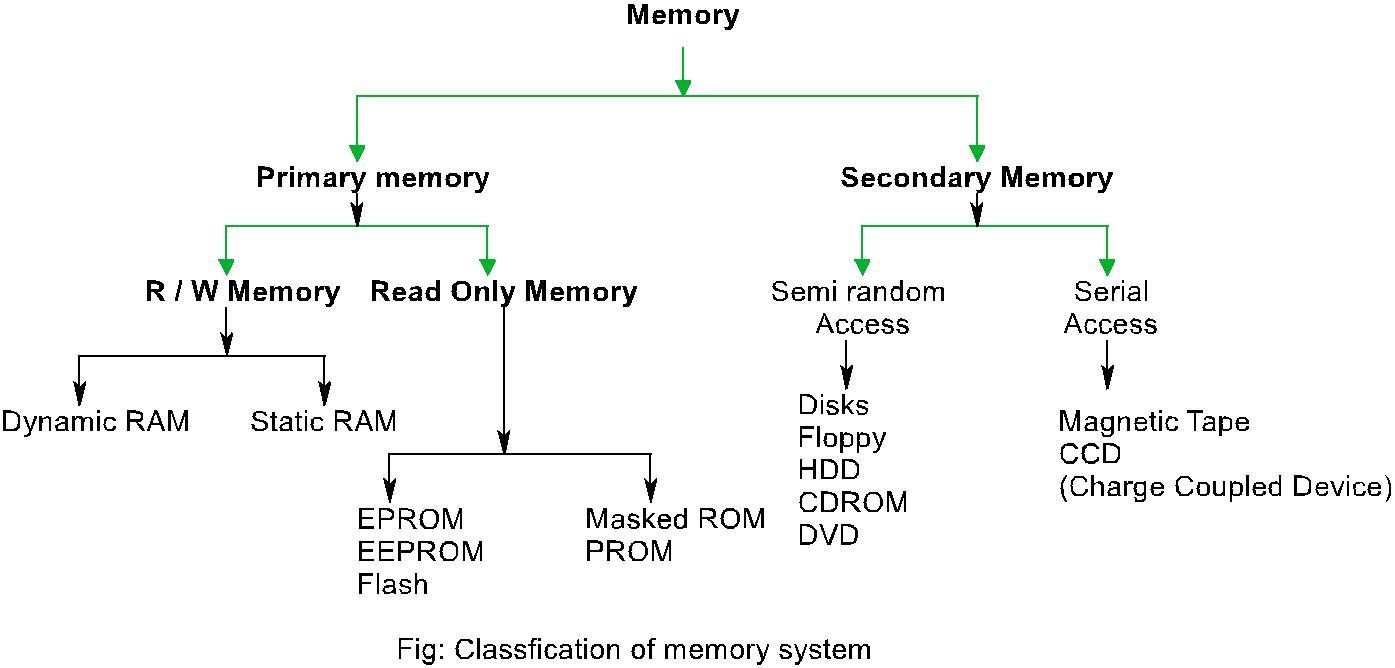
UNIT 5: Basic I/O,Memory R/W and Interrupt Operation

**Introduction:**

* Memory is an essential component of the microcomputer system. It is used to store both instructions and data.
* It is used to store both instructions and data. Memory is made up of registers and the number of bits stored in a register is called memory word
* .Memory word is identified by an address .
* If microprocessor uses 16 bit address , then there will be maximum of 216= 65536 memory addresses ranging from 0000H to FFFFH.

There are various types of memory which can be classified in to two main groups i.e. Primary memory and Secondary memory.

**Memory Devices**



**1. Primary Memory:**

It is the memory used by microprocessor to execute programs. The microprocessor can access only those items that are stored in this memory. Hence, all data and program must be within primary memory prior to its execution. Primary memory is much larger than processer memory that is included in the microprocessor chip.

**Primary memory is divided in to two groups. i. R/W Memory (RAM )**

Microprocessor can read for and write into this memory .This memory is used for information that are likely to be altered such as writing program or receiving data. This memory is volatile i.e. the content will be lost if the power is turned off and commonly known as RAM, RAM are basically of two types.

1. **Static RAM (SRAM)**

This memory is made up of flip flops and it stores bit as voltage. A single flip flop stores binary data either 1 or 0. Each flip flop is called storage cell. Each cell requires six transistors. Therefore, the memory chip has low density but high speed. This memory is more expensive and consumes more power.

1. **Dynamic RAM (DRAM)**

This memory is made up of MOS transistor gates and it stores the bit as charge. The advantage of DRAM are it has high density, low power consumption and cheaper than SRAM. But the bit information leaks therefore needs to be rewritten again every few milliseconds. It is called refreshing the memory and requires extra circuitry to do this. It is slower than SRAM.

**Read Only Memory (ROM):**

ROM contains a permanent pattern of data that cannot be changed. It is non volatile that is no power source is required to maintain the bit values in memory. ROM are basically of 5 types.

1. Masked ROM: A bit pattern is permanently recorded by the manufactures during production.
2. Programmable ROM: In this ROM, a bit pattern may be written into only once and the writing process is performed electrically. That may be performed by a supplier or customer.
3. Erasable PROM (EPROM):

This memory stores a bit in the form of charge by using EPROM programmer which applies high voltage to charge the gate .Information can be erased by exposing ultra violet radiation. It is reusable. The disadvantages are :(i) it must be taken out off circuit to erase it (ii). The entire chip must be erased (iii) the erasing process takes 15 to 20 minutes.

1. Electrically Erasable PROM(EEPROM):

It is functionally same as EPROM except that information can be altered by using electrical signal at the register level rather than erasing all the information. It is expensive compared to EPROM and flash and can be erased in 10 ms.

1. Flash Memory:

It is variation of EPROM. The difference is that EPROM can be erased in register level but flash memory must be erased in register level but flash memory must be erased in its entirety or at block level.

1. **Secondary memory**

The devices that provide backup storage are called secondary memory. It includes serial access type such as magnetic disks and random access type such as magnetic disks. It is nonvolatile memory.

**Performance of memory:**

**1. Access time (ta):**

Read access time: It is the average time required to read the unit of information from memory.

Write access time: It is the average time required to write the unit of information on memory.

Access rate (ra) =/ta

**2. Cycle time (tc):**

It is the average time that lapses between two successive read operation. Cycle rate (rc)= bandwidth = 1/tc

**Access modes of memory:**

1. Random access: In random access mode, the ta is independent of the location from which the data is accessed like MOS memory.

2. Sequential access: In that mode, the ta is dependent of the location form which the data is accessed like magnetic type.

3. Semi random-access: the semi random access combines these two. foreg. In magnetic disk, any track can be accessed at random. But the access within the truck must be in serial fashion.

