Chapter-3

Bus Structure, Memory and I/o Interfacing

Bus Structure:

→ Bus refers to common communication path between microprocessor and peripherals.

→ It is actually a group of wires to carry bits.

→ Fundamentally, in any system bus, the lines can be classified into three functional groups: the address bus, data bus and control bus.

→ The three bus architecture in any microprocessor system or microcontroller system can be as shown in fig. below:-

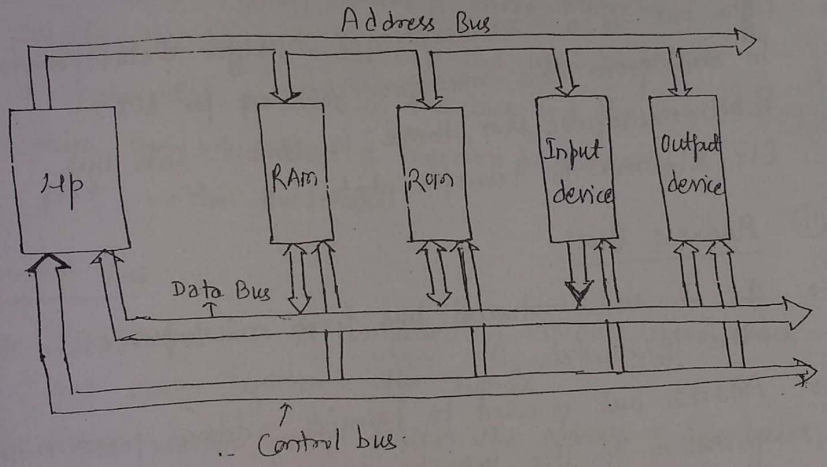


Fig: The three bus architecture of microcontroller

(1) Data Bus:

→ The data bus provides a path for monitoring data between the system modules.

→ The data bus consists of a number of separate lines, generals 8, 16, 32 or 64.
→ The number of lines is referred to as the width of the data bus.
→ Since each line can carry only one bit at a time, the number of lines determines how many bus can be transmitted at a time.
→ The width of the data bus is a key factor in determining the overall system performance.
→ If the data bus is 8 bit wide and each instruction is 16 bits long as in the case of Intel 8088 then the CPU must access the memory module twice during each instruction code.
→ Data bus is a bi-directional bus for data transfer to and from the microprocessor. eg. for 8085 - 8 bit microprocessor, there is 8-bit data bus (ie. 8 wires to transfer data)

(ii) Address Bus:

→ It is unidirectional bus from microprocessor to the peripherals.
→ Address bus is used to transfer address from microprocessor to the peripheral.
→ The address bus which consists of a number of separate lines are used to ~~des~~ designate the source or destination of the data on data bus.
→ For example, if the CPU requires to read a word (8, 16 or 32) bits of data from memory, it put the address of the desired word on the address bus.
→ The width of the address bus clearly determines the maximum possible memory capacity of the system.
→ If there are 'N' no. of address lines(wires), the number of addresses that microprocessor can access is 2^N, or maximum addressing capability of such microprocessor is also 2^N.
eg. for a microprocessor with 16-bit address bus (ie address bus with 16 wires), the microprocessor can generate 2^{16} (=65536) different possible addresses on this bus.
- A memory location or an I/o device can be represented by each of these address.
→ Usually, the higher order bits are used to select a particular modules on the bus and the lower order ~~by~~ bit select a memory location or I/o port within the model.

(iii) Control Bus:

→ The control bus is comprised of various signal lines that carry synchronization signals.
→ These are not groups of lines like address or data buses, but ~~individu~~ individual lines that provide a pulse to indicate the microprocessor operation.
→ The microprocessor unit (MPU) generates specific control signals for every operation (such as memory Read or I/o write, etc) it performs.
→ These signals are used to identify a device type with which the MPU intends to communicate.

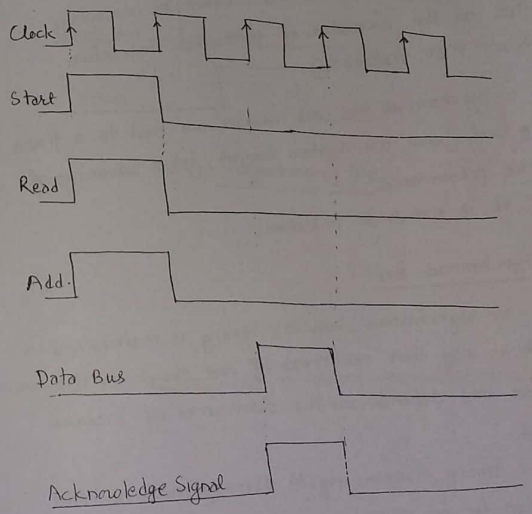
→ The control signals transmit both command and timing information between the system module.
→ The timing signals indicate the validity of data and address information, whereas command signals specify operations to be performed.
→ Some of the control signals are:

- Memory Write:
  It causes data on the bus to be loaded into the address location.
- Memory Read:
  It causes data from the addressed location to be placed on the data bus.
- I/O Write:
  It causes the data on the bus to be output to the addressed I/O port.
- I/O Read
  It causes the data from the addressed I/O port to be placed on the bus.
- Transfer Ack:
  This signal indicates that data ~~from~~ have been accepted from or placed on the bus.
- Bus request:
  Used to indicate that a module wants to gain control of the bus.
- Bus grant:
  Indicate that a requesting model bus has been granted for the control of bus.
- Interrupt Request:
  Indicates that an interrupt has been pending.
- Interrupt Ack:
  Indicates that the pending interrupt has been recognized.

Synchronous and Asynchronous Bus:

① Synchronous Bus:

→ In a synchronous bus, the occurrence of the events on the bus is determined by a clock.
→ The clock transmits a regular sequence of a 0's and 1's of equal duration.



Fig(a) Synchronous Read Operation

The above timing diagram can be described in following steps:

- The CPU ~~starts~~ issues a START signal to indicate the presence of address and control information on the bus.
- Then it issues the memory read signal and places the memory address on the address bus.
- The addressed memory module recognizes the address and after a delay of one clock cycle, it places the data and acknowledgement signals on the buses.

In this way, the synchronous memory read operation completes. In above example, it is shown that bus signals change at the leading edge of the clock signal and most events occupy a single clock cycle so it is the general case and may not be exactly similar to the execution of particular instruction, such as MOV AX,[2035H].

→ In a synchronous bus, all devices are tied to a fixed rate and hence the system cannot take advantage of device performance.

→ But it is easier to implement.

(2) Asynchronous Bus:

→ In an asynchronous bus, the timing is maintained in such a way that occurrence of one event on the bus follows and depends on the occurrence of previous event.

→ In the timing diagram, fig(b) showing memory read operation for an asynchronous bus, the events occur as follows:

- The CPU places the memory read (control) and address signals on the bus.
- After allowing for these signals to stabilize, it issues Master Synchronous Signal (MSYNC ~~ASYNC~~) to indicate the presence of valid address and control signals on the bus.
- The addressed memory module responds with the data and the slave synchronous (SSYNC) signal.

Fig.(b) Asynchronous Bus

Memory

(Contd... -remaining portions)

- Two major semiconductor technologies are used to build integrated-circuits memory devices, which are used to build memory system for today's memory. These are :- bipolar and MOS (metal-oxide semiconductor) technologies.

- Bipolar Memories are seldom used with microprocessor systems. The advantage of bipolar memories is their very fast access time. They have a number of disadvantages when compared with MOS memories. They draw a great deal of power, and there are fewer memory bits for the same silicon chip size. The fabrication is much more complicated and thus are much more expensive than MOS memories. For these reasons, bipolar memory is used only for applications which can afford its great cost.

- MOS memory is the most common microcomputer memory. The MOS memory cells are structurally simple and economical. The MOS cell needs an area of about 25% to 50% of the area required by the bipolar cell. Hence, they can be more densly packed. MOS cell consume much less power than bipolar cells.

- Today's RAM cell are, in common, of two types depending upon the storage elements. MOS technology is applied in both cases. These are :-
  - Static RAM cell
  - Dynamic RAM cell.

① Static RAM (SRAM) cell :

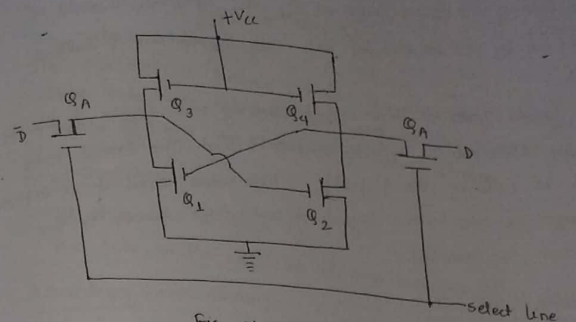


Fig. Static RAM cell

- The figure above shows a NMOS static memory cell which consists of six NMOS transistors.
- The transistors Q_A, called the access transistors are used to access the data during the read and write operation.
- The transistors Q_3 and Q_4 are pull-up load transistors and their cross-connection with Q_2 and Q_1 respectively forms a flip-flop.
- The D and $\bar{D}$ are data lines. There is external driving circuitry which forces D and $\bar{D}$ to opposite values. When reading the bit from the cell the D and $\bar{D}$ serve as output lines, while writing a bit on the cell, they serve as forced input.
- Before reading or writing on cell, the cell must be selected, and the select line is used for the purpose.

Writing On Cell:

- The access transistor Q_A's are turned on via the row select line (high).
- The data lines are forced into a state with D high and $\bar{D}$ low
- The MOSFET Q_1 will be turned on and Q_2 off.

- When forcing signals are removed, Q_1 will continue to hold $\bar{D}$ low (i.e Q_1 will be on) and will keep Q_2 off and D output high.
- The forced state is thus self-sustaining and stable. A similar stable state exists with D low and $\bar{D}$ high. The stored data will be held by the flip-flop in both cases until it is either changed by new forcing signals or until the power to the circuit is removed.

Reading from cell:

- Q_As are turned on via select line (high).
- The stored data appears on $\bar{D}$ and D respectively from Q_1 and Q_2.

② Dynamic RAM (DRAM) cell.

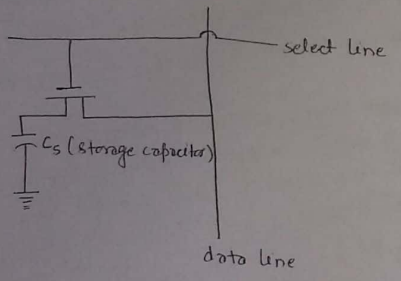


Fig A dynamic RAM cell:

- The figure above shows a dynamic RAM cell which consists of one MOSFET and one capacitor.
  - The capacitor is used to store the bit.
  - The transistor acts as a switch.

Write Operation.

- The data line is high (to store 1)
- The select line is high.
- The MOSFET is turned ON and the capacitor is charged.
- When data and selection line goes low, the MOSFET opens (turns off) and the capacitor retains its charge.

Read Operation.

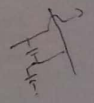
- The data line is high.
- The data in the capacitor is then received at the data line.

Advantages (over SRAM):

① Very simple, allows very large memory arrays to be constructed on a chip at a low cost per bit.
② Power consumption is low.

- Disadvantages:

① The storage capacitor cannot hold its charge over an extended period of time and looses the stored data bit unless its charge is refreshed periodically.
② This process of refreshing requires additional memory circuitry and complicates the operation of the dynamic RAM.



Interfacing Devices.

- These devices are semiconductor chips that are needed to connect peripherals to the bus system.
- Several types of interfacing devices are necessary to interconnect the component of a bus-oriented system. The commonly used devices are tri-state buffer, encoders, decoders and latches.

Tri-state devices:

- In general, a logic device has two states: logic 1 and logic 0. The tri-state device has a third state also i.e. high impedance state, in addition to these two states.
- The device has a third line (other than input and output) called Enable.
- When this line is activated, the tri-state device functions, the same way as ordinary logic devices. When the third line is disabled, the logic device goes into the high-impedance state - as if it were disconnected from the system.
- Ordinarily, current is required to ~~drive~~ drive a device in logic 0 and logic 1 states. In the high impedance state, practically no current is drawn from the system.

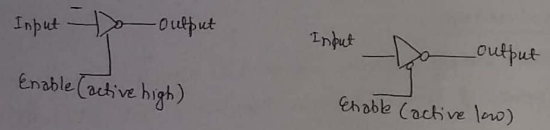


Fig: tri-state inverter

- In a micro-computer systems, peripherals are connected in parallel between the address bus and data bus.
- However, because of tri-state interfacing devices, peripherals do not load the system buses. The microprocessor communicates with one device at a time by enabling the tri-state line of the interfacing device.
- Tri-state logic is critical to proper functioning of the microcomputer.

Buffer:

- The buffer is a logic circuit that amplifies the current or power.
- It has one input line and one output line.

- The logic level of the output is same as that of the input - logic 1 input provides logic 1 output.
- The buffer is used primarily to increase the driving capability of a logic circuit.
- It is also known as driver.

Fig: Buffer

Tri-state Buffer:

- It has similar operation as that of tristate devices.
- This buffer is commonly used to increase the driving capability of the data bus and the address bus.

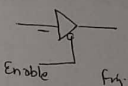


Fig: Tri-state buffer

- As the address bus is unidirectional, this device is commonly used as a driver.
- The octal buffer 74LS244 is a typical example of a tri-state buffer.
- The data bus of a microcomputer is bidirectional, so two tri-state buffers ~~are used to~~ in different directions combine to form a bi-directional buffer.

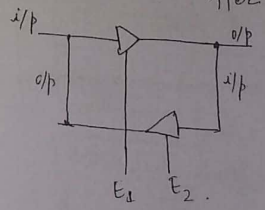


Fig: Bidirectional Buffer

Decoder:

- The decoder is a logic circuit that identifies each combination of the signals present at its input.
- If the input to the decoder has 'n' lines, the decoder will have 2^n output lines. eg. if $n = 2$, the no. of output lines $= 2^n = 2^2 = 4$.
- The two lines can assume four combinations of input signals - 00, 01, 10, 11 - with each combination identified by the output lines 0 to 3. If the input is 11, the output line 3 will be at logic 1 and otherwise will remain at logic 0. This is called decoding.
- Various types of decoders are available, eg. 3 to 8, 4 to 16, etc.
- In general decoders have Enable lines two. The decoder will not function unless enable lines are activated.

