Lecture # 34

Interrupts and related concepts # 3

Covered by these slides and chapter # 17
(section 17.4)

Topics

- Overview
- Hardware Interrupts
- Interrupt Control Instructions
- Writing a Custom Interrupt Handler
- Terminate and Stay Resident Programs
- The No_Reset Program

Overview

- Interrupt handler (interrrupt service routine) – performs common I/O tasks
  - can be called as functions
  - can be activated by hardware events
- Examples:
  - video output handler
  - critical error handler
  - keyboard handler
  - divide by zero handler
  - Ctrl-Break handler
  - serial port I/O

Interrupt Vector Table

- Each entry contains a 32-bit segment/offset address that points to an interrupt service routine
- Offset = *interruptNumber* * 4
- The following are only examples:

Interrupt Number	Offset	Interrupt Vectors
00-03	0000	02C1:5186 0070:0C67 0DAD:2C1B 0070:0C67
04-07	0010	0070:0C67 F000:FF54 F000:837B F000:837B
08-0B	0020	0D70:022C 0DAD:2BAD 0070:0325 0070:039F
0C-0F	0030	0070:0419 0070:0493 0070:050D 0070:0C67
10-13	0040	C000:0CD7 F000:F84D F000:F841 0070:237D

Hardware Interrupts

- Generated by the Intel 8259 Programmable Interrupt Contoller (PIC)
  - in response to a hardware signal
- Interrupt Request Levels (IRQ)
  - priority-based interrupt scheduler
  - brokers simultaneous interrupt requests
  - prevents low-priority interrupt from interrupting a high-priority interrupt

Common IRQ Assignments

- 0 System timer
- 1 Keyboard
- 2 Programmable Interrupt Controller
- 3 COM2 (serial)
- 4 COM1 (serial)
- 5 LPT2 (printer)
- 6 Floppy disk controller
- 7 LPT1 (printer)

Common IRQ Assignments

- 8 CMOS real-time clock
- 9 modem, video, network, sound, and USB controllers
- 10 (available)
- 11 (available)
- 12 mouse
- 13 Math coprocessor
- 14 Hard disk controller
- 15 (available)

Interrupt Control Instructions

- STI – set interrupt flag
  - enables external interrupts
  - always executed at beginning of an interrupt handler
- CLI – clear interrupt flag
  - disables external interrupts
  - used before critical code sections that cannot be interrupted
  - suspends the system timer

Writing a Custom Interrupt Handler

- Motivations
  - Change the behavior of an existing handler
  - Fix a bug in an existing handler
  - Improve system security by disabling certain keyboard commands
- What's Involved
  - Write a new handler
  - Load it into memory
  - Replace entry in interrupt vector table
  - Chain to existing interrupt hander (usually)

Get Interrupt Vector

- INT 21h Function 35h – Get interrupt vector
  - returns segment-offset addr of handler in ES:BX

```
.data
int9Save LABEL WORD
DWORD ?                 ; store old INT 9 address here
.code
mov ah,35h                      ; get interrupt vector
mov al,9                        ; for INT 9
int 21h                         ; call MS-DOS
mov int9Save,BX                 ; store the offset
mov [int9Save+2],ES             ; store the segment
```

