Text Display and Keyboard Programming

Module 12

Dr. Tim McGuire

CS 272

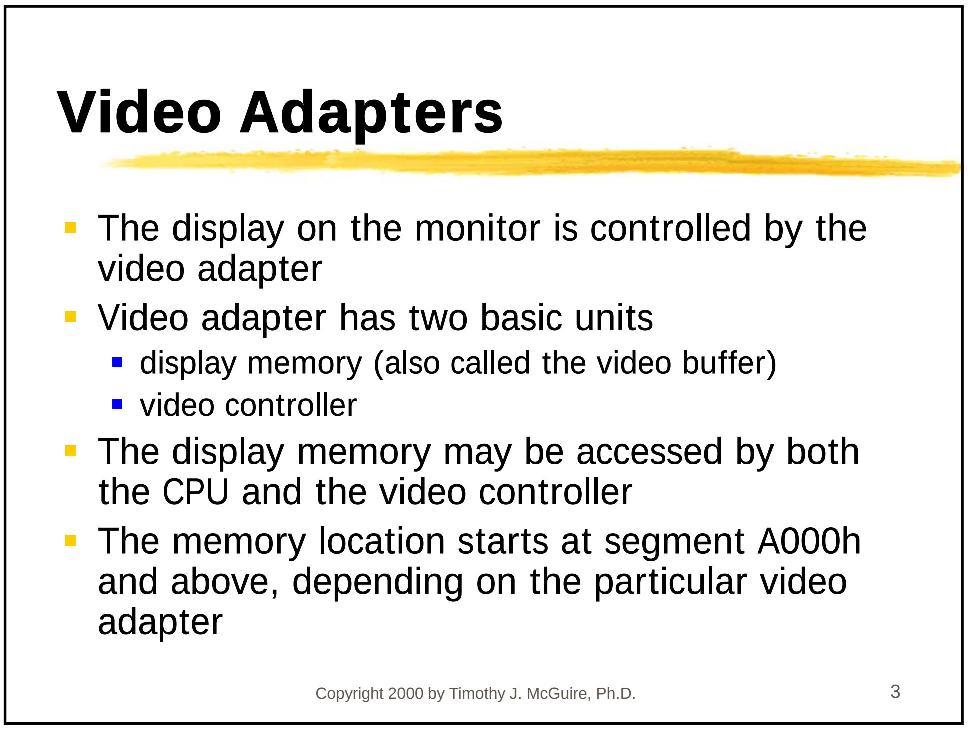
Sam Houston State University

1

Overview

- Assembly language is useful in controlling the monitor display and the keyboard
- We program such operations as:
  - moving the cursor
  - scrolling windows on the screen
  - displaying characters with various attributes
  - reading function keys from the keyboard

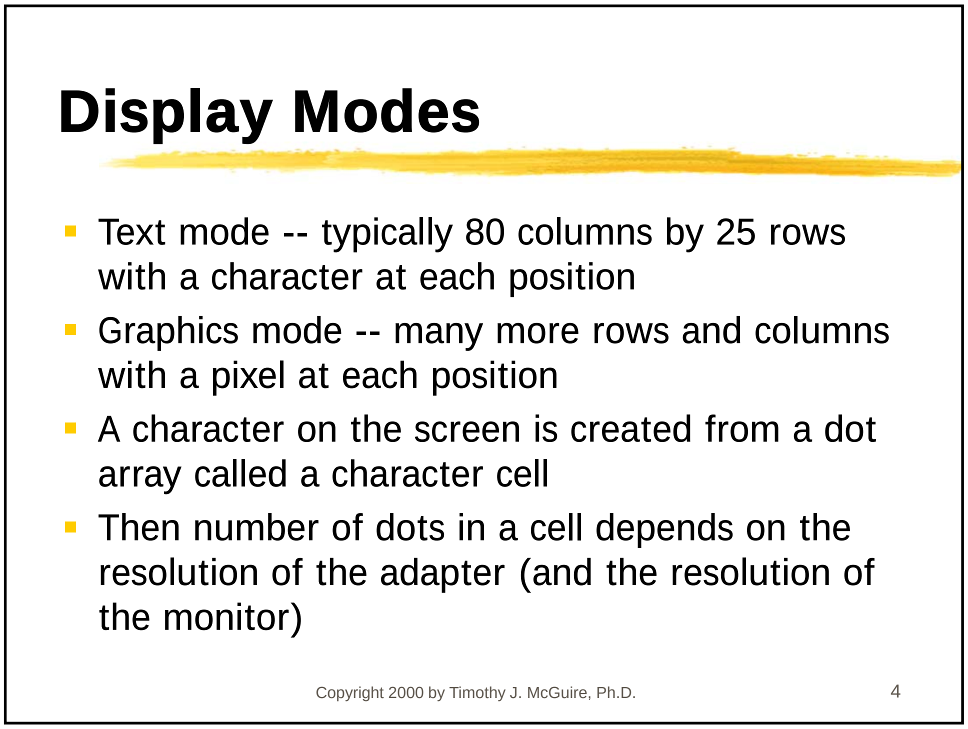
2



Video Adapters

- The display on the monitor is controlled by the video adapter
- Video adapter has two basic units
  - display memory (also called the video buffer)
  - video controller
- The display memory may be accessed by both the CPU and the video controller
- The memory location starts at segment A000h and above, depending on the particular video adapter

3



Display Modes

- Text mode -- typically 80 columns by 25 rows with a character at each position
- Graphics mode -- many more rows and columns with a pixel at each position
- A character on the screen is created from a dot array called a character cell
- Then number of dots in a cell depends on the resolution of the adapter (and the resolution of the monitor)

4

Kinds of Video Adapters

- A little computer archæology here, folks...
- Original PC had
  - MDA (monochrome display adapter) with a 9x14 cell
  - CGA (color graphics adapter) with an 8x8 cell
- EGA (enhanced graphics adapter) had an 8x14 cell
- VGA (video graphics array) has an 8x19 cell