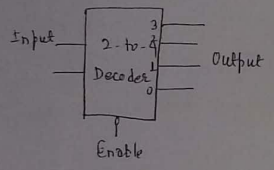


Fig. (a) 2-to-4-decoder

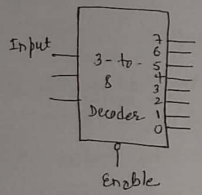


Fig. (b) 3-to-8 decoder

Encoder:

- The encoder is a logic circuit that provides the appropriate code (binary, BCD, etc) as output for each input signals.
- The process is reverse of decoding.

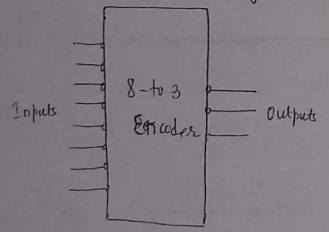


Fig. 8-to-3 Encoder

- Fig. above shows 8-to-3 encoder, it has eight active low inputs and three output lines.
- When 0 goes low, the output is 000, similarly, when input line 5 goes low, the output is 101.
- Encoders are commonly used with keyboards. For each key pressed, the corresponding binary codes are placed on the data bus.

Latches:

- A latch is used commonly to interface output devices.
- When the microprocessor Unit sends an output, data are available on the data bus for only few microseconds, so the latch is used to hold data for display.
- In its simplest form, a latch is a D flip-flop.
- We have other latches (flip-flops) like - RS flip-flop.
  - JK flip-flop.
  - master-slave flip-flop, etc

Internal Structure of a memory:

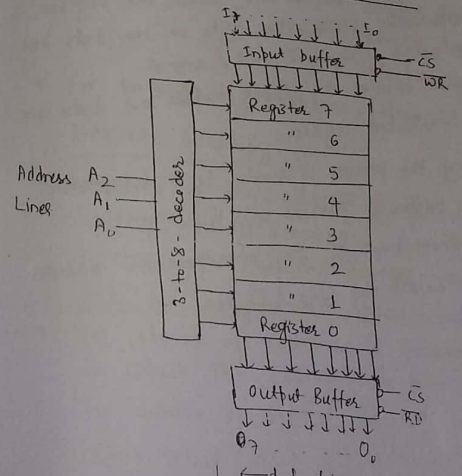


Fig. A 8×8 memory Unit

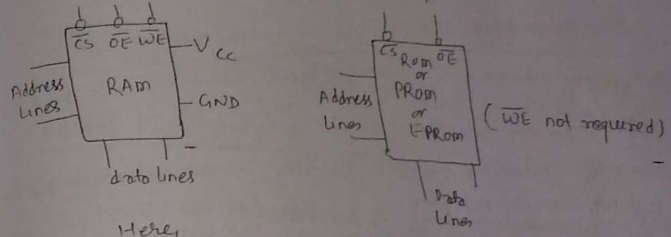
- Internally a memory consists of: address decoder, input buffer, output buffer, registers with address lines, data lines, and $\overline{RD}$, $\overline{WR}$, $\overline{CS}$ control lines.
- The number of address lines will be determined by the memory capability.
- The number of data lines will be determined by memory size.
- For example, for memory with capacity 1k×8 will have 10 address lines and 8 data lines. Similarly, 2k×4 chip will have 11 address lines and 4 data lines.
- For $2^n k \times m$ memory capacity, the number of address lines = n and the number of data lines = m.
- Let's consider the 8×8 memory device with 8-registers, a 3 to 8 decoder, an input buffer and an output buffer. The device will have 3 address lines and 8 data lines. It will also have control lines $\overline{RD}$, $\overline{WR}$ and $\overline{CS}$.
- To write an 8-bit word, the microprocessor places the register address on the 3 address line eg to write in the register 7, the microprocessor places 111 on the address lines.
  - The decoder decodes the address and selects the register 7.
  - Then the microprocessor places the data on the data bus and sends the active low $\overline{WR}$ control signal.
  - The control signal enables the input buffer and data are placed in the selected register.
- To read from memory, the process is similar to write operation, except that output buffer is enabled with $\overline{RD}$ signal.
- The remaining address lines of the microprocessor address bus are used to select the chip. ($\overline{CS}$).

Basic Concept in Memory Interfacing.

→ The primary function of memory interfacing is that the microprocessor should be able to read from and write into a given register of a memory chip.

→ To perform this operation, the microprocessor should,

1. be able to select the chip,
2. Identify the register,
3. Enable the appropriate buffer.



Here,

$\overline{OE}$ → Output Enable Buffer
$\overline{WE}$ → Write Enable Buffer
$\overline{CS}$ → Chip Selection Enable.

* for Interfacing Problems, follow Class notes.

Address Decoding:

→ Detection of the address of the peripheral device is called address decoding.

→ In this method, all the devices like-memory blocks, I/O units are assigned with specific address.

→ The address of a device is determined from the way in which the address lines from the processor are used to derive a special device selection signal known as Chip Select ($\overline{CS}$)

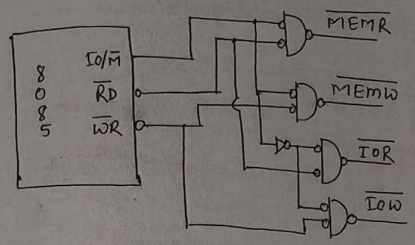
→ For example, if a processor has to write to or read from a memory block, the chip select ($\overline{CS}$) line should be enabled.
→ $\overline{CS}$ at other signals must not be enabled while selecting one chip.
→ There are 2 common methods for mapping address of these devices:

(i) I/O mapped I/O ; (ii) memory-mapped I/O.

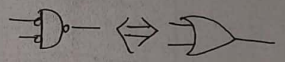
Differences between memory-mapped I/O and I/O mapped I/O.

memory-mapped I/O	I/O mapped I/O
(i) Device address - 16 bits for 8085, 20 bits for 8086.	(i) Device address - 8 bits for 8085 and 16-bits for 8086.
(ii) memory instructions like. STA, LDA, MOV, etc are used. eg STA 2000H (8085), MOV AX,2000H (8086)	(ii) Only IN, OUT instructions are used. eg. IN 02H (8085), IN 0003H (8086)
(iii) $\overline{MEMRD}$, $\overline{MEMWR}$ control signals are used.	(iii) $\overline{IORD}$, $\overline{IOWR}$ control signals are used
(iv) Data transfer is between any registers and I/O device	(iv) Data transfer is between accumulator and I/O device
(v) Some of memory space is used by I/O ports, so not available for memory	(v) Non of the system memory space is used for ports.

Generation of Control Signals



Note:



Address Decoding types:

(i) Absolute :

In this address decoding technique, all address lines that are not used for memory register must be decoded. So, chip select ($\overline{CS}$) ~~must~~ can be asserted by only one address.

(ii) Partial :

In this address decoding technique, some address lines left don't care. This technique reduces hardware but generates multiple addresses resulting in foldback memory space.

Memory Interfacing Examples:

Q.① Explain how 2KB RAM (or 2K*8 RAM or 2048*8 RAM) is interfaced with 8085 microprocessor.

Solution:

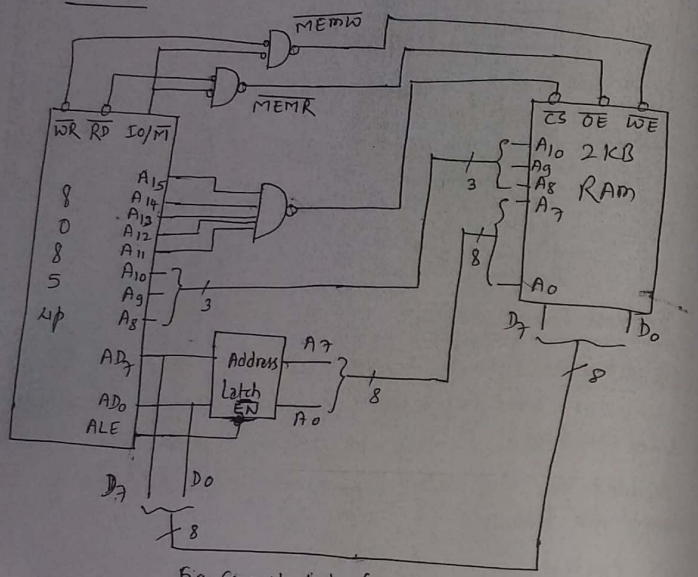


Fig. Circuit interfacing 2KB RAM with 8085 μp.

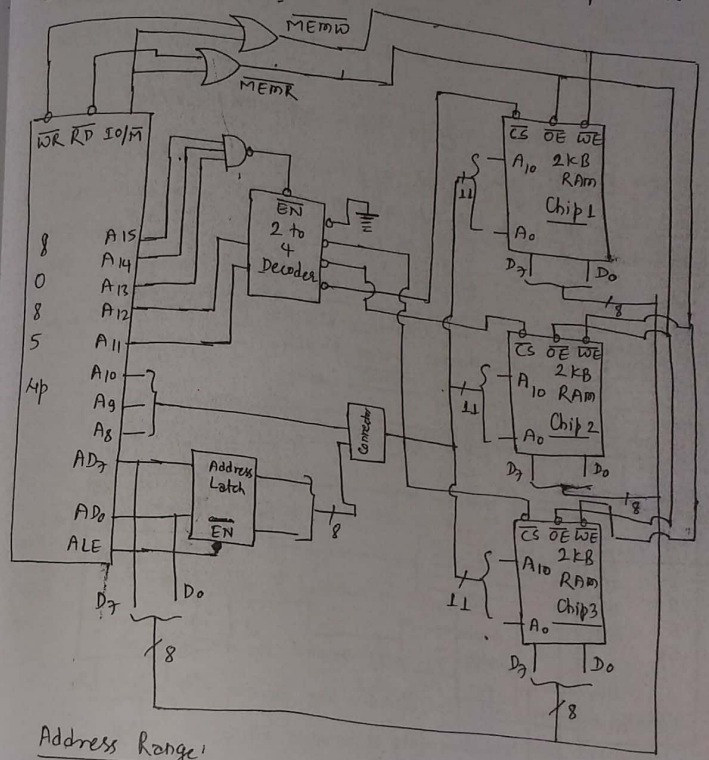
Address Range:

	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	
Starting address	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	= F800H
Final address	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	= FFFFH

Thus, Address for 2KB RAM ranges from F800H - FFFFH.

Q.② Interface three 2KB RAM with 8085 microprocessor.

Solution:



Address Range:

① For Chip 1:

	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	------	A_1	A_0	
Initial address	1	1	1	0	0	0	0	-----	0	0	= E000H
final address	1	1	1	0	0	1	1	.....	1	1	= E7FFH

So, address range of Chip 1 is E000H - E7FFH

② For Chip 2:

	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10} A_9 ---- A_0	
Initial add:	1	1	1	0	1	0 0 ---- 0	= E800H
Final add.	1	1	1	0	1	1 1 ---- 1	= EFFFH

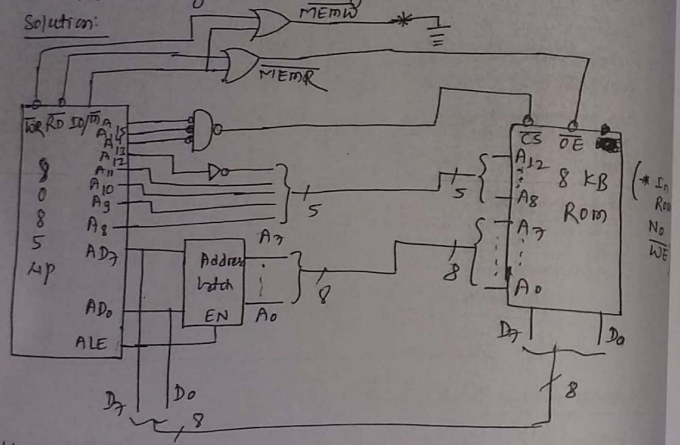
So, address range for chip 2 is E800H – EFFFH.

③ For Chip 3:

	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10} A_9 ---- A_0	
Initial add.	1	1	1	1	0	0 0 --- 0	= F000H
Final add.	1	1	1	1	0	1 1 --- 1	= F7FFH

Q. ③ Design a circuit to interface a 8 KB ROM with 8085 µp. The starting address being 1000H.

Solution:



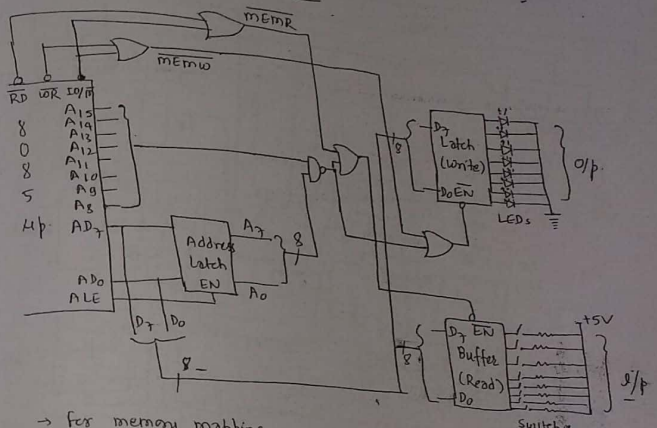
Here:

Address Range:

	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	----	A_1	A_0	
Starting address:	0	0	0	1	0	---	0	0	= 1000H
Final address:	0	0	0	1	1	----	1	1	= 1FFFH.

So, the address range is, 1000H – 1FFFH.

(#1) Interface 8085 microprocessor with 8 LEDs and 8 switches for memory mapped I/O



→ for memory mapping,

Port address for latch i.e. 8 LEDs is → A_{15} A_{14} ---- A_0

⇒ 1 1 --- 1 = FFFFH

and,

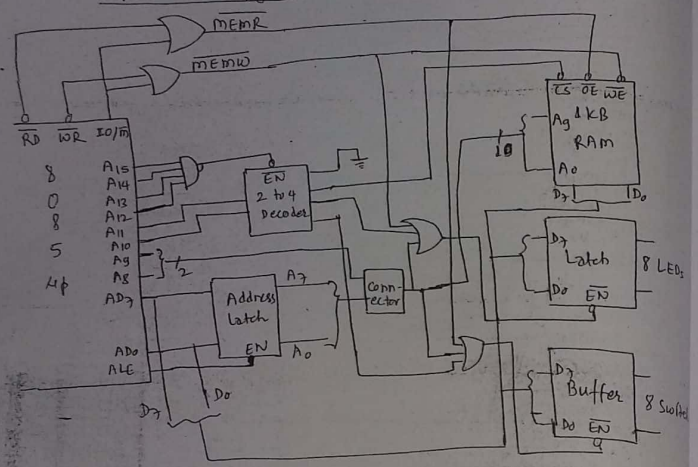
Port address for 8 switches is → A_{15} A_{14} ---- A_0

1 1 --- 1 = FFFFH

So, there is not address range in this case.

