UNIT-III
PART -A
INTERRUPTS AND PROGRAMMABLE INTERRUPT CONTROLLERS

Contents at a glance:

✓ 8086 Interrupts and Interrupt Responses
✓ Introduction to DOS and BIOS interrupts
✓ 8259A Priority Interrupt Controller

8086 Interrupts and Interrupt Responses:

➢ Sometimes it is necessary to have the computer automatically execute one of a collection of special routines whenever certain conditions exist within a program or in the microcomputer system.
➢ For example, it is necessary that microcomputer system should give response to devices such as keyboard, sensor and other components when they request for service.
➢ The most common methods for servicing such device are
  • Polling Method
  • Interrupt method

Polling Method:

➢ In this method the processor must test each device in sequence and in effect ‘ask’ each one if it needs communication with the processor.
➢ It easy to see that a large portion of the main program is looping through this continuous polling cycle.
➢ Such a method would have a serious and decrementing effect on system throughput.

Interrupt Method:

➢ Another method would be the one that allows the microprocessor to execute its main program and only stop to service peripheral devices when it is told to do so by the device itself.
➢ In this method the processor would provide an external asynchronous input that would inform the processor that it should complete whatever instruction that currently being executed and fetch a new routine that will service the requesting device.
➢ Once this servicing is completed, the processor would resume exactly where it left off.
➢ This method is called Interrupt Method.
➢ This method increases the system throughput
➢ When a microprocessor is interrupted, it stops executing its current program and calls special routine which services the interrupt.
➢ The event that causes the interruption is call interrupt and the special routine executed to service the interrupt is called interrupt service routine (ISR) / Procedures.
➢ The interrupt can came from any of the three sources
1. By external signal
2. By a special instruction in the program
3. By the occurrence of some condition

- An interrupt caused by an external signal is referred as **Hardware interrupt**.
- A condition interrupts or interrupts caused by special instructions are called **software interrupts**.
- Other type of interrupts are
  1. Vectored and Non vectored interrupts
     - The interrupts which are having fixed address location for ISR are called vectored interrupts.
     - The interrupts which are not having fixed address location for ISR are called non vectored interrupts.
  2. Maskable and Non maskable interrupts
     - The interrupts which can be ignored are called maskable interrupts.
     - The interrupts which cannot be ignored are called non maskable interrupts.

An 8086 interrupt can come from any one the three sources:

1. **External signal**: An 8086 can get interrupt from an external signal applied to the nonmaskable interrut (NMI) input pin, or the interrupt (INTR) input pin.
2. **Special instruction**: An execution of the Interrupt instruction (INT). This is referred as software interrupt.
3. **Condition produced by Instruction**: An 8086 is interrupted by some condition produced in the program by the execution of an instruction.
4. For example divide by zero: program execution will automatically be interrupted if you attempt to divided an operand by zero.

- At the end of each instruction cycle, 8086 checks to see if any interrupts have been requested.
- If an interrupt has been requested, the 8086 responds to interrupt by stepping through the following series of major steps:
  1. It decrements the stack pointer by 2 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 trap flag (TF) in the flag register.
  4. It decrements the stack pointer by 2 and pushes the current code segment register contents on the stack.
  5. It decrements the stack pointer again by 2 and pushes the current instruction pointer contents on the stack.
  6. It does an indirect far jump to start of the procedure by loading the CS and IP values for the start of the interrupt service routine.
An IRET instruction at the end of the interrupt service routine returns execution to main program.

The 8086 gets the new values of CS and IP register from four memory addresses.

When it responds to an interrupt, the 8086 goes to memory locations to get the CS an IP values to start of the interrupt service routine.

In an 8086 system the first 1 Kb of memory from 00000H to 003FFH is reserved for storing the starting addresses of interrupt service routines.

This block of memory is often called the INTERRUPT VECTOR TABLE or the INTERRUPT POINTER TABLE.

