

8085 - Different methods of data transfer

Microprocessor and Micro-Controller

Course Material – EC 403

By

**Er. Mrs. Saraswati Saha,
Assistant Professor, ECE Department,
RCCIIT, Kolkata**



EC-403

Saraswati Saha
Asst. Prof, FCE, RCCIT

Different methods of Data Transfer in 8085 μ p

Data transfer is essential in any micro-processor-based system. It can take place between the processor and the memory, the processor and an input/output device, or the memory and an I/O device.

The data transfer mechanism is divided into the following types:

• Based on the addressing of the device:

(i) I/O-mapped I/O access (ii) Memory-mapped I/O access.

• Based on the amount of data transferred:

(i) programmed data transfer

(a) polled mode of data transfer (b) Interrupt-driven data transfer

(ii) Direct memory access (DMA)

(a) Burst mode

(b) Cycle stealing mode.

• Based on the method of data transfer and interaction among the devices:

(i) Parallel data transfer

(a) simple data transfer

(b) Handshake mode data transfer

(ii) Serial data transfer

(a) Synchronous data transfer

(b) Asynchronous data transfer.

Questions

Q.1. write the steps or sequence of events in a typical DMA operation. ALSO write the function of HOLD and HLD pin in this context. What is difference between Burst mode & Cycle stealing mode?

Q.2. write a short note on (a) Asynchronous & Synchronous data transfer. (b) Serial & parallel data transfer scheme.

Date:

①

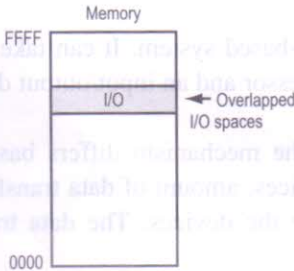
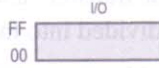
Signature: ^{cache.} 19/2/20

5.2 MEMORY-MAPPED AND I/O-MAPPED DATA TRANSFER

In I/O-mapped device data transfer method, I/O devices and memory are handled separately. A separate address range is assigned for input/output devices. Separate control signals are used for memory access and for I/O device read/write operation. The microprocessor has separate instructions for input and output device access, such as the IN and OUT instructions of the 8085. As memory and I/O device accesses are governed by separate control signals, a single address can be assigned to both an I/O device and a memory location.

In memory-mapped I/O, each input/output device is treated like a memory location. The same control signals are used for I/O device read/write operation and for memory access. Each input or output device is identified by a unique address in the memory address range. All memory-related instructions that are used to read data from the memory are used to access input and output devices. Since the I/O devices use some of the memory address space, the maximum memory addressing capacity is reduced in this system. Table 5.1 compares the two types of data transfer.

Table 5.1 Differences between memory-mapped I/O and I/O-mapped I/O data transfer

Memory-mapped I/O	I/O-mapped I/O
Same address range to address memory and I/O devices In 8085, 16 bit addresses are used.	Different address range for memory and I/O devices The memory map of 8085 is used only by the memory which means all the 64K addresses (0000H to FFFFH) in 8085 can be assigned for memory chips such as RAM and EPROM. The I/O devices are assigned a separate I/O address space which consists of addresses 00H to FFH.
	
Same control signals for memory access is used to select I/O devices. The memory-related control signal \overline{MEMR} is used to read data from memory and also from input device. The memory-related control signal \overline{MEMW} is used to write data into memory and also into the output device.	Separate control signals are available from the CPU for I/O devices selection. The memory-related control signal \overline{MEMR} is used to read data only from the memory and I/O-related control signal \overline{IOR} is used to read data from the input device. The memory-related control signal \overline{MEMW} is used to write data into the memory and the I/O-related control signal \overline{IOW} is used to write data into the output device.

②

(Contd)

Table 5.1 Differences between memory-mapped I/O and I/O-mapped I/O data transfer (Contd)

Memory-mapped I/O	I/O-mapped I/O
Access to the I/O devices using regular memory access instructions. In 8085, data transfer between any general-purpose register and I/O port is possible.	Uses a special class of CPU instructions to access I/O devices such as IN and OUT. In 8085, data transfer is between accumulator and I/O port only.
More number of I/O devices can be accessed as there are more memory address spaces	Fewer I/O devices can be interfaced as the I/O address space is limited compared to the memory.