- Value at $\overline{IO/M}$ $\overline{MEMW}$ and $\overline{IO/M}$ $\overline{MEMR}$ helps to determine whether Read or write operation is performed and thus Latch or buffer is selected.

(#2) Interface 1 KB RAM, 8 LEDs and 8 switches for all memory-mapped I/O configuration:



Here, for RAM:

Address Range is:

Higher address: A_{15} ... A_0 : 1111 1011 1111 1111 = FBFFH (F B F F)

Lower address: 1111 1000 0000 0000 = F800H (F 8 0 0)

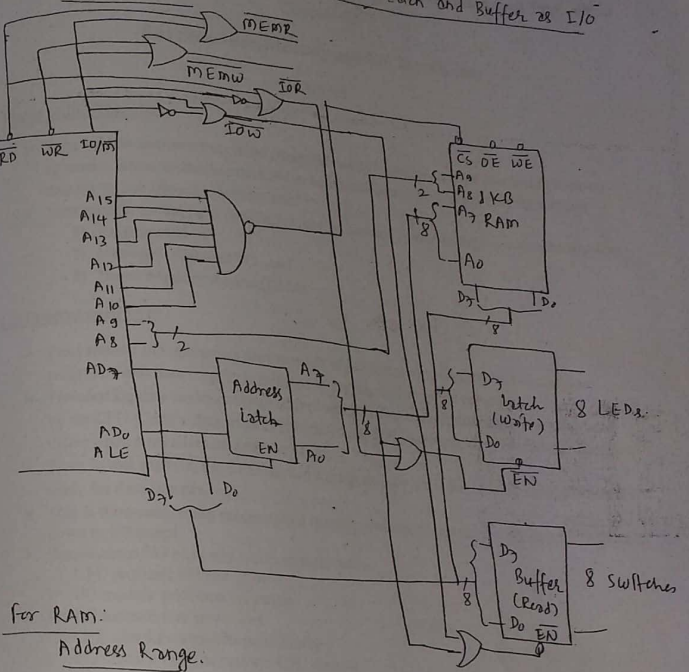
For Latch:

Address is: 1111 0100 0000 0000 i.e. F400H

For Buffer:

Address is: 1111 0000 0000 0000 i.e. F000H

(#3) Interface 1KB RAM as memory, and Latch and Buffer as I/O mapped I/O.



For RAM:

Address Range.

Initial/lower address: A_{15} A_{14} A_{13} A_{12} A_{11} A_{10} A_9 A_8 A_7 A_6 A_5 A_4 A_3 A_2 A_1 A_0
1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0
= FC00H

Final/Higher address: 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
= FFFFH.

For latch/Buffer:

Port address: 0000 0000 ~~0000 0000~~ = 00H.

Chapter-3

Bus Structure, Memory and I/O Interfacing

(13-hrs)

Modes of I/O Transfer:

- 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 the three possible modes:
  1) Programmed I/O,
  2) Interrupt – initiated I/O , and
  3) Direct Memory Access (DMA).

(1.) Programmed I/O:

- Programmed I/O operations are the result of I/O instructions written in the computer program. Each data transfer is initiated by an instruction in the program.
- 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.
- Thus, in this method, the CPU stays in a program loop until the I/O unit indicates that it is ready for data transfer.
- This is time-consuming process since it keeps processor busy needlessly and CPU slowed down to I/O speed.
- Programmed I/O basically works in these ways:
  - CPU requests I/O operation
  - I/O module performs operation
  - I/O module sets status bits
  - CPU checks status bits periodically
  - I/O module does not inform CPU directly
  - I/O module does not interrupt CPU
  - CPU may wait or come back later

- An flowchart for example of data transfer from an I/O device through an interface into the CPU is as shown below:

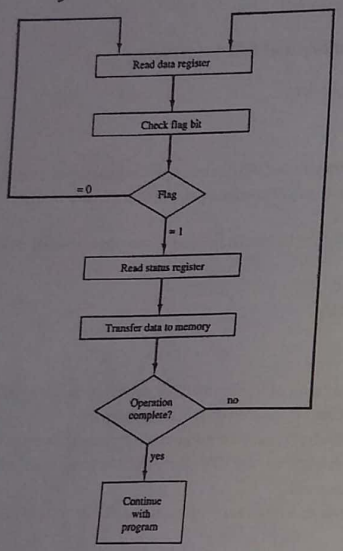


Figure 11-11 Flowchart for CPU program to input data.

- Here, it is assumed that the device is sending a sequence of bytes that must be stored in memory. The transfer of each byte requires 3 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.

- The programmed I/O method is particularly useful in small low- speed computers or in systems that are dedicated to monitor a device continuously.

(2) Interrupt Initiated I/O:

- An alternative to the CPU constantly monitoring the flag is to let the interface inform the computer, as an **interrupt** when it is ready to transfer data.
- In this mode, CPU does not check the flag while it is running. However, when 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. Now, 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 is doing before the interrupt.

- The CPU responds to the interrupt signal by storing the return address from the program counter into 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. There are two methods for accomplishing this:
  - **Vectored Interrupt (** in this the source that interrupts , supplies the branch information to the computer . **)**
  - **Non-Vectored Interrupt (** in this the branch address is assigned to a fixed location in memory . **)**

Below are the basic operations of Interrupt:

- CPU issues read command
- I/O module gets data from peripheral whilst CPU does other work
- I/O module interrupts CPU
- CPU requests data
- I/O module transfers data

(3.) Direct Memory Access(DMA):

- The transfer of data between a fast storage device such as magnetic disks and memory is often limited by the speed of the CPU.
- Removing the CPU from the path and letting the peripheral device manage the memory buses directly would improve the speed of transfer. This transfer technique is called Direct Memory Access (DMA).
- During DMA transfer, the CPU is idle, and has no control of the memory buses. A DMA controller takes over the buses to manage the transfer directly between the I/O device and memory.

- Direct Memory Access needs a special hardware called DMA controller (DMAC) that manages the data transfers and arbitrates access to the system bus. The controllers are programmed with source and destination pointers (where to read/write the data), counters to track the number of transferred bytes, and settings, which includes I/O and memory types, interrupts and states for the CPU cycles.

- DMA increases system concurrency by allowing the CPU to perform tasks while the DMA system transfers data via the system and memory busses.

INTERRUPTS

- An interrupt is a signal that a peripheral board sends to the central processor in order to request attention.
- In response to an interrupt, the processor stops what it is currently doing and executes a service routine.
- When the execution of the service routine is terminated, the original process may resume its previous operation.
- The interrupt is initiated by an external device and is asynchronous, meaning that it can be initiated at any time without reference to system clock. However, the response to an interrupt request is directed or controlled by the microprocessor.
- Interrupts can be classified into 2 types:
  - Maskable (Can be delayed), and
  - Non- maskable (Can't be delayed)

- Interrupts can also be classified into:
  - Vectored (the address of the service-routine is hard-wired i.e. address of ISR is already known to the microprocessor e.gs. RST 7.5, RST 6.5 & RST 5.5),
  - Non-vectored (the address of the service routine needs to be supplied externally i.e. The interrupting device needs to supply the address of ISR to microprocessor. E.g. INTR).

- Interrupts are primarily issued on:
  - Initiation of I/O operation.
  - Completion of I/O operation.
  - Occurrence of hardware or software errors.

Process of interrupt operation:

1. The I/O unit issues an interrupt signal to the processor. An interrupt signal from I/O is the request for exchange of data with the processor.

2. The processor finishes execution of the current instruction before responding to the interrupt.

3. The processor testes for an interrupt, determines that there is one, and sends an acknowledgement signal to the device that issued the interrupt. After receiving this acknowledgement, the device retains its interrupts signal.

4. The processor now begins to transfer the control to the routine which serves the interrupt request from the device. This routine is called '**Interrupt service routine (ISR)**' and it resides at a specified memory location. For this process, the CPU needs to save information needed to reassume the current program at the point of interrupt. The minimum information required is:
   i. The status of the processor, which contained by the **processor status word (PSW)**, and
   ii. the location of the next instruction to be executed which is contained by the **program counter (PC)**,
   these all are pushed onto the stack.

5. The processor then loads the program counter with the entry location of the interrupt service routine that will respond to this interrupt. Once the program counter has been loaded, the control is transferred to the interrupt handler program.

6. The fundamental requirement of the interrupt service routine is that it should begin by saving the contents of all the registers on the stack (as state of the main program should be safe). Suppose the user program is interrupted after the instruction at location N. The contents of all the registers plus the address of the next instruction are saved on the stack. The stack pointed is updated and the programs counter is updated to point to the beginning of the interrupt service routine.

7. The interrupt handler now proceeds to process the interrupt. This will include an examination of status information relating to the I/O operation or the other event that caused an interrupt. It may also involve sending additional commands or acknowledgement to the I/O unit.

8. When interrupt processing is complete the saved register's value (of the main program) are retrieved from the stack and restored to the register.

9. The final function is to restore the PSW and program counter (PC) values from the stack. As a result the next instruction to be executed will be from the previously interrupted main program.

Types of interrupt:

There are three major types of interrupts that cause a break in the normal execution of a program. They can be classified as:

1. External Interrupts,
2. Internal (Exception) Interrupts,
3. Software Interrupts.

External Interrupts:

- External interrupts are initiated via the microprocessor's interrupt pins by external devices such I/O devices, timing device, circuit monitoring the power supply, etc.
- Causes of these interrupts may be; I/O device requesting transfer of data, I/O device finished transfer of data, elapsed time of an event, or power failure.
- Timeout interrupt may result from a program that is an endless loop and thus exceeded its time allocation. Power failure interrupt may have as its service routine a program that transfers the complete state of the CPU into a non-destructive memory in few milliseconds before power ceases.
- External interrupts can be further divided into two types:
  i. Maskable interrupt, and
  ii. Non-maskable interrupt.

(i). **Maskable Interrupt:**

- A maskable interrupt is one which can be enabled or disabled by executing instructions such as EI (Enable interrupts) and DI (Disable interrupt).

- If the microprocessor's 'interrupt enable flip flop' is disabled, it ignores a maskable interrupt.

- In8085, the 1 byte instruction EI sets the interrupt enable flip flop and enables the interrupt process. Similarly the 1 byte instruction DI resets the interrupt enable flip flop and disables the interrupt process. No maskable interrupts are recognized by the processor when the interrupt is disabled.

(ii). ***Non-maskable* Interrupt:**

- This type of interrupt cannot be enabled or disabled by instructions.

- This type has higher priority over maskable interrupt. This means that if both the makable and non maskable interrupts are activated at the same time, then the processor will service the non-maskable interrupt first.

- In 8085 TRAP is an example of non maskable interrupt.

Microprocessors- Chapter-3
Prepared By: Er. Ramu Pandey, Asst. Prof. , NCIT

Internal Interrupts:

- Internal interrupt arise from illegal or erroneous use of an instruction or data. Cause of this interrupt may be: register overflow, attempt to divide by zero, an invalid operation code, stack overflow etc.

- These error conditions usually occur as a result of premature termination of the instruction execution. These are even termed as exceptions.

- The difference between internal and external interrupt are :
  - The internal interrupt is initiated by some exceptional conditions caused by the program itself rather than by external events.
  - Internal interrupts are synchronous with the program, while external interrupts are asynchronous.
  - If the program is return, the internal interrupt will occur in the same place each time. External interrupts depends on external conditions that are independent of the program being executed at the time.

Software Interrupt:

- External and internal interrupts are initiated from signal that occurs in the hardware of the cpu. A software interrupt is initiated by executing an instruction.
- Software interrupt is a special **call** instruction that behaves like an interrupt rather than a subroutine call. It can be used by the programmer to initiate an interrupt procedure at any desired point in the program.
- The most common use of software interrupt is associated with a supervisor call instruction. This instruction provides means for switching from a CPU user mode to the supervisor mode.
- Certain operations in the computer may be assigned to the supervisor mode only, as for example, a complex input or output transfer procedure.
- In 8085 the instruction like RST0, RST1, RST2, RST3…..etc. causes a software interrupt.
-

Interrupt Priority:

- 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 process the required transfer.

- In micro-computer a number of I/O device are attached to the processor, 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.
- An interrupt priority is a system that established a priority over the various sources to determine which condition is to be serviced first when two or more request 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 request which, if delayed or interrupted, could have serious consequences. Device with higher speed transfers such as magnetic disks are given high priority, and slow devices such as keyboards receive low priority.
- When two devices interrupt the processor at the same time, the processor service the device, with the higher priority first.

Interrupts of 8085:

The 8085 has five interrupts which are as follows:

i. TRAP,
ii. RST 7.5,
iii. RST 6.5,
iv. RST 5.5, and
v. INTR

Priority ↓

Interrupt Vectors and the Vector Table:

- An interrupt vector is a pointer to where the ISR (Interrupt Service Routine) is stored in memory.
- All interrupts (vectored or otherwise) are mapped onto a memory area called the Interrupt Vector Table (IVT).
- The IVT is usually located in memory page 00 (0000H - 00FFH).
- The purpose of the IVT is to hold the vectors that redirect the microprocessor to the right place when an interrupt arrives.
- The IVT is divided into several blocks. Each block is used by one of the interrupts to hold its "vector"

Interrupt name	Maskable	Vectored
INTR	Yes	No
RST 5.5	Yes	Yes
RST 6.5	Yes	Yes
RST 7.5	Yes	Yes
TRAP	No	Yes

- There are mainly two ways of servicing multiple interrupts. These are:
  1. Polled Interrupt,
  2. Chained (Vectored) Interrupt.

Polled Interrupt:

- Polled interrupt are handled using software and are therefore slower compared to vectored (hardware) interrupts.
- 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 interrupts sources in sequence.
- The order in which they are tested determines the priority of each interrupt. The highest priority source is tested first, and if its interrupt signal is on, control branches to a service routine for this source. Otherwise the next lower priority source is tested, and so on.
- Thus, the initial service routine for all interrupts consists of a program that tests the interrupt sources in sequence and branches to one of many possible service routines.
- Polled interrupts are very simple. But for large number of devices, the time required to poll each device may exceed the time to service the device.

- In such case, the faster mechanism called chained interrupt is used.

Vectored (Chained) Interrupts:

- 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.