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

**8086 interrupt vector table**

<table>
<thead>
<tr>
<th>ADDRESS</th>
<th>TYPE 255 POINTER</th>
<th>TYPE 33 POINTER</th>
<th>TYPE 32 POINTER</th>
<th>TYPE 31 POINTER</th>
</tr>
</thead>
<tbody>
<tr>
<td>3FFH</td>
<td>(AVAILABLE)</td>
<td>(AVAILABLE)</td>
<td>(AVAILABLE)</td>
<td>(RESERVED)</td>
</tr>
<tr>
<td>3FCH</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AVAILABLE POINTERS (224)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>084H</td>
<td>TYPE 5 POINTER</td>
<td>TYPE 4 POINTER</td>
<td>TYPE 3 POINTER</td>
<td>TYPE 2 POINTER</td>
</tr>
<tr>
<td>080H</td>
<td>(RESERVED)</td>
<td>OVERFLOW</td>
<td>1-BYTE INT INSTRUCTION</td>
<td>NON-MASKABLE</td>
</tr>
<tr>
<td>07FH</td>
<td></td>
<td></td>
<td></td>
<td>SINGLE-STEP</td>
</tr>
<tr>
<td>RESERVED POINTERS (27)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>014H</td>
<td>TYPE 0 POINTER</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>010H</td>
<td>(RESERVED)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DEDICATED POINTERS (5)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>004H</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000H</td>
<td></td>
<td></td>
<td></td>
<td>CS BASE ADDRESS</td>
</tr>
</tbody>
</table>

IP OFFSET

16 BITS
Each interrupt type is given a number between 0 to 255 and the address of each interrupt is found by multiplying the type by 4 e.g. for type 11, interrupt address is $11 \times 4 = 44_{10} = 0002CH$.

Only the first five types have explicit definitions.

The next 27 interrupt types, from 5 to 31, are reserved by Intel for use in future microprocessors.

The upper 224 interrupt types, from 32 to 255, are available for user for hardware or software interrupts.

When 8086 responds to an interrupt, it automatically goes to the specified location in the interrupt vector table to get the starting address of the interrupt service routine.

**8086 Interrupts Types:**

1. **Divide by Zero Interrupt (Type 0):**

   When the quotient from either a DIV or IDIV instruction is too large to fit in the result register; 8086 will automatically execute type 0 interrupt.

2. **Single Step Interrupt (Type 1):**

   The type 1 interrupt is single step trap. In the single step mode, the system will execute one instruction and wait for further direction from the user. The user can examine the contents of registers and memory locations and if they are correct, the user can tell the system to execute the next instruction. This feature is useful for debugging assembly language programs.

3. **Non Maskable Interrupt (Type 2):**

   As the name suggests, this interrupt cannot be disabled by any software instruction. This interrupt is activated by low to high transition on 8086 NMI input pin. In response, 8086 will do a type 2 interrupt.

4. **Break Point Interrupt (Type 3):**

   The type 3 interrupt is used to implement breakpoint function in the system. The type 3 interrupt is produced by execution of the INT 3 instruction. Breakpoint function is often used as debugging aids in cases where single stepping provides more details than wanted.

5. **Overflow Interrupt (Type 4):**

   The type 4 interrupt is used to check overflow condition after any signed arithmetic operation in the system. The 8086 overflow flag, OF, will be represented in the destination register or memory location.

**Software Interrupts (Type 0-255):**

- The 8086 INT instruction can be used to cause 8086 to do one of the 256 possible interrupt types.

- The interrupt type is specified by the number as a part of the instruction.

- You can use an INT 2 instruction to send execution to NMI interrupt service routine.

- This allows you to test the NMI routine without needing to apply an external signal to the NMI input of the 8086.

**Maskable Interrupt (INTR):**

- The 8086 INTR input can be used to interrupt a program execution.
- This interrupt is implemented by using two pins INTR and INTA

**Introduction to DOS and BIOS Interrupts:**

- In IBM PC, part of the operating system is located in the permanent memory (ROM) and part is loaded during power up.
- The part located in ROM is referred to as ROM-BIOS (Basic Input/Output System).
- The other part which is loaded in RAM during power-up from hard disk is known as DOS (Disk Operating System).
- BIOS is located in a 8 Kbyte ROM at the top of memory, the address range being from FE000H to FFFFFH.
- The programs with ROM-BIOS provide the most direct, lowest level interaction with the various devices in the system.
- The ROM-BIOS contains routines for
  1. Power-on self test
  2. System configuration analysis
  3. Time of the day
  4. Print screen
  5. Boot strap loader
  6. I/O support program for
     a. Asynchronous communication
     b. Keyboard
     c. Printer
     d. Display
- Most of these programs are accessible to ALP through software interrupt instruction (INT).
- The design goal for the ROM-BIOS programs is to provide a device-independent interface to the various physical devices in the system.
- Using ROM-BIOS one can output characters to various physical devices like the printer or display, one can read character from keyboard.
- But still few things are not possible with ROM-BIOS
  1. It is not possible to provide ability to load and execute programs directly.
  2. It is not possible to store data on the diskette organized as logical files.
  3. ROM-BIOS have no command-interpreter to allow copying files, print files, deleting files.
- It is the DOS that provides these services.
- When we turn ON our computer, we expect to see message or a prompt.
- We expect to be able to look at the diskette directory to see what data files or programs the diskette contains.
- We expect to run a program by typing name.
- We want to copy programs from one diskette to another, print programs and delete programs.
- All these services are provided by group of programs called DOS.
- The services provided by DOS can be grouped into following categories.
  1. Character Device I/O
  2. File Management
  3. Memory Management
  4. Directory Management
  5. Executive Functions
  6. Command Interpreter
  7. Utility Programs

