Lecture 23: 80x86 Interrupts and 8259

Seyed-Hosein Attarzadeh-Niaki

Based on the slides by Hongzi Zhu

Review

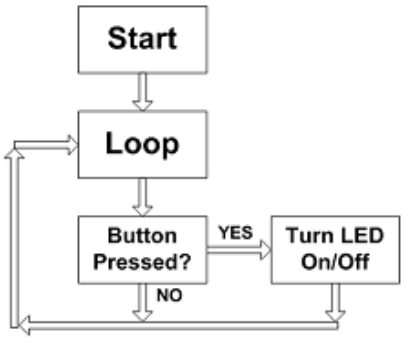
- The 8253 programmable interval timer chip
  - Structure
  - Interfacing
  - Modes of operation

Outline

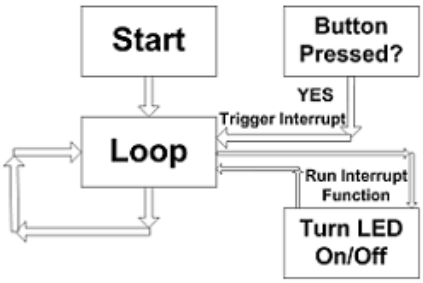
- 80x86 interrupts and the 8259 chip

Reacting to External Events

Polling



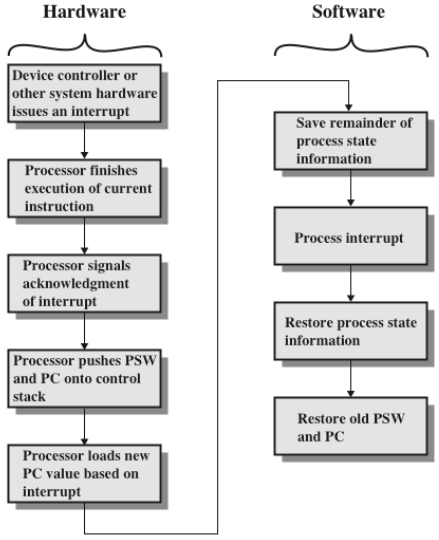
Interrupt



Programming with Interrupts

- An interrupt is the *occurrence of a condition* (an external *event*) which informs the CPU that a device needs its service
  - Causes a *temporary suspension* of a program
  - The event is serviced by another program (*Interrupt Service Routine (ISR)* or Interrupt Handler).
- Interrupt-Driven System
  - Doing more things simultaneously
  - Efficient interfacing to slow devices
  - Quick response to events
  - Suitable for real-time control applications (?)

Simple Interrupt Processing



Interrupts in 8086/8088

- 256 *interrupt TYPEs* in total
  - INT 00 ~ INT 0FFh
- TYPE * 4 = PA of interrupt vector
  - The first 1KB is used to store interrupt vectors, called ***Interrupt Vector Table*** (IVT)
- Interrupt vector points to the entrance address of the corresponding ***interrupt service routine*** (ISR)
- Intel reserves the first 32 interrupt vectors

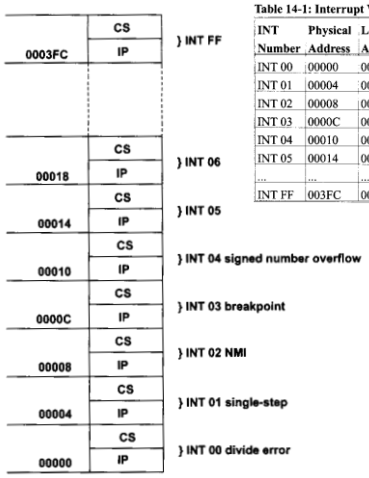
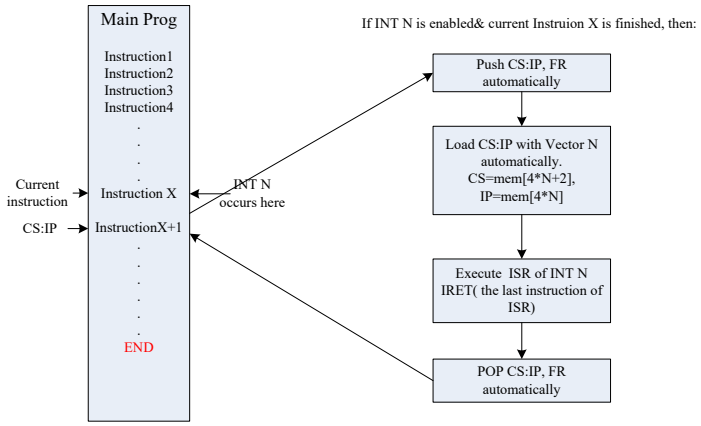


