Interrupts

CO 2103 Assembly Language

- This is a compilation of slides on INT instructions.
- The slides are mostly taken from the slides used by my colleague Dr Md Mahmud Hasan, who taught this module before me.

Interrupts

- Hardware interrupts
  - occur as a response to a hardware device
  - routed through the Intel 8259 Interrupt Controller
- Software interrupts
  - calls to operating system functions, located in BIOS and resident portion of DOS
  - activated by the INT instruction

INT

Interrupts

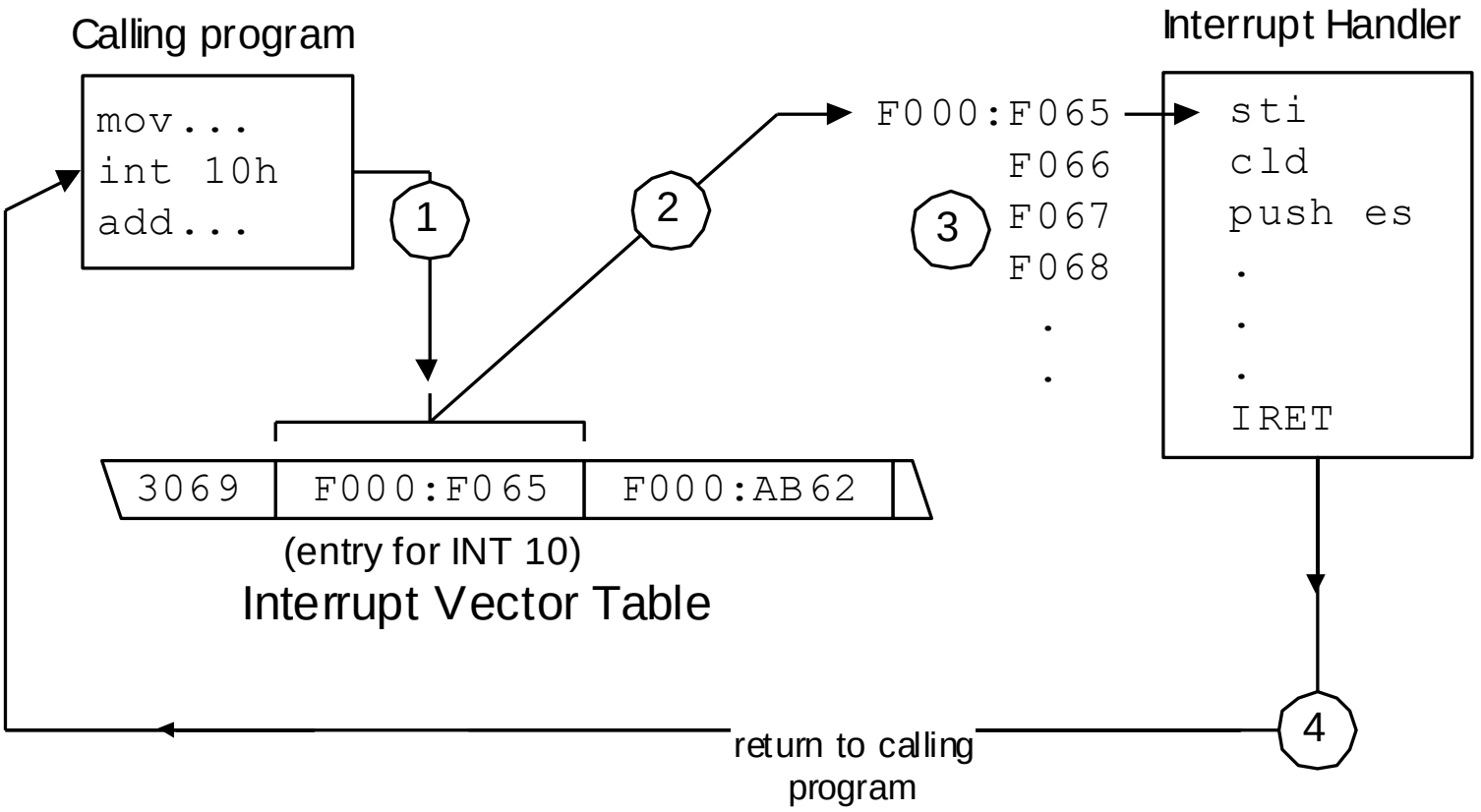
INT

- The INT instruction is used to cause a software interrupt
  - An interrupt interrupts the current program and executes a subroutine, eventually returning control to the original program
  - Interrupts may be caused by hardware or software
- int *interrupt_number* ;software interrupt