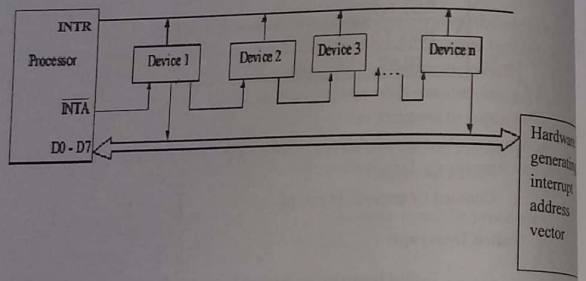


Fig. Block diagram to show Vectored (Chained) Interrupt

- Here the device with the highest priority is 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 ($\overline{INTA}$) signal to the highest priority device, which is device 1 in our case. If this device has generated the interrupt it will accept the $\overline{INTA}$ signal from the processor; otherwise, it will pass $\overline{INTA}$ on to the next device until the $\overline{INTA}$ 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.

➢ For 8085, the four interrupts – TRAP, RST 7.5, RST 6.5 and RST 5.5 are automatically vectored (transferred) to specific locations without any external hardware.
➢ They do not require $\overline{INTA}$ signal or an input port; the necessary hardware is already implemented inside the 8085.
➢ These 8085 vectored interrupts and their call locations are as shown below:

8085 Interrupt	Call Location
TRAP	0024H
RST 7.5	003CH
RST 6.5	0034H
RST 5.5	002CH

INTR:

- This interrupt is maskable. It can be enabled by instruction EI and can be disabled by instruction DI. The INTR interrupt requires external hardware to transfer program sequence to specific CALL locations.
- There are 8 a number of CALL-Locations for INTR interrupt. The hardware circuit generates RST codes for this purpose and places that on the data bus externally.
- When the microprocessor is executing a program, it checks the INTR line (when interrupt enable flip flop is enabled using EI instruction) during the execution of each instruction.
- If the line is high and the interrupt is enabled, the microprocessor completes the currents instruction, disabled the interrupt enable flip flop and sends a $\overline{INTA}$ signal. The processor does not accept any interrupt requests until the interrupt flip flop is enabled again.
- The signal INTA is used to insert a Restart (RST) instruction, (it saves the memory address of the next instruction to the stack. The program is transferred to the call location).
- The RST instruction and their call locations are :

Instruction	Hex-code	Call location
RST 0	C7	0000
RST 1	CF	0008
RST 2	D7	0010
RST 3	DF	0018
RST 4	E7	0020
RST 5	EF	0028
RST 6	F7	0030
RST 7	FF	0038

TRAP:

- It is a non maskable interrupt. It has the highest priority among the interrupt signal.
- It need not be enabled and it cannot be disabled.
- When this interrupt is triggered the program control is transferred to the location 0024 H without any external hardware or the interrupt enable instruction.
- TRAP is generally used for such critical events as power failure and emergency shut off.

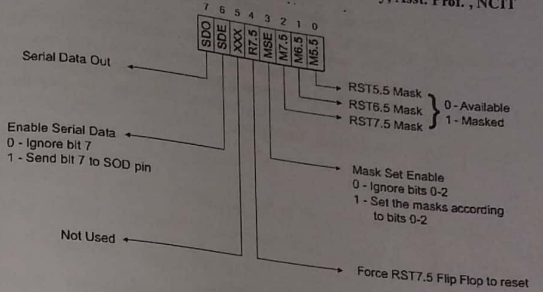
RST 7.5, 6.5, 5.5:

- These interrupts are maskable and are enabled by software using instructions EI and SIM (Set Interrupt Mask).
- The execution of the instruction SIM enables/disables the interrupts according to the bit pattern of the accumulator.

SIM (Set Interrupt Mask) Instruction:

- *It is a 1 byte instruction.*
- *It has 3 main functions.*

Accumulator bit pattern for SIM Instruction:



Functions:

i. Set mask for RST 7.5, 6.5, and 5.5 interrupts. Bit D_3 is a control bit and should be 1 for bits D_0, D_1, and D_2 to be effective. Logic 0 on D_0, D_1, and D_2 will enable the corresponding interrupts and logic 1 will disable the interrupts.

ii. SIM instruction reads the content of accumulator and enables or disables the interrupts according to the contents of accumulator. e.g. D_4 is an additional control bit for M7.5, if D_4=1, then RST 7.5 is reset.

iii. To implement the serial input, output. Bits D_7 and D_6 of the accumulator are used for serial input output and don't affect the interrupts.

Example:

Write instructions to enable interrupt RST 5.5 and mask other interrupts.

Solution:

```
Instruction: MVI A, OEH    ; Bits D3 = 1 and D0 = 0
             SIM           ; Enable RST 5.5
```

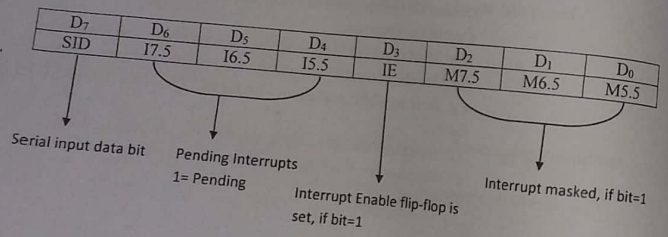
Pending Interrupt:

- When one interrupt request is being served, other interrupt may occur resulting in a pending request.
- When more than one interrupts occur simultaneously the interrupts having higher priority is served and the interrupts with lower priority remain pending.
- The 8085 has an instruction RIM using which the programmer can know the current status of pending interrupts. This instruction gives the current status of only maskable interrupts.

RIM (Read Interrupt Mask) Instruction:

- It is also a 1 byte instruction.

Accumulator bit pattern for RIM Instruction:



- This instruction loads the accumulator with eight bits indicating the current status of the interrupt mask.
- To identify the pending interrupts, Bits D_4, D_5 and D_6 are checked.
- To receive serial data, bit D_7 is used.

Example:

Check whether RST 6.5 is pending or not.

Solution:

```
RIM          ; loads accumulator with the data content of RIM instruction.
AIN 20H ; AND's D5 bit of accumulator to check RST6.5
JNZ Pending; If D5 = 1, then pending RST6.5 is pending
   otherwise not pending.
```

8086 Interrupts:

- Interrupt refers to the disturbance to the processor's task.
- Most microprocessors allow normal program execution to be interrupted by some external signal or by special instruction in the program.
- In response to an interrupt, the microprocessor stops executing the current program and calls a procedure which "services" the interrupt.
- An IRET instruction at the end of the interrupt-service procedure returns execution to the interrupted program.

Sources of 8086 interrupts:

(i) Hardware Interrupt:
- This interrupt is caused by external signal applied to the Non-maskable Interrupt (NMI) input pin or to the Interrupt (INTR) input pin.

(ii) Software Interrupt:
- This interrupt is caused by execution of Interrupt Instruction, INT.

(iii) Internal Interrupt:
- This interrupt is caused by some error conditions produced in the 8086 by the execution of an instruction.
- Examples of these conditions are, divide by zero, register overflow, etc.

Steps of the interrupt response in 8086:

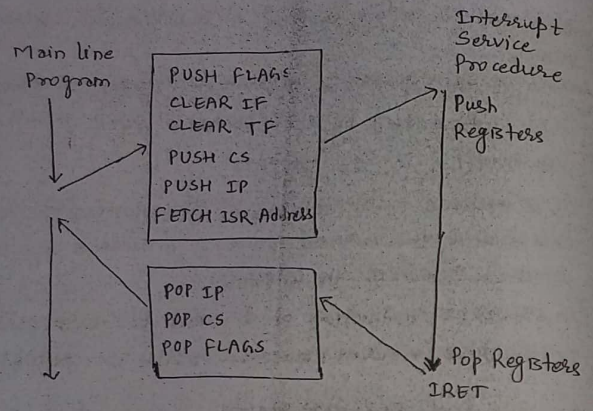


Fig. 8086 Interrupt Response

- At the end of each instruction cycle, the 8086 checks to see if any interrupts have been requested. If an interrupt has been requested, the 8086 responds to the interrupt by stepping through the following series of major actions:

1. It decrements the Stack Pointer by 2 and pushes the flag register on the stack.
2. It disables the 8086-INTR interrupt input by clearing the interrupt flag (IF) in the flag register.
3. It resets the traf flag (TF) in the flag register.
4. It decrements the stack Pointer by 2 and pushes the current code segment register content on the stack.
5. It decrements the stack pointer again by 2 and pushes the current instruction pointer (IP) contents on the stack.
6. It does an indirect far jump to the start of the interrupt procedure.

Interrupt Vector Table (IVT):

→ In 8086 System, the first 1 kB of memory from 00000H to 003FFH, is set aside as a table for storing the starting addresses of interrupt service procedures. This table is called Interrupt Vector Table (IVT).

→ Since 4 bytes are required to store the CS and IP values for each interrupt-service procedures, the table can hold the starting addresses for upto 256 interrupt procedures.

→ The starting address of an interrupt service-procedure is often called the interrupt vector or the interrupt pointer, so the table is referred to as interrupt-vector table or the interrupt-pointer table.

→ Figure below shows the interrupt vector table indicating the arrangement of 256 interrupt vectors.

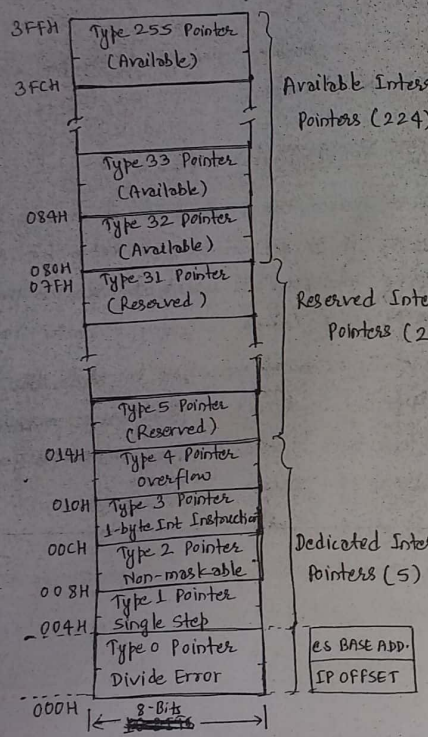


Fig 8086 Interrupt Pointer Table

→ The 256 Interrupt vectors are classified into 3 sections :-

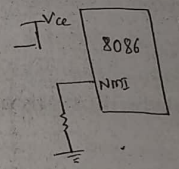
(i) The lowest five types are pre-defined and used for specific functions.

(ii) Next 27 vectors from 5 to 31 are reserved for use in future microprocessors.

(iii) Further 224 vectors from 32 to 255 are available for use as hardware or software interrupts.

How address is obtained of respective interrupt type from Interrupt vector Table (IVT) ?

⇒



NMI = Type 2 Pointer.

→ When NMI occurs,

① microprocessor jumps to IVT table to get the address of ISR.

② first, IP value = 4 * Interrupt type = 4 * 2 = 08H

Second, CS base address = 2 + 4 * (Interrupt type)

= 2 + (4 * 2) = 0AH.

8086 Interrupt Types:

① Software Interrupts:

→ initiated by executing an interrupt instruction -

INT interrupt-type.

Where, interrupt type is an integer in the range 0 to 255.

→ Each interrupt type can be parameterized to provide several services - for example, DOS interrupt service, INT 21H provides more than 80 different services.

Applications:

→ can be used to insert break-point [INT 3] in the program for debugging.

→ Various interrupt service routine can be tested and each routine can be used to perform some specific task such as reading a character from keyboard, writing a character on CRT, etc.

(ii) Hardware Interrupts:

→ initiated by applying an electrical signal to the processor chip.

→ Hardware interrupts can be maskable & Non-maskable.

→ INTR pin of microprocessor is used for maskable interrupt. These interrupts can be enabled by setting IF (Interrupt enable flag) and disabled by clearing IF flag.

→ Non-maskable interrupt can be triggered by applying an electrical signal to the NMI pin of microprocessor. The ~~is~~ microprocessor always responds to this signal and cannot be disabled.

Applications:

- NMI can be used for catastrophic failures such as power failure.
- A large number of devices can be serviced using hardware interrupts with the help of 8259 PIC.

(iii) Pre-defined Interrupts:

→ These includes first five interrupts, which are reserved for specific purpose.

(a) Type 0: Divide-Error Interrupt:

- Issued by INT 00H instruction.
- Type 0 on INTR pin occurs, whenever the result of the division overflows or whenever an attempt is made to divide by zero.

(b) Type 1: Single-Step Interrupt:

→ For single-step interrupt, Trap flag (TF) should be set.

→ CPU automatically generates a type 1 interrupt after executing each instruction, if TF is set.

→ Thus, single-step interrupt is useful in monitoring and debugging programs.

(c) Type 2: Non-maskable Interrupt:

→ This interrupt is initiated when NMI pin of MPU receives a low-to-high transition. This interrupt is normally used for catastrophic conditions such as power failure.

→ Whenever, there is a failure of ac power to the system, it is detected by some external circuitry and an interrupt signal is sent to NMI pin of MPU.

(d) Type 3: Break-point Interrupt:

→ This interrupt is used for break-point and is non-maskable.

→ When we insert a breakpoint, the system executes the instruction upto the breakpoint and then goes to the breakpoint subroutine.

→ Breakpoint interrupt can be generated by executing INT 3 instruction.

→ It is useful in debugging.

③ Type 4 : Overflow Interrupt

→ Overflow interrupt can be generated by two ways:-

- by executing INT 4 (unconditionally generates type-4 interrupt)
- by executing INT 0 instruction (Interrupt is generated only if the overflow flag is set)

→ We donot normally use INT 0 as we can use JO/JNO conditional jumps to take care of overflow.

Interrupt Priority:

→ To handle the situation of occurrence of two or more interrupts at the same time, priority has been fixed for the interrupts. 8086 MPU has the following interrupt priorities:-

Interrupt	Priority
* Divide Error, INT 0	Highest
NMI	⋮
INTR	⋮
Single-Step	Lowest

*[Internal


```
ANI 20H      ; ANDs D5 bit of accumulator to check RST 6.5
JNZ Pending  ; If D5=1, then pending RST 6.5 is pending otherwise not pending.
```

8259 – Programmable/Priority Interrupt Controller (PIC)

- The 8259 is a programmable/priority interrupt controller designed to work with Intel Microprocessors 8085, 8086 and 8088.
- 8259 is used to manage and resolve the interrupts issued from a number of input, output devices at the same time.
- For example we might read ASCII characters in from a keyboard on an interrupt basis; count interrupts from a timer to produce a real-time clock of seconds,minutes,and hours; and detect several emergency or job-done conditions on an interrupt basis. Each of these interrupt applications requires a separate interrupt input. If we are working with 8086, we have a problem here because the 8086 has only two interrupt inputs, NMI and INTR. If we save NMI for power failure interrupt, this leaves only one interrupt input for all other applications.
  For the applications where we have interrupts from multiple sources, we use an external device called PIC to funnel the interrupt signals into a single interrupt input on the processor.
- 8259 is a most common PIC that accepts interrupt requests from up to 8 I/O devices (in single mode) and up to 64 priority interrupts of 64 devices (in cascaded mode).