**Address decoding:**

* Microprocessor is connected with memory and I/O devices via common address and data bus.
* Only one device can send data at a time and other devices can only receive that data.
* If more than one device sends data at the same time, the data gets garbled.
* In order to avoid this situation, ensuring that the proper device gets addressed at proper time, the technique called address decoding is used.
* In address decoding method, all devices like memory blocks, I/O units etc. are assigned with a specific address.
* The address of the device is determined from the way in which the address lines are used to derive a special device selection signal k/a chip select (CS).
* If the microprocessor has to write or to read from a device, the CS signal to that block should be enabled and the address decoding circuit must ensure that CS signal to other devices are not activated.
* Depending upon the no. of address lines used to generate chip select signal for the device, the address decoding is classified as:

1. **I/O mapped I/O**

In this method, a device is identified with an 8 bit address and operated by I/O related functions IN and OUT for that IO/M =1. Si~~nc~~e only 8 bit address is used, at most 256 bytes can be identified uniquely. Generally low order address bits A0-A7 are used and upper bits A8-A15 are considered don‘t care. Usually I/O mapped I/O is used to map devices like 8255A, 8251A etc.

1. **Memory mapped I/O**

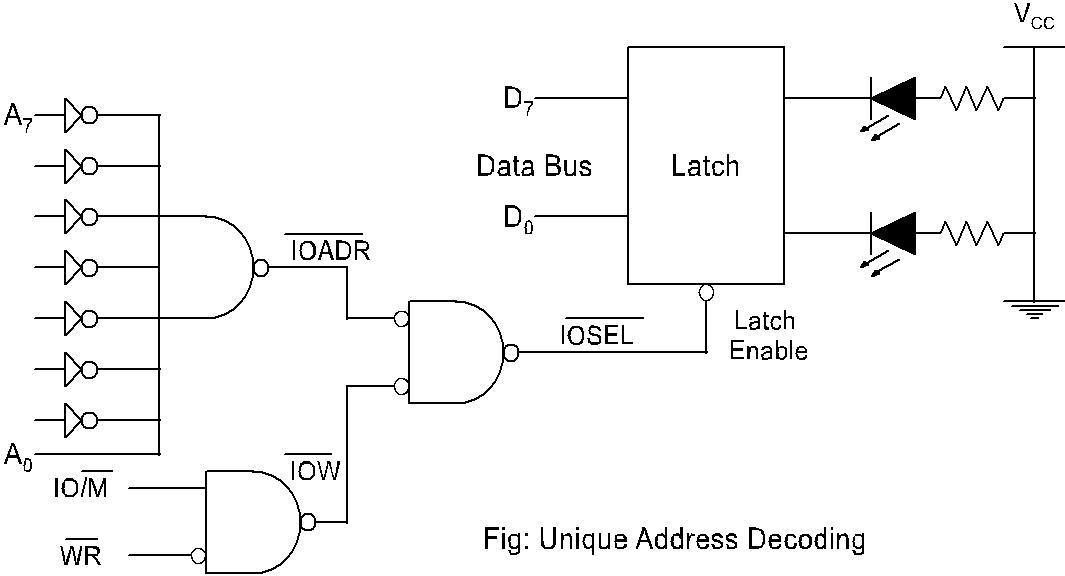
In this method , a device is identified with 16 bit address and enabled memory related functions such as STA , LDA for which IO/M ~~=~~0, here chip select signal of each device is derived from 16 bit address lines thus total addressing capability is 64K bytes . Usually memory mapped I/O is used to map memories like RAM, ROM etc.

**Depending on the address that are allocated to the device the address decoding are categorized in the following two groups.**

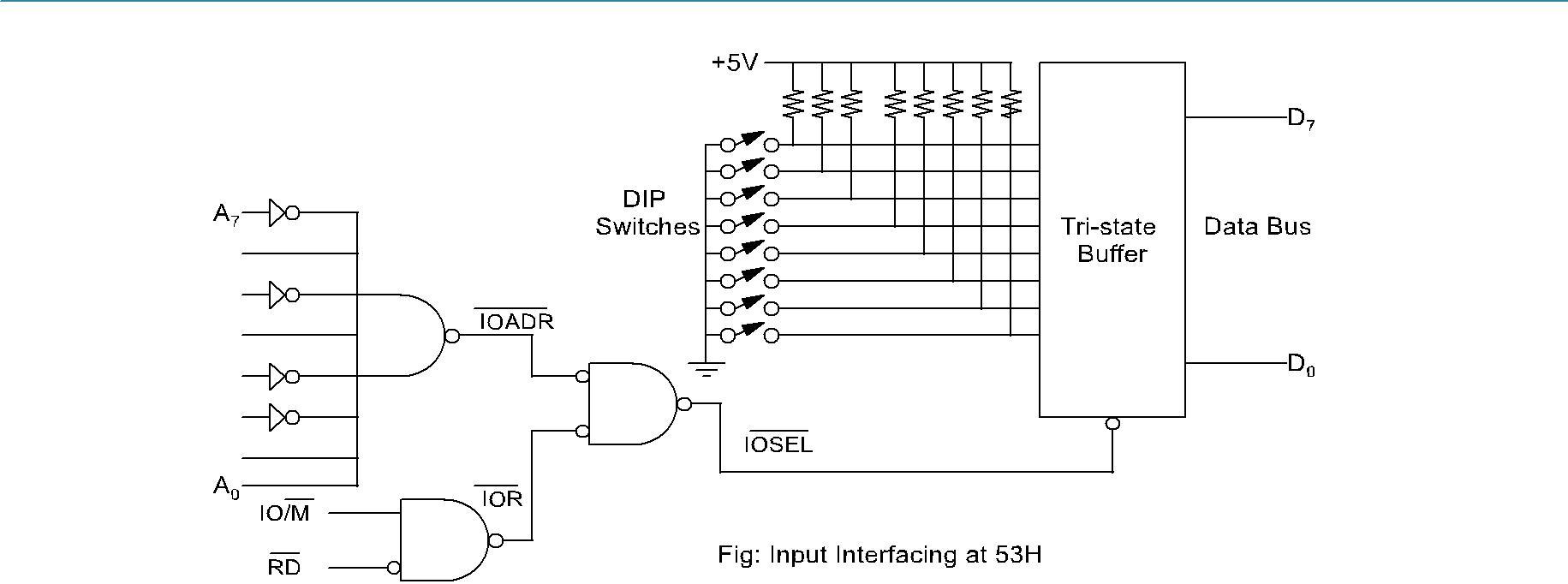
1. **Unique Address Decoding:**

If all the address lines on that mapping mode are used for address decoding then that

decoding is called unique address decoding. It means all 8-lines in I/O mapped I/O and all 16 lines in memory mapped I/O are used to derive **** signal. It is expensive and complicated but fault proof in all cases.

