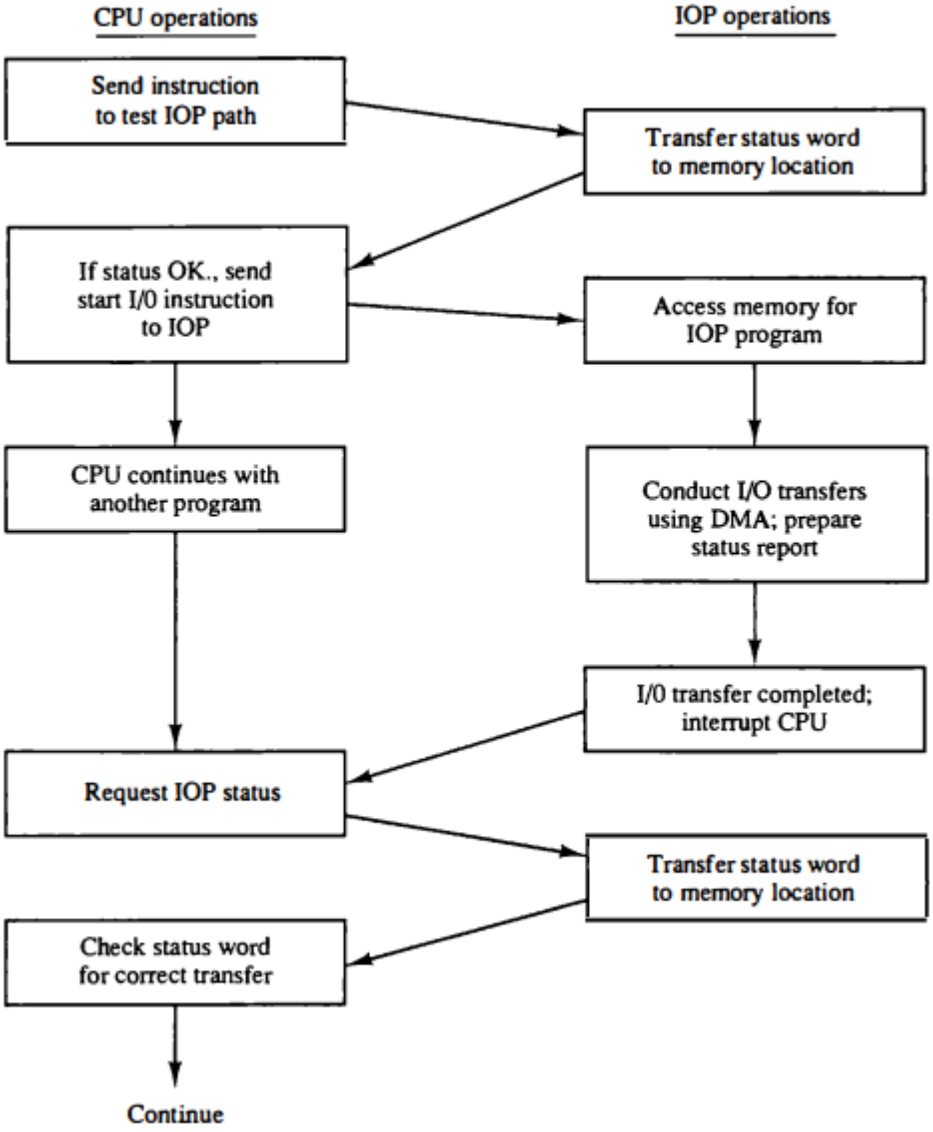
- **To appreciate the operation** of a typical IOP, we will illustrate by a specific example the method by which the CPU and IOP communicate.

- **This is a simplified** example that omits many operating details in order to provide an overview of basic concepts.

- **The sequence** of operations may be carried out as shown in the flowchart of Fig. 20.

- **The CPU** sends an instruction to test the IOP path.

Figure 20 CPU-IOP communication.



- **The IOP responds** by inserting a status word in memory for the CPU to check.

- **The bits of the status word** indicate the condition of the IOP and I/O device, such as IOP overload condition, device busy with another transfer, or device ready for I/O transfer.

- **The CPU refers** to the status word in memory to decide what to do next. If all is in order, the CPU sends the instruction to start I/O transfer.

- **The memory address** received with this instruction tells the IOP where to find its program.

- **The CPU can now continue** with another program while the IOP is busy with the I/O program. Both programs refer to memory by means of DMA transfer.
- **When the IOP terminates** the execution of its program, it sends an interrupt request to the CPU. The CPU responds to the interrupt by issuing an instruction to read the status from the IOP.

- **The IOP responds** by placing the contents of its status report into a specified memory location. The status word indicates whether the transfer has been completed or if any errors occurred during the transfer.

- **From inspection** of the bits in the status word, the CPU determines if the I/O operation was completed satisfactorily without errors.

- **The IOP takes** care of all data transfers between several I/O units and the memory while the CPU is processing another program.

- **The IOP and CPU** are competing for the use of memory, so the number of devices that can be in operation is limited by the access time of the memory.

- **It is not possible** to saturate the memory by I/O devices in most systems, as the speed of most devices is much slower than the CPU.

- **However**, some very fast units, such as magnetic disks, can use an appreciable number of the available memory cycles.

- **In that case**, the speed of the CPU may deteriorate because it will often have to wait for the IOP to conduct memory transfers.