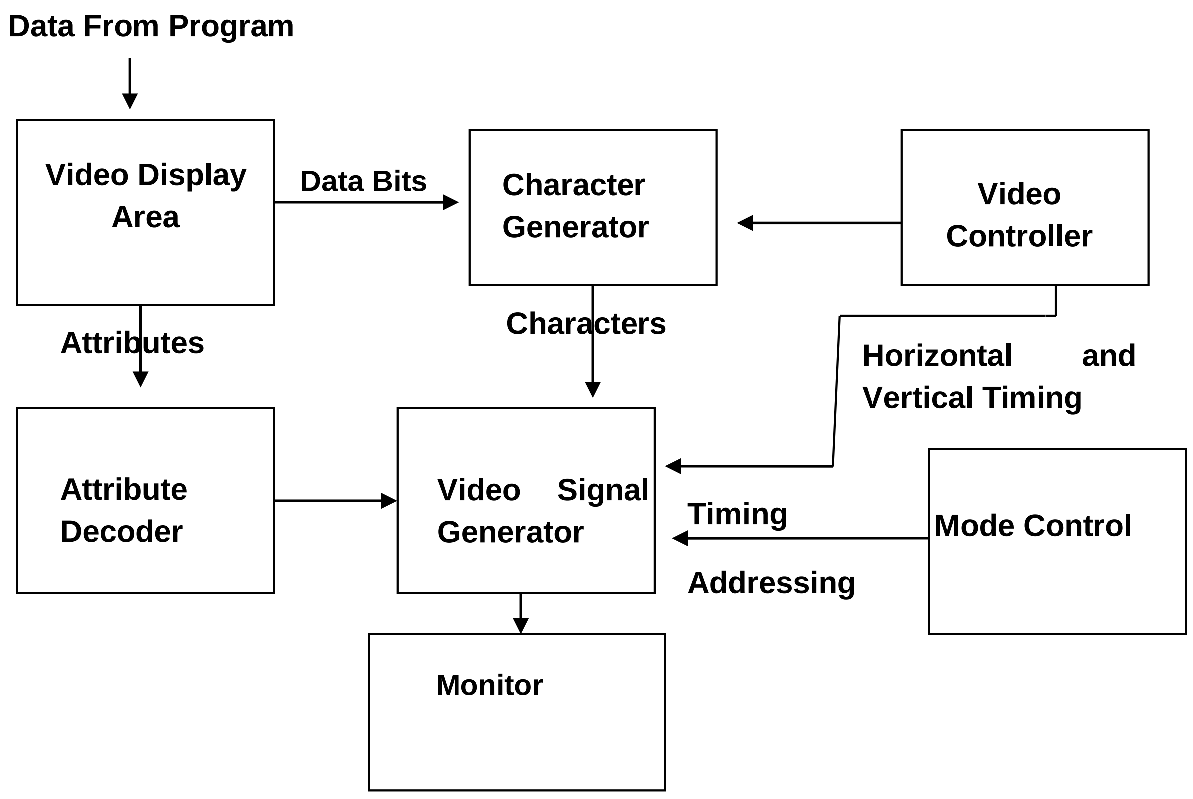
Module 4: Video and Keyboard Interrupts

Lesson 1: Video Interrupts

Learning Outcomes

- LO 4.1.1 Explain the block diagram and the uses of the various components of the Video system
- LO 4.1.2 Determine the uses of various Video modes and attributes then properly apply these on setting up screen display
- LO 4.1.3 Explain the various INT 10H functions and implement it on various screen display manipulation.

Block Diagram of Video System



The common (or once-common) video adapters include MDA (Monochrome display adapter), CGA (color graphics adapter), EGA (enhanced graphics adapter), and VGA (Video graphics array). The VGA and its super VGA successors replaced CGA and EGA video adapters.

The basic components of a video system are monitor, Video Display Area, Video BIOS, and Video Controller. Other integrated devices include Character Generator, Mode Controller, Video Signal Generator, and attribute decoder.

Monitor

The monitor's screen consists of group of closely-related horizontal lines known as the raster. Each line contains hundreds of points called the pixels, which consists of three luminescent phosphor dots for each of the three primary additive colors: red, green, blue.

Three electron beams activate the three colors in the pixels. The beams start at the upper left corner of the screen and scan each line successively from left to right. The varying intensity of the beams set the brightness and color of the pixels. The combinations of red, green, blue, along with intensity, create the various colors and shades.

Video Display Area

A program sends data – characters for text mode and pixels for graphic mode – to the Video Display Area (or Buffer) in RAM either by means of an INT operation or by transferring directly into the area. The data undergoes a complex transformation before it is finally displayed on the screen. The starting address of the Video Area depends on the type of video adapter and the chosen mode. The interrupts that handle screen displays transfer your data directly to this area. Following are the beginning address of the video mode.