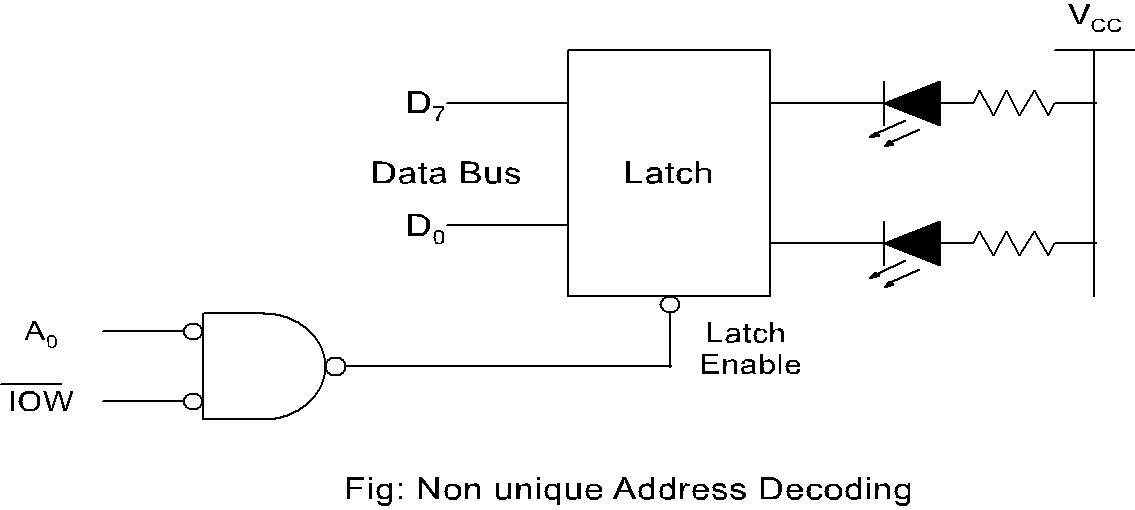


* If A0 is high and A1- A7are low and if IOW becomes low, the latch gets enabled.
* The data to the LED can be transferred in only one case and hence the device has unique Address of 01H.Eight I/P switch interfacing at 53H. (01010011)



**Non Unique Address decoding:**

If all the address lines available on that mode are not used in address decoding then that decoding is called non unique address decoding. Though it is cheaper there may be a chance of address conflict.



* If A0 is low and  is low. Then latch gets enabled. Here A1-A7 is neglected that is any even address can enable the latch.

**DMA (Direct Memory Access)**

**Introduction**

* An **alternative** to the basic and interrupt-driven I/O discussed previously

Direct memory access (DMA) is a feature of modern computer systems that allows certain hardware subsystems to read/write data to/from memory without microprocessor intervention, allowing the processor to do other work.

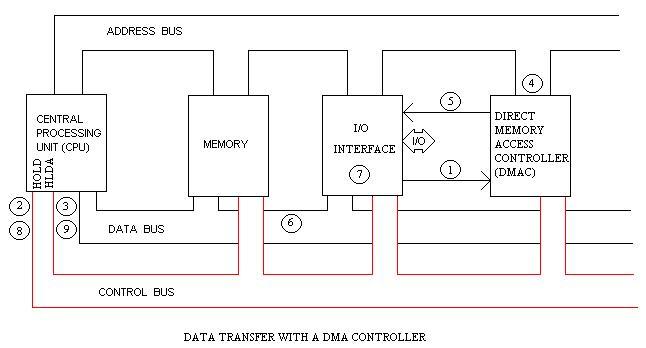
Used in disk controllers, video/sound cards etc, or between memory locations.

* DMA can transfer **large blocks of data** from memory to device or from device to memory
* Uses same Address/Data lines on system bus
* Controls the system bus instead of the processor ("bus master")
* Does not require the processor for data transfer

**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 components capable of becoming masters are processors (and their bus control logic) and DMA controllers.
* Sometimes a DMA controller is associated with a single interface, but they are often designed to accommodate more than one interface.
* The 8086 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 8086 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 8086 will drop the grant on the HLDA pin. One exception to the normal sequence is that if a word, which begins at an odd address is being accessed, then two bus cycles are required to complete the transfer and a grant will not be issued until after the second bus cycle.
* 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. The former is the usual case because this allows the CPU to continue its work until the next data byte is available.

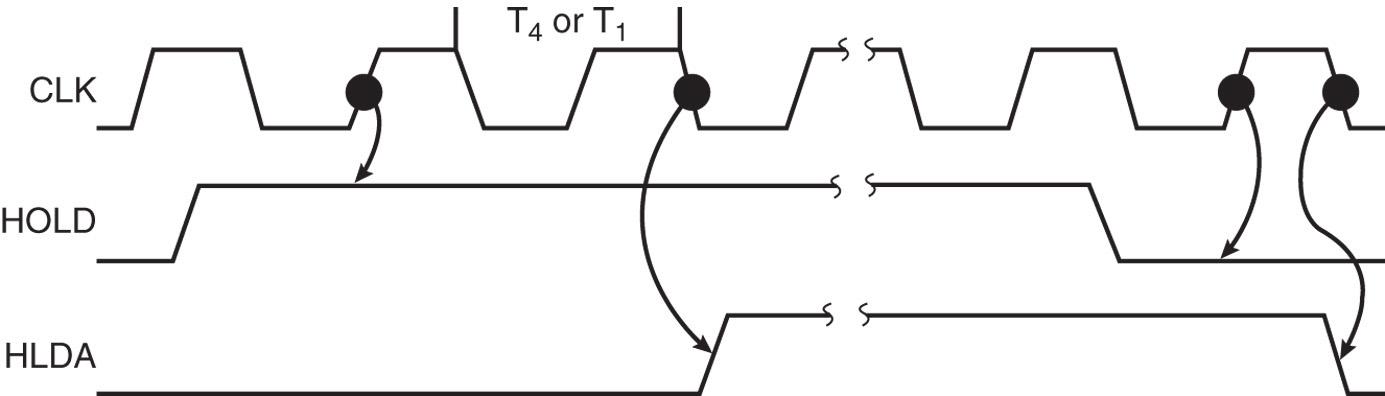
**Basic DMA Operation**



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 8086 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 8086 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. (For an output it signals the interface to latch the next data placed on the bus.)
6. The data byte is transferred to the memory location indicated by the address bus.
7. The interface latches 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 8086 microprocessor is dropped and the HLDA pin goes Low.
10. The address register is incremented by 1.
11. The byte count is decremented by 1.
12. If the byte count is non-zero, return to step 1, otherwise stop.

Following Figure shows HOLD and HLDA timing for the microprocessor.



* + HOLD is sampled in any clocking cycle
  + when the processor recognizes the hold, it stops executing software and enters hold cycles
  + HOLD input has higher priority than INTR or NMI
  + the only microprocessor pin that has a higher priority than a HOLD is the RESET pin
* HLDA becomes active to indicate the processor has placed its buses at high-impedance state.
  + as can be seen in the timing diagram, there are a few clock cycles between the time that HOLD changes and until HLDA changes
* HLDA output is a signal to the requesting device that the processor has relinquished control of its memory and I/O space.
  + one could call HOLD input a DMA request input and HLDA output a DMA grant signal

Direct Memory Access Controller (DMAC) options for data transfer

The DMA Controller has several options available for the transfer of data. They are:

**1) Cycle Steal**

A read or write signal is generated by the DMAC, and the I/O device either generates or latches the data. The DMAC effectively steals cycles from the processor in order to transfer the byte, so single byte transfer is cycle stealing.

We encounter cycle stealing in the context of Direct Memory Access (DMA). Either the DMA controller can use the data bus when the CPU does not need it, or it may force the CPU to temporarily suspend operation. The latter technique is called cycle stealing. Note that cycle stealing can be done only at specific break points in an instruction cycle.

