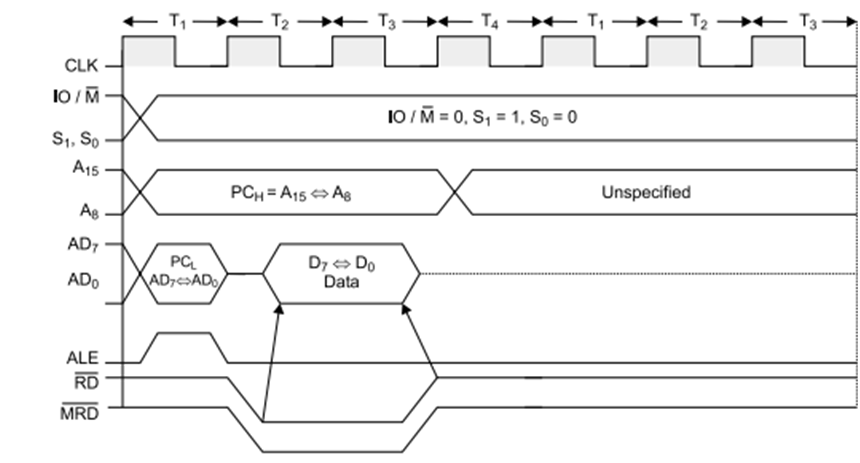
**CHAPTER 5**

Memory Read:

Operation:

 It is used to fetch one byte from the memory.

 It requires 3 T-States.

 It can be used to fetch operand or data from the memory.

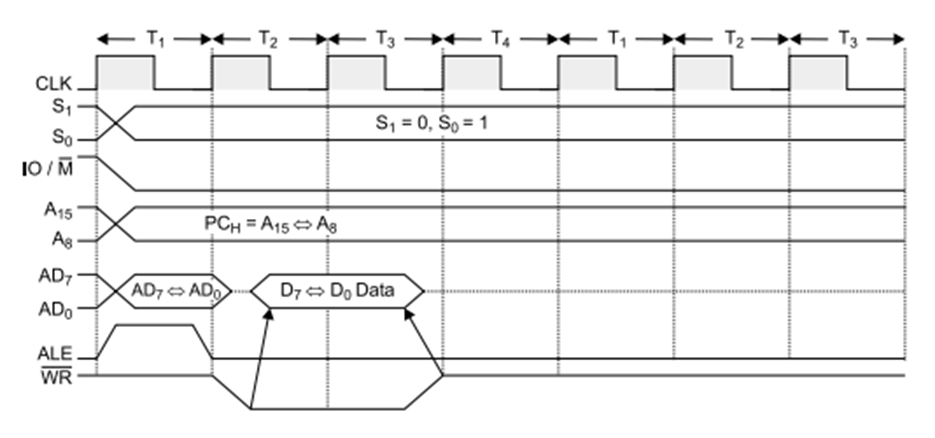
 During T1, A8-A15 contains higher byte of address. At the same time ALE is high. Therefore Lower byte of address A0-A7 is selected from AD0-AD7.

 Since it is memory ready operation, IO/M(bar) goes low.

 During T2 ALE goes low, RD(bar) goes low. Address is removed from AD0-AD7 and data D0-D7 appears on AD0-AD7.

 During T3, Data remains on AD0-AD7 till RD(bar) is at low signal.

Memory Write:



Operation:

 It is used to send one byte into memory.

 It requires 3 T-States.

 During T1, ALE is high and contains lower address A0-A7 from AD0-AD7.