INT Instruction

- The INT instruction is always followed by a hexadecimal number that identifies its type
- Common examples:
  - INT 10h - video BIOS
  - INT 14h - Serial I/O
  - INT 16h - keyboard BIOS
  - INT 17h - printer services
  - INT 1Ah - Time of day
  - INT 1Ch - User timer
  - INT 21h - DOS services

Interrupt Vectoring Process



DOS Function Calls (INT 21h)

- The INT 21h instruction activates a DOS function call
- The function number (0-255) is placed in the AH register before invoking INT 21h
- Some functions require that you assign values to certain registers before invoking INT 21h
- Some functions return values in registers

Output to Monitor

- DOS Interrupts : interrupt 21h
  - This interrupt invokes one of many support routines provided by DOS
  - The DOS function is selected via AH
  - Other registers may serve as arguments
- AH = 2, DL = ASCII of character to output
  - Character is displayed at the current cursor position, the cursor is advanced, AL = DL

Output a String

- Interrupt 21h, function 09h
  - DX = offset to the string (in data segment)
  - The string is terminated with the '$' character
- To place the address of a variable in DX, use one of the following
  - lea DX,theString ;load effective address
  - mov DX, offset theString ;immediate data

Input a Character

- Interrupt 21h, function 01h
- Filtered input with echo
  - This function returns the next character in the keyboard buffer (waiting if necessary)
  - The character is echoed to the screen
  - AL will contain the ASCII code of the non-control character
    - AL=0 if a control character was entered

Additional Input Functions

- 06h : Direct input, no waiting
- 07h : Direct input, no Ctrl-Break
- 08h - Direct input with Ctrl-Break
- 0Ah - Buffered input
- 0Bh - Get input status
- 0Ch - Clear input buffer, invoke input function
- 3Fh - Read from file or device

Direct Input Functions

- No filtering
- No on-screen echo
- Function 6
  - requires DL = 0FFh
  - Does not wait for a character to be input
  - ZF set if no character is waiting
- Function 7
  - Ctrl-Break while waiting for input will not terminate program
- Function 8
  - Ctrl-Break will cancel the input request and terminate the program

Buffered Input

- A buffer of up to 255 characters is used
- Backspace can be used to delete characters
- Other control keys are filtered
- Characers are echoed
- Enter key terminates the input phase
  - DX must contain the offset to a buffer area

```
buffer  db   m, ?, m dup (?)
```

Get Input Status

- AL is set to 0FFh if a character is waiting in the keyboard buffer, AL = 0 otherwise
- The keyboard buffer is a 15-character queue holding keystrokes not yet processed by an application
  - DOS manages this via hardware interrupts
  - A "beep" indicates the buffer is full
    - additional keystrokes are ignored when full

Simple Console I/O

```
mov   ah,1          ; single character input
int   21h

mov   ah,2          ; single character output
mov   dl,'A'
int   21h

mov   ah,9                  ; string output
mov   dx,offset message
int   21h
```