- A000:[0] Used for font descriptors when in text mode and for high resolution graphics for video modes 0DH through 13H.
- B000:[0] Monochrome text for mode 07H
- B800:[0] Text and graphics for modes 00H through 06H

The video circuitry continuously scans the data in the Video Area and refreshes the screen accordingly. Data in the Video Area may be ASCII text (alphanumeric) or graphics format. In text mode, each character in the Video Area requires two bytes: one byte for the character immediately followed by an attribute byte that determines the character's color and intensity. In graphics mode, the Video Area contains of groups of bits that determine the color of each pixel.

The Video Display area allows you to store data in pages. A page stores a screenful of data ans is numbered 0 through 7. Page number 0 is the default and, for text modes, begins in the Video Display Area at B800[0]. Page 1 begins at B900[0], page 2 at BA00[0], page 3 at BB00[0], and so forth.

You may format at any page of the pages in the memory, although you can display only one page at a time. In text mode, each character to be displayed on the screen requires two bytes of memory-one byte for the character and a second for its attribute. In this way, a full page of characters for 80 column and 25 row requires 80x25x2 = 4000 bytes. The amount of memory actually allocated for each page is 4K, or 4,096 bytes, so that a block of 96 unused bytes immediately follows each page.

Video Controller

Video controller generates horizontal and vertical timing signals. It maintains and increments a counter that indicates the current location in the Video Display Area. The counter tells the video circuitry the current data to access, to decode, and to send to the monitor. The Controller has to synchronize the delivery of the data with the timing signals.

Immediately following its horizontal scanning, the Controller issues a vertical synch signal, which causes the monitor to perform vertical scanning beginning at the top left corner. Both horizontal and vertical operations perform overscanning, which result in a border (the overscan area) around the four sides of the screen.

Other tasks of the Video Controller involve handling the size and location of the cursor and selecting the page to be displayed. The controller also contains a

number of register that a program can access for both reading and rewriting their contents.
The ASCII (or Alphanumeric) Character Generator in figure 5.1 converts ASCII codes from the Video Display Area into dot patterns that comprise the characters.
The Attribute Decoder translates

Video BIOS

The video BIOS, which acts as an interface to the video adapter, contains such routines as setting the cursor and displaying characters. Video RAM BIOS supports two Video Data Areas:

1. 40:[49H] contains such data as current mode, number of columns, and size of the Video Display Area.
2. 40:[84H] contains such data as number of rows and character height.

Video Modes

The video mode determines such factors as text or graphics, color or monochrome, screen resolution, and the number of colors. BIOS INT 10H function 00H is used to initialize the mode for the currently executing program or to switch between text and graphics. Setting the mode also clears the screen. An example, mode 03 provides text mode, 25 rows x 80 columns, color and 720x400 dots screen resolution. You can use INT 10H function 0FH, which returns the current video mode in AL. Both functions are covered later.

Text (or alphanumeric) mode is used for displaying the ASCII 256-character. Processing is similar for both color and monochrome, except that color does not support the underline attribute. Following are common text modes, with mode number on the left.

Mode	**Rows x Cols**	**Type**	**Area**	**Pages**	**Resolution**	**Color**
00	25 x 40	Color	B800	0-7	360 x 400	16
01	25 x 40	Color	B800	0-7	360 x 400	16
02	25 x 80	Color	B800	0-3	720 x 400	16
03	25 x 80	Color	B800	0-3	720 x 400	16
07	25 x 80	Monochrome	B000	0	720 x 400	

Attributes

The attribute byte in the text mode determines the characteristics of each displayed character. When a program sets an attribute, it remains set, that is, all subsequent displayed characters have the same attribute until another operation changes it. You can use INT 10H functions to generate a screen attribute and perform such actions as scroll up or down, read attribute or character, or display attribute or character. The attribute byte has the following format:

Background				**Foreground**			
BL	R	G	B	I	R	G	B
7	6	5	4	3	2	1	0

The letters R, G, and B indicate bit positions for red, green, and blue, respectively, for each of the three primary color additive color.

- Bit 7 (BL) sets blinking (may be disabled)
- Bits 6-4 determine the character's background color.
- Bit 3 (I) sets normal (if 0) or high intensity (if 1)
- Bits 2-0 determine the character's foreground color.

The background can display one of eight color and the foreground characters can display one of 16 colors. Blinking and intensity apply only to the foreground, although you can use INT 10H function 10H to override the blinking feature and enable the foreground to display 16 colors. You can also select on of 16 colors for the border.

You can combine the three basic video colors red (R),m green (G), and blue (B) in the attribute byte to form a total of eight colors (including black and white) and can set high intensity (I in the chart below), for a total of 16 colors.

