

# Interrupts

# Interrupts

- Some events occur outside the processor and the processor must be informed about them via a signal.
- Such a signal is called Interrupt.
- Most common example of interrupt signals is from input/output devices like keyboard, mouse, USB storage, speaker, mic, printer, scanners etc.
- Interrupts are **asynchronous** and **unpredictable**.

# Example - A Sample Process and Interrupt

- While CPU is executing a program, what if the user pushes a combination of keys
  - Alt+Tab
  - Ctrl+Alt+Del
  - Esc, Ctrl+C, Ctrl+V etc.
- What will be the response of processor on these different combinations?
- After the response finishes what will processor execute next?

# Example - A Sample Process and Interrupt

- While CPU is executing a program, what if the user pushes a combination of keys

- Alt+Tab
- Ctrl+Alt+Del
- Esc, Ctrl+C, Ctrl+V etc.



Interrupt

- What will be the response of processor on these different combinations?
- After the response finishes what will processor execute next?



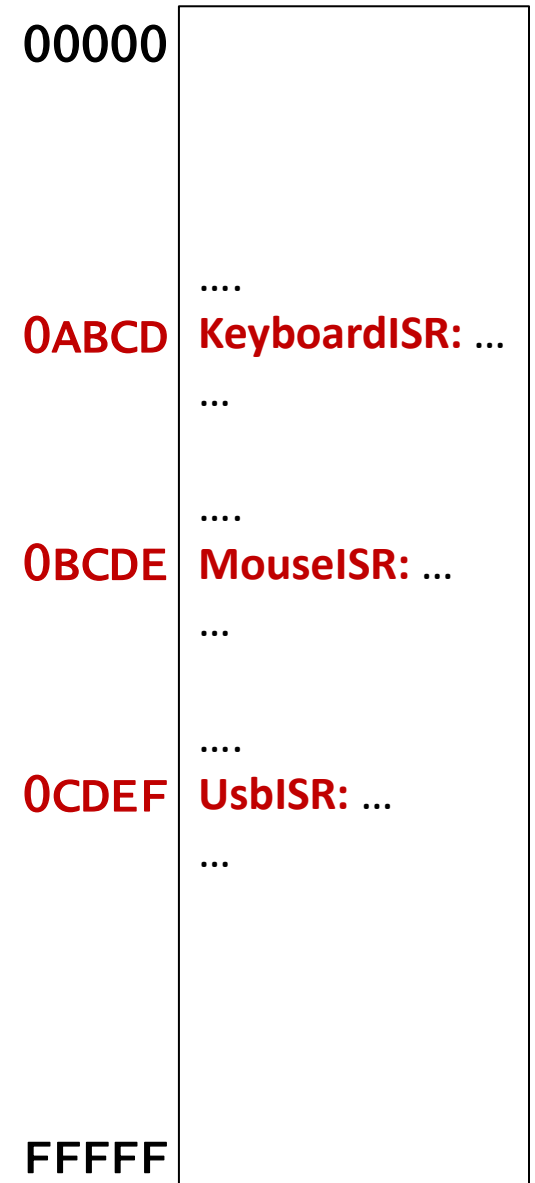
Interrupt  
Handler

# Interrupt Handler

- The routine (code) that executes in response to an interrupt signal is called an **interrupt handler** or an interrupt service routine, **ISR**
- When processor receives an interrupt signal, it **suspends** its current activities, saves its state, and executes the ISR to deal with that event.
- ISR is just a piece of code (list of instructions) residing somewhere in memory.
- Once ISR is completed, processor **resumes** its previously suspended activities.

# Interrupt Handler

- For example, KeyboardISR, handles all the keys and shows you different screens (or performs functionalities) for each key or key-combination pressed.



# How can CPU locate an ISR?

- To make identifying interrupts easier, each kind of interrupt is assigned a number
- Here is a list of interrupts in iAPX-88 architecture.
- There are 256 interrupts in total.

#	Description
00	Divide Error
01	Single Step Debugging
02	Non-maskable Interrupt
03	Debugging Breakpoints
04	INTO Detected Overflow
	...
08	System Timer
<b>09</b>	<b>Keyboard Data Ready</b>
...	...
10	BIOS Video Service
...	
16	BIOS Keyboard Service
...	...
21	DOS API calls
...	...
FF	...