Set Interrupt Vector

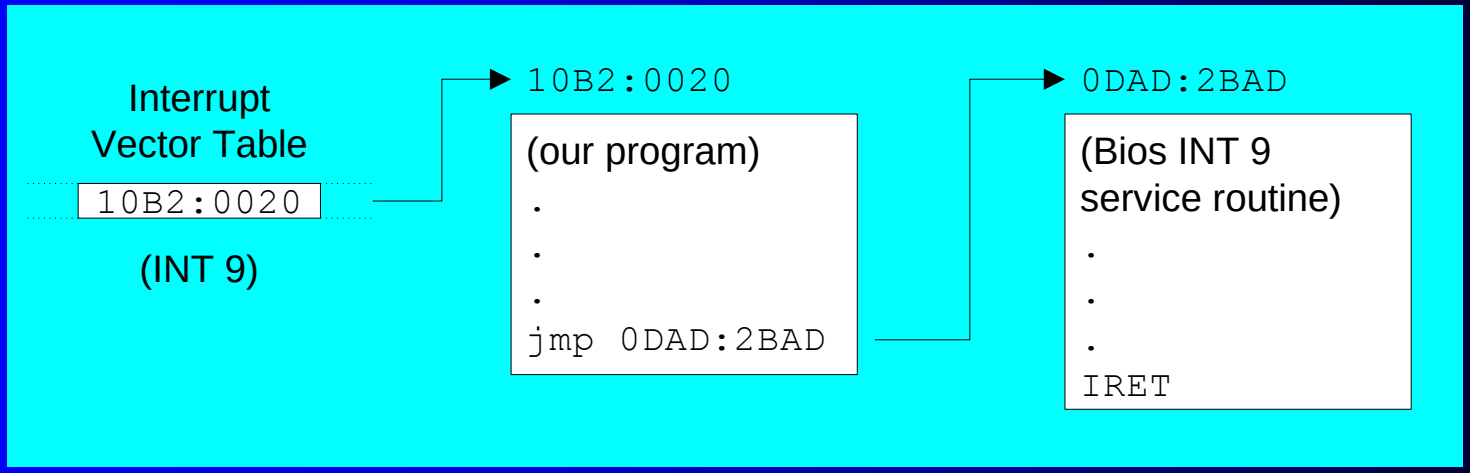
- INT 21h Function 25h – Set interrupt vector
  - installs new interrupt handler, pointed to by DS:DX

```
mov ax,SEG kybd_rtn          ; keyboard handler
mov ds,ax                    ; segment
mov dx,OFFSET kybd_rtn       ; offset
mov ah,25h                   ; set Interrupt vector
mov al,9h                    ; for INT 9h
int 21h
.
.
kybd_rtn PROC                ; (new handler begins here)
```

See the CtrlBrk.asm program.

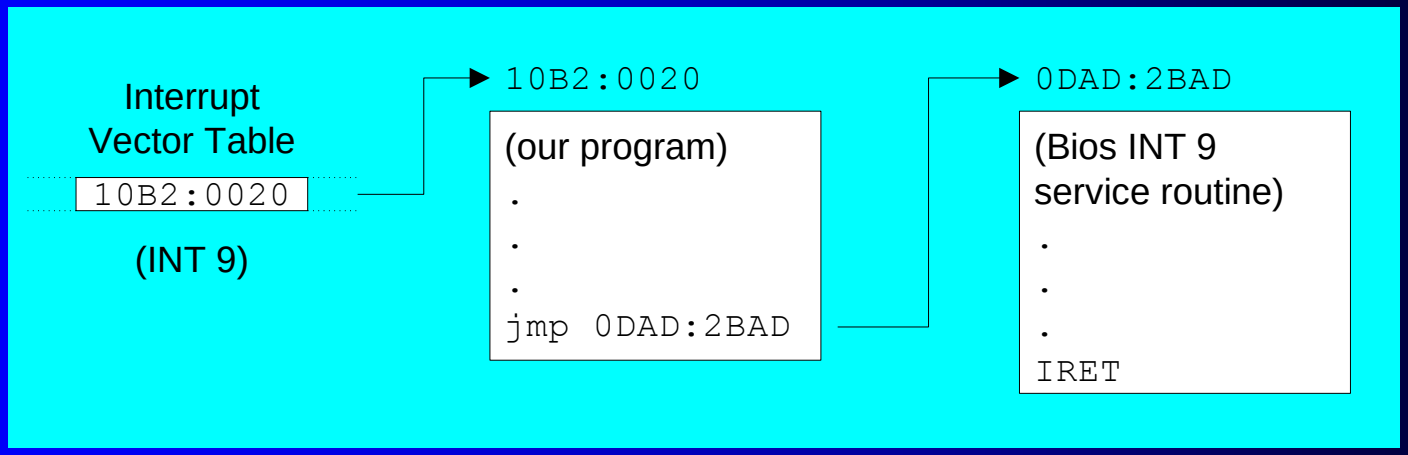
Keyboard Processing Steps

1. Key pressed, byte sent by hardward to keyboard port
2. 8259 controller interrupts the CPU, passing it the interrupt number
3. CPU looks up interrupt vector table entry 9h, branches to the address found there