Color	**I**	**R**	**G**	**B**	**Hex**	**Color**	**I**	**R**	**G**	**B**	**Hex**
Black	0	0	0	0	0	Gray	1	0	0	0	8
Blue	0	0	0	1	1	L. Blue	1	0	0	1	9
Green	0	0	1	0	2	L. Green	1	0	1	0	A
Cyan	0	0	1	1	3	L. Cyan	1	0	1	1	B
Red	0	1	0	0	4	L. Red	1	1	0	0	C
Magenta	0	1	0	1	5	L. Magenta	1	1	0	1	D
Brown	0	1	1	0	6	Yellow	1	1	1	0	E
White	0	1	1	1	7	Bright White	1	1	1	1	F

If the background and foreground color are the same, the displayed character is invisible. You can also use the attribute byte to cause a foreground attribute for the foreground character to blink. The video system causes blinking in this way. It substitutes the background attribute for the foreground attribute about every two seconds, so that the normal character alternates with a blank character.

Background	**Foreground**	**Background**				**Foreground**				
		BL	**R**	**G**	**B**	**I**	**R**	**G**	**B**	**Hex**
Black	Blue	0	0	0	0	0	0	0	1	01

Blue	Red	0	0	0	1	0	1	0	0	14
Green	Cyan	0	0	1	0	0	0	1	1	23
White	Light Magenta	0	1	1	1	1	1	0	1	7D
Green	Gray(Blinking)	1	0	1	0	1	0	0	0	A8

For monochrome monitor, the attribute byte is used the same way as was shown for a color monitor, except that bit 0 sets the underline attribute. To specify attributes, you may set combinations of bits as follows:

- Normal video (black, white) 0000 0111 (07H)
- Reverse video (white, black) 0111 0000 (70H)

The value of the four bits of the attribute byte relate to one of bits 0-3 of the Controller’s Color Plane Enable register then relate to the six RGB signals (three normal and three intense). The bit value in the Palette register specifies on of the 256 DAC (digital-to-analog converter) color registers, which determines the displayed color.

You can generate colors by choosing an attribute for each character. You can also revise the default color in any of all Palette registers by means of INT 10H function 10H.

The attribute remains set until another operation changes it. The INT 10H functions that set the attribute are:

- 06H Scroll Up Screen
- 07H Scroll Down Screen
- 09H Display character with attribute
- 13H Display character string

BIOS INT 10H Operations

INT 10H supports many services (available through function codes in AH) to facilitate video operations. Subject to returned values, the INT operation preserves the contents of BX, CX, DX, DI, SI, DS, ES, and BP.

INT 10H attempts to execute the anything you throw at it, and does not return status codes or error flags. Be especially careful matching proper function codes with INT 10H; you cannot cause any permanent harm, but an error may cause the screen to go blank so that you have to reboot the system.

INT 10H Function 00H: Set Video Mode

The purpose of this function is to set the video mode. Load the function code 00H in AH and the required mode in AL. The following examples set the video mode for standard color text on any type of color monitor.

```
MOV AH, 00H ; Request Set Mode
MOV AL, 03H ; Standard color text
INT 10H           ; Call Interrupt Service
```

The operation returns no values. It also clears the screen, although you can override this feature by setting bit 7 of the mode to 1 using MOV AL, 83H

INT 10H Function 01H: Set Cursor Size

The cursor is no part of the ASCII character set and exists only in text mode. The Video Controller handles the cursor size and location, with special INT 10H operations for its use. The default size for color VGA is 13 for the top of the cursor and 14 for the bottom (6:7 for monochrome). For function 01H to adjust the cursor size vertically, set these registers:

- CH (Bits 4-0) = top of cursor (start scan line)
- CL (Bits 4-0) = bottom of cursor (end scan line)

The code below enlarges the cursor to its maximum size (0:14):

```
        MOV AH, 01H       ; Request set cursor size
        MOV CH, 00        ; Start Scan Line
        MOV CL, 14  ; End Scan Line
        INT 10H           ; Call Interrupt Service
```

INT 10H Function 02H: Set Cursor Position

This operation in text or in graphics mode sets the cursor anywhere on screen according to row:column coordinates. Set these registers: BH=page number (0 is the default), DH=row, and DL=column. This example sets row 12, column 30, for page 0.

```
        MOV AH, 02H       ; Request set cursor
        MOV BH, 00        ; Page Number 0   (Normal)
        MOV DH, 12  ; Row 12
        MOV DL, 30  ; Column 30
                    INT 10H           ; Call Interrupt Service
```

INT 10H Function 03H: Return Cursor Status

You can use function 03H in text or graphics mode to determine the present row: column, and size of the cursor, particularly in situations where a program uses the screen temporarily and has to save a reset the original screen. Set page number in BH, just as for function 02H.

```
              MOV AH, 03H ; Request Cursor location
              MOV BH, 00H ; Page Number
              INT 10H           ; Call Interrupt Service
```

The operation leaves AX and BX unchanged and returns these values:

CH= Starting scan line	CL=Ending Scan Line
DH=Row	DL=Column

INT 10H Function 05H: Select Active Page

Function 05H lets you select the page that is to be displayed in text or graphics mode. You can create different pages and request alternating between pages. The operation is simply a request that returns no values.

```
        MOV AH, 05H ; Request Active Page
        MOV AL, Page#     ; Page Number
        INT 10H           ; Call Interrupt service
```

INT 10H Function 06H: Scroll Up Screen

This operation for text or graphics mode performs a scroll upward of lines in a specified area of the screen (the active video page). Displayed lines scroll off at the top and blank lines appear at the bottom.