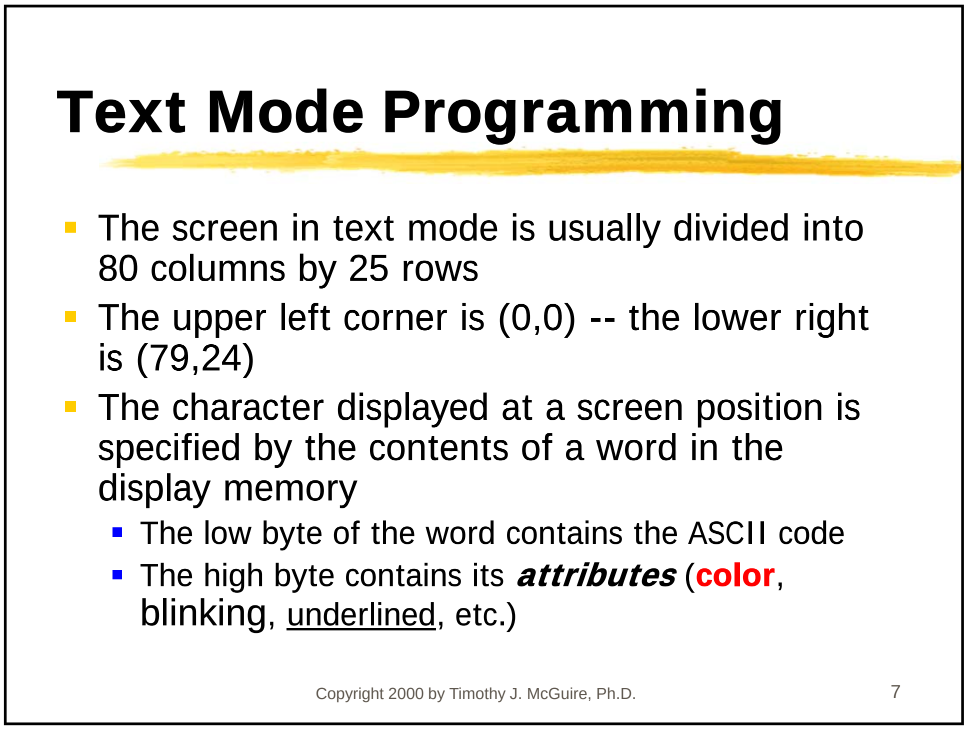
Mode Numbers

- Video Adapter Text Modes

Mode Number	Description	Adapters
0	40x25 16 color text (color burst off)*	CGA,EGA,VGA
1	40x25 16 color text	CGA,EGA,VGA
2	80x25 16 color text (color burst off)*	CGA,EGA,VGA
3	80x25 16 color text	CGA,EGA,VGA
7	80x25 monochrome text	MDA,EGA,VGA

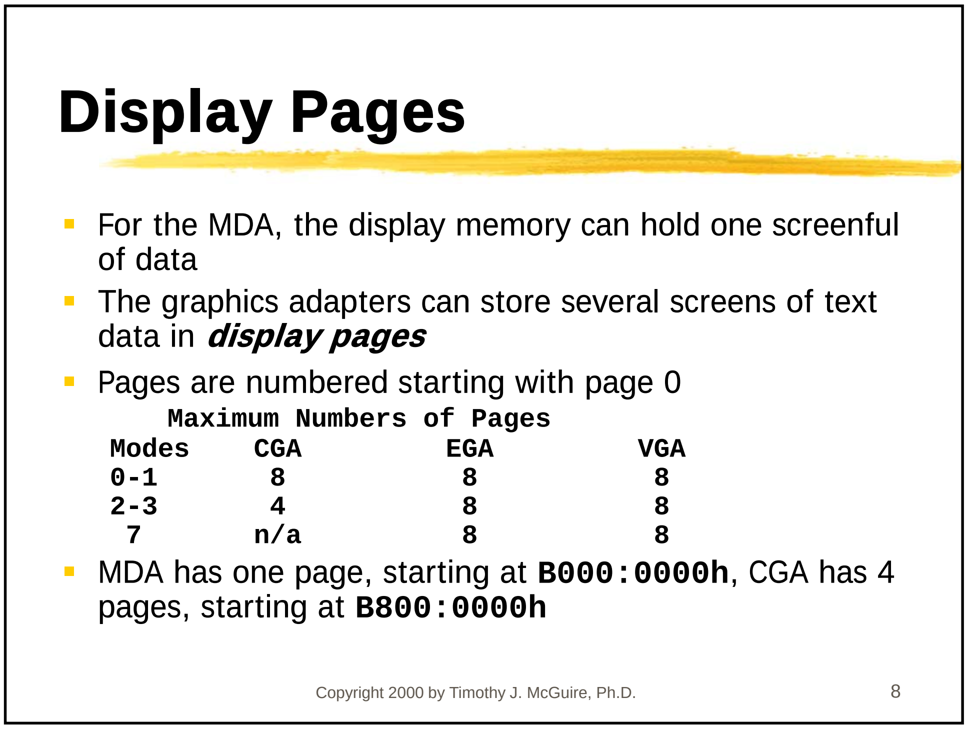
*for composite video only, RGB monitors will display color

fcn 0 fcn Fh



Text Mode Programming

- The screen in text mode is usually divided into 80 columns by 25 rows
- The upper left corner is (0,0) -- the lower right is (79,24)
- The character displayed at a screen position is specified by the contents of a word in the display memory
  - The low byte of the word contains the ASCII code
  - The high byte contains its ***attributes*** (**color**, blinking, underlined, etc.)



Display Pages

- For the MDA, the display memory can hold one screenful of data
- The graphics adapters can store several screens of text data in ***display pages***
- Pages are numbered starting with page 0

Maximum Numbers of Pages

Modes	CGA	EGA	VGA
0-1	8	8	8
2-3	4	8	8
7	n/a	8	8

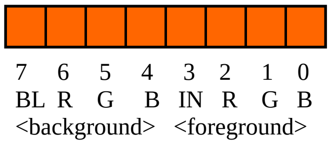
- MDA has one page, starting at **B000:0000h**, CGA has 4 pages, starting at **B800:0000h**

Active Display Page

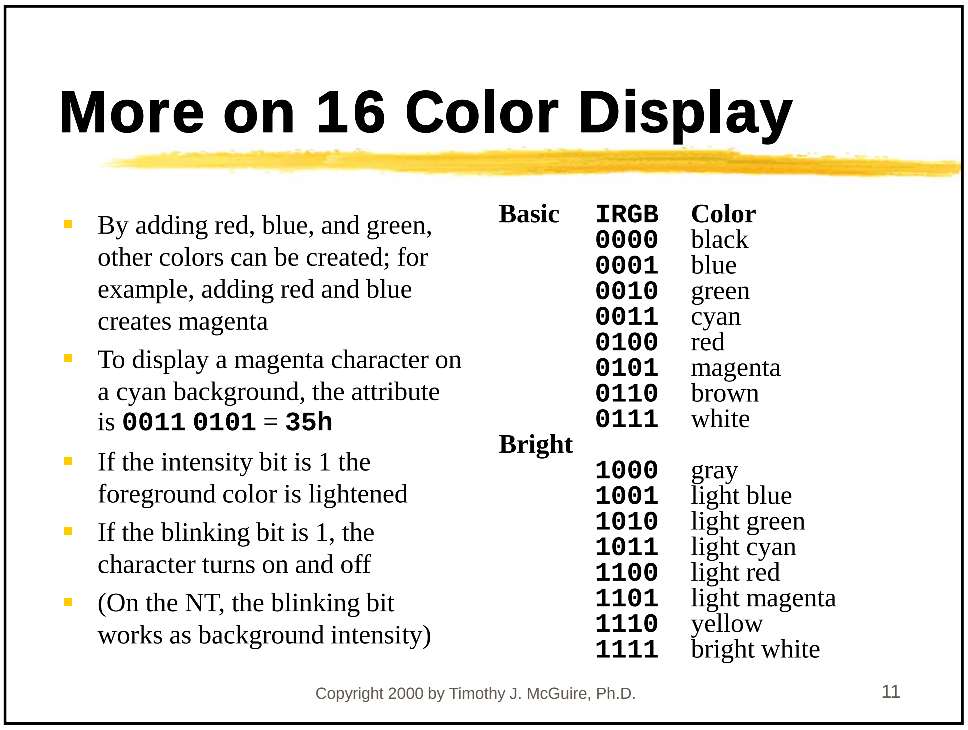
- The active display page is the page currently being displayed
- For an 80x25 mode, the memory requirement is 80x25=2000 words=4000 bytes (the display page is 4K=4096 bytes)
- The video controller displays the first word in the active display page at (row 0,column 0) -- the next word is (row 0, column 1)
- View screen as a 2D row-major order array

