Lecture 10

Dr. Zbigniew Kalbarczyk

University of Illinois at Urbana- Champaign

- Writing and installing your own interrupt service routines (cont)
- Video display

Replacing An Interrupt Handler

```
;install new interrupt vector
%macro setInt 3   ;Num, OffsetInt, SegmentInt
    push  ax
    push  dx
    push  ds

    mov   dx, %{2}
    mov   ax, %{3}
    mov   ds, ax
    mov   al, %{1}
    mov   ah, 25h      ;set interrupt vector
    int   21h

    pop   ds
    pop   dx
    pop   ax
%endmacro
```

```
;store old interrupt vector
%macro getInt 3   ;Num, OffsetInt, SegmentInt
    push  bx
    push  es

    mov   al, %{1}
    mov   ah, 35h      ;get interrupt vector
    int   21h
    mov   %{2}, bx
    mov   %{3}, es

    pop   es
    pop   bx
%endmacro
```


Replacing An Interrupt Handler

```
CR  EQU   0dh
LF  EQU   0ah
……..

Warning    DB  "Overflow - Result Set to ZERO!!!!",CR,LF,0
msgOK      DB  "Normal termination", CR, LF, 0
old04hOffset        RESW    1
old04hSegment       RESW    1
```

```
New04h    ;our new ISR for int 04
          ;occurs on overflow
    sti   ;re-enable interrupts

    mov   ax, Warning
    push  ax
    call  putStr      ;display message
    xor   ax, ax      ;set result to zero
    cwd               ;AX to DX:AX

    iret
```

Replacing An Interrupt Handler

```
..start
    mov   ax, cs
    mov   ds, ax

    ;store old vector
    getInt  04h, [old04hOffset], [old04hSegment]

    ;replace with address of new int handler
    setInt  04h, New04h, cs

    mov   al, 100
    add   al, al
    into  ;calls  int 04  if an overflow occurred
    test  ax, 0FFh
    jz    Error

    mov   ax, msgOK
    push  ax
    call  putStr

Error:
    ;restore original int handler
    setInt 04h, [old04hOffset], [old04hSegment]

    mov   ax, 4c00h
    int   21h
```

NOTES

- INTO is a conditional instruction that acts only when the overflow flag is set
- With INTO after a numerical calculation the control can be automatically routed to a handler routine if the calculation results in a numerical overflow.
- By default Interrupt 04h consists of an IRET, so it returns without doing anything.

Installing an interrupt vector using direct access to the vector table

Assume that we need to install an keyboard interrupt (kbd)

```
KbdVec   EQU      24h      ;kbd generates interrupt type 9
                           ;the location in the vector table corresponding to
                           ;this interrupt is 9*4=36 (24h)
saveOffset        resw     1
saveSegment       resw     1

   cli            ;critical section begins - we do not want to be
                  ;interrupted while updating the interrupt vector table

   mov   ax, 0             ;set ES to point to the vector table
   mov   es, ax

   mov   ax, [es:KbdVec]           ;the same as ES:24h
   mov   [saveOffset], ax          ;save old offset
   mov   ax, [es:KbdVec+2]
   mov   [saveSegment], ax         ;save old segment

   mov   ax, kbdInt                ;offset of our interrupt service routine
   mov   [es:KbdVec], ax

   mov  ax, cs
   mov [es:KbdVec+2], ax

   sti                     ;end of the critical section - enable interrupts
```

DOS function dispatcher

- INT 21h is the DOS function dispatcher. It gives you access to dozens of functions built into the operating system.
- To execute one of the many DOS functions, you can specify a sub-function by loading a value into AH just before calling INT 21
- INT 21h sub-functions
  - AH=3Dh: Open File
  - AH=3Fh: Read File
  - AH=3Eh: Close File
  - AH=13h: Delete File (!)
  - AH=2Ah: Get system date
  - AH=2Ch: Get system time
  - AH=2Ch: Read DOS Version
  - AH=47h: Get Current Directory
  - AH=48h: Allocate Memory block (specified in paragraphs==16 bytes)
  - AH=49h: Free Memory block
  - AH=4Ch: Terminate program (and free resources)