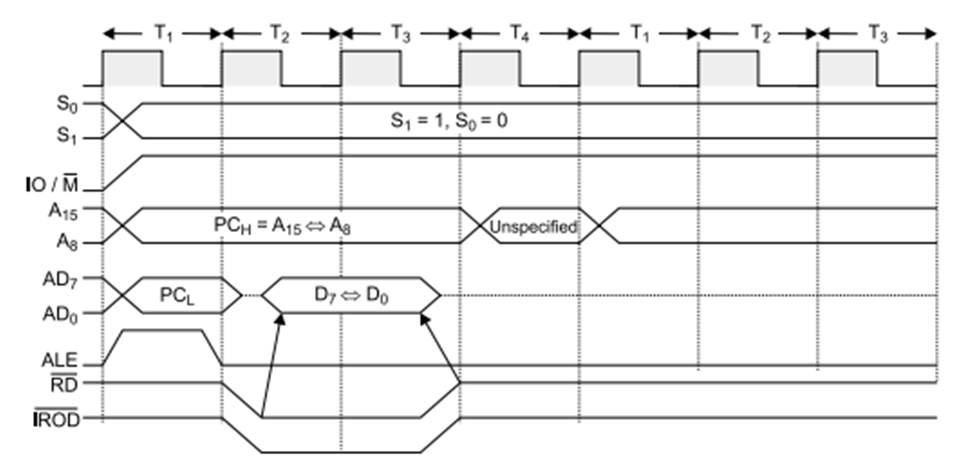
 A8-A15 contains higher byte of address.

 As it is memory operation, IO/M(bar) goes low.

 During T2, ALE goes low, WR(bar) goes low and Address is removed from AD0-AD7 and then data appears on AD0-AD7.

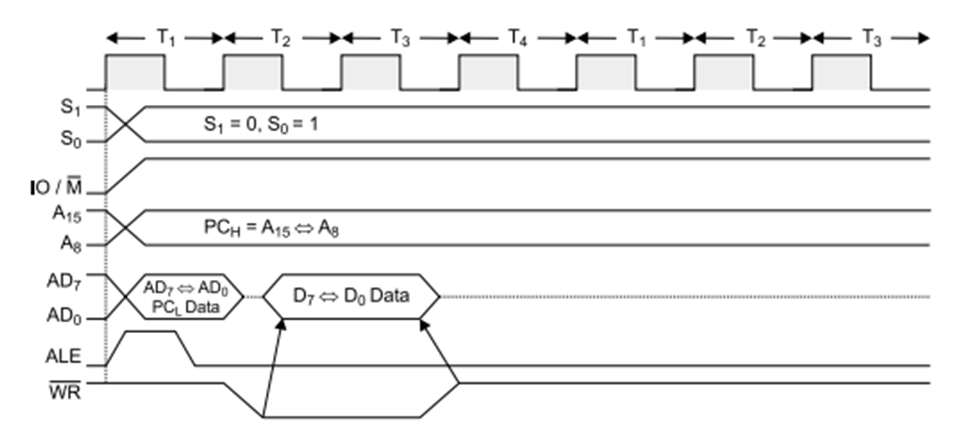
 Data remains on AD0-AD7 till WR(bar) is low.

IO Read:



**Operation:**  
a. It is used to fetch one byte from an IO port.  
b. It requires 3 T-States.  
c. During T1, The Lower Byte of IO address is duplicated into higher order address bus A8-A15.  
d. ALE is high and AD0-AD7 contains address of IO device.  
e. IO/M (bar) goes high as it is an IO operation.  
f. During T2, ALE goes low, RD (bar) goes low and data appears on AD0-AD7 as input from IO device.  
g. During T3 Data remains on AD0-AD7 till RD(bar) is low.

**IO Write:**



Operation:

 It is used to writ one byte into IO device.

 It requires 3 T-States.

 During T1, the lower byte of address is duplicated into higher order address bus A8-A15.

 ALE is high and A0-A7 address is selected from AD0-AD7.

 As it is an IO operation IO/M (bar) goes low.

 During T2, ALE goes low, WR (bar) goes low and data appears on AD0-AD7 to write data into IO device.

 During T3, Data remains on AD0-AD7 till WR(bar) is low.