**2) Burst Transfer:**

To achieve block transfers, some DMAC's incorporate an automatic sequencing of the value presented on the address bus. A register is used as a byte count, being decremented for each byte transfer, and upon the byte count reaching zero, the DMAC will release the bus. When the DMAC operates in burst mode, the CPU is halted for the duration of the data transfer.

**3) Hidden DMA:**

It is possible to perform hidden DMA, which is transparent to the normal operation of the CPU. In other words, the bus is grabbed by the DMAC when the processor is not using it. The DMAC monitors the execution of the processor, and when it recognizes the processor executing an instruction which has sufficient empty clock cycles to perform a byte transfer; it waits till the processor is decoding the op code, and then grabs the bus during this time. The processor is not slowed down, but continues processing normally. Naturally, the data transfer by the DMAC must be completed before the processor starts

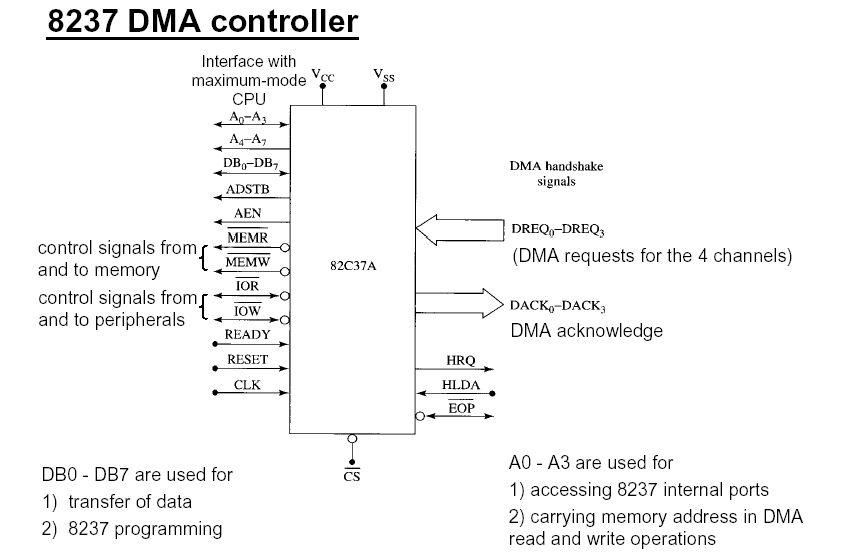
**PROGRAMMABLE DMA CONTROLLER - INTEL 8237**

It is a device to transfer the data directly between IO device and memory without through the CPU. So it performs a high-speed data transfer between memory and I/O device.

The features of 8237 is,

* The 8237 has four channels and so it can be used to provide DMA to four I/O devices.
* Each channel can be independently programmable to transfer up to 64kb of data by DMA.
* Each channel can be independently perform read transfer, write transfer and verify transfer.

It is a 40 pin IC and the pin diagram is,



**8237 Pin Definitions**

1. **CLK**

* **Clock** input is connected to the system clock signal as long as that signal is 5 MHz or less.
  + in the 8086/8088 system, the clock must be inverted for the proper operation of the 8237

1. **CS**

* **Chip select** enables 8237 for programming.
* The CS pin is normally connected to the output of a decoder.
* The decoder does not use the 8086/8088 control signal IO/M(M/IO) because it contains the new memory and I/O control signals (MEMR, MEMW, IOR and IOW).

3. **RESET**

* The **reset** pin clears the command, status, request, and temporary registers.
* It also clears the first/last flip-flop and sets the mask register. This input primes the 8237 so it is disabled until programmed otherwise.

4. **READY**

* Logic 0 on the **ready** input causes the 8237 to enter wait states for slower memory components.

5. **HLDA**

* A **hold acknowledges** signals 8237 that the microprocessor has relinquished control of the address, data, and control buses.

6. **DREQ0–DREQ3**

* **DMA request inputs** are used to request a transfer for each of the four DMA channels. The polarity of these inputs is programmable, so they are either active-high or active-low inputs

**7. DB0 ±DB7**

* DATA BUS: The Data Bus lines are bidirectional three-state signals connected to the system data bus.
* The outputs are enabled in the Program condition during the I/O Read to output the contents of an Address register, a Status register, the Temporary register or a Word Count register to the CPU.
* The outputs are disabled and the inputs are read during an I/O Write cycle when the CPU is programming the 8237A control registers.
* During DMA cycles the most significant 8 bits of the address are output onto the data bus to be strobe into an external latch by ADSTB. In memory-to-memory operations, data from the memory comes into the 8237A on the data bus during the read-from-memory transfer. In the write-to-memory transfer, the data bus outputs place the data into the new memory location.

**8. IOR**

* I/O READ: I/O Read is a bidirectional active low three-state line. In the Idle cycle, it is an input control signal used by the CPU to read the control registers. In the Active cycle, it is an output control signal used by the 8237A to access data from a peripheral during a DMA Write transfer.

**9. IOW**

* I/O WRITE: I/O Write is a bidirectional active low three-state line. In the Idle cycle, it is an input control signal used by the CPU to load information into the 8237A. In the Active cycle, it is an output control signal used by the 8237A to load data to the peripheral during a DMA Read transfer.

**10. A0 ±A3**

* ADDRESS: The four least significant address lines are bidirectional three-state signals. In the Idle cycle they are inputs and are used by the CPU to address the register to be loaded or read. In the Active cycle they are outputs and provide the lower 4 bits of the output address.

**11. A4 ±A7 Output**

* ADDRESS: The four most significant address lines are three-state outputs and provide 4 bits of address. These lines are enabled only during the DMA service.

**12. HRQ Output**

* HOLD REQUEST: This is the Hold Request to the CPU and is used to request control of the system bus. If the corresponding mask bit is clear, the presence of any valid DREQ causes 8237A to issue the HRQ.

**13. DACK0 ±DACK3 Output**

* DMA ACKNOWLEDGE: DMA Acknowledge is used to notify the individual peripherals when one has been granted a DMA cycle. The sense of these lines is programmable. Reset initializes them to active low.

**14. AEN**

* ADDRESS ENABLE: Address Enable enables the 8-bit latch containing the upper 8 address bits onto the system address bus. AEN can also be used to disable other system bus drivers during DMA transfers. AEN is active HIGH.

**15. ADSTB**

* ADDRESS STROBE: The active high, Address Strobe is used to strobe the upper address byte into an external latch.