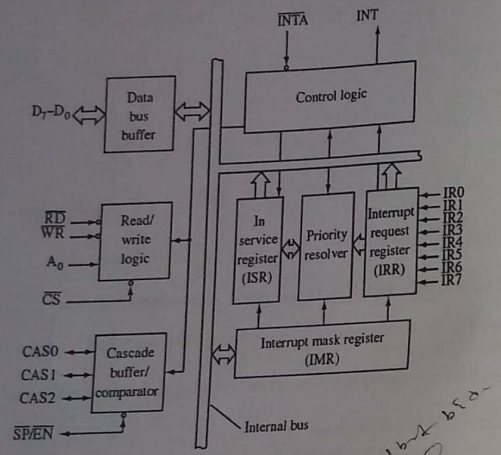
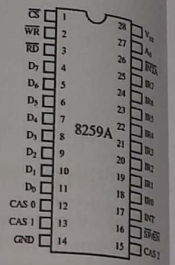


Fig.(1- a) Block Diagram of 8259A PIC



(1-b) Pin Configuration of 8259A PIC

Explanation:

Fig. (1-a) shows the functional block diagram of the 8259A PIC. It is constituted of four sections:

1. Interrupt and Control Logic Section,
2. Data Bus Buffer Section,
3. Read/Write Control Logic Section, and
4. Cascade Buffer/ Comparator Section.

(1) Interrupt and Control Logic Section:

This section consists of five sub-sections as explained below:

(a) Interrupt Request Register(IRR):

- This eight bit register is used to store the status of all eight interrupt input lines ($IR_0 - IR_7$) which are requesting service. The eight interrupt input set corresponding bits of the interrupt request register.

(b) In-Service Register(ISR):

- This register is used to store the information about the interrupt levels that are currently being serviced.

(c) Interrupt Mask Register(IMR):

- This register stores the masking bits of the interrupt lines to be masked.
- The IMR operates on the IRR.
- Masking of a higher priority input will not affect the interrupt request lines of lower quality.
- This register can be programmed by an OCW(Operation Command Word) to store the bits which mask specific interrupts.

(d) Priority Resolver:

- This logic block determines the priorities of the bit set in the IRR.
- The bit corresponding to the highest priority interrupt is selected and strobed into corresponding bit of ISR during $\overline{INTA}$ pulse.

(e) Control Logic Block:

- This block has two pins : INT(Interrupt) as an output and $\overline{INTA}$ (Interrupt Acknowledge) as an input.
- The INT is connected to the interrupt pin of the MPU. Whenever the valid interrupt is asserted , this signal goes high.
- The $\overline{INTA}$ is an interrupt acknowledge signal from the MPU. $\overline{INTA}$ pulse will cause the 8259A to release vectoring information onto the data bus.

(2) Data Bus Buffer Section:

- The 8-bit tri-state, bidirectional 8 bit buffer is used to interface the 8259A to the system data bus.
- Control Words and Status information are transferred through the Data Bus Buffer.

(3) Read/Write Control Logic Section :

- The function of this block is to accept output commands from the CPU.
- It contains the Initialization Command Word (ICW) and Operation Command Word (OCW) registers which store the various control formats for device operation and setup 8259 to operate in various modes.
- The different input lines to Read/Write Control Logic are as follows:

  $\overline{CS}$ **(CHIP SELECT)**:

  A LOW on this input enables the 8259A. No reading or writing will occur unless the device is selected.

$\overline{WR}$ (WRITE):
A LOW on this input enables the CPU to write control words (ICWs and OCW_s) to the 8259A.

$\overline{RD}$(READ):
A LOW on this input enables the 8259A to send the status of the Interrupt Request Register(IRR), In-Service Register(ISR), the Interrupt Mask Register(IMR), or the interrupt level on the data bus.

A_0:
This input signal is used in conjunction with $\overline{WR}$ and $\overline{RD}$ signals to write commands into various command registers, as well as reading the various status registers of the chip. This line can be tied directly to one of the address lines.

(4) Cascade Buffer/Comparator Section:

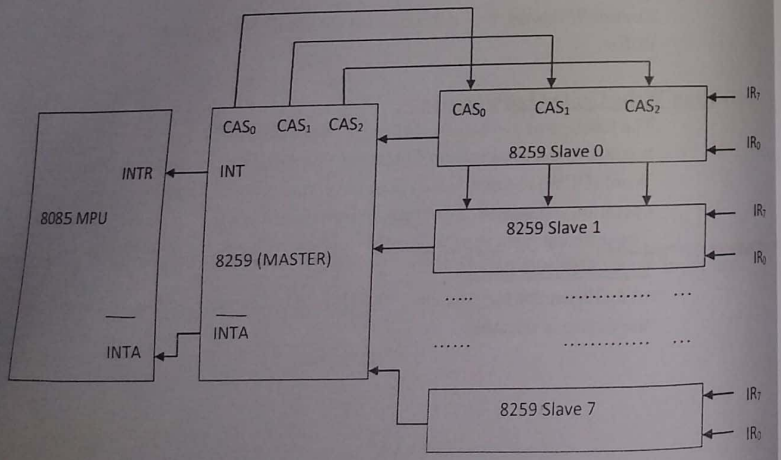


Fig. 2 – Cascading the 8259A

- The fig. 2 above shows how 8259s chips are connected in cascaded format in master-slave pattern, which makes it possible to accept upto 64 interrupt lines from different devices.
- The Cascade Buffer/Comparator block stores and compares the IDs of all 8259As used in the system.
- The associated three I/O pins (CAS_{0-2}) are outputs when the 8259A is used as a master and are inputs when 8259A is used as slave.
- The master puts out an identification code of 3 bits (8 devices) on the CAS_0 to CAS_2 lines. Slave 8259As accept these three signal inputs compared with the codes assigned during initialization. The slave thus selected will send its preprogrammed subroutine address onto the Data Bus during the next one or two consecutive $\overline{INTA}$ pulse.

$\overline{SP}$/ $\overline{EN}$(SLAVE PROGRAM / ENABLE BUFFER):
This is a dual function pin. When in the buffered mode, it can be used as an output to control buffer transceivers (EN). When not in buffered mode, it is used as an input to designate a master (SP=1 /SP =0).

- Buffered or non-buffered mode of operation of 8259 can be specified at the time of initialization of 8259.

8259
LS

Sequence of Operation of 8259A:

- Peripheral raises one or more INTERRUPT REQUEST (IR7-0) lines high, setting the corresponding IRR bit(s).
- Priority resolver checks three registers i.e. IRR for interrupt requests, IMR for masking bit and ISR for Interrupt Request being serviced. It resolves the priority and sets the INT high.
- The CPU or microprocessor acknowledges the INT and responds with an $\overline{INTA}$ pulse
- Upon receiving the $\overline{INTA}$ pulse from the CPU, the highest priority ISR bit is set and the corresponding IRR bit is reset. In case of 8085, the 8259A will also

release a CALL instruction code (11001101) onto the 8-bit Data Bus through its D7-D0 pins.

- This CALL instruction will initiate two more $\overline{INTA}$ pulses to be sent to the 8259A from the CPU group.
- These two $\overline{INTA}$ pulses allow the 8259A to release its preprogrammed subroutine address onto the Data Bus. The lower 8-bit address is released at the first $\overline{INTA}$ pulse and higher 8-bit address is released at the second $\overline{INTA}$ pulse.
- This completes the interrupt cycle. In the AEOI (Automatic End of Interrupt) mode, the ISR bit is reset at the end of the third INTA pulse. Otherwise, the ISR bit remains set until an appropriate EOI (End of Interrupt) command is issued at the end of the interrupt sequence.

8259 Programming:

Microprocessors- Chapter-3
Prepared By: Er. Ramu Pandey, Asst. Prof. , NCIT

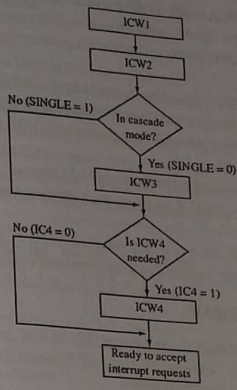


Fig. 3: 8259A Initialization Flowchart

- The 8259A accepts two types of command words generated by the CPU:

1. **Initialization Command Words(ICWs):**
   - Before normal operation can begin, each 8259A in the system must be brought to a starting point- by a sequence of 2- 4 bytes times by $\overline{WR}$ pulses.

2. **Operation Command Words(OCWs):**
   - These are the command words which command the 8259A to operate in various interrupt modes. These modes are:
   (a) Fully Nested Mode (FNM),
   (b) Rotating Priority Mode,
   (c) Special Mask Mode, and
   (d) Polled Mode.

   - The OCWs can be written into the 8259A anytime after initialization.

Modes of Operation of 8259A:

1. **Fully Nested Mode(FNM)**:

- It is the default mode set after initialization i.e. this mode is entered after initialization, unless another mode is programmed or changed through OCW.
- The interrupt requests are ordered in priority from 0 to 7 (0 highest and 7 lowest priorities).
- When interrupt request is acknowledged, the 8259A determines the interrupt of the highest priority and set corresponding bit in the ISR.
- ISR bit remains set until End of Interrupt (EOI) command through an OCW is issued by CPU.
- If the Automatic End of Interrupt (AEOI) mode is set in ICW4 during initialization, the ISR bit is automatically reset on the trailing edge of the last $\overline{INTA}$ pulse.
- While the ISR bit is set, all further interrupts of same or lower priority are inhibited, while higher levels will generate an interrupt.

End of Interrupt (EOI):

- EOI is used to reset the In Service (IS) bit that indicates the service bit.
- Two types: (i). Non-specific EOI, (ii) Specific EOI

(i) **Non-specific EOI**:
- When 8259A is operated in modes which preserve the fully nested structure, it can determine which IS bit to reset on EOI.
- issued by OCW2 (EOI=1, SL=0, R=0)
- If FNM is not used, specific EOI command will have to be issued.
- In Specific EOI the L0, L1 and L2 of OCW2 specifies the level on which EOI command is to cut on.
- Resets the highest IS bit of these interrupt requested level which were set automatically before the end of service routine.

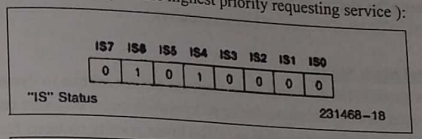
(ii) ***Specific EOI:***
- When a mode is used which may disturb the fully nested structure, the 8259A may no longer be able to determine the last level acknowledged. In this case a Specific End of Interrupt must be issued which includes as part of the command the IS level to be reset.
- A specific EOI can be issued with OCW2 (EOI =1, SL=1, R=0, and L0-L2 is the binary level of the IS bit to be reset).

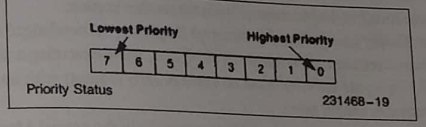
Special Fully Nested Mode (SFNM):
- Used in Cascade connection.
- allows further interrupt requests from slaves.
- SFNM is set up by ICW4 during initialization.

2. **Rotation Priority Mode:**
   (a) Automatic Rotation,
   (b) Specific Rotation

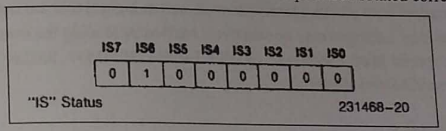
(a) **Automatic Rotation** (Equal Priority Devices):
- In some applications there are a number of interrupting devices of equal priority. In this mode, a device after being serviced, receives the lowest priority, so, a device requesting an interrupt will have to wait , in the worst case until each of 7 other devices are serviced at most once.
- For example, if the priority and 'in-service' status is:

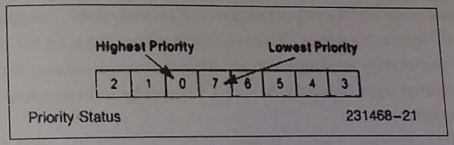
Before Rotate (IR4 the highest priority requesting service):





After Rotate (IR4 was serviced, all other priorities rotated correspondingly):





- There are 2 ways to accomplish Automatic Rotation using OCW2, the rotation on Non-specific EOI command (R=1, SL=0, EOI=1) and the Rotate in Automatic EOI Mode which is set by (R=1, SL=0, EOI=0) and cleared by (R=0, SL=0, EOI=0)

(b) Specific Rotation(Specific Priority):

- The programmer can change priorities by programming the bottom priority and thus fixing all other priorities; i.e., if IR5 is programmed as the bottom priority device, then IR6 will have the highest one.
- The Set Priority command is issued in OCW2 where: R=1, SL=1, L0- L2 is the binary priority level code of the bottom priority device.
- Observe that in this mode internal status is updated by software control during OCW2. However, it is independent of the End of Interrupt (EOI) command (also executed by OCW2). Priority changes can be executed during an EOI command by using the Rotate on Specific EOI command in OCW2 (R =1, SL=1, EOI=1 and LO- L2= IR level to receive bottom priority).

3. **Special Mask Mode:**

- Some applications may require an interrupt service routine to dynamically alter the system priority structure during its execution under software control. For example, the routine may wish to inhibit lower priority requests for a portion of its execution but enable some of them for another portion.
- The difficulty here is that if an Interrupt Request is acknowledged and an End of Interrupt command did not reset its IS bit (i.e., while executing a service routine), the 8259A would have inhibited all lower priority requests with no easy way for the routine to enable them.
- That is where the Special Mask Mode comes in. In the special Mask Mode, when a mask bit is set in OCW1, it inhibits further interrupts at that level and enables interrupts from all other levels (lower as well as higher) that are not masked.
- Thus, any interrupts may be selectively enabled by loading the mask register.
- The special Mask Mode is set by OWC3 where: SSMM=1, SMM=1, and cleared where SSMM=1, SMM=0.

4. **Polled Mode:**

- In Poll mode the INT output functions as it normally does. The microprocessor should ignore this output. This can be accomplished either by not connecting the INT output or by masking interrupts within the microprocessor, thereby disabling its interrupt input. Service to devices is achieved by software using a Poll command.
- The Poll command is issued by setting P=`1" in OCW3. The 8259A treats the next $\overline{RD}$ pulse to the 8259A (i.e., $\overline{RD}$=0, $\overline{CS}$=0) as an interrupt acknowledge, sets the appropriate IS bit if there is a request, and reads the priority level. Interrupt is frozen from $\overline{WR}$ to $\overline{RD}$.