The Attribute Byte

- The high byte of the word that specifies a display character is called the **attribute byte**
- Attribute byte for 16-color text display (modes 0-3)



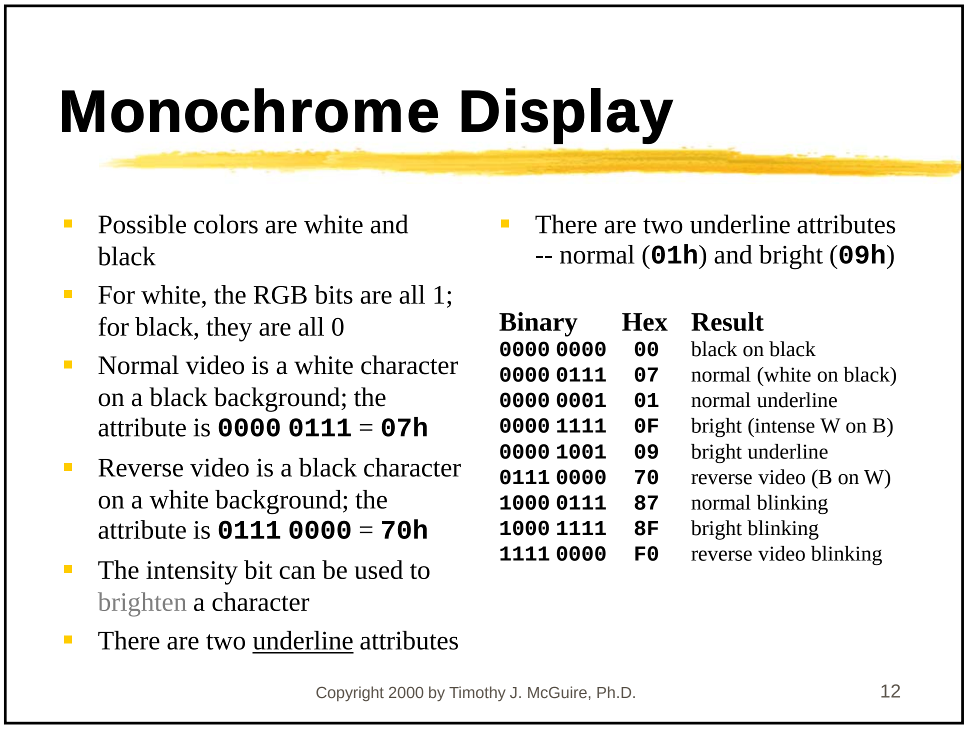
- BL = Blinking
  IN = Intensity
  R=red G=green B=blue
- A 1 in a bit position selects an attribute characteristic
- For example, to display a red character on a blue background, the attribute byte should be **`0001 0100`** = **`14h`**



More on 16 Color Display

- By adding red, blue, and green, other colors can be created; for example, adding red and blue creates magenta
- To display a magenta character on a cyan background, the attribute is **`0011 0101`** = **`35h`**
- If the intensity bit is 1 the foreground color is lightened
- If the blinking bit is 1, the character turns on and off
- (On the NT, the blinking bit works as background intensity)

Basic	**IRGB**	**Color**
	0000	black
	0001	blue
	0010	green
	0011	cyan
	0100	red
	0101	magenta
	0110	brown
	0111	white
Bright		
	1000	gray
	1001	light blue
	1010	light green
	1011	light cyan
	1100	light red
	1101	light magenta
	1110	yellow
	1111	bright white



Monochrome Display

- Possible colors are white and black
- For white, the RGB bits are all 1; for black, they are all 0
- Normal video is a white character on a black background; the attribute is **`0000 0111`** = **`07h`**
- Reverse video is a black character on a white background; the attribute is **`0111 0000`** = **`70h`**
- The intensity bit can be used to brighten a character
- There are two underline attributes

- There are two underline attributes -- normal (**`01h`**) and bright (**`09h`**)

Binary	**Hex**	**Result**
0000 0000	00	black on black
0000 0111	07	normal (white on black)
0000 0001	01	normal underline
0000 1111	0F	bright (intense W on B)
0000 1001	09	bright underline
0111 0000	70	reverse video (B on W)
1000 0111	87	normal blinking
1000 1111	8F	bright blinking
1111 0000	F0	reverse video blinking