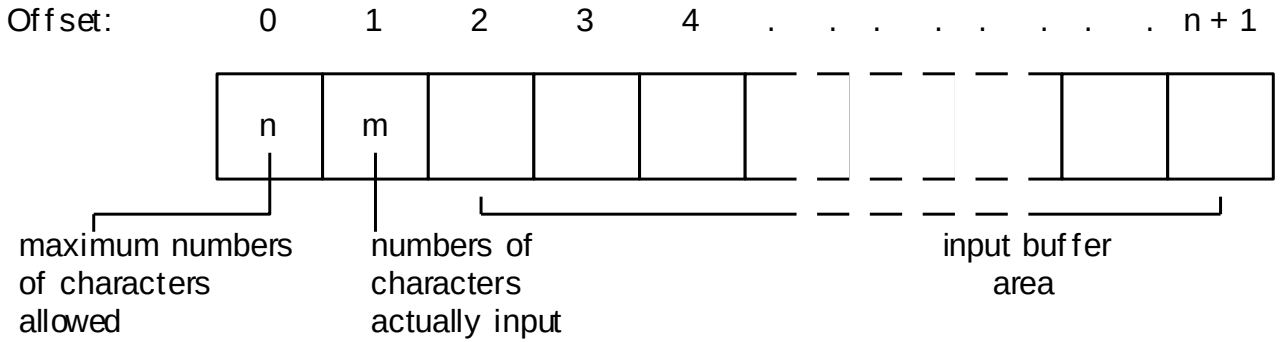
INT 21h: Standard Input

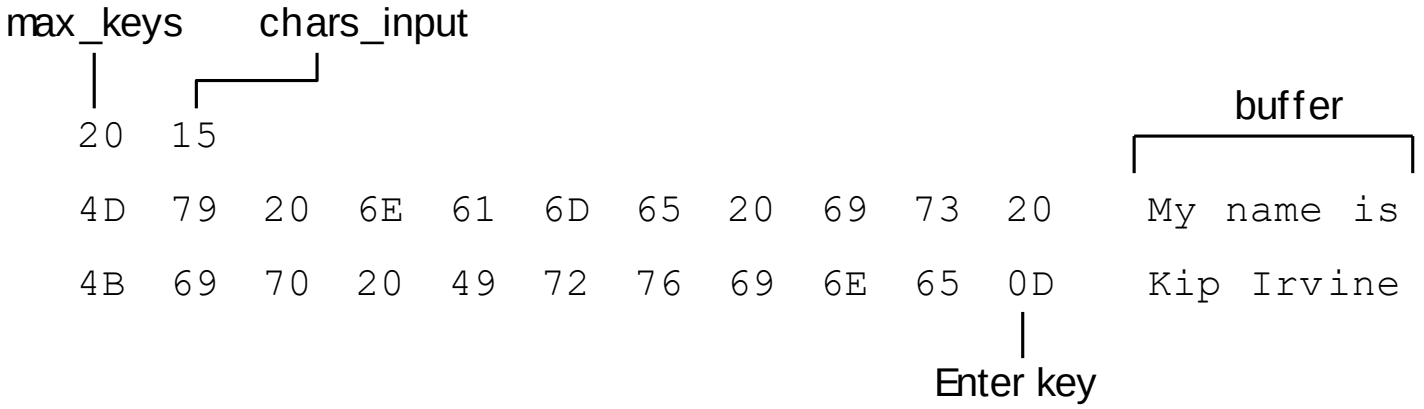
- 01h Filtered Input With Echo
- 06h Direct Input Without Waiting
- 07h Direct Input, No Ctrl-Break
- 08h Direct Input with Ctrl-Break
- 0Ah Buffered Input
- 0Bh Get Input Status
- 0Ch Clear Input Buffer, Invoke Input Function
- 3Fh Read From File or Device

Comparison of Standard Input

DOS Function Number	1	6	7	8
Waits for keystroke?	Y	N	Y	Y
Echoes character?	Y	N	N	N
Ctrl-Break recognized?	Y	N	N	Y
Filters control characters?	Y	N	N	N

Keyboard Parameter Record (Function 0Ah)





3Fh: Read from File or Device

When the user presses Enter at the end of the input, two bytes (0Dh,0Ah) are appended to the string in the input buffer and the count in AX includes the extra characters.

```
buffer db 127 dup(0)
.
.
mov  ah,3Fh                 ; read from file/device
mov  bx,0                   ; device = keyboard
mov  cx,127                 ; request 127 bytes maximum
mov  dx,offset buffer
int  21h                    ; AX = number chars typed + 2
```

40h: Write to File or Device

```
buffer db 127 dup(0)
count  dw ?
.
mov  ah,40h                ; read from file/device
mov  bx,1                  ; device = console
mov  cx,count              ; number of chars to write
mov  dx,offset buffer
int  21h
```

2Ah: Get Date, 2Bh: Set Date

```
mov   ah,2Ah
int   21h
mov   year,cx
mov   month,dh
mov   day,dl
mov   dayOfWeek,al
```

```
mov   ah,2Bh
mov   cx,year
mov   dh,month
mov   dl,day
int   21h
cmp   al,0
jne   badDate
```

Requires administrator privileges under Windows NT.