System BIOS functions

- All PCs come with a BIOS ROM (or EPROM).
- The BIOS contains procedures that provide basic functions such as bootstraping and primitive I/O.
  - INT 19h: Reboot system
  - INT 11h: Get equipment configuration
  - INT 16h: Keyboard I/O

Video BIOS functions

- Video cards come with procedures stored in a ROM
- Collectively known as the video BIOS
- Located at C0000-C7FFF and holds routines for handling basic video adapter functions
- To execute a function in video BIOS ROM, do an INT 10h with video sub-function number stored in AX
- INT 10h, Sub-function examples
  - AH=0, AL=2h: 80 column x 25 row text display mode
  - AH=0, AL=13h: 320x200 pixel, 256-color graphics display mode

Where is all this stuff???

Address	Contents	Location
FFFFF – D0000	System BIOS functions	ROM or EPROM on system motherboard
C7FFF – C0000	Video BIOS functions	ROM or EPROM on the video display adapter
BFFFF – A0000	Video RAM	
9FFFF – 00400	Memory usable by your real mode programs	640KB of RAM
003FF – 00000	Interrupt vector table	

Video output

- Video is the primary form of communication between a computer and a human.
- The monochrome (single color) monitor uses one wire for video data, one for horizontal sync, and one for vertical sync.
- A color video monitor uses three video signals: **red**, **green**, & **blue**
  - these monitors are called RGB monitors and convert the analog RGB signals to an optical image.
- The RGB monitor is available as either an analog or TTL (digital) monitor.

TTL RGB monitor

- Uses TTL level signals (0 or 5V) as video inputs (RGB) and an extra signal called intensity to allow change in intensity
- Used in the CGA (Color Graphics adaptor) system found in older computers
- Can display a total of 16 different colors (eight are generated at high intensity and eight at low intensity)
- Example:

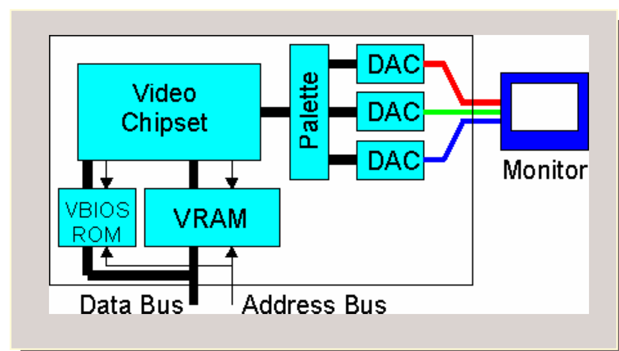
Intensity	Red	Green	Blue	Color
0	0	0	0	Black
1	0	0	0	Gray
1	0	1	0	Light green
1	1	0	0	Light red
1	1	1	1	Bright white

The analog RGB monitor

- Uses analog signals - any voltage between 0.0 V and 0.7 V
- This allow an infinite number of colors to be displayed
- In practice a finite number of levels is generated (16K, 256K, 16M, colors depending on the standard)
- Analog displays use a digital-to-analog converter (DAC) to generate each color video voltage
- A common standard uses a 6-bit DAC to generate 64 different video levels between 0 V and 0.7 V
  - this allows 64x64x64 colors to be displayed, or 262,144 (256 K) colors
- 8-bit DACs allow 256x256x256 or 16M colors

The analog RGB monitor

- The Video Adapter converts digital information from the CPU to analog signals for the monitor.
- VRAM/DRAM Video/Dynamic Random Access Memory
  - Stores screen content
- Video BIOS
  - Stores character mappings
- Palette registers
  - Defines R/G/B color values
- Graphic accelerator
  - Hardware implemented graphic routines
- DAC
  - Generates analog Red/Green/Blue signals



The analog RGB monitor

- Example: video generation used in video standards such as EGA (enhanced graphic adapter) and VGA (variable graphics array)
- A high-speed palette SRAM (access time less than 40ns) stores 256 different codes that represent 256 different hues (18-bit codes)
- This 18-bit code is applied to the DACs
- The SRAM is addressed by 8-bit code that is stored in the computer VRAM to specify color of the pixel
- Once the color code is selected, the three DACs convert it to three video voltages for the monitor to display a picture element (pixel)

The Analog RGB Monitor
Example of Video Generation (cont.)