<table>
<thead>
<tr>
<th>S.No</th>
<th>DOS</th>
<th>BIOS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>DOS is loaded from the bootable diskette.</td>
<td>BIOS is located in an 8 Kbyte ROM.</td>
</tr>
<tr>
<td></td>
<td>DOS program offer different degree of flexibility, portability and hardware independence.</td>
<td>The programs within the ROM-BIOS provide the most direct, lowest level interaction with the various devices in the system. Using these programs require hardware knowledge.</td>
</tr>
<tr>
<td>2.</td>
<td>DOS has ability to load and execute programs directly.</td>
<td>ROM-BIOS does not have ability to load and execute programs directly.</td>
</tr>
<tr>
<td>3.</td>
<td>DOS can store data on the diskette organized as a logical files.</td>
<td>ROM-BIOS cannot store data on the diskette organized as a logical files.</td>
</tr>
<tr>
<td>4.</td>
<td>DOS has command interpreter to allow us to copy files, print files and delete files.</td>
<td>ROM-BIOS have no command interpreter to allow us to copy files, print files and delete files.</td>
</tr>
</tbody>
</table>

- The MS-DOS API is an API which originated with 86-DOS and is used in MS-DOS/PC DOS and other DOS-compatible operating systems. Most calls to the DOS API are invoked using software interrupt 21H (INT 21H).
- By calling INT 21h with a sub function number in the AH processor register and other parameters in other registers, one invokes various DOS services.
- DOS services include keyboard input, video output, disk file access, executing programs, memory allocation, and various other things.
<table>
<thead>
<tr>
<th>AH</th>
<th>Description</th>
<th>AH</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>Read character from STDIN</td>
<td>02</td>
<td>Write character to STDOUT</td>
</tr>
<tr>
<td>05</td>
<td>Write character to printer</td>
<td>06</td>
<td>Console Input/ Output</td>
</tr>
<tr>
<td>07</td>
<td>Direct char read (STDIN), no echo</td>
<td>08</td>
<td>Char read from STDIN, no echo</td>
</tr>
<tr>
<td>09</td>
<td>Write string to STDOUT</td>
<td>0A</td>
<td>Buffered input</td>
</tr>
<tr>
<td>0B</td>
<td>Get STDIN status</td>
<td>0C</td>
<td>Flush buffer for STDIN</td>
</tr>
<tr>
<td>0D</td>
<td>Disk reset</td>
<td>0E</td>
<td>Select default drive</td>
</tr>
<tr>
<td>19</td>
<td>Get current default drive</td>
<td>25</td>
<td>Set interrupt vector</td>
</tr>
<tr>
<td>2A</td>
<td>Get system date</td>
<td>2B</td>
<td>Set system date</td>
</tr>
<tr>
<td>2C</td>
<td>Get system time</td>
<td>2D</td>
<td>Set system time</td>
</tr>
<tr>
<td>2E</td>
<td>Set verify flag</td>
<td>30</td>
<td>Get DOS version</td>
</tr>
<tr>
<td>35</td>
<td>Get Interrupt vector</td>
<td>3B</td>
<td>Set working directory</td>
</tr>
<tr>
<td>36</td>
<td>Get free disk space</td>
<td>3D</td>
<td>Open file</td>
</tr>
<tr>
<td>3A</td>
<td>Remove subdirectory</td>
<td>3C</td>
<td>Create file</td>
</tr>
<tr>
<td>3E</td>
<td>Close file</td>
<td>3F</td>
<td>Read file</td>
</tr>
<tr>
<td>40</td>
<td>Write file</td>
<td>41</td>
<td>Delete file</td>
</tr>
<tr>
<td>42</td>
<td>Seek file</td>
<td>43</td>
<td>Get/Set file attributes</td>
</tr>
<tr>
<td>47</td>
<td>Get current directory</td>
<td>4C</td>
<td>Exit program</td>
</tr>
<tr>
<td>4D</td>
<td>Get return code</td>
<td>54</td>
<td>Get verify flag</td>
</tr>
<tr>
<td>56</td>
<td>Rename file</td>
<td>57</td>
<td>Get/Set file date</td>
</tr>
</tbody>
</table>
8259A Priority Interrupt Controller:

- Interrupts can be used for a variety of applications.
- Each of these interrupt applications require a separate interrupt input.
- If we are working with an 8086, we get only two interrupt inputs INTR and NMI.
- For applications where we have multiple interrupt sources, we use external device called a priority interrupt controller (PIC)

Features of 8259:

1. It can manage eight priority interrupts. This is equivalent to provide eight interrupt pins on the processor in place of INTR pin.
2. It is possible to locate vector table for these additional interrupts anywhere in the memory map.
3. By cascading 8259s it is possible to get 64 priority interrupts.
4. Interrupt mask register makes it possible to mask individual interrupt request.
5. With the help of 8259A user can get information of pending interrupts, in-service interrupts and masked interrupts.
6. The 8259A is designed to minimize the software and real time overhead in handling multilevel priority interrupts.

8259A includes eight blocks:

1. Data Bus Buffer
2. Read/Write Logic
3. Control Logic
4. Three Registers (IRR, ISR and IMR)
5. Priority Resolver
6. Cascade Buffer
1. **Data Bus Buffer**: The data bus allows the 8086 to send control words to 8259A and read a status word from the 8259A. The 8-bit data bus also allows the 8259A to send interrupt types to 8086.

2. **Read/Write Logic**: The RD and WR inputs control the data flow on the data bus when the device is selected by asserting its chip select input low.

3. **Control Logic**: This block has an input and output line. If 8259A is properly enabled the interrupt request will cause 8259A to assert its INT output pin high. If this pin is connected to the INTR pin of 8086 and if 8086 interrupt flag is set, then this high signal cause the 8086 respond INTR.

4. **Interrupt Request Register (IRR)**: The IRR is used to store all the interrupt levels which are requesting service. The eight interrupt inputs set corresponding bits of Interrupt Request Register.

5. **Interrupt Service Register (ISR)**: The ISR stores all the levels that are currently being serviced.

6. **Interrupt Mask Register (IMR)**: The IMR stores the masking bits of the interrupt lines to be masked. This register can be programmed by an OCW. An interrupt which is masked by software will not be recognized and serviced even if it set corresponding bits in the IRR.

7. **Priority Resolver**: The priority resolver determines the priorities of the bits set in the IRR. The bit corresponding to the highest priority interrupt input is set in the ISR during the INTA input.

8. **Cascade Buffer Comparator**: This section generates control signals necessary for cascade operations. It also generates Buffer-Enable signals. The 8259A can be cascaded with other 8259s in order to expand the interrupt handling capacity to 64 levels. In such case, the former is called a **master**, and the latter are called **slaves**. The 8259 can set up as a master or a slave by the \( SP/EN \)
CS*: This is an active-low chip select signal for enabling RD* and WR* operations of 8259A. INTA` function is independent of CS*.

WR*: This pin is an active-low write enable input to 8259A. This enables it to accept command words from CPU.

RD*: This is an active-low read enable input to 8259A. A low on this line enables 8259A to release status onto the data bus of CPU. D7-D0 These pins form a bidirectional data bus that carries 8-bit data either to control word or from status word registers. This also carries interrupt vector information.

CAS0-CAS2: Cascade Lines A single 8259A provides eight vectored interrupts. If more interrupts are required, the 8259A is used in cascade mode. In cascade mode, a master 8259A along with eight slaves 8259A can provide up to 64 vectored interrupt lines. These three lines act as select lines for addressing the slaves 8259A.

SP'/EN': This pin is a dual purpose pin. When the chip is used in buffered mode, it can be used as buffer enable to control buffer transreceivers. If this is not used in buffered mode then the pin is used as input to designate whether the chip is used as a master (SP = 1) or a slave (EN = 0).

INT: This pin goes high whenever a valid interrupt request is asserted. This is used to interrupt the CPU and is connected to the interrupt input of CPU.

IRO-IR7 (Interrupt requests): These pins act as inputs to accept interrupt requests to the CPU. In edge triggered mode, an interrupt service is requested by raising an IR pin from a low to a high state and holding it high until it is acknowledged, and just by latching it to high level, if used in level triggered mode.