A Display Page Demo

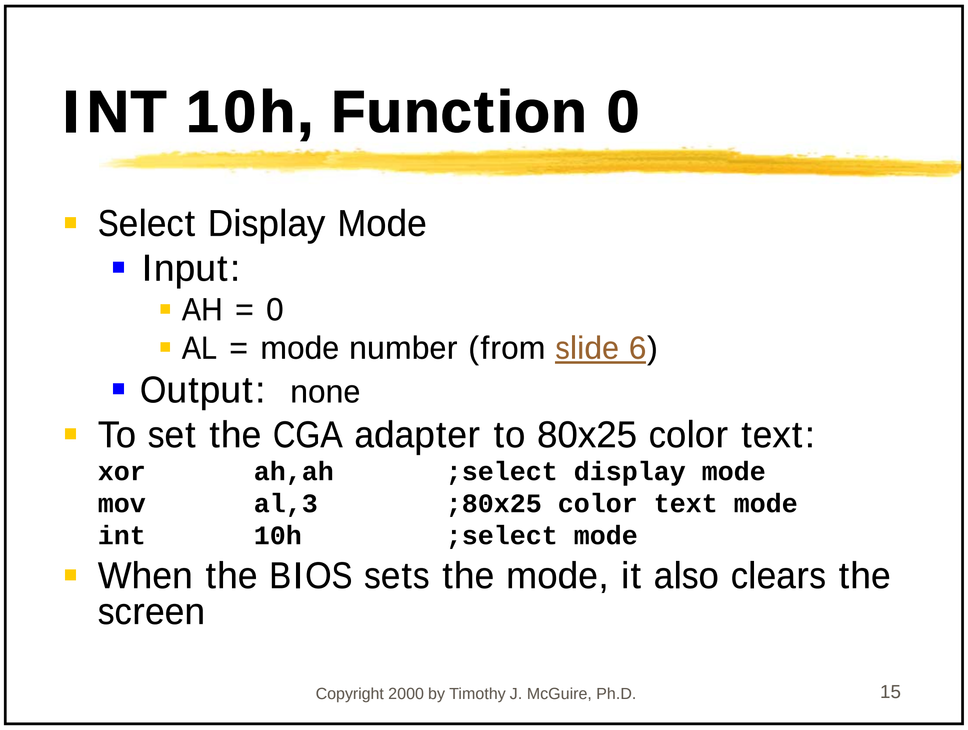
- This code fills the color screen with yellow "**A**"s on a blue background

```
;set DS to active display page
        mov     ax,0B800h  ;color active display page
        mov     ds,ax
        mov     cx,2000    ;80*25 = 2000 words
        mov     di,0       ;initialize DI
        mov     ax,1E41h   ;yellow A on blue
;fill active display page
fillbuf:
        mov     [di],ax    ;char in al, attribute in ah
        add     di, 2      ;go to next word
        loop    fillbuf    ;loop until done
```

- The code is found in **`scr_disp.asm`**

INT 10h

- It's tedious to display data by moving it directly into the active display page
- Instead we use the BIOS video screen routine invoked by the **`INT 10h`** instruction
- As with **`INT 21h`** (DOS system calls) the particular function is selected by putting a function number in the AH register
- We will look first at the most important **`INT 10h`** functions used in text mode

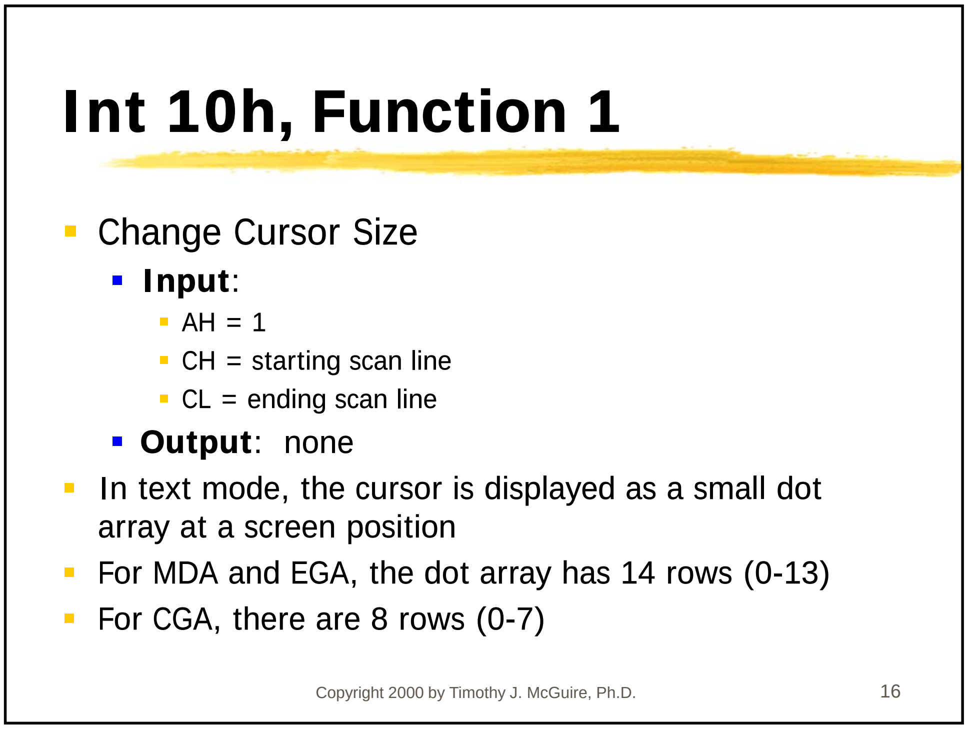


INT 10h, Function 0

- Select Display Mode
  - Input:
    - AH = 0
    - AL = mode number (from slide 6)
  - Output: none
- To set the CGA adapter to 80x25 color text:

```
xor     ah,ah       ;select display mode
mov     al,3        ;80x25 color text mode
int     10h         ;select mode
```

- When the BIOS sets the mode, it also clears the screen



Int 10h, Function 1

- Change Cursor Size
  - **Input**:
    - AH = 1
    - CH = starting scan line
    - CL = ending scan line
  - **Output**: none
- In text mode, the cursor is displayed as a small dot array at a screen position
- For MDA and EGA, the dot array has 14 rows (0-13)
- For CGA, there are 8 rows (0-7)

Cursor Size Example

- To make the cursor as large as possible in EGA mode:

```
mov         ah,1        ;cursor size function
mov         ch,0        ;starting row
mov         cl,13       ;ending row
int         10h         ;change cursor size
```

INT 10h, Function 2

- Move Cursor
  - Input:
    - AH = 2
    - DH = new cursor row (0-24)
    - DL = new cursor column (0-79 for 80x25 mode)
    - BH = page number
  - Output: none
- To move the cursor to the center of page 0:

```
mov       ah,2        ;move cursor function
xor       bh,bh       ;page 0
mov       dx,0C27h    ;row = 12, column = 39
int       10h         ;move cursor
```

INT 10h, Function 3

- Get Cursor Position and Size
  - Input:
    - AH = 3
    - BH = page number
  - Output:
    - DH = cursor row
    - DL = cursor column
    - CH = cursor starting scan line
    - CL = cursor ending scan line
- For some applications, such as moving the cursor up one row, we need to know its current location

An Move Cursor Example

- Move the cursor up one row if not at the top of the screen on page 0:

```
    mov     ah,3        ;read cursor location  fcn
    xor     bh,bh       ;page 0
    int     10h         ;dh = row, dl = column
    or      dh,dh       ;cursor at top of screen?
    jz      exit        ;yes, exit
    mov     ah,2        ;move cursor function
    dec     dh          ;row = row -1
    int     10h         ;move cursor
exit:
```