**16. MEMR**

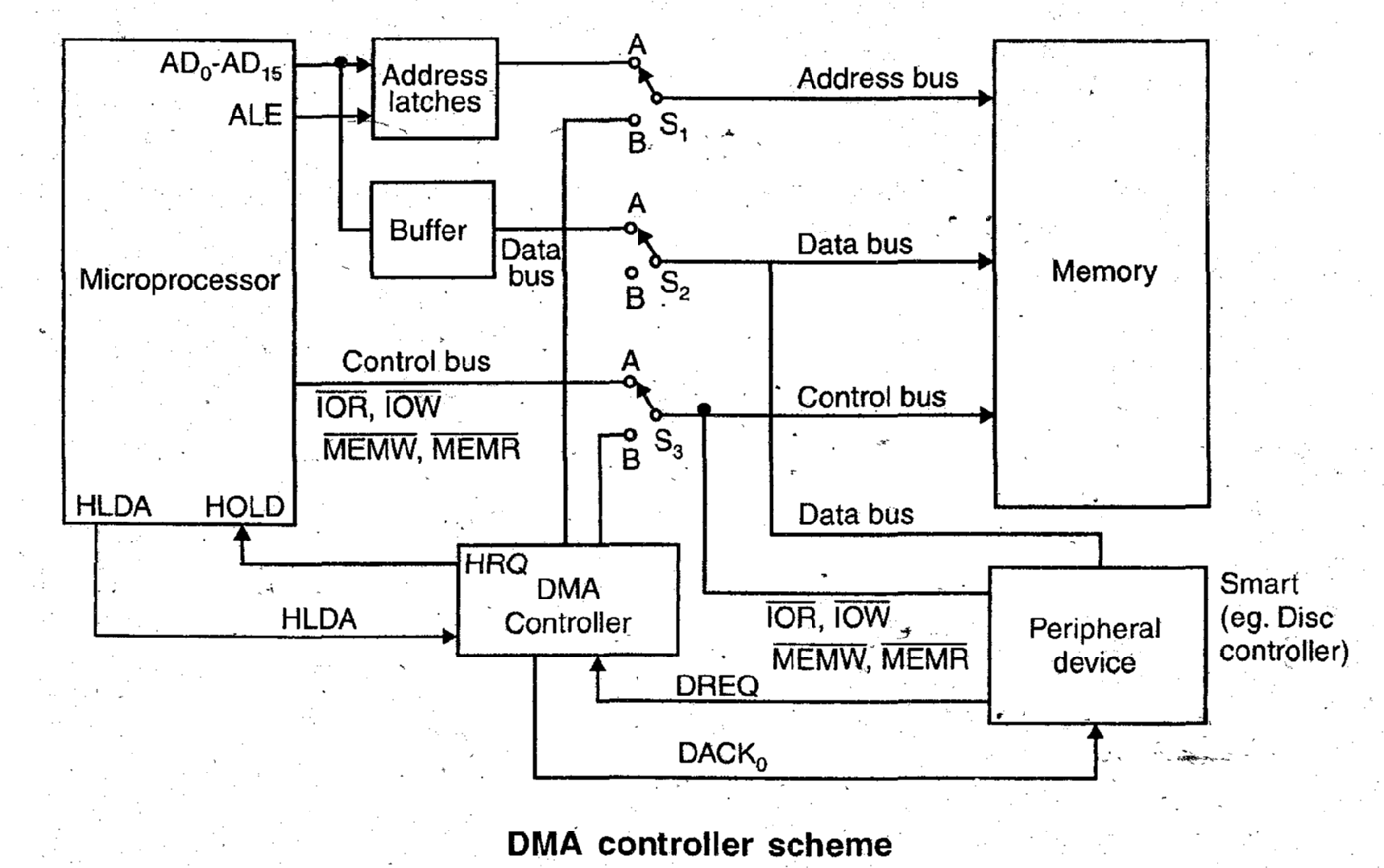
* MEMORY READ: The Memory Read signal is an active low three-state output used to access data from the selected memory location during a DMA Read or a memory-to-memory transfer.

**17. MEMW**

* MEMORY WRITE: The Memory Write is an active low three-state output used to write data to the selected memory location during a DMA Write or a memory-to-memory transfer.

**DMA Interfacing with Microprocessor**

In I/O data transfer data is transferred by using microprocessor .The microprocessor will read data from I/O device and then will write data to memory

* In this case there are two operations for single data transfer.
* If the data is less, then micro process will not waste its time; transferring data from I/O to memory or back. But suppose, data is huge, then the transfer rate from I/O to memory or back will slow down because of microprocessor intervention. In such case, to speed up the process of transferring the data, we can think, Can I/O have direct access to memory and the answer is, yes.
* It can have Direct memory access (DMA), but under Supervision. The device which supervises, data transfer is named as DMA controller.
* Now let’s have diagrammatic representation of the scheme, which depicts microprocessor, DMA controller, memory and I/O device.

**System Interface**

* The DMA is used to transfer data bytes between I/O (such as floppy disk) and system memory (or from memory to memory) at high speed. It includes eight data lines, four control signals (IOR, IOW, MEMR, and MEMW), and eight address lines (A7-A0). However, it needs 16 address lines to access 64K bytes. Therefore, an additional eight lines must be generated as shown in figure 15.34.
* When a transfer begins, the DMA places the low-order byte on the address bus and the high-order byte on the data bus and asserts AEN (Address Enable) and ADSTB (Address Strobe). These two signals are used to latch the high-order byte from the data bus: thus, it places the 16-bit address on the system bus. After the transfer of first byte, the latch is updated when the lower byte generates a carry (or borrows). Figure 15.34 shows two latches: one latch (373 #1) to latch a high-order address from the data bus by using the AEN and ADSTB signals, and the second latch (373 #2) to demultiplex the 8085 bus and generate the low-order address bus by using the ALE (Address Latch Enable from the 8085) signal. The AEN signal is connected to the OE signal of the second latch to disable the low-order address bus from the 8085 when the first latch is enabled to latch the high-order byte of address.

**Programming The 8237**

To implement the DMA transfer, the 8237 should be initialized by writing into various control registers discussed earlier in the DMA channels and interfacing section. To initialize the 8237, the following steps are necessary:

1. Write a control word in the Mode register that selects the channel and specifies the type of transfer (Read, Write or Verify) and the DMA mode (block, single-byte, etc.).

2. Write a control word in the Command Register that specifies parameters such as priority among four channels. DREQ and DACK active levels, and timing, and enables the 8237.

3. Write the starting address of the data block to be transferred in the channel Memory Address Register (MAR).

4. Write the count (the number of the bytes in the data block) in the channel Count register.

**Advantages of DMA**

* + Computer system performance is improved by direct transfer of data between memory and I/O devices, bypassing the CPU.
  + CPU is free to perform operations that do not use system buses.
  + Quick data transfer because a dedicated piece of hardware transfers data from one computer location to another and only one or two bus read/write cycles are required per piece of data transferred.
  + Minimizes latency in servicing a data acquisition device because the dedicated hardware responds more quickly than interrupts and transfer time is short.
  + Minimizes latency reduces the amount of temporary storage (memory) required on an I/O device.
  + Processor is not used for holding the data transfer activity and is available for other processing activity.
  + Also in systems where the processor primarily operates out of its cache, data transfer actually occurring in parallel, thus increasing overall system utilization**.**

**Application:**

* Extensively used for computer-based data acquisition applications including streaming data to disk, real-time screen data display and continuous data acquisition applications.
* DMA was used for floppy disk I/O in the original PC and for hard disk I/O in later versions.
* PC-based DMA technology, along with high-speed bus technology, is driven by data storage, communications and graphics needs-all of which require the highest rates of data transfer between system memory and I/O devices.
* Data acquisition applications have the same needs and therefore can take advantage of the technology developed for larger markets.

**Disadvantages of DMA**

* In case of Burst Mode data transfer, the CPU is rendered inactive for relatively long periods of time.