Table 14-1: Interrupt Vector

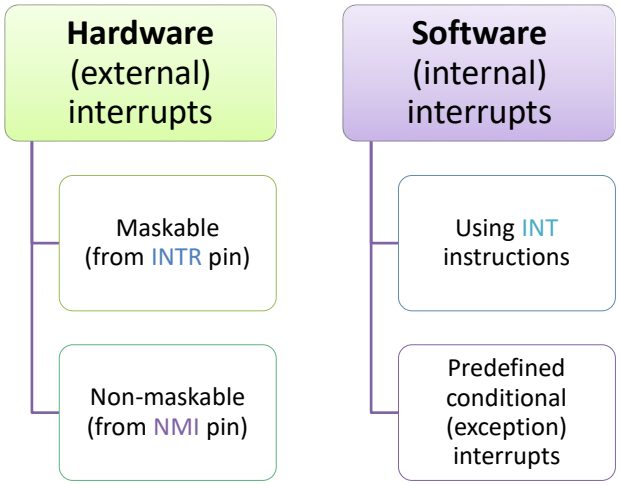
INT Number	Physical Address	Logical Address
INT 00	00000	0000:0000
INT 01	00004	0000:0004
INT 02	00008	0000:0008
INT 03	0000C	0000:000C
INT 04	00010	0000:0010
INT 05	00014	0000:0014
...	...	...
INT FF	003FC	0000:03FC

Main Program and ISR



An ISR is launched by an interrupt event (internal 'int xx', external NMI and INTR) . So, ISR is 'separated' from main.

Categories of Interrupts



Hardware Interrupts

Non-maskable interrupt

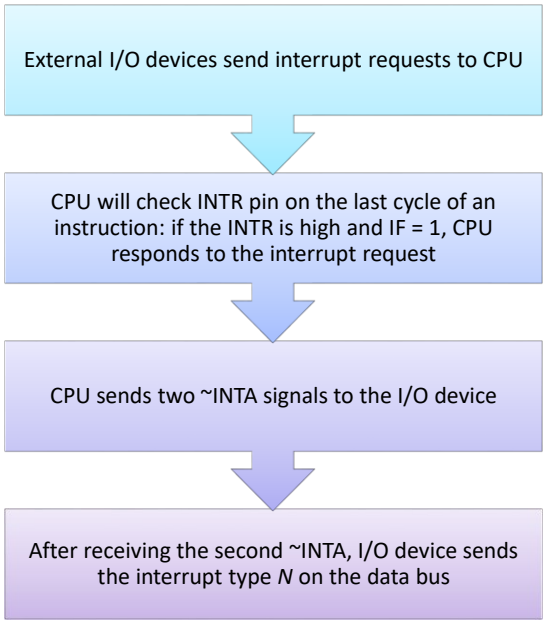
- Trigger: NMI pin, input-signal, rising edge and two-cycle high activate
- TYPE: INT 02
- Not affected by the IF
- Reasons:
  - RAM parity check error, interrupt request from co-CPU 8087, etc.

Maskable interrupt

- Trigger: INTR pin, input-signal, high active
- TYPE: No predefined type
- IF = 1, enable; IF = 0, disable
  - **STI** sets IF, **CLI** clears IF
- Reasons:
  - Interrupt requests of external I/O devices

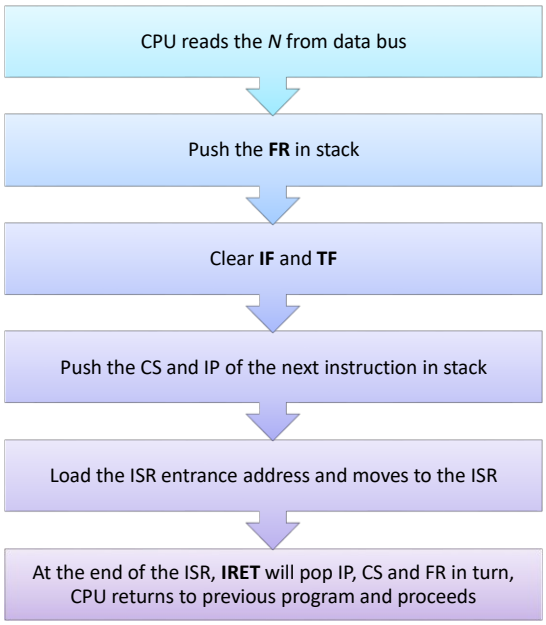
Procedure for Processing Maskable Interrupts