2Ch: Get Time, 2Dh: Set Time

```
mov   ah,2Ch
int   21h
mov   hours,ch
mov   minutes,cl
mov   seconds,dh
```

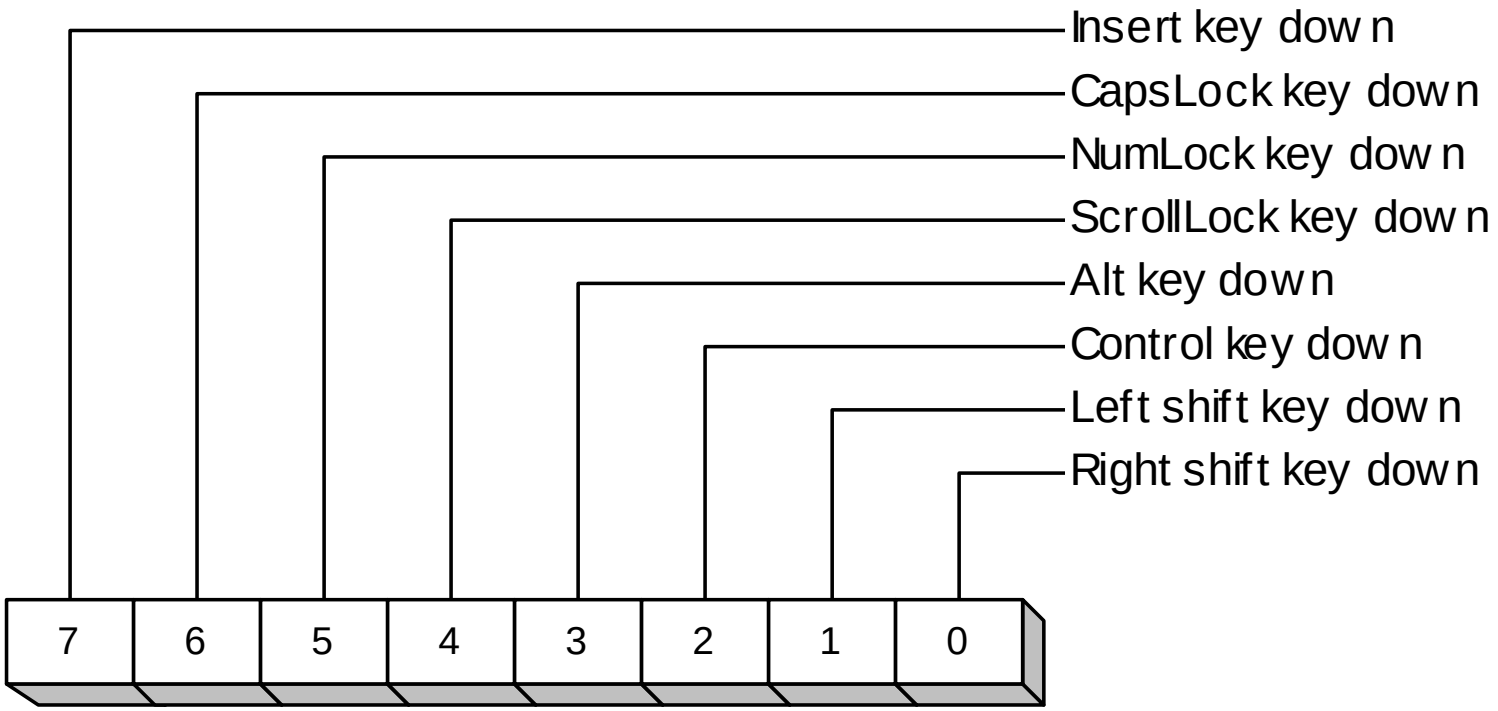
```
mov   ah,2Dh
mov   ch,hours
mov   cl,minutes
mov   dh,seconds
int   21h
cmp   al,0
jne   badTime
```

Requires administrator privileges under Windows NT.