INTA' (Interrupt acknowledge): This pin is an input used to strobe-in 8259A interrupt vector data on to the data bus. In conjuction with CS, WR, and RD pins, this selects the different operations like, writing command words, reading status word, etc.

Interrupt Sequence in an 8086 System

The interrupt sequence in an 8086-8259A system is described as follows:

1. One or more of the interrupt request lines (IRO-IR7) are raised high, setting the corresponding IRR bits.

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

3. The CPU acknowledges the INT and responds with INTA pulse.

4. Upon receiving an INTA from the CPU, the highest priority ISR bit is set and the corresponding IRR bit is reset. The 8259A does not drive data bus during this cycle.

5. A selection of priority modes is available to the programmer so that the manner in which requests are processed by the 8259A can be configured to match his system requirements. The priority modes can be changed dynamically at any time during the main program.

6. The 8086 will initiate a second INTA pulse. During this pulse 8259A releases interrupt type onto the DATA bus where it is read by CPU.

7. This completes the interrupt cycle. In the AEOI mode the ISR bit is reset at the end of the second INTA pulse. Otherwise the ISR bit remains set until an appropriate EOI command is issued at the end of interrupt subroutine.

Programming the 8259A:
The 8259A requires two types of command words.

1. Initialization Command Words (ICWs)

2. Operational Command Words (OCWs)

The 8259A can be initialized with four ICWs; the first two are compulsory and the other two are optional based on the modes being used.

After initialization, the 8259A can be set up to operate in various modes by using three different OCWs

- The initialization sequence of 8259A is described in from of a flow chart in Fig.

**Initialization Command Word 1 (ICW1):**

---

**Fig. 6.14 Initialization Sequence of 8259A**

---

**Initialization Command Word 1 (ICW1):**

- 0 = ICW4 Needed
- 1 = No ICW4 Needed

**Call Address Interval**

- 1-Interval of 4 bytes
- 0-Interval of 8 bytes

**ADH = 1 for 8086 based system**

---

**Fig. 6.15 Initialization Command Word 1 (ICW1)**

---
- A write command issued to 8259A with $A_0=0$ and $D_4=1$ is interpreted as ICW1, which starts the initialization sequence.

It specifies

1. Single or multiple 8259As in the system.
2. 4 or 8 bit interval between the interrupt vector locations.
3. The address bit $A_7-A_5$ to CALL instruction.
4. Edge triggered or level triggered interrupts.
5. ICW4 is needed or not.

Initialization Command Word 2 (ICW2):

- A write command following ICW1, with $A_0=1$ is interpreted as ICW2. This is used to load high order byte of the interrupt vector address of all interrupts.

Initialization Command Word 3 (ICW3):

- ICW3 is required only if there is more than one 8259A in the system and if they are cascaded. An ICW3 operation loads a slave register in the 8259.
- For master each bit in ICW3 is used whether it has a slave 8259 attached to it on its corresponding IR input.
- For slave, bits $D_0-D_2$ of ICW3 are used to assign a slave identification code to the 8259A.
Initialization Command Word 4 (ICW4):

- It is loaded only if the D0 bit of ICW1 is set.

It specifies

1. Whether to use special fully nested mode or non special fully nested mode.
2. Whether to use buffered mode or non buffered mode.
3. Whether to use automatic EOI or normal EOI.
4. CPU used 8086/8088

- After initialization the 8259A is ready to process the interrupt request. However during operation it might be necessary to change the mode of processing the interrupts.
- Operation Command Words (OCWs) are used for this purpose

Operation Command Word 1 (OCW1):

- A write command to the 8259 with A0=1 (after ICW2) is interpreted as OCW1. OCW1 is used for enabling or disabling the recognition of specific interrupt requests by programming the IMR.

Operation Command Word 2 (OCW2):

- A write command to the 8259 with A0=1 and D4D3=00 is interpreted as OCW2.
Operation Command Word 3 (OCW3):

- OCW3 is used to read the status of registers, and to set or reset the Special Mask and Polled modes

Priority Modes and Other Features:

The various modes of operation of the 8259 are:

a) Fully Nested Mode
b) Special Fully Nested Mode
c) Rotating Priority Mode
d) Special Masked Mode
e) Polled Mode
Interfacing and programming 8259:

Example: Show 8259A interfacing connections with 8086 at the address 074x. Write an ALP to initialize the 8259A in single level triggered mode, with call address interval of 4, Non buffered, no special fully nested mode. Then set the 8259A to operate with IR6 masked, IR4 as bottom priority level, with special EOI mode. Set special mask mode of 8259A. Read IRR and ISR into registers BH and BL respectively. IR0 of 8259A will have type 80H.