The word enabled onto the data bus during $\overline{RD}$ is:

D7	D6	D5	D4	D3	D2	D1	D0
I	—	—	—	—	W2	W1	W0

W0–W2: Binary code of the highest priority level requesting service.

I: Equal to "1" if there is an interrupt.

- This mode is useful if there is a routine command common to several levels so that the INTA sequence is not needed (saves ROM space). Another application is to use the poll mode to expand the number of priority levels to more than 64.

8255A- PROGRAMMABLE PERIPHERAL INTERFACE (PPI)

- The 8255A is a widely used, programmable, parallel I/O device that provides three 8-bit input/output ports in one 40-pin IC package.
- It can be programmed to transfer data under various conditions, from simple I/O to interrupt I/O.
- Mostly it is used to manage the handshake operation automatically, for which it can be programmed to automatically receive an $\overline{STB}$ signal from a peripheral, send an interrupt signal to the processor, and send an ACK signal to the peripheral at proper times.
- It is flexible, versatile, and economical (when multiple I/O ports are required).
- It is an important general-purpose I/O device that can be used with almost any microprocessor.
- It has 24 I/O pins that can be grouped primarily in two 8-bits parallel ports: A and B , with the remaining 8 bits as port C.
- The eight bits of Port C can be used as individual bits or be grouped in two 4-bit ports : C_{UPPER} (C_U) and C_{LOWER}(C_L).
- The function of these ports is defined by writing a control word in the control register.
- Basically the functions of 8255A are classified into two modes:
  1. Bit Set/Reset (BSR) mode, and
  2. I/O mode.
- The BSR mode is used to set or reset the bits in port C.
- The I/O mode is further divided into three modes:
  1. Mode 0,
  2. Mode 1, and
  3. Mode 2.
- In Mode 0, all ports function as simple I/O ports.
- Mode 1 is a handshake mode whereby ports A and/or B, use bits from port C as handshake signals. In the handshake mode, two types of I/O data transfer can be implemented: status check and interrupt.
- In Mode 2, port A can be set up for bidirectional data transfer using handshake signals from port C, and port B can be set up either in Mode 0 or Mode 1.

Various Applications:

(i) *Printer Interface,*
(ii) *Keyboard and Display Interface,*
(iii) *ADC and DAC Interface,*
(iv) *Floppy Disk Interface, etc.*

Why 8255A is used?

- to expand the I/O ports,
- no use of external combinational logic.

INTERNAL BLOCK DIAGRAM OF 8255A PPI:

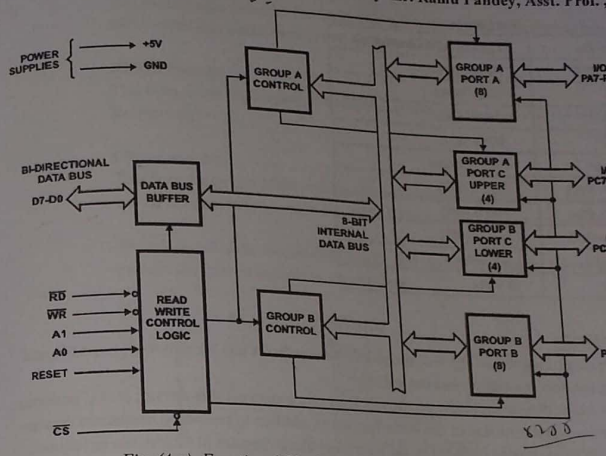
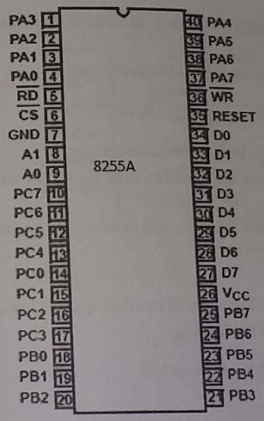
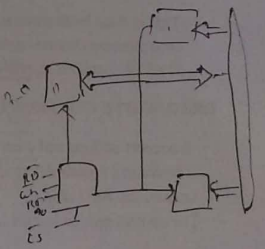


Fig. (4-a). Functional Block Diagram of 8255A PPI

Pin Configuration





PIN NAMES WITH DESCRIPTIONS:

Pin Names	
D_7-D_0	Data Bus (Bi-Directional)
RESET	Reset Input
$\overline{CS}$	Chip Select
$\overline{RD}$	Read Input
$\overline{WR}$	Write Input
A0, A1	Port Address
PA7–PA0	Port A (BIT)
PB7–PB0	Port B (BIT)
PC7–PC0	Port C (BIT)
V_{CC}	+5 Volts
GND	0 Volts

DESCRIPTION:

- The block diagram in Fig. 4-a, shows two 8-bit ports (A and B), two 4-bit ports (C_U and C_L), the data bus buffer, and control logic.
- This block diagram includes all the elements of a programmable device; port C performs functions similar to that of the status register in addition to providing handshake signals.
- The different blocks included in the functional block diagram of 8255A are as follows:

DATA BUS BUFFER:

- This is 8-bit bi-directional buffer used to interface the internal data bus of 8255 with external (system) data bus.
- The CPU transfers data to and from the 8255 through this buffer.

READ/WRITE CONTROL LOGIC:

- It accepts address and control signals from the microprocessor.
- The control signals determine whether it is a read or write operation and also set or reset the 8255 chip.
- The control section has six lines. Their functions and connections are as follows:

$\overline{RD}$(Read):
This control signal enables the Read Operation. When the signal is low, the MPU reads data from a selected I/O port of the 8255A.

$\overline{WR}$(Write):

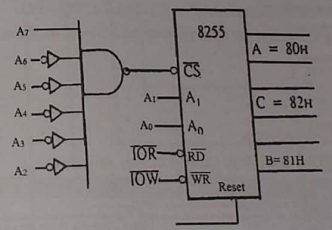
This control signal enables the Write Operation. When the signal goes low, the MPU writes into a selected I/O port or the control register.

RESET(Reset):
This is an active high signal; it clears the control register and sets all the ports in the input mode.

$\overline{CS}$, A_0 and A_1:
These are the device select signals. $\overline{CS}$ is connected to decoded address and A_0 and A_1 are generally connected to MPU address lines, A_0 and A_1, respectively.

The $\overline{CS}$ signal is the master Chip Select, and A_0 and A_1 specify one of the I/O ports or the control register as given below:

$\overline{CS}$	A_1	A_0	Selected
0	0	0	Port A
0	0	1	Port B
0	1	0	Port C
0	1	1	Control Register
1	X	X	8255A is not selected



Data Bus
Read & w r

$\overline{CS}$		Hex Address	Port
A_7 A_6 A_5 A_4 A_3 A_2 1 0 0 0 0 0	A_1 A_0 0 0 0 1 1 0 1 1	= 80H = 81H = 82H = 83H	A B C Control Register

As an example, the port addresses in above fig. are determined by the CS, A_0 and A_1 lines. The CS line goes low when $A_7 = 1$ and A_0 through A_2 are at logic 0. When these signals are combined with A_0 and A_1, the port addresses range from 80H to 83H.

Group A Control:

- This control block controls Port A and Port C_{UPPER} i.e. PC_7- PC_4
- It accepts control signals from the control word and forwards them to respective ports.

Group B Control:

- This control block controls Port B and Port C_{LOWER} i.e. PC_3- PC_0
- It accepts control signals from the control word and forwards them to the respective ports.

Port A, Port B and Port C:

- *These are 8-bit bidirectional ports.*
- *They can be programmed to work in various modes as follows:*

Port	Mode 0	Mode 1	Mode 2
Port A	Yes	Yes	Yes
Port B	Yes	Yes	No *(Mode 0 or Mode 1)*
Port C	Yes	No *(Handshake signals)*	No *(Handshake signals)*

CONTROL WORD:

- The content of the control register is called control word.
- It specifies an I/O functions for each port.

- The control register can be accessed to write a control word when $\overline{A_0}$ and A_1 are at logic 1. This register is not accessible for a Read Operation.

To communicate with peripherals through the 8255A, three steps are necessary:

1. Determine the addresses of Ports A, B and C and of the control register according the Chip Select Logic and address lines A_0 and A_1.
2. Write a control word in the control register.
3. Write I/O instructions to communicate with peripherals through ports A, B and C.

CONTROL WORD FOR I/O MODE:

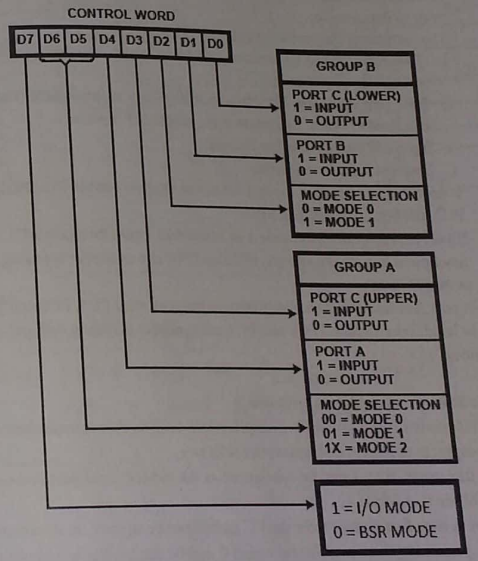


Fig. 8255A Control Word Format for I/O Mode

The various I/O modes of 8255A are as follows:

1. **MODE 0: Simple Input or Output**
   - In this mode, ports A and B are used as two simple 8-bit I/O ports and port C as two 4 bit ports.
   - Each port (or half port, in case of C) can be programmed to function as simply as input port or an output port.
   - The input/output features in Mode 0 are as follows:
     1. Outputs are latched.
     2. Inputs are not latched.
     3. Ports do not have handshake or interrupt capability.
   - A common example is a keyboard used for data entry.

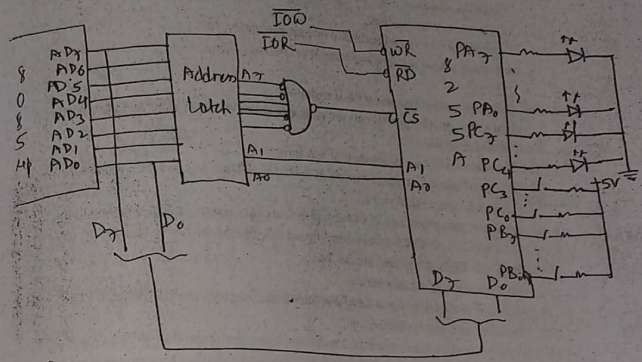
2. **MODE 1: Input or Output with Handshake**
   - In mode 1, handshake signals are exchanged between the MPU and peripherals prior to data transfer.
   - The features of this mode include the following:
     1. Two ports (A and B) function as 8-bit I/O ports. They can be configured either as input or output ports.
     2. Each port uses three lines from port C as handshake signals. The remaining two lines of port C can be used for simple I/O functions.
     3. Input and Output data are latched.
     4. Interrupt logic is supported.
   - If port B is initialized in mode 1 for either input or output, Pins PC0, PC1 and PC2 function as handshake lines.
   - If port A is initialized in mode 1 as handshake input, then pins PC3, PC4 and PC5 function as handshake signals. (PC6 and PC7 are available for using as input lines or output lines)
   - If port A is initialized as handshake output port, then PC3, PC6 and PC7 function as handshake signals. (PC4 and PC5 are available for using as input or output lines)

3. ***Mode 2: Bidirectional data transfer***
   - *This mode is used primarily in applications such as data transfer between two computers or floppy disk controller interface.*
   - *In this mode, port A can be configured as the bidirectional port and port B either in Mode 0 or Mode 1.*
   - *Port A uses five signals from port C as handshake signals for data transfer.*
   - *The remaining three signals from port C can be used either as simple I/O or as handshake for port B.*
   - *If port B is in mode 0, then PC0, PC1 and PC2 used for I/O. If port B is in mode 1, then PC0, PC1 and PC2 used as handshake lines.*

8255 Programs

① Write a program for 8255A PPI to read the DIP switches and display the reading from Port B at Port A and from port C_L at port C_U using the given circuit

(C_{Lower}) PC_3-PC_0 (C_{Upper}) PC_7-PC_4



Solution:

Basic addressing

Here, Port B and Port C_{LOWER} = input
Port A and Port C_{UPPER} = output

Control Word:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	
1	0	0	0	0	0	1	1	= 83H
I/O mode	Port A in mode 0		Port A o/p	Port C_U = o/p		Port B in mode 0	Port B I/P	Port C_L input

1. **MODE 0: Simple Input or Output**
   - In this mode, ports A and B are used as two simple 8-bit I/O ports and port C as two 4 bit ports.
   - Each port (or half port, in case of C) can be programmed to function as simply as input port or an output port.
   - The input/output features in Mode 0 are as follows:
     1. Outputs are latched.
     2. Inputs are not latched.
     3. Ports do not have handshake or interrupt capability.
   - A common example is a keyboard used for data entry.

2. **MODE 1: Input or Output with Handshake**
   - In mode 1, handshake signals are exchanged between the MPU and peripherals prior to data transfer.
   - The features of this mode include the following:
     1. Two ports (A and B) function as 8-bit I/O ports. They can be configured either as input or output ports.
     2. Each port uses three lines from port C as handshake signals. The remaining two lines of port C can be used for simple I/O functions.
     3. Input and Output data are latched.
     4. Interrupt logic is supported.
   - If port B is initialized in mode 1 for either input or output, Pins PC0, PC1 and PC2 function as handshake lines.
   - If port A is initialized in mode 1 as handshake input, then pins PC3, PC4 and PC5 function as handshake signals. (PC6 and PC7 are available for using as input lines or output lines)
   - If port A is initialized as handshake output port, then PC3, PC6 and PC7 function as handshake signals. (PC4 and PC5 are available for using as input or output lines)

3. **Mode 2: Bidirectional data transfer**
   - This mode is used primarily in applications such as data transfer between two computers or floppy disk controller interface.
   - In this mode, port A can be configured as the bidirectional port and port B either in Mode 0 or Mode 1.
   - Port A uses five signals from port C as handshake signals for data transfer.
   - The remaining three signals from port C can be used either as simple I/O or as handshake for port B.
   - If port B is in mode 0, then PC0, PC1 and PC2 used for I/O. If port B is in mode-1, then PC0, PC1 and PC2 used as handshake lines.

Address Calculation:

Port A: 0011 0000 = 30H

Port B: 0011 0001 = 31H

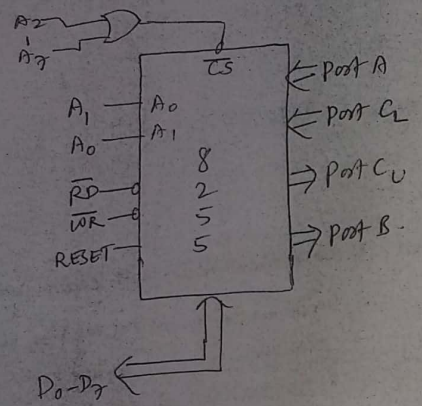
Port C: 0011 0010 = 32H

Control register: 0011 0011 = 33H.