INT 16h BIOS Keyboard Input

AH	Description
03h	*Set Typematic Repeat Rate*. Call with AH = 3, AL = 5, BH = repeat delay, BL = repeat rate. The delay values in BH are: (0 = 250 ms; 1 = 500 ms; 2 = 750 ms; 3 = 1000ms). The repeat rate in BL varies from 0 (fastest) to 1Fh (slowest).
05h	*Push Key into Buffer*. Pushes a keyboard character and corresponding scan code into the keyboard typeahead buffer. Call with AH = 5, CH = scan code, and CL = character code. If the typeahead buffer is already full, the Carry flag will be set, and AL = 1.
10h	*Wait for Key*. If a keystroke is waiting, its scan code is returned in AH and its character code is returned in AL. If no key is waiting, the routine waits in a loop for a key to be pressed.
11h	*Check Keyboard Buffer*. Examines the keyboard typeahead buffer to see if a key is waiting; if one is, this function returns the scan code in AH and the character code in AL, and clears the Zero flag. Otherwise, ZF = 1. The keystroke is not removed from the buffer.
12h	*Get Keyboard Flags*. Shows the keyboard status byte, which is bit mapped. See Figure 5.

Keyboard Status Byte



Keyboard Input Using INT 16h

Use INT 16h to input any key, including function keys, arrow keys, and other extended keys.

```
mov  ah,10h   ; wait for key
int  16h      ; AH=scan code, AL=ASCII code
```

Keyboard Input Using INT 16h

INT 16h function 11h detects the presence of a key in the keyboard typeahead buffer. The following loop uses a conditional jump (JNZ), explained in Chapter 6.

```
L1:
   mov  ah,11h          ; key waiting?
   int  16h
   jnz  keyWaiting   ; yes: process it
   jmp  L1           ; no: continue loop

keyWaiting:
  mov scanCode,ah
  mov ASCIICode,al
```

Keyboard Scan Codes

- A keyboard scan code is a unique 8-bit binary number associated with a particular keyboard key.
- A list of frequently used codes is inside the front cover of the book. Here are samples:

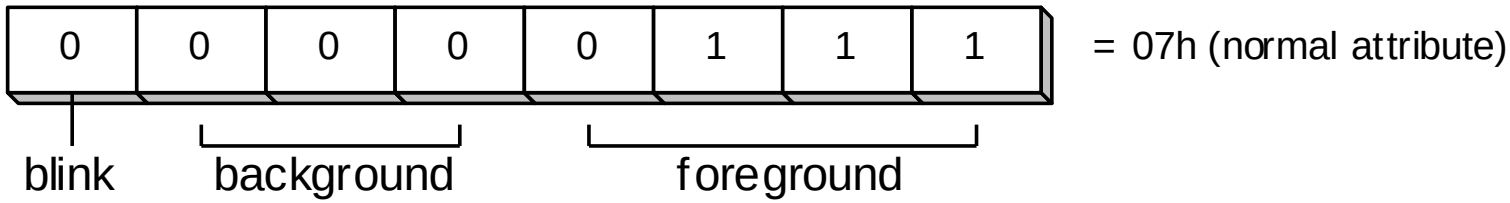
F1 function key	3Bh	Home	47h
F2 function key	3Ch	End	4Fh
F3 function key	3Dh	PgUp	49h
F4 function key	3Eh	PgDn	51h

Common ASCII Control Characters

Hexadecimal	Decimal	Description
08	08	Backspace
09	09	Horizontal tab
0A	10	Line feed
0C	12	Form feed (printer only)
0D	13	Carriage return (Enter key)
1B	27	Escape

Video Attribute Layout

(MSDOS mode only)



If your program is running in an MS-DOS window under Windows/NT, your background color is stored in bits 4-7 of the attribute bit, and blinking is disabled.