INT 10h, Function 5

- Set Active Display Page
  - Input:
    - AH = 5
    - AL = active display page
      0 - 7 for modes 0, 1
      0 - 3 for CGA modes 2, 3
      0 - 7 for EGA, VGA modes 2, 3
      0 - 7 for VGA mode 7
  - Output: none
- To select page 1 for the CGA:

```
mov       ah,5        ;select active display page
mov       al,1        ;page 1
int       10h         ;select page
```

INT 10h, Function 6

- Scroll the Screen or a Window Up
  - Input:
    - AH = 6
    - AL = number of lines to scroll (0 means the whole window)
    - BH = attribute for blank lines
    - CH = row for upper left corner of window
    - CL = column for upper left corner of window
    - DH = row for lower right corner of window
    - DL = column for lower right corner of window
- Scrolling the screen up one line means moving each display line up one row and bringing in a blank line at the bottom

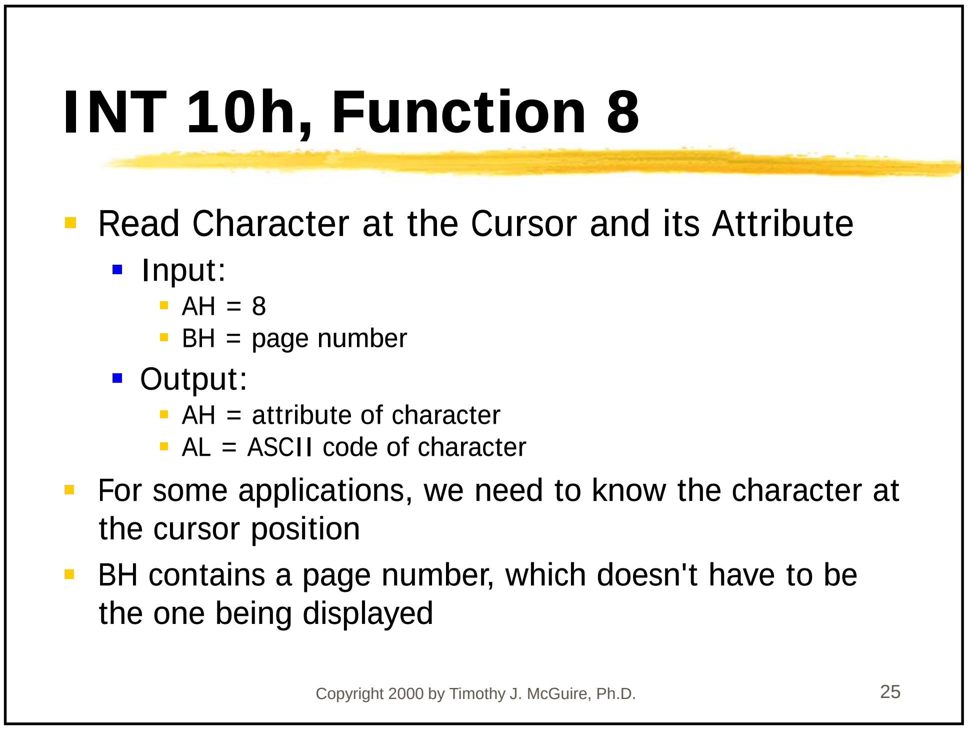
Screen Scrolling Example

- Clear the screen to black for the 80x25 display:

```
mov     ah,6          ;scroll up function
mov     al,0          ;clear whole screen
mov     cx,0          ;upper left corner is (0,0)
mov     dx,184Fh      ;lower right is (4Fh,18h)
mov     bh,07h        ;normal video attribute
int     10h           ;clear screen
```

INT 10h, Function 7

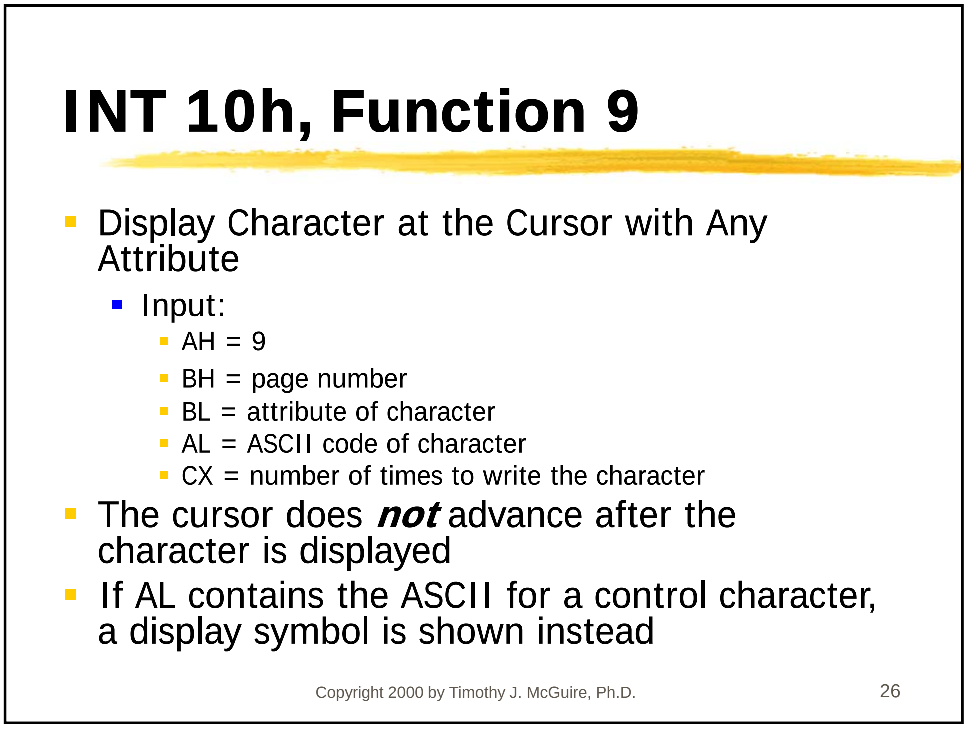
- Scroll the Screen or a Window Down
  - Input:
    - AH = 7
    - AL = number of lines to scroll (0 means the whole window)
    - BH = attribute for blank lines
    - CH = row for upper left corner of window
    - CL = column for upper left corner of window
    - DH = row for lower right corner of window
    - DL = column for lower right corner of window
- When the screen or window is scrolled down one line, each line moves down one row, a blank line is brought in at the top, and the bottom row disappears



INT 10h, Function 8

- Read Character at the Cursor and its Attribute
  - Input:
    - AH = 8
    - BH = page number
  - Output:
    - AH = attribute of character
    - AL = ASCII code of character
- For some applications, we need to know the character at the cursor position
- BH contains a page number, which doesn't have to be the one being displayed

25



INT 10h, Function 9

- Display Character at the Cursor with Any Attribute
  - Input:
    - AH = 9
    - BH = page number
    - BL = attribute of character
    - AL = ASCII code of character
    - CX = number of times to write the character
- The cursor does ***not*** advance after the character is displayed
- If AL contains the ASCII for a control character, a display symbol is shown instead