Program:

```
MVI A, 83H   ; Load Accumulator with control word
OUT 33H      ; Write word in control register to initialize
               the ports.
IN 31H       ; Read switches at Port B.
OUT 30H      ; Display the reading at Port A.
IN 32H       ; Read switches at Port C
ANI 0FH      ; Mask the upper four bits of Port C.
RLC          ; Rotate and place data in upper half
RLC            of the Accumulator.
RLC
RLC
OUT 32H      ; Display data at Port C UPPER
HLT
```

Q.② Write a program to transfer the input data from Port A to Port B and Port C_{LOWER} to Port C_{UPPER}



Solution:

Control word:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	
1	0	0	1	0	0	0	1	= 91H.

Address Calculation:

			(A_1 A_0)	
Port A:	0000	00	01	= 00H
Port B:	0000	00	10	= 02H
Port C:	0000	00	01	= 01H
Control register:	0000	00	11	= 03H.

Program:

```
MVI A, 91H
OUT 03H
IN 00H
OUT 02H
IN 01H
ANI 0FH
RLC
RLC
RLC
RLC
OUT 01H
HLT
```

BSR (Bit Set/Reset Mode):

- Any bit of Port C can be Set or Reset using this mode.
- BSR mode doesn't alter any previously transmitted control word with bit $D_7 = 1$. Thus the input and output operations of port A and B are not affected by a BSR control word.
- In the BSR mode, individual bits of port C can be used for applications such as an on/off switch.
- This is very useful as it provides 8 individually controllable lines which can be used while interfacing with devices like an A to D Converter or a 7-segment display, etc.

BSR CONTROL WORD:

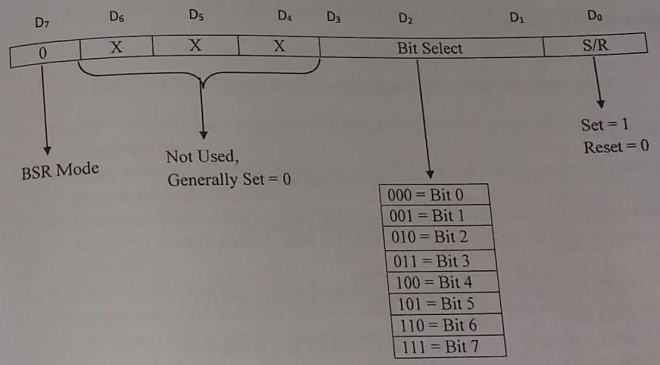


Fig. 8255A Control Word Format in the BSR Mode

Example 1:
Write BSR Control Word sub-routines to set bits PC_7 and PC_3 and reset them after 10ms.
(Assume control register address = 83H and delay subroutine is available)

Solution:
Here,

BSR CONTROL WORDS:

		D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	
To set Bit PC_7	=	0	0	0	0	1	1	1	1	= 0FH
To reset Bit PC_7	=	0	0	0	0	1	1	1	0	= 0EH
To set bit PC_3	=	0	0	0	0	0	1	1	1	= 07H
To reset bit PC_3	=	0	0	0	0	0	1	1	0	= 06H

Control Register Address = 83H

SUBROUTINE:

```
BSR:    MVI   A, 0FH     ; Load byte in accumulator to set PC7
        OUT 83H          ; Set PC7 = 1
        MVI A, 07H       ; Load byte in accumulator to set PC3
        OUT 83H          ; Set PC3 = 1
        CALL DELAY       ; this is a 10ms delay
        MVI A, 06H       ; Load accumulator with the byte to reset PC3
        OUT 83H          ; Reset PC7
        MVI A, 0EH       ; Load accumulator with the byte to reset PC7
        OUT 83H          ; Reset PC7
        HLT
```

From the analysis of above routine, following points can be noted:

1. To set/reset bits in port C, a control word is written in control register and not in port C.
2. The BSR Control Word affects only one bit in port C.
3. The BSR Control Word doesn't affect the I/O mode.

SERIAL I/O (8250/8251)

Specially, there are two basic approaches for serial data communication. They are:
- software- controlled serial I/O, and
- Hardware-controlled serial I/O.

The 8085 microprocessor has two pins specially designed for software-controlled serial I/O. One is called SOD (Serial Output Data) and the other is called SID (Serial Input Data). Data Transfer is controlled through two instructions: SIM and RIM. Instructions RIM and SIM are used for two different processes: interrupt and serial I/O.

But for the hardware-controlled serial I/O, different integrated circuits are used. The two basic chips used for serial communication are as follows:

1. 8250 UART (Universal Asynchronous Receiver Transmitter),
2. 8251 USART (Universal Synchronous Asynchronous Receiver Transmitter)

8250 UART (Universal Asynchronous Receiver Transmitter)

- 8250 UART is used as an interface by 8086 microprocessor, to perform serial communication.
- While transmitting, it converts data from parallel to serial. While receiving, it converts the data from serial to parallel.
- It supports signals for RS-232 which is a popular serial communication bus standard.
- Moreover, it was commonly used in PCs and related equipments such as printers or modems.
- The 8250 included an on-chip programmable bit rate generator, allowing use for both common and special-purpose bit rates which could be accurately derived from an arbitrary crystal oscillator reference frequency.

INTERFACE WITH 8086:

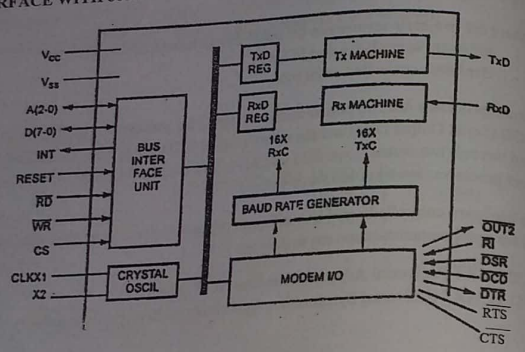


Fig. Interfacing 8086 with 8250 UART for serial communication

- In this block diagram, the internal bus of 8250 UART transfers data between the transmit/receive registers and the bus interface unit.
- Generally, 8250 UART can be used for serial communication using Modem, so the pins required for serial communication through Modem are also available in 8250 UART.
- For asynchronous serial data transfer, 8250 UART has Baud Rate Generator that selects appropriate Baud Rate of data transmission.

The 8251A Programmable Communication Interface

- The 8251A is a programmable chip designed for synchronous and asynchronous serial data communication, packaged in a 28-pin DIP.
- The 8251A is an enhanced version of its predecessor, the 8251, and is compatible with the 8251.
- It is compatible with a wide range of microprocessors such as 8048, 8080, 8085, 8086 and 8088.
- The 8251A is used as a peripheral device and is programmed by the CPU to operate using virtually any serial data transmission technique presently in use (including IBM 'bi-sync').
- The USART accepts data characters from the CPU in parallel format and then converts them into a continuous serial data stream for transmission.
- Simultaneously, it can receive serial data streams and converts them into parallel data characters for the CPU.

Block Diagram of 8251A USART:

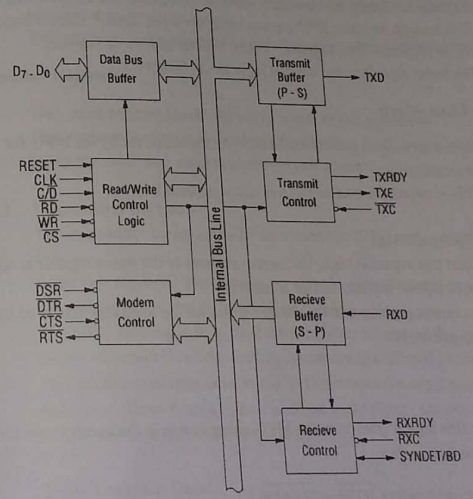


Fig. Functional Block Diagram of 8251A USART

- Fig. above shows the functional block diagram of 8251A UART. It includes five sections which are as follows:
  1. Read/Write Control Logic and Registers,
  2. Transmitter Section,
  3. Receiver Section,
  4. Data Bus Buffer Section, and
  5. Modem Control Section.

- Although, 8251A can operate in both synchronous and asynchronous mode, for the sake of clarity, here we are focused with only asynchronous serial data communication.
- The asynchronous mode is often used for data communication between the MPU and serial peripherals such as terminals and floppy disks.
- The different sections included in the functional block diagram of 8255A are described below:

1. Read/Write Control Logic and Registers:

- This control logic interfaces the chip with the MPU, determines the functions of the chip according to the control word in its register, and monitors the data flow.
- This section includes R/W control logic, six input signals, control logic, and three buffer registers: data register, control register and status register.
- The input signals to the control logic are as follows:

$\overline{CS}$ – Chip Select:

- When this signal goes low, the 8251A is selected for by the MPU for communication.
- This is usually connected to a decoded address bus.

C/$\overline{D}$ – Control/Data:

- When this signal is high, the control register or the status register is addressed; when it is low, the data buffer is addressed.
-The control register and the status register are selected differentiated by $\overline{WR}$ and $\overline{RD}$ respectively.

$\overline{WR}$ – Write:

-When this signal goes low, the MPU either writes in the control register or sends output to the data buffer.

- This is connected to $\overline{IOW}$ or $\overline{MEMW}$.

$\overline{RD}$ – Read:

- When this signal goes low, the MPU either reads a status from the status register or accepts (inputs) data from the data buffer.

- This is connected to either $\overline{IOR}$ or $\overline{MEMR}$.

RESET – Reset:

- A high on this input resets the 8251A and forces it into the idle mode.

CLK – Clock:

- This is the clock input, usually connected to the system clock.

- This clock does not control either the transmission or the reception rate. The clock is necessary for communication with the microprocessor.

Control Register:

- This 16-bit register for a control word consists of two independent bytes; the first byte is called the **mode instruction** (word) and the second byte is called the **command instruction** (word).
- This register can be accessed as an output port when C/$\overline{D}$ pin is high.

Status Register:

- This input register checks the ready status of the peripheral.
- This register is addressed as an input port when the C/$\overline{D}$ pin is high
- It has the same port address as the control register.

2. Transmitter Section:

- The transmitter accepts parallel data from the MPU and converts them into serial data.
- It has two registers: a buffer register to hold eight bits and an output register to convert eight bits into a stream of serial bits.
- The MPU writes a byte in the buffer register; whenever the output register is empty, the contents of the buffer register are transferred to the output register.
- This section transmits data on the TxD pin with the appropriate framing bits (Start and Stop). Three output signals and one input signal are associated with the transmitter section, which are as follows:

TxD- Transmit Data:

-Serial bits are transmitted on this line.

TxC- Transmitter Clock:
- This input signal controls the rate at which bits are transmitted by the USART.
- The clock frequency can be 1, 16, or 64 times the baud.

TcRDY – Transmitter Ready:
- When this output signal is high, it indicates that the buffer register is empty and the USART is ready to accept the byte.
- This signal is reset when a data byte is loaded into the buffer.

TxE – Transmitter Empty:
- When this output signal is high, it indicates that the output register is empty.
- This signal is reset when a byte is transferred from the buffer to the output registers.

Summary of Control Signals for the 8251A:

$\overline{CS}$	C/$\overline{D}$	$\overline{RD}$	$\overline{WR}$	Function
0	1	1	0	MPU writes instructions in the control register
0	1	0	1	MPU reads status from the status register
0	0	1	0	MPU outputs data to the data buffer
0	0	0	1	MPU accepts data from the data buffer
1	X	X	X	USART is not selected.

3. Receiver Section:
- The receiver accepts serial data on the RxD line from a peripheral and converts them in to parallel data.
- This section has two registers: the receiver input register and the buffer register.
- When the RxD line goes low, the control logic assumes it is a Start bit, waits for half a bit time, and samples the line again.
- If the line is still low, the input register accepts the following bits, forms a character, and loads it into the buffer register.
- Subsequently, the parallel byte is transferred to the MPU when requested.

- In the asynchronous mode, two input signals and one output signal are necessary, as described below:

$\overline{RxD}$- Receive Data:
- Bits are received serially on this line and converted into a parallel byte in the receiver input register.

RxC – Receiver Clock:
- This is a clock signal that controls the rate at which bits are received by the USART.
- In the asynchronous mode, the clock can be set to 1, 16, or 64 times the baud.

RxRDY – Receiver Ready:
-This output signal goes high when the USART has a character in the buffer register and is ready to transfer it to the MPU.

INITIALIZING THE 8251A:

- To implement the serial communication, the MPU must inform the 8251A of all details, such as mode, baud, Stop bits, parity, etc.
- So, prior to data transfer, a set of control words must be loaded into the 16-bit control register of the 8251A.
- In addition, the MPU must check the readiness of a peripheral by reading the status register.
- The control words are divided into two formats:
  1. Mode words, and
  2. Command words.

1. Mode Word:

- Mode word is used for setting the function of the 8251.
- Mode instruction will be in "wait for write" at either internal reset or external reset. That is, the writing of a control word after resetting will be recognized as a "mode instruction."
- Items set by mode instruction are as follows:

  • Synchronous/asynchronous mode

  • Stop bit length (asynchronous mode)

  • Character length

- Parity bit
- Baud rate factor (asynchronous mode)
- Internal/external synchronization (synchronous mode)
- Number of synchronous characters (Synchronous mode)

- The mode word format for 8251A chip is as shown below:

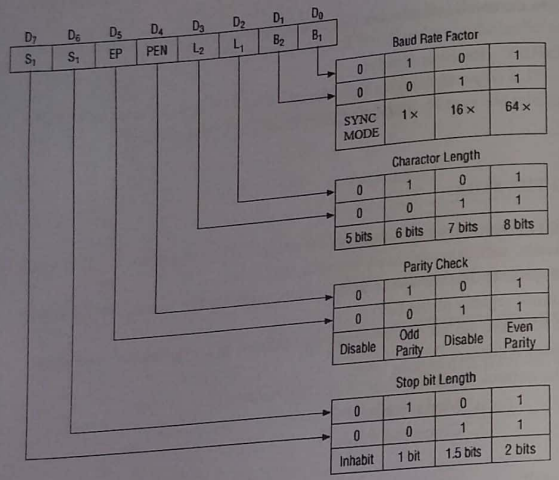
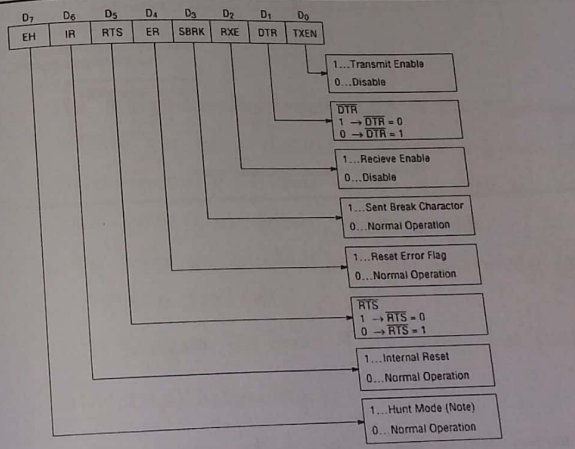


Fig. 2 Bit Configuration of Mode Instruction (Asynchronous)

2. Command Word:

- Command is used for setting the operation of the 8251.
- It is possible to write a command whenever necessary after writing a mode instruction and sync characters.
- Items to be set by command are as follows:

  - Transmit Enable/Disable
  - Receive Enable/Disable
  - DTR, RTS Output of data.
  - Resetting of error flag.
  - Sending to break characters
  - Internal resetting
  - Hunt mode (synchronous mode)



Note: Seach mode for synchronous charactors in synchronous mode

Fig. 4 Bit Configuration of Command

STATUS WORD

- It is possible to see the internal status of the 8251 by reading a status word.
- The bit configuration of status word is shown in Fig. 5.