1- CPU responds to INTR interrupt requests



Procedure for Processing Maskable Interrupts

2- CPU executes the ISR of INT *N*



Software Interrupts

- INT xx instruction
  - An ISR is called upon instruction such as "INT xx"
    - E.g., int 21h ; DOS service
  - CPU always responds and executes the corresponding ISR
    - Not affected by the IF
  - You can "CALL" any ISR by using the INT instruction

Difference Between INT & CALL

1. CALL FAR can jump anywhere within 1MB vs. INT jumps to a fix location (finding the corresponding ISR)
2. CALL FAR is in the sequence of instructions vs. an external interrupt can come in at any time
3. CALL FAR cannot be masked (disabled) vs. an external interrupt can be masked
4. CALL FAR saves CS:IP of next instruction vs. INT saves FR + CS:IP of next instruction
5. last instruction: RETF vs. IRET. RETI also pops FR from the stack.

Software Interrupts

- Predefined conditional interrupts
  - "INT 00" (divide error)
    - Reason: dividing a number by zero, or quotient is too large
  - "INT 01" (single step)
    - If TF = 1, CPU will generate an INT 1 interrupt after executing each instruction for debugging
  - "INT 03" (breakpoint)
    - When CPU hits the breakpoint set in the program, CPU generates INT 3 interrupt for debugging
  - "INT 04" (signed number overflow)
    - **INTO** instruction
    - Check the OF after an arithmetic instruction

```
;How to clear TF?
    PUSHF
    POP AX
    AND AX, 0FEFFH
    PUSH AX
    POPF
```

```
MOV  AX,0009H
ADD AX,0080H
INTO
```


Procedure for Processing Non-Maskable & Software Interrupts

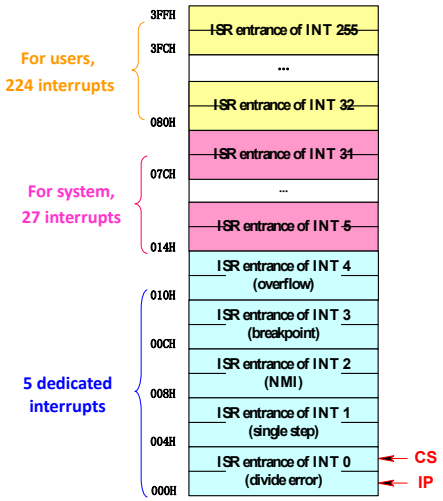
For NMI

- CPU checks NMI, generates INT 02 interrupt automatically regardless of **IF** and executes to the ISR

For software (internal) interrupts

- CPU generates INT *N* interrupt automatically and executes the corresponding ISR

Interrupt Vector Table of 8086/8088 in IBM PC



- 256 interrupts
  - 0 ~ 4 dedicated
  - 5 ~ 31reserved for system use
    - 08H ~ 0FH：8259A
    - 10H ~ 1FH：BIOS
  - 32 ~ 255 reserved for users
    - 20H ~ 3FH：DOS
    - 40H ~ FFH：open

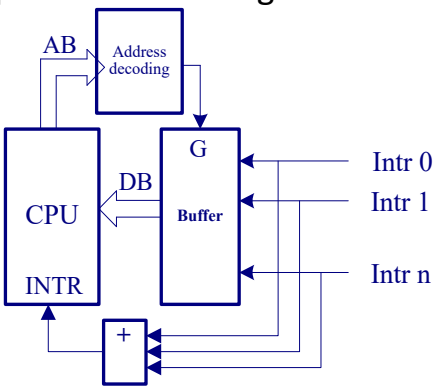
Example: INT 12H for checking the RAM size

Interrupt Priority

- INT instruction has higher priority than INTR and NMI
- NMI has higher priority than INTR
- For different external interrupt requests, different strategies can be used to determine their priorities.