**Direct Memory Access (DMA)**

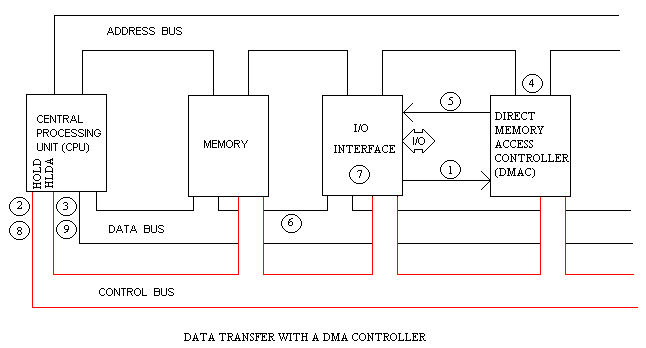
During any given bus cycle, one of the system components connected to the system bus is given control of the bus. This component is said to be the master during that cycle and the component it is communicating with is said to be the slave. The CPU with its bus control logic is normally the master, but other specially designed components can gain control of the bus by sending a bus request to the CPU. After the current bus cycle is completed the CPU will return a bus grant signal and the component sending the request will become the master.

Taking control of the bus for a bus cycle is called cycle stealing. Just like the bus control logic, a master must be capable of placing addresses on the address bus and directing the bus activity during a bus cycle.

The 8085 microprocessor receives bus requests through its HOLD pin and issues grants from the hold acknowledge (HLDA) pin. A request is made when a potential master sends a 1 to the HOLD pin. Normally, after the current bus cycle is complete the 8085 will respond by putting a 1 on the HLDA pin. When the requesting device receives this grant signal it becomes the master. It will remain master until it drops the signal to the HOLD pin, at which time the 8085 will drop the grant on the HLDA pin.

When a DMA controller becomes master it places an address on the address bus and sends the interface the necessary signals to cause it to put data on, or receive data from, the data bus. Since the DMA controller determines when the bus request is dropped, it can return control to the CPU after each data byte is transferred and then request control again when the next data byte is ready, or it can retain control until the entire block is moved.

DMA Block Transfer



During a block input byte transfer, the following sequence occurs as the data byte is sent from the interface to the memory:

1. The interface sends the DMA controller a request for DMA service.
2. A Bus request is made to the HOLD pin (active High) on the 8085 microprocessor  and the controller gains control of the bus.
3. A Bus grant is returned to the DMA controller from the Hold Acknowledge (HLDA) pin (active High) on the 8085 microprocessor.
4. The  DMA controller places contents of the address register onto the address bus.
5. The controller sends the interface a DMA acknowledgment, which tells the interface to put data on the data bus.
6. The data byte is transferred to the memory location indicated by the address bus.
7. The interface handle the data.
8. The Bus request is dropped, the HOLD pin goes Low, and the controller relinquishes the bus.
9. The Bus grant from the 8085 microprocessor is dropped and the HLDA pin goes Low.

**Interrupt**

 Interrupt is a process where an external device can get the attention of the microprocessor. The process starts from the I/O device. An interrupt is considered to be an emergency signal that may be serviced. The Microprocessor may respond to it as soon as possible.

 When the Microprocessor receives an interrupt signal, it suspends the currently executing program and jumps to an Interrupt Service Routine (ISR) to respond to the incoming interrupt. Each interrupt will most probably have its own ISR.

 Responding to an interrupt may be immediate or delayed depending on whether the interrupt is maskable or non-maskable and whether interrupts are being masked or not. There are two ways of redirecting the execution to the ISR depending on whether the interrupt is vectored or non-vectored.

 **Vector Interrupt**: In this type of interrupt, Processor knows the address of Interrupt. In other word processor knows the address of interrupt service routine.

The examples of vector interrupt are RST 7.5, RST 6.5, RST 5.5, TRAP.

 **Non-Vector Interrupt**: In this type of interrupt, Processor cannot know the address of Interrupt. It should give externally. In this, device will have to send the address of interrupt service routine to processor for performing Interrupt.

The example of Non-vector interrupt is INTR.

 When a device interrupts, it actually wants the MP to give a service which is equivalent to asking the MP to call a subroutine. This subroutine is called ISR (Interrupt Service Routine).

**Software Interrupt**: It is an instruction based Interrupt which is completely controlled by software. That means programmer can use this instruction to execute interrupt in main program.

There are eight software interrupts available in 8085 microprocessor. See the example with their hex code and vector address.