**Interrupts**

**Introduction**

* Interrupt is signals send by an external device to the processor, to request the processor to perform a particular task or work.
* Mainly in the microprocessor based system the interrupts are used for data transfer between the peripheral and the microprocessor.
* The processor will check the interrupts always at the 2nd T-state of last machine cycle.
* If there is any interrupt it accept the interrupt and send the INTA (active low) signal to the peripheral.
* The vectored address of particular interrupt is stored in program counter.
* The processor executes an interrupt service routine (ISR) addressed in program counter.
* It returned to main program by RET instruction.

**Need for Interrupt:** Interrupts are particularly useful when interfacing I/O devices that provideor require data at relatively low data transfer rate.

**Interrupt Operations**

The transfer of data between the microprocessor and input /output devices takes place using various modes of operations like programmed I/O, interrupt I/O and direct memory access. In programmed I/O, the processor has to wait for a long time until I/O module is ready for operation. So the performance of entire system degraded. To remove this problem CPU can issue an I/O command to the I/O module and then go to do some useful works. The I/O device will then interrupt the CPU to request service when it is ready to exchange data with CPU. In response to an interrupt, the microprocessor stops executing its current program and calls a procedure which services the interrupt.The interrupt is a process of data transfer whereby an external device or a peripheral can inform the processor that it is ready for communication and it requests attention. The response to an interrupt request is directed or controlled by the microprocessor.

**Process of interrupt Operation**

**From the point of view of** I/O **unit**

* I/O device receives command from CPU
* The I/O device then processes the operation
* The I/O device signals an interrupt to the CPU over a control line.
* The I/O device waits until the request from CPU.

**From the point of view of processor**

* The CPU issues command and then goes off to do its work.
* When the interrupt from I/O device occurs, the processor saves its program counter & registers of the current program and processes the interrupt.
* After completion for interrupt, processor requires its initial task.

**Interrupt structures:**

A processor is usually provided with one or more interrupt pins on the chip. Therefore a special mechanism is necessary to handle interrupts from several devices that share one of these interrupt lines. There are mainly two ways of servicing multiple interrupts which are polled interrupts and daisy chain (vectored) interrupts.

* + 1. **polled interrupts:**

Polled interrupts are handled by using software which is slower than hardware interrupts. Here the processor has the general (common) interrupt service routine (ISR) for all devices. The priority of the devices is determined by the order in which the routine polls each device. The processor checks the starting with the highest priority device. Once it determines the source of the interrupt, it branches to the service routine for that device.

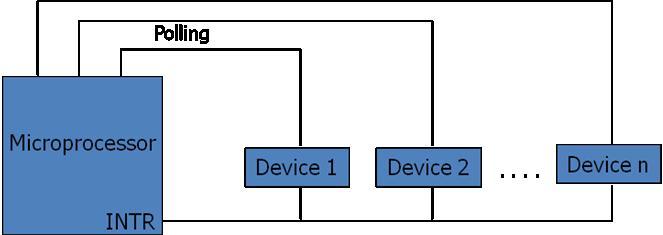


Fig: Polled Interrupt

Here several eternal devices are connected to a single interrupt line (INTR) of the microprocessor. When INTR signal goes up, the processor saves the contents of PC and other registers and then branches to an address defined by the manufactures of the processor. The user can write a program at this address to find the source of the interrupt by starting the polled from highest priority device.

* + 1. Daisy chain (vectored) interrupt:

This is hardware concept of handling the multiple interrupts. In this technique, the devices are connected in a chain fashion as shown in figure below for setting up the priority system.

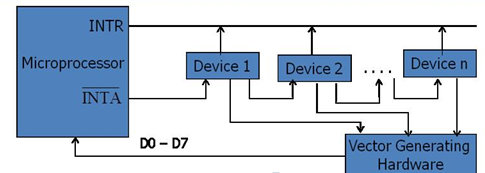


Fig: Vectored (Daisy Chain) Interrupt

Here the device with he highest priority placed in the first position, followed by lower priority devices. Suppose that one or more devices interrupt the processor at a time. In response, the processor saves its current status and then generates an interrupt acknowledge (INTA) signal to the highest priority device, which is device 1 in our case. If this device has generated the interrupt it will accept the INTA signal from the processor; otherwise, it will pass INTA on to the next device until the INTA is accepted by the interrupting device. Once accepted, the device provides a means to the processor for finding the interrupt address vector using external hardware. Usually the requesting device responds by placing a word on the data lines. With the help of hardware it generates interrupts vector address. This word is referred to as vector, which the processor used as a pointer to the appropriate device service routine. This avoids the need to execute a general interrupt service routine first. So this technique is also referred to as vectored interrupts.

**Basic Interrupt Processing**

The occurrence of interrupt triggers a number of events, both in processor hardware and in software. The interrupt driven I/O operation takes the following steps.

* The I/O unit issues an interrupt signal to the processor for exchange of data between them.
* The processor finishes execution of the current instruction before responding to the interrupt.
* The processor sends an acknowledgement signal to the device that it issued the interrupt.
* The processor transfers its control to the requested routine called ―Interrupt Service Routine (ISR) by saving the contents of program status word (PSW) and program counter (PC).
* The processor now loads the PC with the location of interrupt service routine and the fetches the instructions. The result is transferred to the interrupt handler program.
* When interrupt processing is completed, the saved register‘s value are retrieved from the stack and restored to the register.
* Finally it restores the PSW and PC values from the stack.

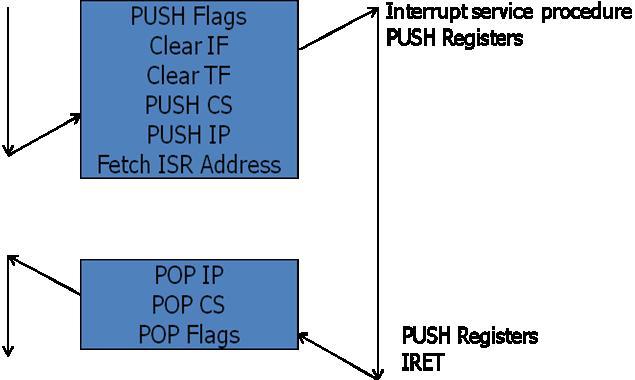


Fig: Interrupt Response for 8086 Microprocessor

The figure summarizes these steps. The processor pushes the flag register on the stack, disables the INTR input and does essentially an indirect call to the interrupt service procedure. An IRET function at the end of interrupt service procedure returns execution to the main program

**Interrupt priority:**

Microcomputers can transfer data to or from an external devices using interrupt through INTR pin. When device wants to communicate with the microcomputer, it connects to INTR pin and makes it high or low depending on microcomputer. The microcomputer responds by sending signal via its pin called interrupt acknowledgement INTA. In differentiation with the occurrence of interrupts, basically following interrupts exist.

1. **External interrupts:**

These interrupts are initiated by external devices such as A/D converters and classified on following types.

* Maskable interrupt :

It can be enabled or disabled by executing instructions such as EI and DI. In 8085, EI sets the interrupt enable flip flop and enables the interrupt process. DI resets the interrupt enable flip flop and disables the interrupt.

* Non-maskable interrupt:

It has higher priority over maskable interrupt and cannot be enabled or disabled by the instructions.