# How can CPU locate an ISR?

- For each of the 255 interrupts, the addresses of the interrupt handlers (ISRs) are stored in a special area of memory, called Interrupt Vector Table (IVT).
- Each entry (row) in table is of 4 bytes
  - one row per interrupt
- First two byte in row are offset of ISR
- 3rd and 4th byte are segment address of ISR

int 0	offset	segment
int 1	offset	segment
int 2	offset	segment
int 3	offset	segment
.....		
int FF	offset	segment

Interrupt Vector Table  
(simplified tabular view)

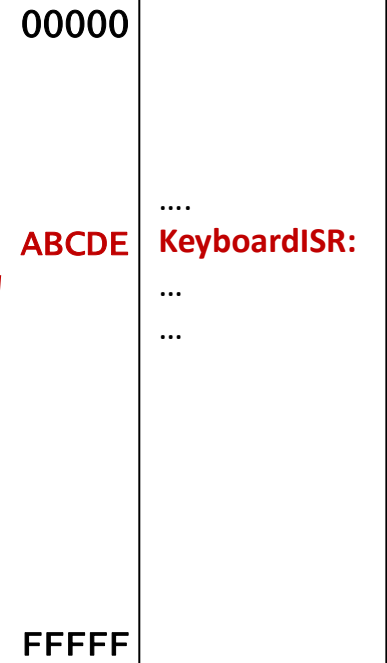


# Using an IVT entry

	(IP)	(CS)
INT 0	Offset0	Segment0
INT 1	Offset1	Segment1
INT 2	Offset2	Segment2
...	...	...
INT 9	Offset9 <b>000E</b>	Segment9 <b>ABCD</b>
...	...	...
INT N	OffsetN	SegmentN
...	...	...
INT 255	Offset255	Segment255

$$\begin{array}{r} ABCD0 \\ + \text{0000E} \\ \hline = ABCDE \end{array}$$

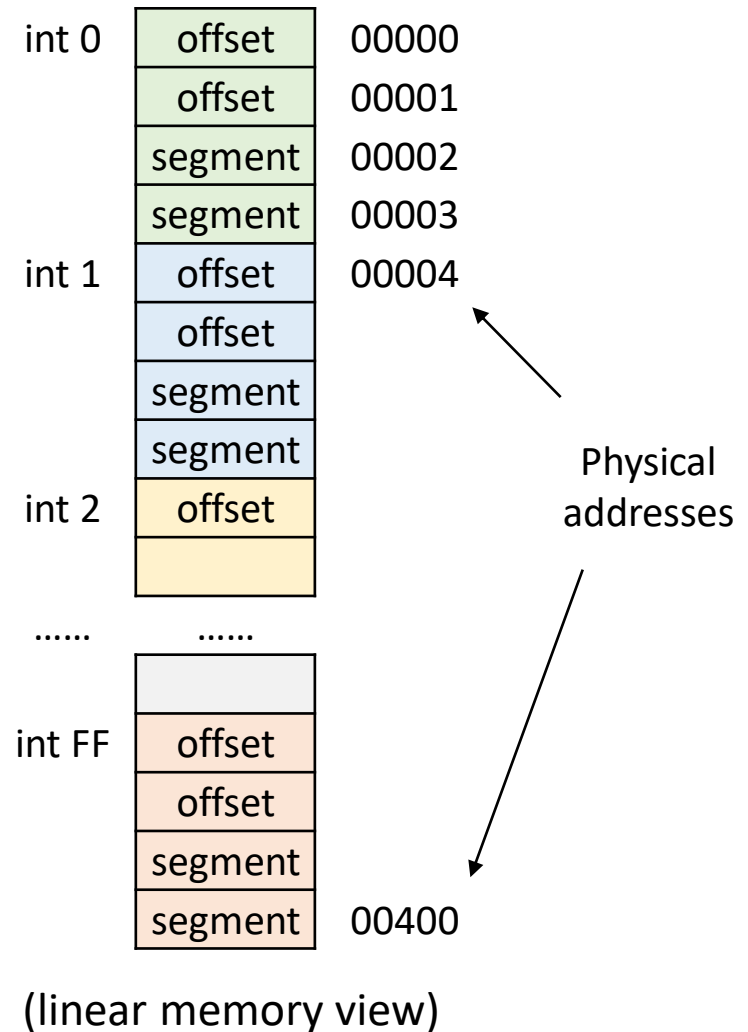
**Segment9:Offset9**  
gives us physical address  
of KeyboardISR



IVT is like a table of contents for interrupt handlers.