**Hardware Interrupt**: As name suggests it is interrupt which can get the interrupt request in hardware pin of microprocessor 8085. There are mainly six dedicated pins available for interrupt purpose.



Those are TRAP, RST 7.5, RST 6.5, RST 5.5, INTR, INTA (It is not an Interrupt pin but it is used to send acknowledgement of the Interrupt request getting from other interrupt pin.)

**8085 Interrupt Pins and Interrupt Priority**

There are five interrupt pins in 8085 and one interrupt acknowledge (INTA) pin.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Pin No. | Name | Type | Priority | ISR Location |
| 6 | TRAP | Vectored | Highest | CALL 0024H (3-byte call) |
| 7 | RST 7.5 | Vectored |  | CALL 003CH (3-byte call) |
| 8 | RST 6.5 | Vectored |  | CALL 0034H (3-byte call) |
| 9 | RST 5.5 | Vectored |  | CALL 002CH (3-byte call) |
| 10 | INTR | Non-Vectored | Lowest | RST (Restart instructions) – 1 byte call |

Pin 6 to pin 10 interrupts have the priorities from highest to lowest in decreasing order.

 Priority means which interrupt gets the acknowledgement first if more than one are interrupting the microprocessor.

**Maskable and Non-Maskable Interrupt**

Maskable interrupts: An interrupt which can be disabled by software that means we can disable the interrupt by sending appropriate instruction, is called a maskable interrupt.

RST 7.5, RST 6.5, and RST 5.5 are the examples of Maskable Interrupt.

Non-Maskable interrupts: As name suggests we cannot disable the interrupt by sending any instruction is called Non Maskable Interrupt.

TRAP interrupt is the non-maskable interrupt for 8085. It means that if an interrupt comes via TRAP, 8085 will have to recognize the interrupt; we cannot mask it.

**Interrupts vs. Polling**  
• A single microcontroller can serve several devices.  
That are two ways to do that: interrupts or polling.  
• The program which is associated with the interrupt is called the interrupt service routine (ISR) or interrupt handler

A) The first method is the simple one - Polling:   
In the Polling method, the microcontroller must "access by himself" the device and “ask” for the information it needs for processing. In fact we see that in the Polling method the external devices are not independent systems; they depend on the microcontroller, and only the micro is entitled to obtain access to the information it needs.   
The main drawback of this method when writing program is waste of time of microcontroller, which needs to wait and check whether the new information has arrived.   
The microcontroller continuously monitors the status of a given device. When the condition is met, it performs the device. After that, it moves on to monitor the next device until everyone is serviced. The microcontroller checks all devices in a round robin fashion.

B) The second method is - Interrupt:   
Interrupt is the signal sent to the micro to mark the event that requires immediate attention. Interrupt is “requesting" the processor to stop to perform the current program and to “make time” to execute a special code. Whenever any device needs its service, the device notifies the microcontroller by sending it an interrupt signal. Upon receiving an interrupt signal, the microcontroller interrupts whatever it is doing and serves the device. The program which is associated with the interrupt is called the interrupt service routine (ISR) or interrupt handler.  
The “request” for the microcontroller to “free itself” to execute the interrupt could come from several sources:

Key Differences Between Interrupt and Polling in OS

1. In interrupt, the device notifies the CPU that it needs servicing whereas, in polling CPU repeatedly checks whether a device needs servicing.
2. Interrupt is a **hardware** **mechanism** as CPU has a wire,**interrupt-request line** which signal that interrupt has occurred. On the other hands, Polling is a **protocol** that keeps checking the **control bits** to notify whether a device has something to execute.
3. **Interrupt handler** handles the interrupts generated by the devices. On the other hands, in polling, **CPU** services the device when they require.
4. In interrupts, CPU is only disturbed when any device interrupts it. On the other hand, in polling, CPU waste lots of CPU cycles by repeatedly checking the command-ready bit of every device.
5. An interrupt can occur at**any instant of time** whereas, CPU keeps polling the device at the **regular intervals**.
6. Polling becomes inefficient when CPU keeps on polling the device and rarely finds any device ready for servicing. On the other hands, interrupts become inefficient when the devices keep on interrupting the CPU processing repeatedly.