**2. Internal interrupts:**

* These are indicated internally by exceptional conditions such as overflow, divide by zero, and execution of illegal op-code. The user usually writes a service routine to take correction measures and to provide an indication in order to inform the user that exceptional condition has occurred.
* There can also be activated by execution of TRAP instruction. This interrupt means TRAP is useful for operating the microprocessor in single step mode and hence important in debugging.
* These interrupts are used by using software to call the function of an operating system. Software interrupts are shorter than subroutine calls and they do not need the calling program to know the operating system‘s address in memory.

If the processor gets multiple interrupts, then we need to deal these interrupts one at a time and the dealing approaches are:

1. **Sequential processing of interrupts**

When user program is executing and an interrupt occurs interrupts are disabled immediately. After the interrupt service routine completes, interrupts are enabled before resuming the user program and the processor checks to see if additional interrupts have occurred

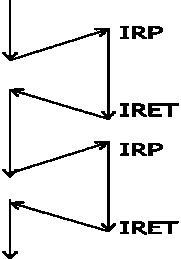
. 

Fig: Sequential Interrupt Service

1. **Priority wise processing of interrupts:**

The drawback of sequential processing is that it does not take account of relative priority or time critical needs. The alternative form of this is to define priorities for interrupts and to allow an interrupt of higher priority to cause a lower priority interrupts pause until high priority interrupt completes its function.

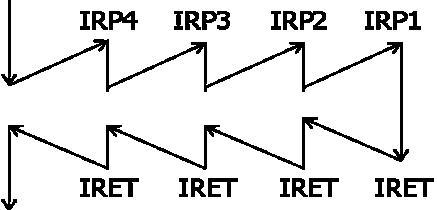


Fig: Priority wise Interrupt service

**Interrupt Service Routine**

* An interrupt service routine (ISR) is a software routine that hardware invokes in response to an interrupt.
* ISRs examine an interrupt and determine how to handle it.
* ISRs handle the interrupt, and then return a logical interrupt value.
* Its central purpose is to process the interrupt and then return control to the main program.
* An ISR must perform very fast to avoid slowing down the operation of the device and the operation of all lower priority ISRs.
* As in procedures, the last instruction in an ISR should be iret.

ISR is responsible for doing the following things:

1. Saving the processor context

Because the ISR and main program use the same processor registers, it is the responsibility of the ISR to save the processor‘s registers before beginning any processing of the interrupt. The processor context consists of the instruction pointer, registers, and any flags. Some processors perform this step automatically.

2. Acknowledging the interrupt

The ISR must clear the existing interrupt, which is done either in the peripheral that generated the interrupt, in the interrupt controller, or both.

3. Restoring the processor context

After interrupt processing, in order to resume the main program, the values that were saved prior to the ISR execution must be restored. Some processors perform this step automatically.

**Interrupt Processing in 8085**

* Interrupt is signals send by an external device to the processor, to request the processor to perform a particular task or work.
* Mainly in the microprocessor based system the interrupts are used for data transfer between the peripheral and the microprocessor.
* The processor will check the interrupts always at the 2nd T-state of last machine cycle.
* If there is any interrupt it accept the interrupt and send the INTA (active low) signal to the peripheralThe vectored address of particular interrupt is stored in program counter.
* The processor executes an interrupt service routine (ISR) addressed in program counter.
* It returned to main program by RET instruction.

**Types of Interrupts:**

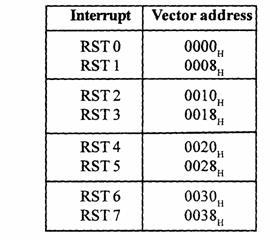
It supports two types of interrupts.

1. Hardware
2. Software

**Software interrupts:**

The software interrupts are program instructions. These instructions are inserted at desired locations in a program.

The 8085 has eight software interrupts from RST 0 to RST 7. The vector address for these interrupts can be calculated as follows.



Interrupt number \* 8 = vector address

For RST 5; 5 \* 8 = 40 = 28H

Vector address for interrupt RST 5 is 0028H

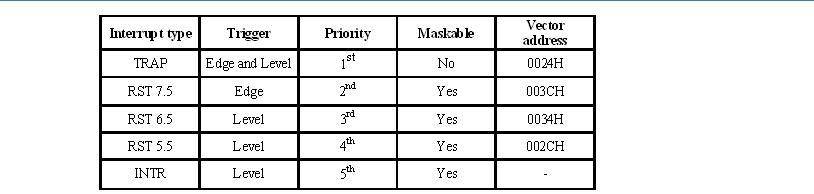
The Table shows the vector addresses of all interrupts.

**Interrupt Pins and Priorities (Hardware interrupts)**

An external device initiates the hardware interrupts and placing an appropriate signal at the interrupt pin of the processor.

If the interrupt is accepted then the processor executes an interrupt service routine.

|  |  |  |  |
| --- | --- | --- | --- |
| The 8085 has five hardware interrupts | | |  |
| (1) TRAP | (2) RST 7.5 | (3) RST 6.5 | (4) RST 5.5(5) INTR |



**TRAP:**

* This interrupt is a non-maskable interrupt. It is unaffected by any mask or interrupt enable.
* TRAP bas the highest priority and vectored interrupt.
* TRAP interrupt is edge and level triggered. This means hat the TRAP must go high and remain high until it is acknowledged.
* In sudden power failure, it executes a ISR and send the data from main memory to backup memory.
* The signal, which overrides the TRAP, is HOLD signal. (i.e., If the processor receives HOLD and TRAP at the same time then HOLD is recognized first and then TRAP is recognized).
* There are two ways to clear TRAP interrupt.
  1. By resetting microprocessor (External signal)
  2. By giving a high TRAP ACKNOWLEDGE (Internal signal)

**RST 7.5:**

* The RST 7.5 interrupt is a maskable interrupt.
* It has the second highest priority.
* It is edge sensitive. ie. Input goes to high and no need to maintain high state until it recognized.
* Maskable interrupt. It is disabled by,
  1. DI instruction
  2. System or processor reset.
  3. After reorganization of interrupt.
* Enabled by EI instruction.

**RST 6.5 and 5.5:**

* The RST 6.5 and RST 5.5 both are level triggered. . ie. Input goes to high and stay high until it recognized.
* Maskable interrupt. It is disabled by,
  1. DI, SIM instruction
  2. System or processor reset.
  3. After reorganization of interrupt.
* Enabled by EI instruction.
* The RST 6.5 has the third priority whereas RST 5.5 has the fourth priority.

**INTR:**

* INTR is a maskable interrupt.
* It is disabled by,
  1. DI, SIM instruction
  2. System or processor reset.
  3. After reorganization of interrupt.
* Enabled by EI instruction.
* Non- vectored interrupt. After receiving INTA (active low) signal, it has to supply the address of ISR.
* It has lowest priority.
* It is a level sensitive interrupts. ie. Input goes to high and it is necessary to maintain high state until it recognized.
* The following sequence of events occurs when INTR signal goes high.
  + 1. The 8085 checks the status of INTR signal during execution of each instruction.
    2. If INTR signal is high, then 8085 complete its current instruction and sends active

Low interrupt acknowledge signal, if the interrupt is enabled.