Keyboard Processing Steps

4. Our handler executes, intercepting the byte sent by the keyboard
5. Our handler jumps to the regular INT 9 handler
6. The INT 9h handler finishes and returns
7. System continues normal processing



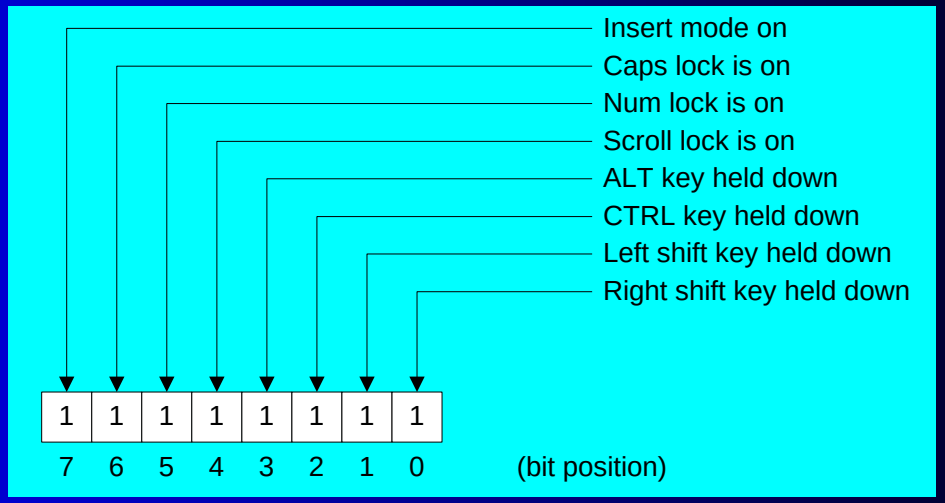
Terminate and Stay Resident Programs

- (TSR): Installed in memory, stays there until removed
  - by a removal program, or by rebooting
- Keyboard example
  - replace the INT 9 vector so it points to our own handler
  - check, or filter certain keystroke combinations, using our handler
  - forward-chain to the existing INT 9 handler to do normal keyboard processing

The No_Reset Program (1 of 5)

- Inspects each incoming key
- If the Del key is received,
  - checks for the Ctrl and Alt keys
  - permits a system reset only if the Right shift key is also held down

The keyboard status byte indicates the current state of special keys:



The No_Reset Program (2 of 5)

- View the source code
- Resident program begins with:

```
int9_handler PROC FAR
    sti                         ; enable hardware interrupts
    pushf                       ; save regs & flags
    push  es
    push  ax
    push  di
```

The No_Reset Program (3 of 5)

- Locate the keyboard flag byte and copy into AH:

```
L1: mov   ax,40h                ; DOS data segment is at 40h
    mov   es,ax
    mov   di,17h                ; location of keyboard flag
    mov   ah,es:[di]            ; copy keyboard flag into AH
```

- Check to see if the Ctrl and Alt keys are held down:

```
L2: test  ah,ctrl_key           ; Ctrl key held down?
    jz    L5                    ; no: exit
    test  ah,alt_key            ; ALT key held down?
    jz    L5                    ; no: exit
```

The No_Reset Program (4 of 5)

- Test for the Del and Right shift keys:

```
L3: in    al,kybd_port          ; read keyboard port
    cmp   al,del_key            ; Del key pressed?
    jne   L5                    ; no: exit
    test  ah,rt_shift           ; right shift key pressed?
    jnz   L5                    ; yes: allow system reset
```

- Turn off the Ctrl key and write the keyboard flag byte back to memory:

```
L4: and   ah,NOT ctrl_key       ; turn off bit for CTRL
    mov   es:[di],ah            ; store keyboard_flag
```

The No_Reset Program (5 of 5)

- Pop the flags and registers off the stack and execute a far jump to the existing BIOS INT 9h routine:

```
L5: pop   di                          ; restore regs & flags
    pop   ax
    pop   es
    popf
    jmp   cs:[old_interrupt9]         ; jump to INT 9 routine
```