Setting a non-zero value in AL causes that number of lines to scroll up. Load the following registers:

AL=Number of rows (00 for full screen)	CX=Starting row: column
BH=Attribute or pixel value	DX=Ending row: column

The example below would create a window (with its own attributes) of 7 rows and 30 columns, with the top left at 12:25, the top right at 12:54, the bottom left at 18:25 and the bottom right at 18:54:

```
        MOV AX, 0607H     ; Request scroll up 7 lines
        MOV BH, 61H ; Brown BG, blue FG
        MOV CX, 0C19H      ; From 12:25 through
        MOV DX, 1236H  ; 18:54
        INT 10H             ; Call interrupt service
```

INT 10H Function 07H: Scroll Down Screen

For text and graphics mode, scrolling down the screen causes the bottom lines to scroll off and blank lines to appear at the top. Other than the fact that this scroll down, it works the same function 06H, which scrolls up.

AL=Number of rows (00 for full screen)   CX=Starting row: column
BH=Attribute or pixel value   DX=Ending row: column

INT 10H Function 08H: Read Character and Attribute at Cursor

Function 08H can read both a character and its attribute from the Video Display Area in either text of graphics mode. The position of the cursor determines the character that is read. Set the page number in BH, as follows:

```
        MOV AH, 08H       ; Request read char/Attribute
        MOV BH, 00  ; Page Number
        INT 10H           ; Call interrupt service
```

The operation delivers the character to AL and its attribute to AH. In graphics mode, the operation returns 00H for a non-ASCII character.

INT 10H Function 09H: Display Character and Attribute at Cursor

This useful operation displays a specified number of characters in text or graphics mode according to a given attribute. The position of the cursor determines where the character is to display.

AL= ASCII Characters   BL= Attribute / Pixel Value
BH= Page Number   CX= Count

```
MOV AH,09H  ; Request Display
MOV AL,01H  ; Happy Face for Display
MOV BH, 00  ; Page Number 0
MOV BL, 16H ; Blue BG, Black FG
MOV CX, 60  ; No. repeated characters
INT 10H     ; Call Interrupt service
```

The operation does not advance the cursor or respond to the Bell, Carriage Return, Line Feed, Tab characters; instead, it attempts to display them as ASCII characters. In text mode, when the display exceeds the rightmost column, the operation automatically continues the display on the next row at column 00.

INT 10H Function 0AH: Display Character at Cursor

The only difference between function 0AH and 09H is that function 09H sets the attribute whereas function 0AH uses the current value. Here is the code for function 0AH.

```
        MOV AH, 0AH       ; Request Display
        MOV AL, char            ; Character to Display
        MOV BH, page#           ; Page Number
        MOV BL, Value           ; Pixel Value
        MOV CX, Repetition      ;# char repeated
                    INT 10H                 ; Call interrupt
```

The operation returns no values.

INT 10H Function 0CH: Write Pixel Dot

Function 0CH is used to display a selected color (Background and palette) in graphics mode. Set these registers.

AL= Color of the pixel CX=Column
BH=Page number DX=Row

The minimum value for the column or row is 0, and the maximum value depends on the video mode. The following example sets a pixel at column 200, row 50:

```
MOV AH, 0CH       ; Request write dot
MOV AL, 03        ; Color of pixel
MOV BH, 0               ; Page number 0
MOV CX, 200       ; Horizontal x-coordinate (column)
MOV DX, 50        ; Vertical y-coordinate (row)
INT 10H                 ; call interrupt service
```

INT 10H Function 0DH: Read Pixel Dot

This operation, the opposite of function 0CH, reads a dot to determine its color value. Set page number in BH, column in CX, and row in DX. The minimum value for column or row is 0, and the maximum value depends on the video mode.

```
MOV AH, 0DH ; Request read pixel
MOV BH, 0         ; Page # 0
MOV CX, 80  ; X Coordinate
MOV DX, 110 ; Y Coordinate
INT 10H           ; Call Interrupt
```

The operation returns the pixel value to AL.