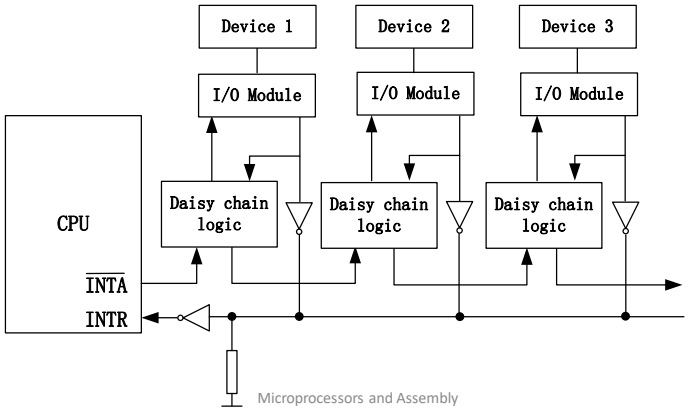
Priority of INTR Interrupts

- Software polling
  - The sequence of checking determines the priority



Priority of INTR Interrupts

- Hardware checking
  - The location in the daisy chain counts



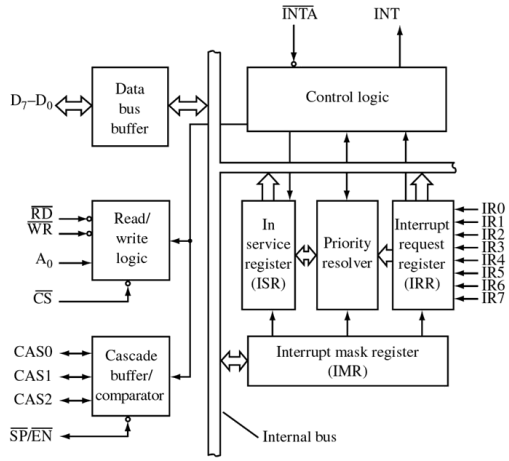
Priority of INTR Interrupts

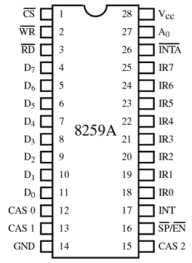
- Vectored interrupt controller
  - E.g., 8259

8259

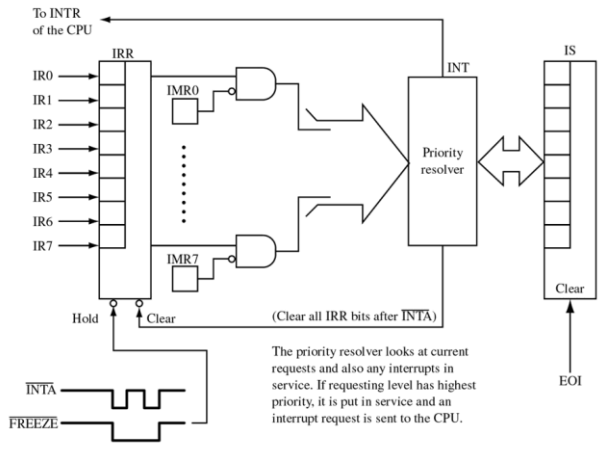
- 8259 is Programmable Interrupt Controller (PIC)
- A tool for managing the interrupt requests.
- A flexible peripheral controller chip:
  - PIC can deal with up to 64 interrupt inputs
  - interrupts can be masked
  - various priority schemes can also programmed.
- Originally (in PC XT) it is available as a separate IC
- Later the functionality of (two PICs) is integrated in the motherboards chipset.
- In some of the modern processors, the functionality of the PIC is built in.