1. In response to the acknowledge signal, external logic places an instruction OPCODE on the data bus. In the case of multi byte instruction, additional interrupt acknowledge machine cycles are generated by the 8085 to transfer the additional bytes into the microprocessor.
2. On receiving the instruction, the 8085 save the address of next instruction on

stack and execute received instruction.

**THE 8259A PROGRAMMABLE INTERRUPT CONTROLLER**

The 8259A programmable interrupt controller designed to work with Intel microprocessors 8085, 8086 and 8088. The 8259A interrupt controller can

1. Manage eight interrupts according to the instructions written into its control registers. This is equivalent to proving eight interrupt pins on the processor in place of one INTR (8085) pin.

2. Vector can interrupt request anywhere in the memory map. However, all eight interrupts are spaced at the interval of either four or eight locations. This eliminates all the major drawback of the 8085 interrupts in which all interrupts are vectored to memory locations on page 00H

3. Resolve eight levels of interrupt priorities in a variety of modes, such as fully nested mode, automatic rotation mode, and specific rotation mode.

4. Mask each interrupt request individually.

5. Read the status of pending interrupts, in-service interrupts, and masked interrupts.

6. Be set up to accept either the level-triggered or the edge-triggered interrupt request

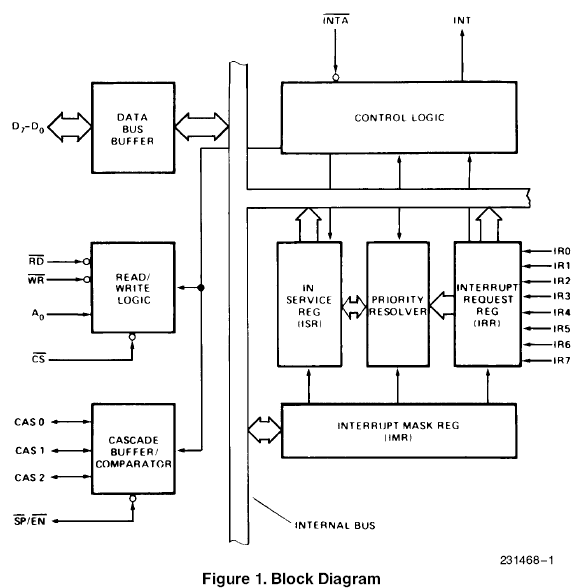
7. Be expanded to 64 priority levels by cascading additional 8259As.

8. Be set up to work with either the 8085 microprocessor mode or the 886/8088 microprocessor mode

The 8259A is upward-compatible with its predecessor, the 8259. The main difference between the two is that the 8259A can be used with Intel's 8086/88 16-bit microprocessor. It also includes additional features such as the level-triggered mode, buffered mode, and automatic-end-of interrupt mode. To simplify the explanation of the 8259A, illustrative examples will not include the cascade mode or the 8086/88 mode and will be limited to modes continuously used with the 8085.

**BLOCK DIAGRAM OF THE 8259A**

Figure 15.29 shows the internal block diagram of the 8259A. It includes eight blocks: control logic, Read/Write logic, data bus buffer, three registers (IRR, ISR and IMR), priority resolver, and cascade buffer. This diagram shows all the elements of a programmable device, plus additional blocks. The functions of these blocks are given below:



**DATA BUS BUFFER** If the data format is not matched it temporarily holds the data. For eg: Printer can accept only 1 bit data but at a time more than 1 bit data may be sent. In such case data bus buffer stores the bits that printer cannot accept at the moment.

**READ/WRITE LOGIC** This is a typical Read/Write control logic. When the address line A0 is at logic 0, the controller is selected to write a command or read a status. The Chip Select logic and A0 determine the port address of the controller.

**CONTROL LOGIC**

This block has two pints: INT (Interrupt) as an output, and INTA (Interrupt Acknowledge) as an input. The INT is connected to the interrupt pin of the MPU. Whenever a valid interrupt is asserted, this signal goes high.

**INTERRUPT REGISTERS AND PRIORITY RESOLVER**

The interrupt Request Register (IRR) has eight input lines (IR0-IR7) for the interrupts. When these lines go high, the requests are stored in the register. The In-Service Register (ISR) stores all the levels that are currently being serviced, and the Interrupt Mask Register (IMR) stored the masking bits of the interrupt lines to be masked. The Priority Resolver (PR) examines these three registers and determines whether INT should be sent to the MPU.

**CASCADE BUFFER/COMPARATOR** This block is used to expand the number of interrupt levels by cascading two or more 8259As. To simplify this discussion, this block will not be mentioned again.

**Interrupt Operation**

To implement interrupts, the Interrupt Enable flip-flop in the microprocessor should be enabled by writing the EI instruction, and the 8259A should be initialized by writing control words in the control register. The 8259A requires two types of control words: Initialization Command Words (ICWs) and Operational Command Words (OCWs). The ICWs are used to set up the proper conditions and specify RST vector addresses. The OCWs are used to perform functions such as masking interrupts, setting up status-read operations, etc. After the 8259A is initialized, the following sequence of events occurs when one or more interrupt request lines go high.

1. The IRR stores the requests.

2. The priority resolver checks three registers: the IRR for interrupt requests, the IMR for masking bits, and the ISR for the interrupt request being served. It resolves the priority and sets the INT high when appropriate.

3. The MPU acknowledges the interrupt by sending INTA.

4. After the INTA is received, the appropriate priority bit in the ISR is set to indicate which interrupt level is being served, and the corresponding bit in the IRR is reset to indicate that the request is accepted. Then, the opcode for the CALL instruction is placed on the data bus.

5. When the MPU decodes the CALL instruction, it places two or more INTA signals on the data bus.

6. When 8259A receives the second INTA, it places the low-order byte of the CALL address on the data bus. At the third INTA, it places the high-order byte on the data bus. The CALL address is the vector memory location for the interrupt: this address is placed in the control register during the initialization.

7. During the third INTA pulse, the ISR bit is reset either automatically (Automatic-End-of-Interrupt—AEOI) or by a command word that must be issued at the end.

**Priority Modes and Other Features** Many types of priority modes are available under software control in the 8259A, and they can be changed dynamically during the program by writing appropriate command words. Commonly used priority modes are discussed below:

**1. Fully Nested Mode:** This is a general-purpose mode in which all IRS (interrupt Requests) are arranged from highest to lowest, with IR0 as the highest and IR7 as the lowest.

In addition, any IR can be assigned the highest priority in this mode; the priority sequence will then begin at that IR. In the example below, IR4 has the highest priority, and IR3 has the lowest priority:

IR0 IR1 IR2 IR3 IR4 IR2 IR3 IR4

4 5 6 7 0 1 2 3

**2.Automatic Rotation Mode:** In this mode, a device, after being serviced, receives the lowest priority. Assuming that the IR2 has just been serviced, it will receive the seventh priority, as shown below:

IR0 IR1 IR2 IR3 IR4 IR2 IR3 IR4

1 6 7 0 1 2 3 4

**3. Specific Rotation Mode:** This mode is similar to the automatic rotation mode, except that the user can select any IR for the lower priority.