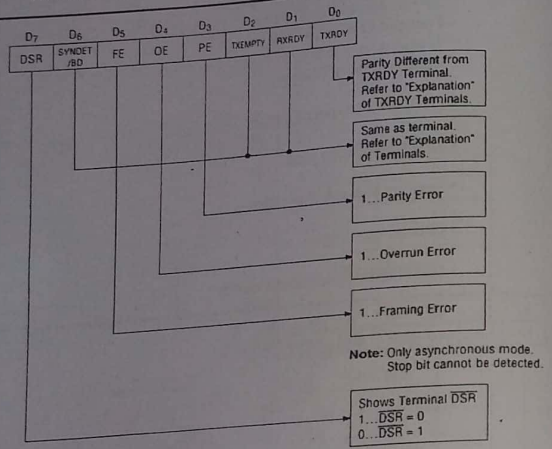


Fig. 5 Bit Configuration of Status Word

Direct memory Access (DMA):

→ Direct memory Access (DMA) is an I/O technique commonly used for high-speed data transfer, for example, data transfer between system memory and a floppy disk.

→ In status check I/O and interrupt I/O, data transfer is relatively slow because each instruction needs to be fetched and executed.

→ In DMA, the MPU releases the control of the buses to a device called a DMA Controller.

→ The controller manages data transfer between memory and a peripheral under its control, thus bypassing the MPU.

→ DMA Transfer uses two signals - HOLD and HLDA in 8085 microprocessor.

HOLD:

- This is an active high input signal to 8085 from another master requesting the use of address and data buses.
- After receiving the Hold request, the MPU relinquishes the buses in the following machine cycle
- All buses are tri-stated and Hold Acknowledge (HLDA) signal is sent out.
- MPU regains the control of the buses after HOLD goes low.

HLDA (Hold Acknowledge):

- This is an active high output signal indicating that MPU is relinquishing control of the buses.

→ A DMA Controller uses these signals as if it were a peripheral requesting the MPU for the control of the buses.

Concept of DMA:

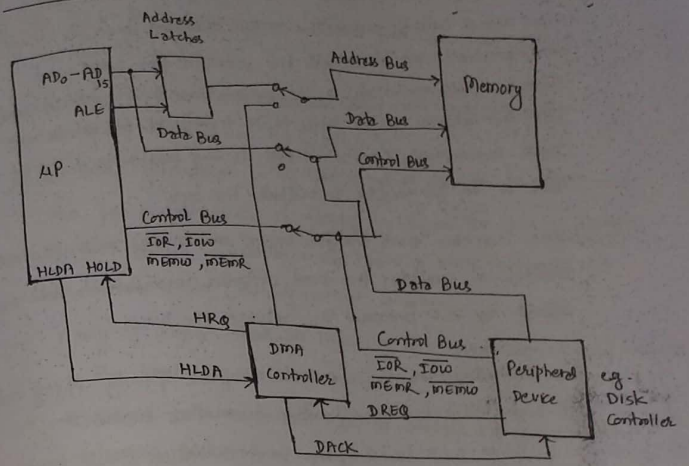


Fig. Block Diagram showing DMA Transfer

The sequence of DMA transfer as shown in above block diagram can be explained below:-

(I) originally, microprocessor is connected to the memory as shown in fig. above with switches closed for address, data and control buses. When peripheral wants to transfer data using DMA Transfer, it sends DMA request, DREQ, signal to the DMA Controller.

(II) If the input (channel) of the DMA controller is unmasked, the DMA Controller will send a hold-request, HRQ signal to the microprocessor's HOLD input.

(III) The microprocessor (μP) finishes the current machine cycle and floats its buses, sending out a hold-acknowledge signal, HLDA, to the DMA Controller.

(iv) When DMA controller receives HLDA signal, it will send out a control signal which throws the 3 bus switches down to their DMA position. This disconnects the processor from buses and connects DMA controller to the buses. Nao, DMA controller sends out the address of the byte to be transferred and sends out DMA-Acknowledge (DACK) signal to the peripheral device to tell it to get ready to output the byte.

(v) ~~When~~ Then the DMA transfer begins and finally when the data transfer is complete, the DMA controller unasserts its hold-request signal to the processor and releases the buses.

→ The DMA controller is a processor capable of only copying data at high speed from one location to another location.

→ To perform DMA transfer, the DMA controller should have,
- a data bus,
- an address bus,
- Read/write control signals, and
- control signals to disable its role as a peripheral and to enable its role as a processor.

This process is called switching from slave mode to master mode.

→ For the illustration of DMA transfer, we study a programmable DMA Controller, Intel 8237.

The 8237 DMA Controller:

→ The 8237 is a programmable DMA Controller packaged in a 40-pin IC package.

→ It has 4 independent channels with each channels ~~capt~~ capable of transferring 64K bytes.

→ It must interfaces with two types of devices: the MPU and the peripheral such as floppy disks, and be able to operate in two modes: master mode (during DMA transfer) and in slave mode (during normal condition).

→ many of signals that are input in the I/O mode become outputs in the processor mode.

→ ~~The~~ 8237 DMA controller can be studied categorizing the following sections!

DMA CHANNELS AND INTERFACING:

→ The 8237 has 4 independent channels, CH_0 to CH_3.

→ Internally, two 16-bit registers are associated with each channel:-

① Count Register
- It is used to load a count of the number of bytes to be copied.

⑪ Memory Address Register:
- It is used to load the starting address of the byte to be copied.

→ The address of these registers are determined by four address lines: A_3 to A_0 and the Chip select ($\overline{CS}$) signal.

→ Address 0000 on lines A_3-A_0 selects CH0 memory Address Register (MAR) and address 0001 selects the next register, CH0 count.

→ Similarly, all the remaining registers are selected in sequential order.

→ The last eight registers are used to write commands or read status.

→ The addresses of the internal registers range from 00H to 0FH

A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	
0	0	0	0	0	0	0	0	= 00 CH0 MAR
↓				↓			↓	
Chip Select				1	1	1	1	= 0F mask Register

→ The different registers and pins of 8237 DMA controller are as shown in fig. 5.33.

DMA SIGNALS:

The different important signals required to understand DMA Transfer are as explained below:-

DREQ0 – DREQ3 – DMA Request:

- These are 4 independent, asynchronous signal inputs to the DMA channels from peripherals such as floppy disk and hard disks.
- To obtain DMA service, a request is generated by activating the DREQ line of the channel.

DACK0 – DACK3 – DMA Acknowledge:

→ These are output lines to inform the individual peripherals that a DMA is granted.

→ DREQ and DACK are equivalent to handshake signals in I/O devices.

AEN and ADSTB – Address Enable and Address Strobe:

→ These are active high output signals that are used to latch a high-order address byte to generate a 16-bit address.

$\overline{MEMW}$ and $\overline{MEMR}$ – Memory Write & Memory Read:

→ These are output signals used during DMA cycle to write and read from memory.

A_3-A_0 and A_7-A_4 – Address:

→ A_3-A_0 are bidirectional address lines. They are used as inputs to access control registers.

→ During DMA cycle, these lines are used as output lines to generate a low-order address that is combined with the remaining address lines A_7-A_0.

HRQ and HLDA – Hold Request and Hold Acknowledge:

→ HRQ is an output signal used to request the MPU control of the system bus.

→ After receiving the HRQ, the MPU completes the bus cycle in process and issues the HLDA signal.

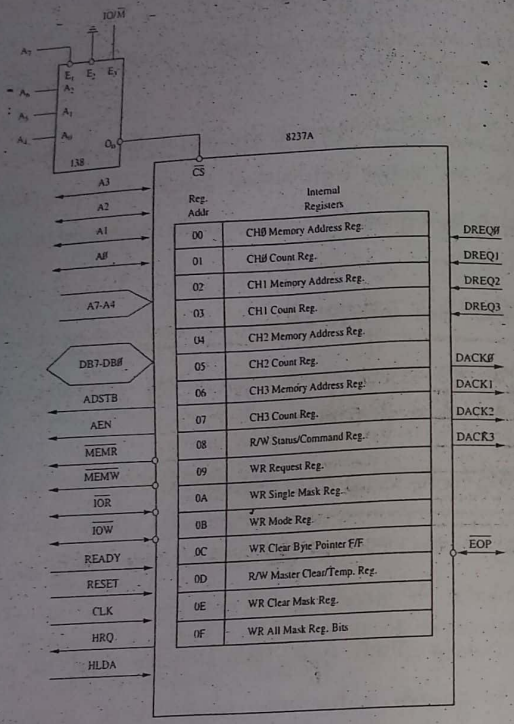


FIGURE 15.33
8237A—DMA Controller with Internal Registers

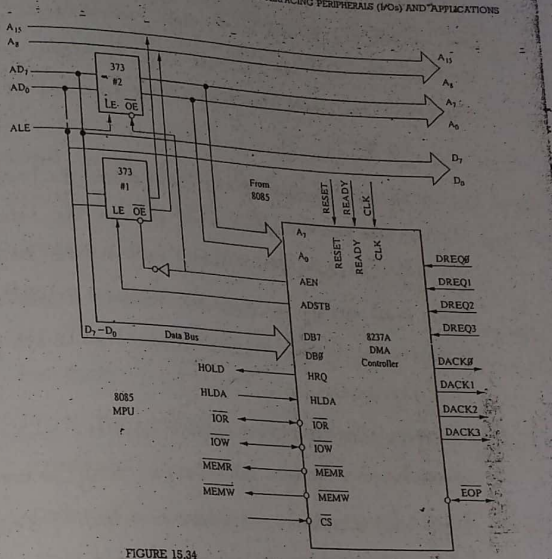


FIGURE 15.34
Interfacing 8237A—DMA Controller with the 8085

These steps are illustrated in the following example.

Example 15.8 Write initialization instructions for the DMA controller in Figure 15.33 to meet the following specifications. Use the same port (register) addresses (00 to 0FH) as in Figure 15.33.

1. Disable the DMA controller and begin writing initialization instructions.
2. Initialize Channel #3 (CH3) to transfer 1K of bytes from the system memory to the floppy disk assigned to CH3.

SYSTEM INTERFACE:

→ DMA Transfer includes eight data lines, four control signals ($\overline{IOR}$, $\overline{IOW}$, $\overline{MEMR}$ and $\overline{MEMW}$) and eight address lines (A_7-A_0). However, it needs 16 address lines to access 64K bytes. So, eight additional lines must be generated as shown in fig 15.34.

→ When a transfer begins, the DMA places the low-order byte on the address bus and 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 the first byte, the latch is updated when the lower byte generates a carry (or borrow).

→ Fig. 15.34 shows two latches: one latch (373 #1) is to latch a high-order address from the data bus by using AEN and ADSTB signals, and the second latch (373 #2) to demultiplex the 8085 bus and generate the low-order address bus using ALE from 8085.

→ The AEN signal is connected to the $\overline{OE}$ signal of the second latch to disable the low-order address bus from 8085 when the first latch is enabled to latch the high-order byte of the address.

PROGRAMMING 8237:

→ To implement the DMA Transfer, the 8237 should be initialized by writing into various control registers.

→ To initialize 8237, the following steps are necessary:-

(i) Write a control word in mode register that selects the channel and specifies the type of transfer (Read, write or verify) and DMA mode (block, single-byte, etc)

(ii) Write a control word in command register that specifies parameters such as priority among 4 channels, DREQ and DACK active levels and timing and enables the 8237.

(iii) Write the starting address of data block to be transferred in channel MAR.

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

DMA EXECUTION:

→ There are 2 modes of DMA controller: slave mode and the master mode.

Slave mode:

→ In slave mode, DMA controller is treated as a peripheral, using the following steps:

(i) MPU selects the DMA controller through chip: Select

(ii) The MPU writes the control words in control registers and command/status registers using control signals $\overline{IOW}$ and $\overline{IOR}$.

Master mode:

After the initialization, the 8237 in master mode keeps checking for a DMA request, and the steps in data transfer can be listed as follows:-

(I) When the peripheral is ready, it sends high signal to DRQ.

(II) When DRQ has been received and channel enabled, the control logic sets HRQ (Hold Request) high. HRQ is connected to the HOLD signal of 8085.

(III) In next cycle, the MPU relinquishes the bus and sends HLDA (Hold Acknowledge) signal to the 8237.

(IV) After receiving the HLDA signal, the DMA asserts AEN (Address Enable) signal high. The high AEN signal disable 373 Latch #2, thus disconnecting the demultiplexed bus A_7-A_0 of the MPU and enables 373 Latch #1 through an inverter.

Next, the DMA asserts ADSTB (Address Strobe) high, that is connected to Latch Enable (LE) of 373 Latch #1 and places the contents of the data bus, which is a high order byte of the starting address, on A_{15}-A_8. At the same time, DMA also outputs the low order addr A_7-A_0 on the low-order address bus.

(V) When the entire A_{15}-A_0 is available on the address bus DMA sends DACK to the peripheral.

(VI) DMA controller continues data transfer by asserting necessary control signals ($\overline{IOR}$, $\overline{IOW}$, $\overline{MEMR}$ & $\overline{MEMW}$) until DACK=1

(VII) At the end of data transfer, DMA asserts $\overline{EOP}$ (End of Process) signal low to inform peripheral that data transfer is complete.