3-bit Background Colors

The following background colors are used only when running in full-screen mode or in pure MSDOS mode (by rebooting).

Binary	Hex	Color
000	00	black
001	01	blue
010	02	green
011	03	cyan
100	04	red
101	05	magenta
110	06	brown
111	07	white

4-bit Foreground Colors

Binary	Hex	Color	Binary	Hex	Color
0000	00	black	1000	08	gray
0001	01	blue	1001	09	light blue
0010	02	green	1010	0A	light green
0011	03	cyan	1011	0B	light cyan
0100	04	red	1100	0C	light red
0101	05	magenta	1101	0D	light magenta
0110	06	brown	1110	0E	yellow
0111	07	white	1111	0F	bright white

4-bit Background Colors

Binary	Hex	Color	Binary	Hex	Color
0000	00	black	1000	08	gray
0001	01	blue	1001	09	light blue
0010	02	green	1010	0A	light green
0011	03	cyan	1011	0B	light cyan
0100	04	red	1100	0C	light red
0101	05	magenta	1101	0D	light magenta
0110	06	brown	1110	0E	yellow
0111	07	white	1111	0F	bright white

Table 9. Listing of INT 10h Functions (1 of 2)

Function Number (in AH)	Description
0	*Set Video Mode*. Set the video display to monochrome, text, graphics, or color mode.
1	*Set Cursor Lines*. Identify the starting and ending scan lines for the cursor.
2	*Set Cursor Position*. Position the cursor on the screen.
3	*Get Cursor Position*. Get the cursor's screen position and size.
4	*Read Light Pen*. Read the position and status of the light pen.
5	*Set Display Page*. Select the video page to be displayed.
6	*Scroll Window Up*. Scroll a window on the current video page upward, replacing scrolled lines with blanks.
7	*Scroll Window Down*. Scroll a window on the current video page downward, replacing scrolled lines with blanks.
8	*Read Character and Attribute*. Read the character and its attribute at the current cursor position.
9	*Write Character and Attribute*. Write a character and its attribute at the current cursor position.
0Ah	*Write Character*. Write a character only (no attribute) at the current cursor position.
0Bh	*Set Color Pallete*. Select a group of available colors for the video adapter.
0Ch	*Write Graphics Pixel*. Write a graphics pixel when in graphics mode.
0Dh	*Read Graphics Pixel*. Read the color of a single graphics pixel at a given location.
0Eh	*Write Character*. Write a character to the screen and advance the cursor.
0Fh	*Get Video Mode*. Get the current video mode.
11h	*Load Default ROM Font*. While in text mode, load one of three default ROM fonts and display on the EGA and VGA displays.

Table 9. Listing of INT 10h Functions (2 of 2)

0	*Set Video Mode*. Set the video display to monochrome, text, graphics, or color mode.
1	*Set Cursor Lines*. Identify the starting and ending scan lines for the cursor.
2	*Set Cursor Position*. Position the cursor on the screen.
3	*Get Cursor Position*. Get the cursor's screen position and size.
4	*Read Light Pen*. Read the position and status of the light pen.
5	*Set Display Page*. Select the video page to be displayed.
6	*Scroll Window Up*. Scroll a window on the current video page upward, replacing scrolled lines with blanks.
7	*Scroll Window Down*. Scroll a window on the current video page downward, replacing scrolled lines with blanks.
8	*Read Character and Attribute*. Read the character and its attribute at the current cursor position.
9	*Write Character and Attribute*. Write a character and its attribute at the current cursor position.
0Ah	*Write Character*. Write a character only (no attribute) at the current cursor position.
0Bh	*Set Color Pallete*. Select a group of available colors for the video adapter.
0Ch	*Write Graphics Pixel*. Write a graphics pixel when in graphics mode.
0Dh	*Read Graphics Pixel*. Read the color of a single graphics pixel at a given location.
0Eh	*Write Character*. Write a character to the screen and advance the cursor.
0Fh	*Get Video Mode*. Get the current video mode.
11h	*Load Default ROM Font*. While in text mode, load one of three default ROM fonts and display on the EGA and VGA displays.