INT 10H Function 0EH: Display in Teletype Mode

This operation lets you use the monitor as a terminal for simple displays in text and graphics mode, used as follows:

```
MOV AH, 0EH     ; Request Display

MOV AL, 41H     ; Display Capital letter A

MOV BL, Color   ; FG Color

INT 10H         ; Call Interrupt
```

INT 10H Function 10H: Access Palette Registers

This operation provides a number of functions concerned with reading and changing the Palette registers, the Overscan (Border) register, and Video DAC. Load a sub function in AL to specify the activity

Subfunction 03H: Select Background Intensity: This operation let you enable or disable the blinking attribute. Load a code in BL (00H=disable and 01H = enable). The operation access the Attribute Controller's Mode Control Register. With blinking disabled, all 16 Palette registers are available for background

```
MOV AX,1003H      ; Request
MOV BL,00H  ; disable blinking
INT 10H           ; Call interrupt service
```

INT 10 Function 13H: Display Character String

This powerful operation in text and graphics modes displays strings of any length with operations for setting the attribute and moving the cursor. Load ES:BP with the segment:offset address of the string to display. The operation acts on the Backspace, Bell, Carriage Return, and Line Feed control characters, but not Tab.

```
MOV AH, 13H       ; Request Display String
MOV AL, 01        ; Subfunction 00,01,02,03
MOV BH, 00        ; Page Number
MOV BL, 1E        ; Blue BG, Yellow FG
LEA BP, address         ; Address in ES:BP
MOV CX, Length          ; Length Of String
MOV DH, 05H       ; Screen Row
MOV DL, 05        ; Screen Column
INT 10H           ; Call Interrupt
```

The four sub-functions that you set in AL are:
00 Display String And Attribute, no cursor advance
01 Display String And Attribute, advance cursor
02 Display Character then attribute, no cursor advance
03 Display Character then attribute, advance cursor

Using Graphics Mode

Graphics mode uses pixel (picture elements or pels) to generate color patterns. We saw earlier that an attribute in text mode consists of four bits for background and four bits for foreground. The video system interprets pixels in a similar way but, depending on mode, may represent a pixel from one to eight bits.

Setting graphics mode causes the cursor to disappear, although it is still accessible.

The table below provides the common graphics mode.

Mode	**Type**	**Display Area**	**Pages**	**Resolution**	**Colors**
04H	Color	B800	8	320x200	4
05H	Color	B800	8	320x200	4
06H	Color	B800	8	640x200	2
0DH	Color	B800	8	320x200	16
0EH	Color	A000	4	640x200	16
0FH	Monochrome	A000	2	640x350	1
10H	Color	A000	2	640x350	16
11H	Color	A000	1	640x480	2
12H	Color	A000	1	640x480	16

13H	Color	A000	1	320x200	256

- Graphics mode 04H and 05H. Original CGA modes also used by VGA for upward compatibility. One byte represent four pixels (two bits per pixel). Two bits can provide 2^2, or four, different colors at the same time.
- Graphics mode 06H. Original CGA mode used by VGA for upward compatibility. One byte represents eight pixels (one bit per pixel), giving two colors at one time.
- Graphics modes 0DH, 0EH, and 10H. Original EGA modes also used by VGA for upward compatibility. In 16-color graphics mode, each screen character is contained in a 8x8 matrix of pixels. The Video Display Area represent the value with 32 bytes of data; that is, four bits per pixel x 8 x 8 = 256 bits = 32 bytes
- Graphics mode 0FH. Original EGA monochrome mode also used by VGA for upward compatibility.
- Graphics modes 11H and 12H. Modes specifically designed for VGA. These modes are similar to modes 0DH and 0EH in operation. Technically, mode 11H needs only one bit map to represent a screen of pixels. However, where mode 11H has access to two palette colors, mode 12H has access to 16
- Graphics mode 13H. Mode specifically designed for VGA. This mode also uses pixel maps, but represent each pixel by a byte in the video map. The eight bits provide 2^8, or 256, different colors.

Use the INT 10 Function 00H to set graphics mode.

Lesson 2: Keyboard Interrupts

Learning Outcomes:

- LO 4.2.1 Describe how the keyboard would handle input prior to be processes as a useful data.
- LO 4.2.2 Explain the various INT 21H functions and implement it on various keyboard input manipulation
- LO 4.2.3 Explain the various INT 16H functions and implement it on various keyboard input manipulation.

On this part of Module 4, describes many operations for handling keyboard input, some which have specialized uses. Of these operations, INT 21H function 0AH and INT 16H should provide almost all the keyboard operations you'll require. Other topics include the keyboard shift status bytes, scan codes, and the keyboard buffer area.

- ***Shift Status*** – bytes in the BIOS keyboard area enables the program to determine whether the Ctrl, Shift, Alt, keys are pressed.
- ***Scan Code*** – A unique number assigned to each key on the keyboard that enables the system to identify the source of a pressed key and enables a program to check whether the pressed key is an extended function such as Home, PageUp, or Arrow.
- ***Keyboard Buffer Area*** – provides space in memory for you to type ahead before the program actually request input.