26

Example of Functions 8&9

- Change the attribute of the character under the cursor to reverse video for monochrome display:

```
mov     ah,8         ;read character
xor     bh,bh        ; from page 0
int     10h          ;char in AL, attrib in AH
mov     ah,9         ;display char with attrib
mov     cx,1         ;display 1 character
mov     bl,70h       ;reverse video attribute
int     10h          ;display character
```

INT 10h, Function 0Ah

- Display Character at the Cursor with Current Attribute
  - Input:
    - AH = 0Ah
    - BH = page number
    - AL = ASCII code of character
    - CX = number of times to write the character
- This function is like function 9, except that the attribute byte is not changed, so the character is displayed with the current attribute

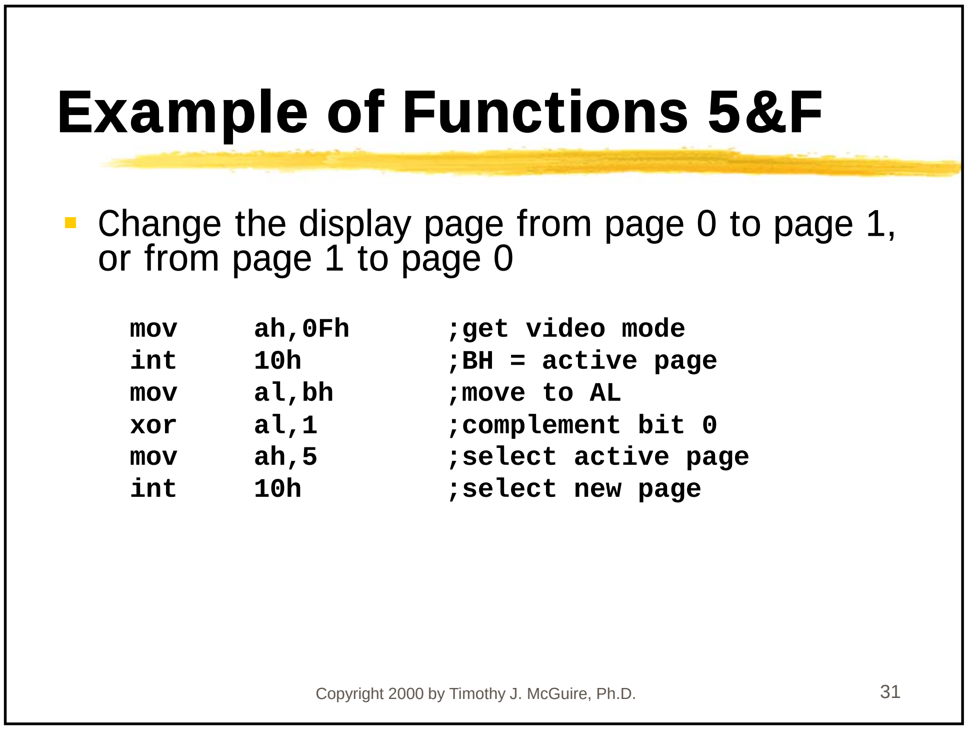
INT 10h, Function Eh

(no, it's not Canadian… eh?)

- Display Character and Advance Cursor
  - Input:
    - AH = 0Eh
    - BH = page number
    - AL = ASCII code of character
- This function displays the character in AL and advances the cursor to the next position in the row
- If at the end of a row, it sends it to the beginning of the next row
- This is the **BIOS** function used by INT 21h, function 2 to display a character
- Control characters are interpreted

INT 10h, Function 0Fh

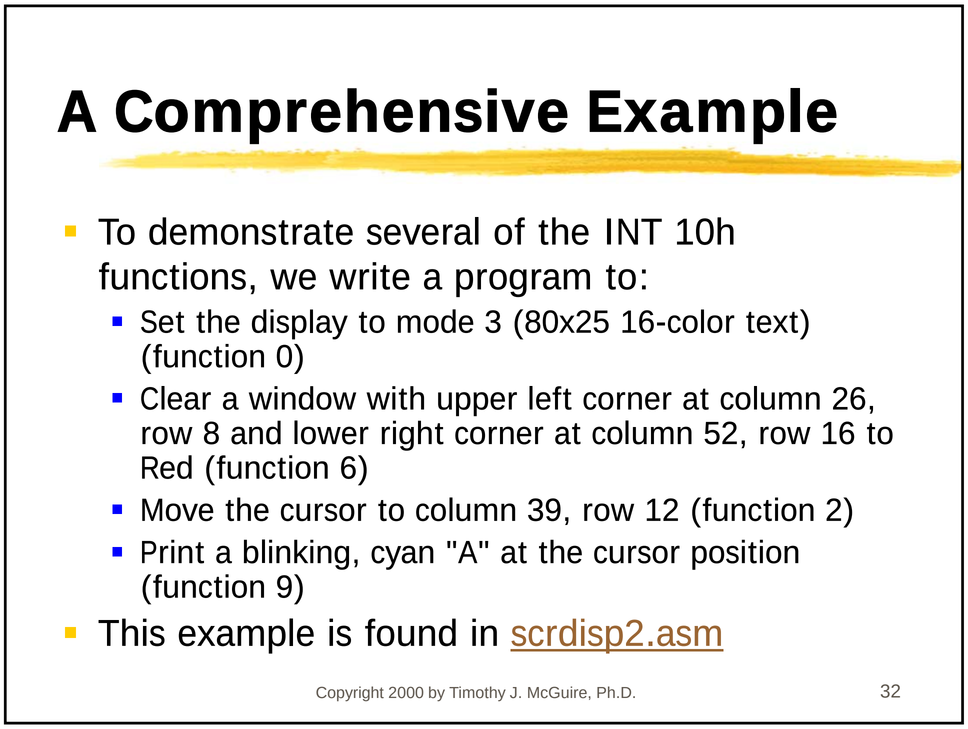
- Get Video Mode
  - Input:
    - AH = 0Fh
  - Output:
    - AH = number of screen columns
    - BH = active display page
    - AL = display mode (see page 6)
- This function can be used with function 5 to switch between pages being displayed



Example of Functions 5&F

- Change the display page from page 0 to page 1, or from page 1 to page 0

```
mov     ah,0Fh      ;get video mode
int     10h         ;BH = active page
mov     al,bh       ;move to AL
xor     al,1        ;complement bit 0
mov     ah,5        ;select active page
int     10h         ;select new page
```