8259A Programmable Interrupt Controller (PIC) Block Diagram and Pin Definitions



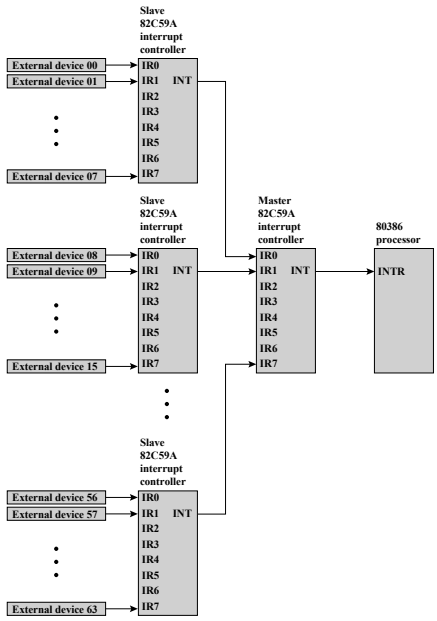


8259A Operation



All interrupt requests must pass through the PIC's interrupt request register (IRR) and interrupt mask register (IMR). If put in service, the appropriate bit of the in-service (IS) register is set.

Cascading 82C59A Interrupt Controllers



Programming 8259A

- 8259A is programmed by *initialization* and *operation* command words.
- Initialization command words (ICWs) are programmed **before** the 8259A is able to function in the system.
  - dictate the basic configuration of the 8259A
- Operation command words (OCWs) are programmed **during** the normal course of operation.
  - control the operation of the 8259A (masking&priority)

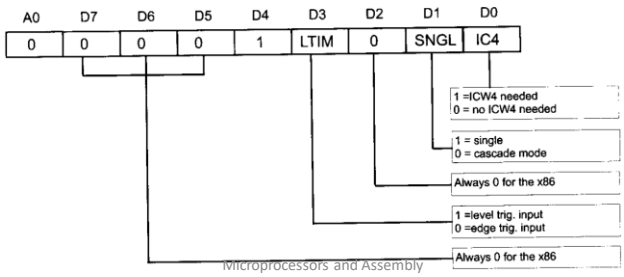
8259 Initialization Command Words

- The four initialization command words (ICWs) are selected when the A0 pin is logic 1
- If a single 8259A is used in a system, ICW1, ICW2, and ICW4 must be programmed when powered up
- If programmed in cascade mode by ICW1, then ICW3 must be programmed

$\overline{CS}$	A0	Initialization
0	0	ICW1
0	1	ICW2, ICW3, ICW4
1	x	8259 is not addressed

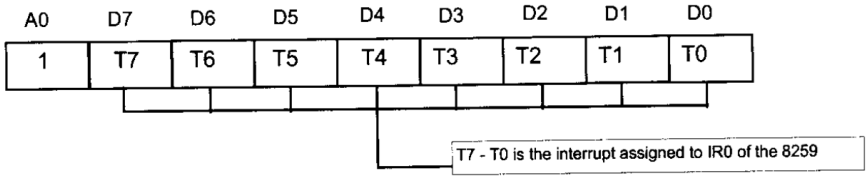
ICW Descriptions - ICW1

- **ICW_1** programs basic operation of the 8259A.
  - selects single or cascade operation by programming the SNGL bit
  - if cascade operation is selected, ICW_3 must also be programmed
  - the LTIM bit determines whether interrupt request inputs are positive edge-triggered or level-triggered



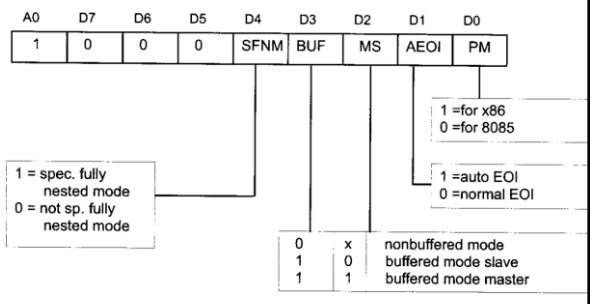
ICW Descriptions - ICW2

- **ICW_2** selects the vector number used with the interrupt request inputs.
  - if programming 8259A so it functions at vector locations 08H–0FH, place 08H into this command word
  - if programming for vectors 70H–77H, place 70H in this ICW



ICW Descriptions - ICW4

- **ICW_4** is programmed for use with the 8086 processors, the rightmost bit must be logic 1 to select operation with them.
- **SFNM**—Selects the special fully nested mode of operation if logic 1 is placed in this bit
- **BUF** and **M/S**—Buffered and master slave
- **AEOI**—Selects automatic or normal end of interrupt. The EOI commands of OCW2 are used only if AEOI mode is not selected by ICW_4
  - if AEOI is selected, the interrupt automatically resets the interrupt request bit and does not modify priority (no need for EOI instruction before IRET)
  - preferred mode of operation for the 8259A