The keyboard provides basic types of keys

1. **Standard Characters** – consists of letters A-Z, numbers 0-9, some characters %, $ and #.
2. **Extended Function Keys**
   - Program function keys, such as F1 and Shift+F1

   - Numeric keypad keys with NumLock, toggled off: Home, End, Arrow, Del, Ins, PageUp, and PageDown, and the duplicate keys for them on the extended keyboard
   - ALT + alphabetics, ALT+ program-function keys.
3. **Special Keys** - Alt, Ctrl and Shift which normally work in association of other keys as well as Caps Lock, Num Lock, Scroll Lock, which indicates a condition. BIOS does not deliver these keystrokes as ASCII characters to the program. Instead, BIOS treats these differently from other keys by updating their current state in the shift status bytes in the BIOS keyboard data areas.

The original PC with its 83 keys suffered from a short-sighted design decision that caused keys on the so-called numeric keypad to perform two actions. Thus numbers share keys with Home, End, Arrows, Del,Ins,PageUp, and PageDn, with the numlock toggling between them. To overcome the problems caused by this layout, designers produced and enhanced keyboard with 101 keys and subsequently 104 keys for Windows. Of the 18 added keys,, only F11 and F12 provide a new function; the rest duplicate the function of keys on the original keyboard.

BIOS Keyboard Data Areas

The BIOS Data Area at segment 40[0]H in low memory contains a number of useful data items. These include two Keyboard Data Areas that indicate the current status of the control keys. Keyboard Data Areas that indicate the current status of the control keys. Keyboard Data Areas contains two bytes. The first byte is at 40:17H, where bits set to 1 indicate the following:

First Byte 40:17H

Bit	Action	Bit	Action
7	Insert Active	3	Right Alt Pressed
6	CapsLock State Active	2	Right Ctrl Pressed
5	NumLock State Active	1	Left Shift Pressed
4	Scroll Lock State Active	0	Right Shift Pressed

You may use INT 16H function 02H, to check the values. “Pressed” means that the user is currently holding down the key; releasing the keys causes BIOS to clear the bit value.

Second Byte 40:18H

For the second byte of Keyboard Data Area 1 at 40:18H, bits set to indicate the following

Bit	Action	Bit	Action
7	Insert Pressed	3	Caps/NumLock Active
6	CapsLock Pressed	2	SysReq Pressed
5	NumLock Pressed	1	Left Alt Pressed
4	Scroll Lock Pressed	0	Left Ctrl Pressed

The Keyboard Buffer

The BIOS Data Area at location 40:1EH contains the keyboard buffer. This feature allows you to type up to 15 characters even before a program requests keyboard input. When you press a key, the keyboard processor automatically the key’s scan code (its unique assigned number) and requests BIOS INT 09H

In simple terms, the INT 09H routine gets the scan code from the keyboard , converts it to an ASCII character, and delivers it to the keyboard buffer area. Subsequently, INT 16H reads the character from the buffer and delivers it to your program. Your program need never request INT 09H because the processor performs it automatically.

Interrupt 21H Functions

There are various INT 21H services that handle keyboard input. All of these operations require a function code in AH and accept only one input character. In the discussion that follows, the term to “respond to Ctrl+Break request” means that the system terminates the program if the user presses the Ctrl+Break or Ctrl+C keys together.

INT 21H Function 01H:Keyboard Input with echo

This operation accepts a character from the keyboard buffer or, if none is present, waits for keyboard entry.

Example:

```
MOV AH, 01H ; Request Keyboard display
INT 21H           ; Call interrupt service
```

This operation returns 2 Status codes to AL. AL= Non-zero means that standard ASCII characters are present and echoes it on the screen.AL=Zero means extended function keys such as Home or F1, and AH still contains the original function. The operation handles extended functions clumsily, attempting to echo them on the screen. And to get the scan code for the function key in AL, you immediately have to repeat the INT 21H operation The operation respond to Ctrl+Break request.

INT 21H Function 07H: Direct keyboard input without echo

This operation works like function 01H, except that the entered character does not echo on the screen and does not respond to CTRL+Break request. It could be used to key in a password that is to be invisible.

Example:

```
MOV AH, 07H; Request Keyboard display
INT 21H     ; Call interrupt service
```

INT 21H Function 08H: Keyboard Input without echo

This operation works like function 01H except that the entered character does not echo on the screen.

Example

```
MOV AH, 08H ; Request Keyboard display
INT 21H           ;Call interrupt service
```

INT 21H Function 0AH: Buffered keyboard input

INT 21H function 0AH for accepting data from the keyboard is particularly powerful. The input area for keyed-in characters requires a parameter-list containing specific fields that the INT operation is to process. (Comparable to a record or structure on a high level language). First, the operation needs to know the maximum number of input characters. The purpose is to prevent users from keying in too many characters; if so, the operation the speaker and does not accept further input. Second, the operation delivers to the parameter list the number of bytes actually entered.