5.3 PROGRAMMED DATA TRANSFER

The instructions for programmed data transfer are written and controlled by the programmer and executed by the processor. The data transfer between the processor and I/O devices (and vice versa) takes place by executing the corresponding instructions. Programmed I/O data transfers are identical to read and write operations for memories and device registers. An example of programmed I/O is a device driver writing one data byte at a time directly into the device's memory.

Programmed data transfer can take place at a time determined by the programmer. Based on the time of execution of the data transfer instruction, programmed data transfer is divided into two types—polled mode of data transfer and interrupt-driven data transfer.

In polled mode of data transfer, data is read from an input device when the processor or CPU is ready. The processor then executes the data transfer instruction. If the input device is not ready, the processor waits until the device is ready with data. Similarly, data is written into an output device by the processor when it executes the write instruction corresponding to that output device. The program is written in such a way that the processor waits in a loop until the output device is ready to receive data. Clearly, in waiting for the device to be ready, processor time is wasted in this mode of data transfer.

In interrupt-driven data transfer, data is read from the input device only when it is ready with data. When the device is ready, it gives an interrupt signal to the processor, indicating that the data is ready. In the interrupt service routine (ISR), a program is executed to read the data from the corresponding input device. Similarly, the output device gives an interrupt to the processor when it can accept data. The programmers have to write an ISR for data transfer to the corresponding output device. Interrupt-driven data transfer is advantageous as data transfer is done only when the device is ready; the processor need not wait until the device is ready. The processor can execute some other main routines while the data transfer program is executed as an ISR. It is an efficient technique because processor time is not wasted in waiting while an I/O device is getting ready or not ready. Slow I/O devices can be interfaced for data transfer using the interrupt-driven technique.

5.4 DIRECT MEMORY ACCESS

In programmed I/O data transfer, the processor is actively involved in the entire data transfer process. So, the data transfer rate is limited. The processor is tied up and processor time is wasted. To overcome these disadvantages, the direct memory access (DMA) method of data transfer is used.

Direct memory access is a technique to transfer data between the peripheral I/O devices and the memory, without the intervention of the processor. The basic idea is to transfer *blocks of data* directly between the memory and the peripherals. Even though the transfer is done without the processor, the processor initiates the DMA operation. This technique is generally used to transfer large blocks of data between memory and I/O. During DMA data transfer, the processor/CPU is kept in an idle suspended state known as Hold state. DMA performs high-speed data transfer to and from mass storage peripheral devices such as hard disk drives, magnetic tapes, CD ROMs, and video controllers. A hard disk may have a transfer rate of 5 Mbps, that is, one byte every 200 ns. Performing such data transfer using the CPU is not only undesirable but also unnecessary, since the CPU transfer rate is limited by the speed of the memory and peripheral devices.

Under normal circumstances, the CPU has full control of the address and data buses in the system. When direct memory access occurs, an external device or DMA controller takes over the temporary control of the system bus from the CPU. The CPU writes necessary control words into the DMA controller, to indicate the following details about the data transfer: read or write operation, device address involved, starting address of the data memory block and the amount of data to be transferred. After this initialization, the DMA controller takes care of the data transfer. In the 8085, the hold request is received and acknowledged using the HOLD and HLDA pins, respectively.

The sequence of events in a typical DMA process is as follows:

- (i) The peripheral or the DMA controller asserts one of the request pins (e.g., HOLD) for holding the processor.
- (ii) The processor completes its current instruction and enters into the Hold state. In the Hold state, the processor temporarily stops the execution of the instruction and releases the address and data buses by making them enter into a high impedance state.
- (iii) The processor issues a Hold Acknowledge (HLDA) signal to indicate the release of bus control to the peripheral or the DMA controller.
- (iv) The DMA operation starts.
- (v) Upon completion of the DMA operation, the peripheral or the DMA controller removes the Hold signal applied to the processor and relinquishes bus control.

In general, a DMA controller can interface several peripherals that may request DMA with the processor. It is the controller that decides the priority of DMA requests that are received simultaneously from many peripherals. It then communicates with the peripheral device and the processor, and provides memory addresses for transferring data. The 8237 programmable DMA controller is the

controller device that is most commonly used with the 8085 and 8088. It is a four-channel device, with each channel being dedicated to a particular peripheral device. In addition, each channel is capable of addressing 64 KB of memory.