A Comprehensive Example

- To demonstrate several of the INT 10h functions, we write a program to:
  - Set the display to mode 3 (80x25 16-color text) (function 0)
  - Clear a window with upper left corner at column 26, row 8 and lower right corner at column 52, row 16 to Red (function 6)
  - Move the cursor to column 39, row 12 (function 2)
  - Print a blinking, cyan "A" at the cursor position (function 9)
- This example is found in scrdisp2.asm

Screen Display Example

```
; Screen Display Program
   section  .text
Start:
;set video mode
   mov     ah,0                 ;select mode function
   mov     al,3                 ;80x25 color text
   int     10h                  ;select mode
;clear window to red
   mov     ah,6                 ;scroll up function
   mov     cx,081Ah             ;upper left corner (1Ah,08h) (26,8)
   mov     dx,1034h             ;lower right corner (34h,10h)(52,16)
   mov     bh,43h               ;cyan chars on red background
   mov     al,0                 ;scroll all lines
   int     10h                  ;clear window
```

Example, continued

```
;move cursor
   mov     ah,2                 ;move cursor function
   mov     dx,0C27h             ;center of screen (27h,0Ch) (39,12)
   xor     bh,bh                ;page 0
   int     10h                  ;move cursor
;display character with attribute
   mov     ah,09                ;display character function
   mov     bh,0                 ;page 0
   mov     bl,0C3h              ;blinking cyan char, red background
   mov     cx,1                 ;display one character
   mov     al,'A'               ;character is 'A'
   int     10h                  ;display character
;return to DOS
Exit:
   mov     ah,04Ch              ;DOS function: Exit program
   mov     al,0                 ;Return exit code value
   int     21h                  ;Call DOS.  Terminate program
   END     Start                ;End of program / entry point
```

The Keyboard

- The original keyboard for the PC had 83 keys, but most now have 101 or more
- In general, we can group the keys into three categories:
  - ASCII keys -- printable and control characters
  - Shift keys -- left and right shifts, CapsLock, Ctrl, Alt, NumLock, ScrollLock
  - Function keys -- F1-F12, arrow keys, Home, PgUp, PgDn, End, Ins, Del

Scan Codes

- The first computer keyboards were ASCII keyboards, but extended keyboard needs more codes than that
- Each key on the keyboard is assigned a unique number called a **scan code**
- When a key is pressed, the keyboard circuit sends the corresponding scan code to the computer
- Scan codes start at 1 instead of 0

Some Important Scan Codes

Hex	Decimal	Key	Hex	Decimal	Key
1D	29	Ctrl	50	80	Down Arrow
2A	42	LeftShift	51	81	PgDn
38	56	Alt	52	82	Ins
3A	58	CapsLock	53	83	Del
3B-44	59-68	F1-F10	54-5D	84-93	Shift F1-F10
45	69	NumLock	5E-66	94-103	Ctrl F1-F10
46	70	ScrollLock	67-71	104-113	Alt F1-F10
47	71	Home	85-86	133-134	F11-F12
48	72	Up Arrow	87-88	135-136	Shft F11-F12
49	73	PgUp	89-8A	137-138	Ctrl F11-F12
4B	75	Left Arrow	8B-8C	139-140	Alt F11-F12
4C	76	Keypad 5			
4D	77	Rght Arrow			
4F	79	End			

Key Combinations

- How does the computer detect a combination of keys like Ctrl-Alt-Del?
- There is a mechanism for indicating that a key has been pressed but not released
- When a key is released, the keyboard circuit sends another code called a **break code**, derived from the key's scan code by setting the msb to 1 (the scan code - or "make code" - for Ctrl is 1Dh, the break code is 9Dh)

Keyboard Flags

- Information on the status of certain keys pressed and not yet released is stored in the *keyboard flags* status byte in memory address **`0040:0017`**
- This is true for Ins and the Shift keys
- A program can call a BIOS routine to investigate these flags

The Keyboard Buffer

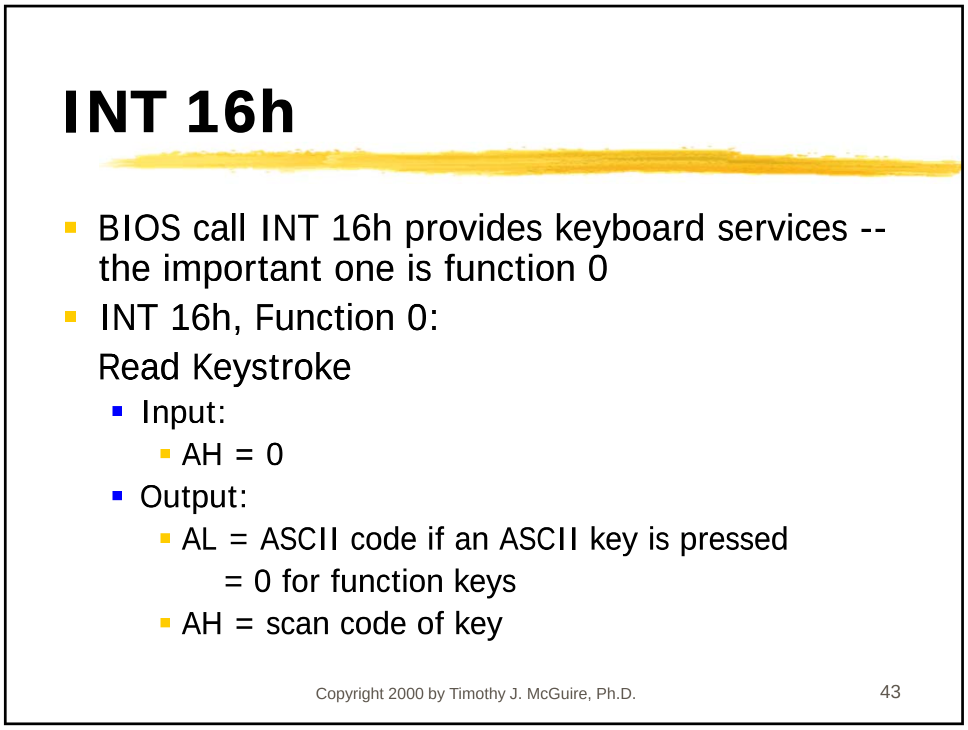
- The computer uses a 15-word block of memory called the **keyboard buffer** to store keys that have been typed but not yet read by the program
- Each keystroke is stored as a word with the high byte containing the key's scan code, and the low byte containing its ASCII code if it's an ASCII key, or 0 if it is a function key
- A shift key is not stored in the buffer

Processing the Buffer

- The contents of the buffer are released when a program requests key inputs
- The key values are passed to the program in the same order they arrive; that is, the keyboard buffer is a *queue*
- If a key input is requested and the buffer is empty, the system waits until a key is pressed
- If the buffer is full and the user presses a key, the computer sounds a tone