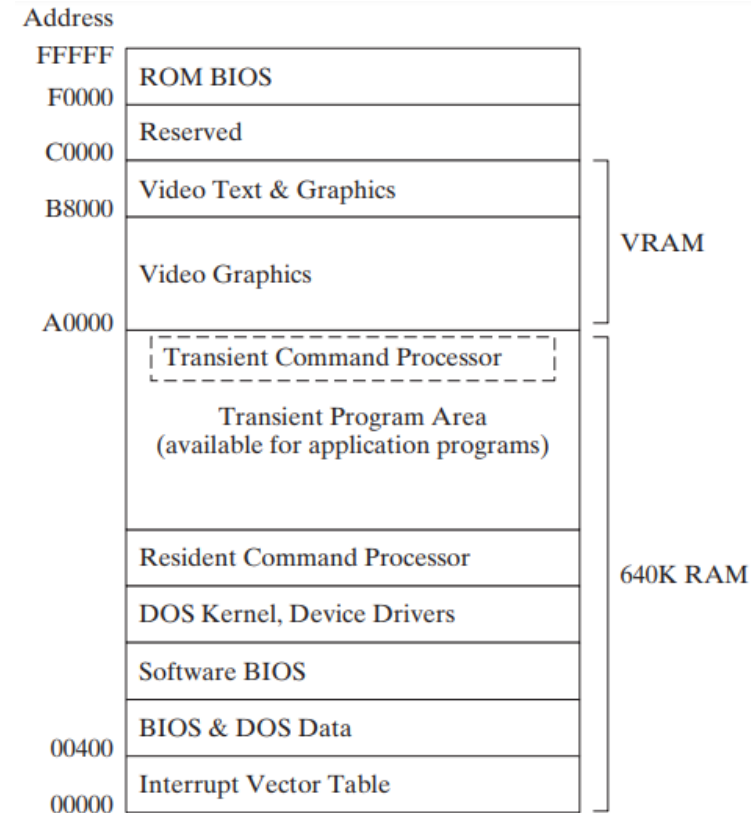
# Interrupt Vector Table (IVT)

- Offset of n-th Interrupt is at  $n*4$ th byte in IVT
- Segment of n-th interrupt is at  $n*4+2$  byte in IVT
- Each entry in the table is called a vector
- The actual contents in this table vary from one machine to another



# Where is IVT located?

- IVT requires  $256 \times 4 = 1024$  bytes (1KB) of physical memory
- Just like ToC is at the start of a book, IVT is located in the very beginning of physical memory.
- From physical address 00000 to 00399 hex, there is interrupt vector table.
- In memory, after IVT there are blocks reserved by system use.
- Programs are loaded and executed in Transient Program Area



Physical memory  
organization in an IBM PC

(note: memory drawn in reverse here)

# Interrupt-response of CPU

1. Complete current Instruction
  2. pushf (push flags register)
  3. cli (clear interrupt flag – disable external interrupts)
  4. clt (clear trap flag – disable line by line debugging)
  5. push CS
  6. push IP
  7.  $IP \leftarrow [n*4]$  (for n-th interrupt)
  8.  $CS \leftarrow [n*4+2]$
- Saving the return location (Note: IP points to next instruction)*
- Far jump to ISR of interrupt n*

# CPU returning back from ISR

- The last instruction of each interrupt handler must be IRET.
- Using IRET, the interrupt handler tells CPU to resume the previously suspended activities.
- It is very similar to how RET instruction is paired with CALL to return from a subroutine

```
ISR_n: .....  
        ; assembly code here  
        .....  
        .....  
        IRET ; Return from ISR
```

Typical ISR structure

# IRET operation

The operation of IRET can be written as:

1. pop IP
  2. pop CS
  3. popf (restore the flags register)
- Far jump to previous (interrupted) code location*

# How CPU gets an Interrupt?

- An interrupt may be triggered by
  1. Device external to CPU, for example, moving the mouse or pressing a key on keyboard, inserting a USB.
  2. Special interrupt instructions, e.g.

```
mov ax, 04c00h  
int 21h
```
  3. under certain conditions, by the CPU itself, e.g. divide by zero

# How CPU gets an Interrupt?

- An interrupt may be triggered by

## Hardware Interrupts

1. Device external to CPU, for example, moving the mouse or pressing a key on keyboard, inserting a USB.

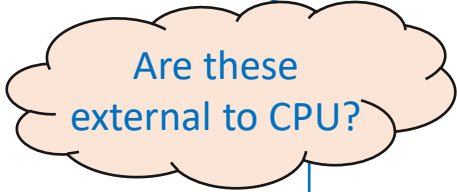
## Software Interrupts

2. Special interrupt instructions, e.g.

```
mov ax, 04c00h
```

```
int 21h
```

3. under certain conditions, by the CPU itself, e.g. divide by zero



Are these  
external to CPU?