DMA data transfer can be divided into two types:

- (i) Burst or block transfer mode
- (ii) Cycle stealing or interleaved mode

In burst mode of DMA data transfer, a complete block of data is transferred in a single DMA cycle. The system bus is released by the peripheral or DMA controller only after the required bytes of data are transferred. In cycle stealing mode of data transfer, a block of data is transferred over many DMA cycles. The system bus is released to the processor after a byte or a set of bytes are transferred in one DMA cycle. Thus, the processor is not suspended from its activities for a long time. It takes several DMA cycles to complete the transfer of one block of data.

5.5 PARALLEL DATA TRANSFER

In parallel mode of data transfer, all the bits in a word are simultaneously transmitted. Since the 8085 word consists of eight bits, all the eight bits are transmitted and received in parallel form. In some special cases, the number of data bits transferred will be lesser than eight. In general, parallel data transfer is used for transfer of data over short distances such as within a system, within a printed circuit board (PCB), etc. It can be done either in polled mode or in interrupt-driven mode. In polled method, data is read from the input device by the processor at a time determined by the processor. This polled mode of data transfer can be done in two ways—synchronous or simple I/O and handshake I/O.

In simple or synchronous mode, data is read from the input device by the processor irrespective of the status of the input device. It is assumed that the input device is in synchronism with the processor and that it is ready with data whenever the processor reads the data. Similarly, the data is written into the output device irrespective of its status. The processor assumes that the output device is in synchronism with the processor.

In handshake I/O mode, the processor checks for the status of the I/O devices before data transfer. An input device gives a signal to the processor, indicating that it is ready with the data. The processor checks continuously for the reception of this signal and upon reception can read the data. Similarly, an output device gives a signal to the processor, indicating that it can accept data. The processor, before writing data to the output device, checks for this signal. If the signal indicating readiness of the output device is available, the processor can write the data to the output device. The signals that are transferred between the devices and the processor are called handshake signals.

5.6 SERIAL DATA TRANSFER

Parallel data transfer has the drawback of needing several wires to transfer all the bits of data. So, it can not be used effectively for long distance transfers. As one

wire is used for each bit, byte-wise data transfers are eight times more expensive than a single bit transfer. Serial data transfer is the solution for data transfers over long distances. It is a low-cost way to send data over long distances. In serial data transfer, only one bit is transferred over a data transfer line. All the bits in a data word can be transmitted by using a shift register and transferring the data bit by bit. Parallel-to-serial data conversion is done by a device called Universal Asynchronous Receiver-Transmitter (UART).

In serial data transfer, the three aspects are important—first, the speed or frequency at which the bits are transmitted into the serial data line. The frequency at which the data is transmitted serially is technically called baud rate. Baud rate is the measure of the number of bits transmitted over a second. Second, the mode of data transfer—serial data transfer can be done in two modes (synchronous and asynchronous). Third, the voltage levels for logic 1 and logic 0 for the data being transmitted plays a role. Various serial communication protocols define these aspects as standards for proper communication.

In synchronized data transfer, the device that sends the data and the device that receives the data are synchronized with the common clock. In synchronous mode, data transfer takes place with a fixed and known time frame. In asynchronous data transfer, data words are transmitted with a random time frame between them. Most microprocessor- and computer-related data communications are based on the asynchronous mode of transmission. Microprocessors use interrupts and other software techniques to synchronize random timing between data words, so as to receive the data completely.

The modem plays an important role in serial transmission. It is a device that allows transmission of serial data over communication lines such as telephone lines. In general, communication lines are incapable of carrying the voltage changes required for a direct digital connection. A modem overcomes this limitation by modulating digital information into analog signals using one of the modulation techniques and demodulating it back into digital information upon reception.

The computer or a microprocessor terminal that initiates the serial communication is called *data terminal equipment* (DTE). The final equipment that receives the serial data is also called data terminal equipment. *Data communication equipment* (DCE) is a device that connects the DTE to a transmission line. So, the transmitting DTE sends the serial data to the DCE. The DCE is generally a modem. This helps in level shifting and transmitting the serial data over the chosen transmission line. Similarly, at the receiving station, a DCE (generally a modem) receives the signal and transfers the serial data to the receiving DTE.

This section introduces the RS-232, RS-485, General-Purpose Interface Bus (GPIB), and IEEE 488 standards, which are used for data transfer between two computer or processor systems. RS-232 is a common serial communication protocol used in computer systems.

⑥

5.6.1 Introduction to RS-232 Standard

RS-232 is a serial communication standard given by the Electronic Industries Association (EIA), an organization represented by a group of electronic industries.