Keyboard Operation

- When you press a key:
  - The keyboard sends a request (interrupt 9)
  - The interrupt 9 service routine obtains the scan code from the keyboard i/o port an stores it in a word in the keyboard buffer
  - The current program may use INT 21h, function 1, to read the ASCII code

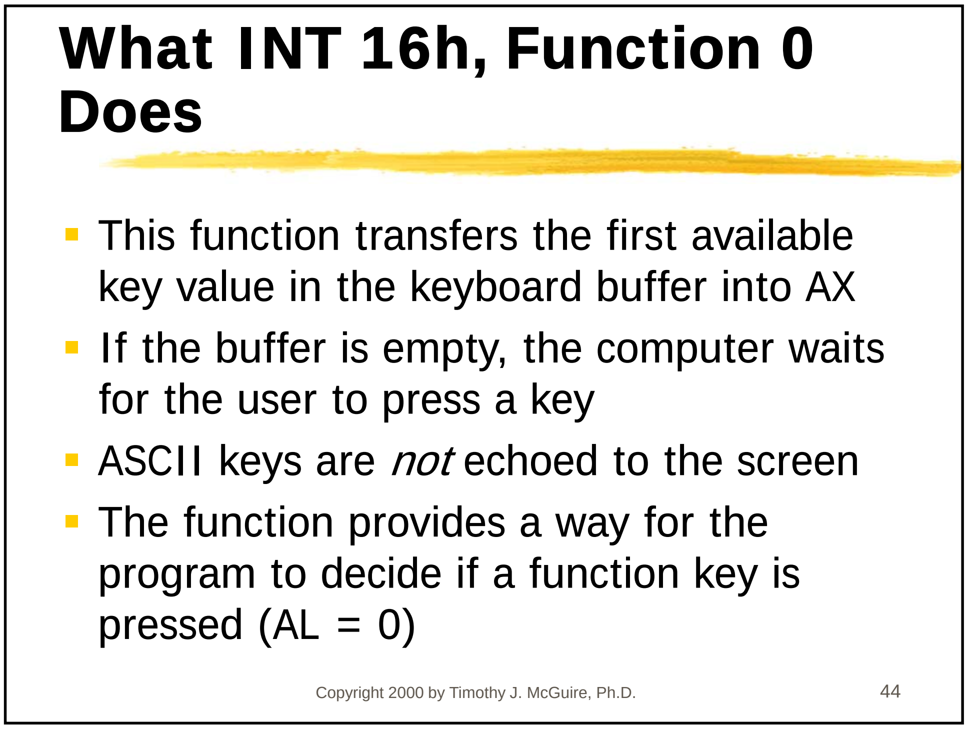


INT 16h

- BIOS call INT 16h provides keyboard services -- the important one is function 0
- INT 16h, Function 0:

  Read Keystroke
  - Input:
    - AH = 0
  - Output:
    - AL = ASCII code if an ASCII key is pressed

      = 0 for function keys
    - AH = scan code of key



What INT 16h, Function 0 Does

- This function transfers the first available key value in the keyboard buffer into AX
- If the buffer is empty, the computer waits for the user to press a key
- ASCII keys are *not* echoed to the screen
- The function provides a way for the program to decide if a function key is pressed (AL = 0)

Example

- Move the cursor to the upper left corner if the F1 key is pressed; and to the lower right corner if any other function key is pressed
- If a character key is pressed, do nothing

45

Example Source:

```
   mov     ah,0                ;read keystroke function
   int     16h                 ;al = ASCII code or 0,
                               ;ah = scan code
   or      al,al               ;al = 0? (function key?)
   jne     Exit                ;no, character key
   cmp     ah,3Bh              ;scan code for F1?
   je      F1                  ;yes, go to move cursor
;other function key
   mov     dx,184Fh            ;lower right corner (4Fh,18h)
   jmp     Execute             ;go to move cursor
F1:
   xor     dx,dx               ;upper left corner
Execute:
   mov     ah,2                ;move cursor function
   xor     bh,bh               ;page
   int     10h                 ;move cursor
Exit:
```

46

A Comprehensive Example

- To show how the function keys may be programmed, here is a program that does some of the things that a basic word processor does
- It first clears the screen and puts the cursor in the upper left corner, then lets the user type text on the screen, operate some of the function keys, and finally exits when the ESC is pressed

Screen Editor Algorithm

```
Clear the screen
Move the cursor to the upper left corner
Get a keystroke
WHILE key is not the Esc key DO
   IF function key THEN
        perform function
   ELSE                    // key is a character key
        display character
   ENDIF
   Get a keystroke
ENDWHILE
```

How to Handle Function Keys

- ESC -- detected by checking for ASCII code 1Bh
- Up arrow
  - cursor moves up one row unless it's at the top of the screen
  - if at top, screen scrolls down one line
- Down arrow
  - cursor moves down one row unless it's at the bottom of the screen
  - if at bottom, screen scrolls up one line

Left and Right Arrow Keys

- Right arrow
  - cursor moves right one column unless it's at the right margin
  - if at right margin, it moves to the beginning of the next row
  - if it's in the lower right corner, screen scrolls up one line
- Left arrow
  - cursor moves left one column unless it's at the left margin
  - if at left margin, it moves to the end of the previous row
  - if it's in the upper left corner, screen scrolls down one line

Algorithm for Function Keys

```
DO_FUNCTION_KEY algorithm
Get cursor position
Examine scan code of last key pressed
CASE scan code OF
 up arrow:
  IF cursor at top of screen THEN
     scroll screen down
  ELSE
     move cursor up one row
  ENDIF
 down arrow:
  IF cursor at bottom of screen THEN
     scroll screen up
  ELSE
     move cursor down one row
  ENDIF
 left arrow:
  IF cursor not at beginning of row THEN
     move cursor to the left
  ELSE
     IF cursor in in row 0 THEN
       scroll screen down
     ELSE
       more cursor to the end of previous row
     ENDIF
  ENDIF
 right arrow:
  IF cursor not at end of row THEN
     move cursor to the right
  ELSE
     IF cursor in in last row THEN
       scroll screen up
     ELSE
       more cursor to the start of next row
     ENDIF
  ENDIF
ENDCASE
```

Program Listing

- The listing is long and is contained in scr_edit.asm

We will look at graphics mode programming later…

Some additional BIOS and DOS Interrupts

- Interrupts 0 to 0Fh

Interrupt	Usage	Interrupt	Usage
0h	Divide by 0	9h	Keyboard
1h	Single step	0Ah	Reserved
2h	NMI	0Bh	Serial (COM2)
3h	Breakpoint	0Ch	Serial (COM1)
4h	Overflow	0Dh	Fixed disk
5h	PrintScreen	0Eh	Floppy disk
6h	Reserved	0Fh	Parallel port
7h	Reserved		
8h	Timer tick		