Solution: Let the starting address is 0000:0200H. The interconnections of 8259A with 8086 are as shown in Fig 1.10. The 8259 is interfaced with lower byte of the 8086 data bus, hence A0 line of the microprocessor system is abandoned and A1 of the microprocessor system is connected with A0 of the 8259A. Before going for an ALP, all the initialisation command words (ICWS) and Operation command word (OCWS) must be decided.

ICW1 decides single level triggered, address interval of 4 as given below.

<table>
<thead>
<tr>
<th>A0</th>
<th>D7</th>
<th>D6</th>
<th>D5</th>
<th>D4</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

= 1FH

- D0: ICW4 Needed
- D1: Single 8259A
- D2: Call Address Interval 4
- D3: Level Triggered
- D4: Always set to 1
- D5 D6 D7: Don’t care for 8086 system
- A0: Always set to 0

ICW2 Vector address = 0000:0010 for IR3

<table>
<thead>
<tr>
<th>T7</th>
<th>T6</th>
<th>T5</th>
<th>T4</th>
<th>T3</th>
<th>A10</th>
<th>A9</th>
<th>A8</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

= 83H

There is no slave hence the ICW3 is as given below

<table>
<thead>
<tr>
<th>A0</th>
<th>D7</th>
<th>D6</th>
<th>D5</th>
<th>D4</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

ICW3 = 00H

Actually ICW3 is not at all needed, because in ICW1 the 8259A is set for single mode.

The ICW4 should be set as shown below:

<table>
<thead>
<tr>
<th>A0</th>
<th>D7</th>
<th>D6</th>
<th>D5</th>
<th>D4</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

ICW4 = 01H

- D0: For 8086 system
- D1: Normal EOI
- D2 D3: Non buffered mode
- D4: For special fully nested mode masking
OCW1 sets the mask of IR6 as shown below

<table>
<thead>
<tr>
<th>A₀</th>
<th>D₇</th>
<th>D₆</th>
<th>D₅</th>
<th>D₄</th>
<th>D₃</th>
<th>D₂</th>
<th>D₁</th>
<th>D₀</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

OCW₁=40H

IR6 is masked

OCW₂ sets the modes and rotating priority as shown below

<table>
<thead>
<tr>
<th>A₀</th>
<th>D₇</th>
<th>D₆</th>
<th>D₅</th>
<th>D₄</th>
<th>D₃</th>
<th>D₂</th>
<th>D₁</th>
<th>D₀</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

OCW₂=E4H

D₀ D₂ Bottom priority Level set at IR4
D₅ D₇ Specific EOI Command with rotating priority

OCW3 sets the special mask mode and reads ISR and IRR using the following control words.

For reading IRR

<table>
<thead>
<tr>
<th>A₀</th>
<th>D₇</th>
<th>D₆</th>
<th>D₅</th>
<th>D₄</th>
<th>D₃</th>
<th>D₂</th>
<th>D₁</th>
<th>D₀</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

OCW₃=6AH

D₀ D₁ Read IRR
D₂ No Poll command
D₅ D₇ Special mask mode

For reading ISR

<table>
<thead>
<tr>
<th>A₀</th>
<th>D₇</th>
<th>D₆</th>
<th>D₅</th>
<th>D₄</th>
<th>D₃</th>
<th>D₂</th>
<th>D₁</th>
<th>D₀</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

OCW₃=6BH

D₀ D₁ Read ISR
D₂ No Poll command
D₅ D₇ Special mask mode
Assembly language program:

CODE SEGMENT
ASSUME CS: CODE
START:  MOV AL, 1FH
         MOV DX, 0740H
         OUT DX, AL ; WRITE ICW1
         MOV DX, 0742H
         MOV AL, 83H
         OUT DX, AL ; WRITE ICW2
         MOV AL, 01H
         OUT DX, AL ; WRITE ICW4
         MOV AL, 040H
         OUT DX, AL ; WRITE OCW1
         MOV AL, 0E4H
         MOV DX, 0740H
         OUT DX, AL ; WRITE OCW2
         MOV AL, 6AH
         OUT DX, AL ; WRITE OCW3 FOR READING IRR
         IN AL, DX
         MOV BH, AL
         MOV AL, 6BH ; ; WRITE OCW3 FOR READING ISR
         OUT X, AL
         IN AL, DX
         MOV BL, AL
         MOV AH, 4CH
         INT 21H
CODE ENDS
END START