The Parameter List Consists:

1. Provide the Name of the parameter list in form of label.
2. The First byte of the parameter list contains your maximum number of characters in one byte field format (0-FFH)
3. The second byte is for operation to store in binary format the actual number of characters
4. The third byte begins a field that is to contain the typed characters form left to right.

```
PARA_LIST LABEL BYTE; Start a parameter list
      MAX_LEN DB 20           ; Maximum number input characters
```

```
ACT_LEN DB ?            ; Actual number of input characters
KB_DATA DB 20 DUP(‘ ‘);Characters entered in Kb
```

To request keyboard input, set function 0AH in AH load the parameter list into DX and issue INT 21H.
Example:

```
MOV AH,0AH        ; Request keyboard input
LEA DX, PARA_LIST       ; Load address of parameter list
INT 21H                 ; Call interrupt service
```

The INT operation waits for a user to type characters and checks that the number does not exceed the maximum of 20. The operation echoes each typed character onto the screen where the cursor is situated and advances the cursor. The <enter> key on the keyboard would signal the end of a keyboard entry. The operation also transfers the Enter character (0DH) to the input field KB_DATA, but does not count its entry in the actual length. If you key in a name such as Wilson+<enter>, the parameter list appears like this:

ASCII	20	W	i	l	s	o	n	#		….
HEX	14	06	57	69	6C	6F	6E	0D	20	…

The operation delivers the length of the input name, 06H, into the second byte of the parameter list, named ACT_LEN in the example. The enter character (0DH) is at KB_DATA+6. (The # sign represents 0DH since this character is non-printable). Given that the maximum length of 20 includes the 0DH, the user may type only up to 19 characters.

This operation accepts and acts on the Backspace character, but does not add it to the count. Other than Backspace, the operation does not accept more than the maximum number of characters.

INT 21 Function 0BH: Check Keyboard status

This operation returns FFH in AL if an input character is available in the keyboard buffer area and 00H if no character is available. Note that the operation does not expect the user to press a key; rather it simply checks the buffer

INT 21H Function 0CH: Clear Keyboard buffer and invoke function

You may use this operation in association with function 01H, 06H, 07H, 08H, or 0AH. Load the function in AL.
Example:

```
MOV AH, 0CH       ; Request keyboard function
MOV AL, function  ; Required Function
INT 21H                 ; Call Interrupt
```

This operation clears the keyboard buffer, executes the function AL, and accepts a character according to the function request.

Interrupt 16H Functions

INT 16H is the basic BIOS keyboard operation used extensively by software developers and provides the following services according to a function code you load in AH. Note functions 03H and 12H are not supported by the emulator.

INT 16H Function 00H: Get Keystroke from keyboard

This interrupt would wait for an input from the user, and returns the following:
AH = BIOS scan code.
AL = ASCII character.
Example

```
MOV AH,00H  ; Request get keystroke
INT 16H           ; Call interrupt function
CMP AH,1EH  ; Check if compare BIOS scan code with ‘A’ scan code.
JE EXIT           ; If the same to exit
```

If the keystroke is present, it is then removed from the keyboard buffer.

INT 16H Function 01H: Check Keystroke from the keyboard buffer

This interrupt would only check the keyboard buffer if a character is present, and not do any input. And after that it would return the following:
ZF=1 if keystroke is not available
ZF=0 if keystroke is available
AH=BIOS Scan code
AL=ASCII character

INT 16H Function 03H: Set Typematic Mode

When you hold down a key for more than ½ second, the keyboard enters typematic mode and automatically repeats the character. To change this:

```
MOV AH, 03H             ; Set typematic repeat rate
MOV AL, 05H             ; Required subfunction
MOV BH, repeat-delay    ; Delay before start
MOV BL, repeat-rate            ; Speed Repetition
INT 16H
```

Repeat delay (BH) – 0 = ¼ sec, 1= ½ sec, 2=3/4 sec. Repeat rate (BL) – 0 (fastest) through 31 (Slowest)

INT 16H Function 12H: Return Keyboard Shift Status

This operation delivers the keyboard status byte from BIOS Data Area 1 at location 40:17H to AL and the byte from 40:18 to AH. The example below tests AL to determine whether the Left Shift (bit 1) or Right Shift (bit 0) keys are pressed:
Example:

```
MOV AH, 12H       ; Request Shift Status
INT 16H                 ; Call interrupt service
AND AL,03H        ; Left or Right shift pressed
JZ Exit                 ; yes…..
```

For the status byte in AH, 1-bits mean the following:

Bit	**Key**	**Bit**	**Key**
7	SysReq pressed	3	Right Alt pressed
6	Caps Lock Pressed	2	Right Ctrl Pressed

5	Num Lock Pressed	1	Left Alt Pressed
4	Scroll Lock Pressed	0	Left Ctrl Pressed

Keyboard Scan Codes and ASCII codes

For each category the column shows the format for a normal key (not combined with another key) and formats when the key is combined with the Shift, Ctrl, and Alt keys. Under columns headed Normal, Shift, Ctrl, and Alt are two hex bytes as they appear when keyboard operation delivers to the AH and AL registers.