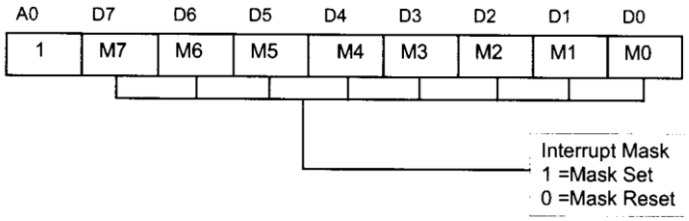
8259 Operation Command Words

- Used to direct 8259A operation once programmed with the ICW
- OCWs are selected when the A0 pin is at logic 0 level
- Except OCW1, which is selected when A0 is logic 1

$\overline{CS}$	A0	Operation Command Word
0	0	OCW2, OCW3
0	1	OCW1
1	x	8259 is not addressed

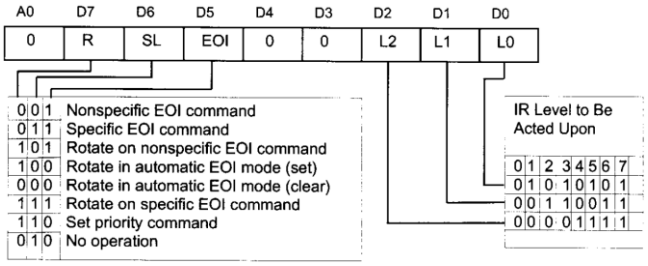
OCW Descriptions - OCW1

- **OCW_1** is used to set and read the interrupt mask register.
  - When a mask bit is set, it will turn off (mask) the corresponding interrupt input
  - The mask register is read when OCW_1 is read
  - Must be programmed after ICW upon initialization



OCW Descriptions - OCW2

- **OCW_2** is programmed to assign priorities only when the AEOI mode is not selected.
- Three modes
  - Fully nested: highest priority to R0 and lowest to R7
  - Automatic rotation: when an IR is served, takes the lowest priority and will not be served until others take a chance
  - Specific rotation: like automatic, but specifies the rotation scheme

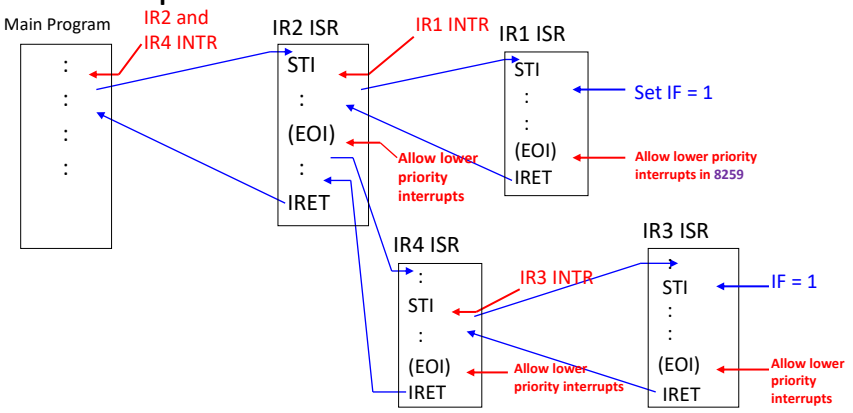


OCW Descriptions - OCW2

- **Nonspecific End-of-Interrupt**—Command sent by the interrupt service procedure to signal the end of the interrupt
  - 8259A determines which interrupt level was active and resets the correct interrupt status register bit
  - resetting the bit allows the interrupt to take action again or a lower priority interrupt to take effect
- **Specific End-of-Interrupt**—A command that allows a specific interrupt request to be reset
  - exact position determined with bits L_2–L_0 of OCW_2
- **Rotate-on-Nonspecific EOI**—functions exactly like the Nonspecific End-of-Interrupt command, except it rotates interrupt priorities after resetting the interrupt status register bit
  - the level reset by this command becomes the lowest priority interrupt
  - if IR_4 was just serviced by this command, it becomes the lowest priority interrupt input and IR_5 becomes the highest priority
- **Rotate-on-Automatic EOI**—A command that selects automatic EOI with rotating priority
  - must only be sent to the 8259A once if this mode is desired.
  - if this mode must be turned off, use the clear command
- **Rotate-on-Specific EOI**—Functions as the specific EOI, except that it selects rotating priority.
- **Set priority**—Allows the programmer to set the lowest priority interrupt input using the L_2–L_0 bits.