- Any change in the color codes is accomplished during retrace (moving the electron beam to the upper left-hand corner for vertical retrace and to the left margin of the screen for horizontal retrace)
- The resolution and color depth of the display (e.g., 640x400) determines the amount of memory required by the video interface card
- 640x400 resolution with 256 colors (8 bits per pixel) 256K bytes of memory are required to store all the pixels for the display

Text mode video

- There is not a single common device for supporting video displays
- There are numerous display adapter cards available for the PC
- Each supports several different display modes
- We'll discuss the 80x25 text display mode which is supported by most of display adapters
- The 80x25 text display is a two dimensional array of words with each word in the array corresponding to a character on the screen
- Storing the data into this array affects the characters appearing on the display

Text mode video

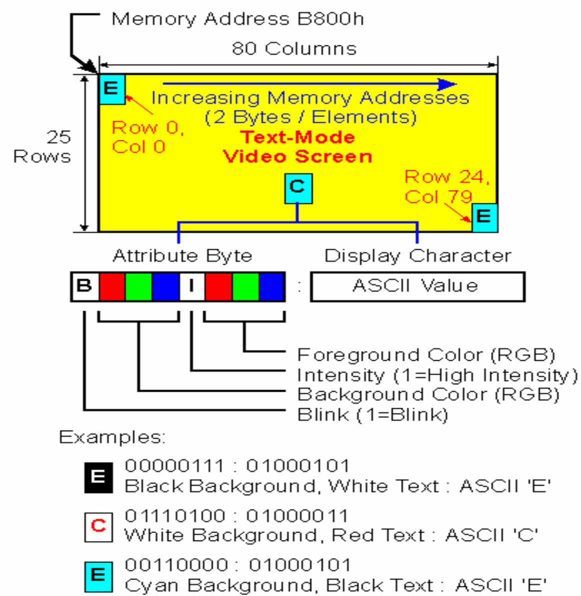
- Each text page occupies under 4K bytes of memory
- 80(columns) x 25 (rows) x 2 (bytes) = 4000 bytes
- The LO byte contains the ASCII code of the character to display
- The HO byte contains the attribute byte
- Display adapters provide 32 K for text displays and let you select one of eight different pages
- Each display begins on a 4K boundary, at address:
  - B800:0000, B800:1000, B800:2000, …. B800:7000

Text mode video

The attribute byte controls underlying background and foreground colors, intensity and blinking video

Choose your colors with care (some combinations of foreground and background colors are not readable)

Do not overdo blinking text on the screen



The cursor

- A pointer to the “insertion point” on the screen
- When you use DOS/BIOS functions to display a character, it displays where the cursor points
- The cursor then moves to the next column
- Other functions let you move backwards or up/down

Using DOS to Control Display – INT 21h

- DOS provides **INT 21h**, which is called the DOS function dispatcher and supports functions such as: read from the keyboard, write to the screen, write to the printer, read and write to disk files, etc.
- INT 21h must be told which function is being requested
  - this information is passed by placing the function number in the **AH register**
  - depending on the function being used, other information may be needed

INT 21h, AH = 02h

- WRITE CHARACTER TO STANDARD OUTPUT
- AH = 02h
- DL = character to write
- Return: AL = last character output (despite the official docs which state nothing is returned)

INT 21h, AH = 6

- DIRECT CONSOLE OUTPUT
- AH = 06h
- DL = character to output (except FFh)
- Return: AL = character output
- Note: When DL = 0FFh then the function reads the console .
  If DL = ASCII character, then the function displays the ASCII character on the console

Displaying A Single Character Using INT 21h

Example:

- Suppose that the letter 'A' is to be printed to the screen at the current cursor position
- Can use function 02h or 06h
- INT 21h must also be told which letter to print
  - the ASCII code must be placed into DL register

```
MOV     DL, 'A'
MOV     AH, 06h
INT     21h
```

INT 21h, AH = 09h

- WRITE STRING TO STANDARD OUTPUT

- AH = 09h
- DS:DX -> '$'-terminated string

- Return: AL = 24h (the '$' terminating the string, despite official docs which state that nothing is returned)

Displaying A Character String Using INT 21h

- DOS function 09h displays a character string that ends with '$'

```
MSG     DB      'This is a test line','$'
...
MOV     DX, MSG
MOV     AH, 09h
INT     21h
```

- The string will be printed beginning at the current cursor position

Using DOS to Control Display – INT 10h

- The DOS function calls allow a key to be read and a character to be displayed but the cursor is difficult to position at a specific location on the screen.
- The BIOS function calls allow more control over the video display and require less time to execute than the DOS function calls
- BIOS provides interrupt **INT 10h**, known as the video interrupt, which gives access to various functions to control video screen
- Before we place information on the screen we should get the position of the cursor:

  function 03h reads cursor position (DH=Row, DL=Column, BH=Page #)

  function 02h sets cursor position (DH=Row, DL=Column, BH=Page #)

INT 10h

- Function 0Fh finds the number of the active page
  - the page number is returned in the BH register
- The cursor position assumes that
  - the left hand page column is column 0 progressing across a line to column 79
  - the row number corresponds to the character line number on the screen, the top line being line 0

Writing characters directly

- Since the VRAM is memory mapped, you can use MOV instructions to write data directly to the display
- Typically, we set the ES register to B800h so that the extra segment can be used to address the VRAM
- Now video display can be accessed just like a 2D word array

Example

- Calculate the offset from the beginning of the VRAM segment (B8000h) for an arbitrary page (P), row (Y) and column (X) in an 80x25 text display mode
  - Offset = 1000h * page + 160 * Y + 2*X

String instructions

- Idea: Setup a data transfer and go
- Do an operation on source [DS:SI] and destination [ES:DI] and change SI and DI depending on the direction flag
- Transfer data much more quickly than loops and movs
- Think of the following instructions in terms of their equivalents
  - You can't do memory to memory operations with other opcodes.
  - The add's at the end of the equivalent code do not affect the flags.

String instructions

- MOVS – Move source to destination
- ```
  mov byte [es:di], byte [ds:si]
  add si, 1  ; if CLD
  add di, 1
  ```
- CMPS – Compare source to destination and set ZF if [ES:DI] = [DS:SI]
- ```
  cmp byte [es:di], byte [ds:si]
  add si, 1  ; If CLD
  add di, 1
  ```

String instructions

- STOS – Store AL/AX/EAX into destination

```
mov byte [es:di], al
add di, 1  ; If CLD
```

- LODS – Load destination into AL/AX/EAX

```
mov al, byte [ds:si]
add si, 1  ; If CLD
```

- SCAS – Compare destination to AL… and set ZF if [ES:DI] = AL…

```
cmp byte [es:di], al
add di, 1                    ; If CLD
```


String instructions

- Each of the instructions should be appended with B, W or D for byte, word, or double word sized transfers.
- REP – What makes all this useful
  - This is a prefix to the above opcodes
  - REP/REPE/REPZ
    - DEC CX
    - LOOP until CX = 0 while ZF = 1
  - REPNE/REPNZ
    - DEC CX
    - Loop until CX = 0 while ZF = 0

String instructions

```
;  Example: Copying a display buffer to the screen
CLD                              ; clear dir flag so we go up

; setup source
MOV SI, DisplayBuffer            ; offset with respect to DS

; setup destination
MOV AX, VidGrSeg                 ; A800
MOV ES, AX                       ; set destination segment as ES
MOV DI, 0                        ; start of screen

; setup counter
MOV CX, (320*200 / 4)            ; moving 4 bytes at a time
REP MOVSD                        ; this takes awhilek
```