LETTERS	**NORMAL**		**SHIFT**		**CTRL**		**ALT**	
a and A	1E	61	1E	41	1E	1	1E	0
b and B	30	62	30	42	30	2	30	0
c and C	2E	63	2E	43	2E	3	2E	0
d and D	20	64	20	44	20	4	20	0
e and E	12	65	12	45	12	5	12	0
f and F	21	66	21	46	21	6	21	0
g and G	22	67	22	47	22	7	22	0
h and H	23	68	23	48	23	8	23	0
i and I	17	69	17	49	17	9	17	0
j and J	24	6A	24	4A	24	0A	24	0
k and K	25	6B	25	4B	25	0B	25	0
l and L	26	6C	26	4C	26	0C	26	0
m and M	32	6D	32	4D	32	0D	32	0
n and N	31	6E	31	4E	31	0E	31	0
o and O	18	6F	18	4F	18	0F	18	0
p and P	19	70	19	50	19	10	19	0
q and Q	10	71	10	51	10	11	10	0
r and R	13	72	13	52	13	12	13	0
s and S	1F	73	1F	53	1F	13	1F	0

t and T	14	74	14	54	14	14	14	0
u and U	16	75	16	55	16	15	16	0
v and V	2F	76	2F	56	2F	16	2F	0
w and W	11	77	11	57	11	17	11	0
x and X	2D	78	2D	58	2D	18	2D	0
y and Y	15	79	15	59	15	19	15	0
z and Z	2C	7A	2C	5A	2C	1A	2C	0
Spacebar	39	20	39	20	39	20	39	0

FUNCTION KEYS	**NORMAL**		**SHIFT**		**CTRL**		**ALT**	
F1	3B	0	54	0	5E	0	68	0
F2	3C	0	55	0	5F	0	69	0
F3	3D	0	56	0	60	0	6A	0
F4	3E	0	57	0	61	0	6B	0
F5	3F	0	58	0	62	0	6C	0
F6	40	0	59	0	63	0	6D	0
F7	41	0	5A	0	64	0	6E	0
F8	42	0	5B	0	65	0	6F	0
F9	43	0	5C	0	66	0	70	0
F10	44	0	5D	0	67	0	71	0
F11	85	0	87	0	89	0	8B	0
F12	86	0	88	0	8A	0	8C	0

TOP ROW	**NORMAL**		**SHIFT**		**CTRL**		**ALT**	
` and ~	29	60	29	7E			29	0

1 and !	2	31	2	21			78	0
2 and @	3	32	3	40	3	0	79	0
3 and #	4	33	4	23			7A	0
4 and $	5	34	5	24			7B	0
5 and %	6	35	6	25			7C	0
6 and ^	7	36	7	5E	7	1E	7D	0
7 and &	8	37	8	26			7E	0
8 and *	9	38	9	2A			7F	0
9 and (	0A	39	0A	38			80	0
0 and)	0B	30	0B	29			81	0
- and _	0C	2D	0C	5F	0C	1F	82	0
= and +	0D	3D	0D	2B			83	0

Operation Keys	**Normal**		**Shift**		**Ctrl**		**Alt**	
Esc	1	1B	1	1B	1	1B	1	0
Backspace	0E	8	0E	8	0E	7F	0E	0
Tab	0F	9	0F	0	0F	0	0F	0
Enter	1C	0D	1C	0D	1C	0A	1C	0

Punctuation	**Normal**		**Shift**		**Ctrl**		**Alt**	
[and {	1A	5B	1A	7B	1A	1B	1A	00
] and }	1B	5D	1B	7D	1B	1D	1B	00
; and :	27	3B	27	3A			27	00
‘ and “	28	5C	28	22			28	00
\ and |	2B	27	2B	7C	2B	1C	2B	00
, and <	33	2C	33	3C			33	00

. and >	34	2E	34	3E			34	00
/ and ?	35	2F	35	3F			35	00

Key	**Normal**		**Shift**		**Ctrl**		**Alt**	
Slash (/)	E0	2F	E0	2F	95	00	A4	00
Del	53	E0	53	E0	93	E0	A3	00
Down Arrow	50	E0	50	E0	91	E0	A0	00
End	4F	E0	4F	E0	75	E0	9F	00
Enter	E0	0D	E0	0D	E0	0A	A6	00
Home	47	E0	47	E0	77	E0	97	00
Ins	52	E0	52	E0	92	E0	A2	00
Left Arrow	4B	E0	4B	E0	73	E0	9B	00
Page Down	51	E0	51	E0	76	E0	A1	00
Page Up	49	E0	49	E0	84	E0	99	00
Right Arrow	4D	E0	4D	E0	74	E0	9D	00
Up Arrow	48	E0	48	E0	8D	E0	98	00