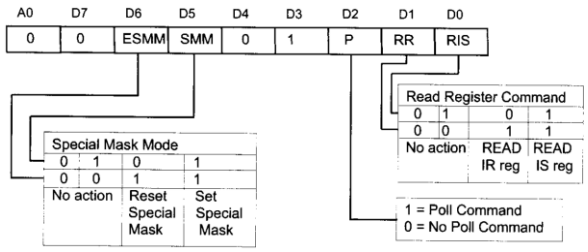
Interrupt Nesting

- Higher priority interrupts can interrupt lower interrupts



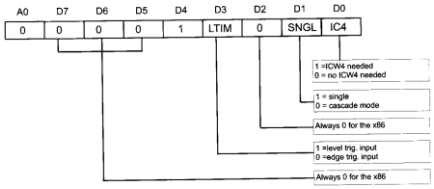
OCW Descriptions - OCW3

- **OCW_3** selects register to be read, operation of the special mask register & poll command.
  - if polling is selected, the P bit must be set and then output to the 8259A; the next read operation will read the poll word
  - the rightmost three bits of the word indicate the active interrupt request with the highest priority
  - the leftmost bit indicates if there is an interrupt and must be checked to determine whether the rightmost three bits contain valid information



Example I

Find the ICWs of the 8259 if it is used with an 8088/86 CPU, single, level triggering IRs, and IR0 is assigned "INT 50H". The 8259 is in slave buffered mode with normal EOI.

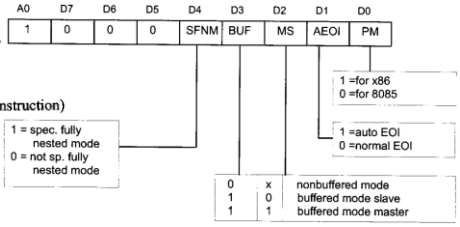


ICW1

D0 =1	ICW4 needed
D1 =1	single
D2 =0	this is always zero for 80x86 CPUs
D3 =1	level triggering
D4 =1	required by the ICW1 itself
D5 =D6 =D7 =0	this is always zero for 80x86 CPUs

ICW2 = 01010000 = 50H

No ICW3 is needed since it is single and not cascaded.



ICW4

D0 =1	8088/86
D1 =0	normal (we must issue EOI before IRET instruction)
D2 =0;D3=1	slave buffered mode
D4 =0	not nested
D5 =D6 =D7 =0	required by the ICW4

We get ICW4 =00001001 =09H.

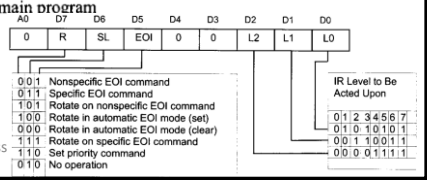
Example II

Show the program to initialize the 8259 using the port addresses in Example 1

```
MOV   AL,1BH      ;ICW1
OUT   26H,AL      ;TO PORT 26H
MOV   AL,50H      ;ICW2
OUT   27H,AL      ;TO PORT 27H
MOV   AL,09       ;ICW4
OUT   27H,AL      ;TO PORT 27H
```

Show the last 3 instructions of the interrupt service routine for IR1 of the 8259 in Example

```
INT_SERV    PROC  FAR       ;routine for IR1
            ...   ...
            MOV   AL,20H    ;the EOI byte for OCW2
            OUT   26H,AL    ;to port designated for OCW2
            IRET            ;return from ISR to the main program
```



Registering an ISR

```
INITIALIZE:
      XOR AX,AX
      MOV ES,AX
      CLI        ; Disable interrupts
      MOV WORD PTR ES:[136], OFFSET INT22   ;setups offset
                                            ;of handler 22h
      MOV WORD PTR ES:[138], CS             ; assuming
                                            ; current CS
      STI        ; Reenable interrupts
       ; End of setup
…

INT22  PROC FAR
       ; Here goes the body of your handler
       IRET
INT22  ENDP
```


Next Lecture

- Bus Interfaces
- Direct Memory Access