INT 10h (06h) Scroll Window Up

When you scroll a window up, existing lines of text are moved upward and one or more blank lines are created. You can assign a color to the blank lines.

```
mov   ah,6          ; scroll window up
mov   al,5          ; scroll 5 lines
mov   ch,0          ; upper left row
mov   cl,0          ; upper left column
mov   dh,24         ; lower right row
mov   dl,79         ; lower right column
mov   bh,7          ; attribute for blank lines
int   10h           ; call BIOS
```

Scroll (clear) Entire Window

If you set AL to zero, all lines in the window are scrolled. This clears the window.

```
mov   ah,6          ; scroll window up
mov   al,0          ; entire window
mov   ch,0          ; upper left row
mov   cl,0          ; upper left column
mov   dh,24         ; lower right row
mov   dl,79         ; lower right column
mov   bh,7          ; attribute for blank lines
int   10h           ; call BIOS
```

INT 10h (07h) Scroll Window Down

The following scrolls all lines within a window in the downward direction by one row. The blank line's attribute is blue text on a white background (11110001):

```
mov   ah,7        ; scroll window down
mov   al,1        ; scroll one row
mov   ch,0        ; upper left row
mov   cl,0        ; upper left column
mov   dh,24       ; lower right row
mov   dl,79       ; lower right column
mov   bh,0F1h     ; blank line's attribute
int   10h         ; call BIOS
```

INT 10h (2h) Set Cursor Position, and INT 10h (08h) Read Character and Attribute

```
locate:
    mov   ah,2              ; set cursor position
    mov   bh,0              ; on video page 0
    mov   dx,0501h          ; at row 5,column 1
    int   10h
getchar:
    mov   ah,8              ; read char/attribute
    mov   bh,0              ; on video page 0
    int   10h
    mov   char,al           ; save the character
    mov   attrib,ah         ; save the attribute
```

Advance the Screen Cursor

Strategy: Get the current cursor position, add 1 to DL, and set the new cursor position.

```
AdvanceCursor proc
   pusha
   mov  ah,3        ; get cursor position
   mov  bh,0
   int  10h
   inc  dl          ; increment column
   mov  ah,2        ; set cursor position
   int  10h
   popa
   ret
AdvanceCursor endp
```

INT 10h (09h) Write Character and Attribute

This function does not advance the cursor, so you have to do that separately

```
mov   ah,9        ; write character and attribute
mov   al,0Ah      ; ASCII character 0Ah
mov   bh,0        ; video page 0
mov   bl,2        ; color (attribute) = green
mov   cx,1        ; display it one time
int   10h
```

Example: Write a Color String

```
string db "ABCDEFGHIJKLMOP"
count  = ($-string)
color  db 1
    .
    mov  cx,count
    mov  si,offset string
L1:
    push  cx          ; save loop counter
    mov   ah,9        ; write character and attribute
    mov   al,[si]     ; character to display
    mov   bh,0        ; video page 0
    mov   bl,color    ; get the color
    mov   cx,1        ; display it one time
    int   10h
    call  AdvanceCursor
    inc   color       ; next color
    inc   si          ; next character position
    pop   cx          ; restore loop counter
    Loop  L1
```

Table 10. Direct Video Procedures in the Link Library

Under Windows 2000, you can see the output of these functions while debugging in CodeView, but if you run the program in a Command window, they do not generate any output.

Procedure	Description
Set_videoseg	Set the current video segment address. The default is B800h, which is appropriate for a color display, including all types of VGA. The alternative is B000h, the default for the older monochrome display. Input: AX contains the segment value.
Writechar_direct	Write a single character to VRAM. Input: AL = character, AH = attribute, DH/DL = row (0-24) and column (0-79) on screen.
Writestring_direct	Write a null-terminated string to VRAM, all characters in the same color. Input: DS:SI points to the string, AH = attribute, DH/DL = row (0-24) and column (0-79) on screen.