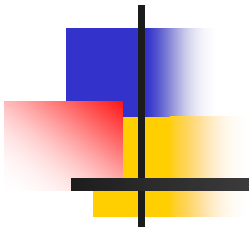
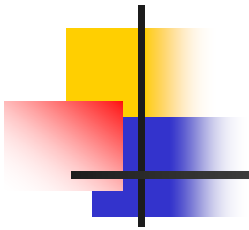
Computer Organization & Assembly Languages



MS-DOS & BIOS-level Programming

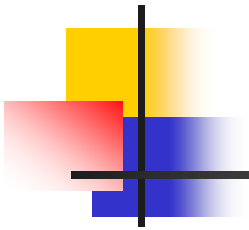
Pu-Jen Cheng

Adapted from the slides prepared by Kip Irvine for the book,
Assembly Language for Intel-Based Computers, 5th Ed



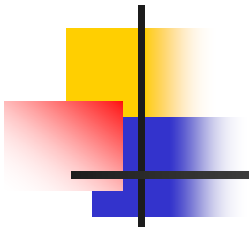
Chapter Overview

- **MS-DOS and the IBM-PC**
- MS-DOS Function Calls (INT 21h)
- Standard MS-DOS File I/O Services



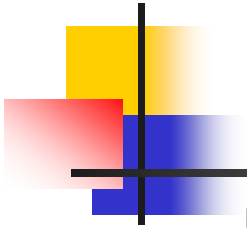
MS-DOS and the IBM-PC

- Real-Address Mode
- MS-DOS Memory Organization
- MS-DOS Memory Map
- Redirecting Input-Output
- Software Interrupts
- INT Instruction
- Interrupt Vectoring Process
- Common Interrupts



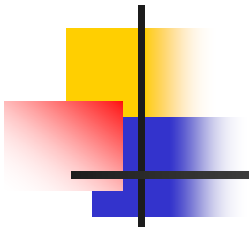
Real-Address Mode

- Real-address mode (16-bit mode) programs have the following characteristics:
  - Max 1 megabyte addressable RAM
  - Single tasking
  - No memory boundary protection
  - Offsets are 16 bits
- IBM PC-DOS: first Real-address OS for IBM-PC
  - Has roots in Gary Kildall's highly successful Digital Research CP/M
  - Later renamed to MS-DOS, owned by Microsoft



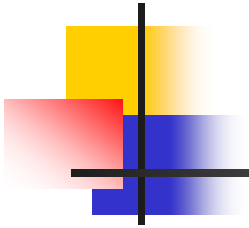
Memory Models

Model	Description
tiny	A single segment, containing both code and data. This model is used by .com programs.
small	One code segment and one data segment. All code and data are near, by default.
medium	Multiple code segments and a single data segment.
compact	One code segment and multiple data segments.
large	Multiple code and data segments.
huge	Same as the large model, except that individual data items may be larger than a single segment.
flat	Protected mode. Uses 32-bit offsets for code and data. All data and code (including system resources) are in a single 32-bit segment.



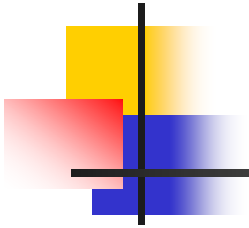
NEAR and FAR Segments

- NEAR segment
  - requires only a 16-bit offset
  - faster execution than FAR
- FAR segment
  - 32-bit offset: requires setting both segment and offset values
  - slower execution than NEAR



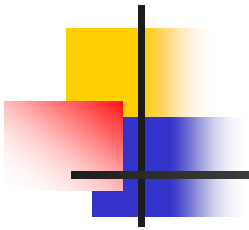
.MODEL Directive

- The .MODEL directive determines the names and grouping of segments
- .model tiny
  - code and data belong to same segment (NEAR)
  - .com file extension
- .model small
  - both code and data are NEAR
  - data and stack grouped into DGROUP
- .model medium
  - code is FAR, data is NEAR



.MODEL Directive

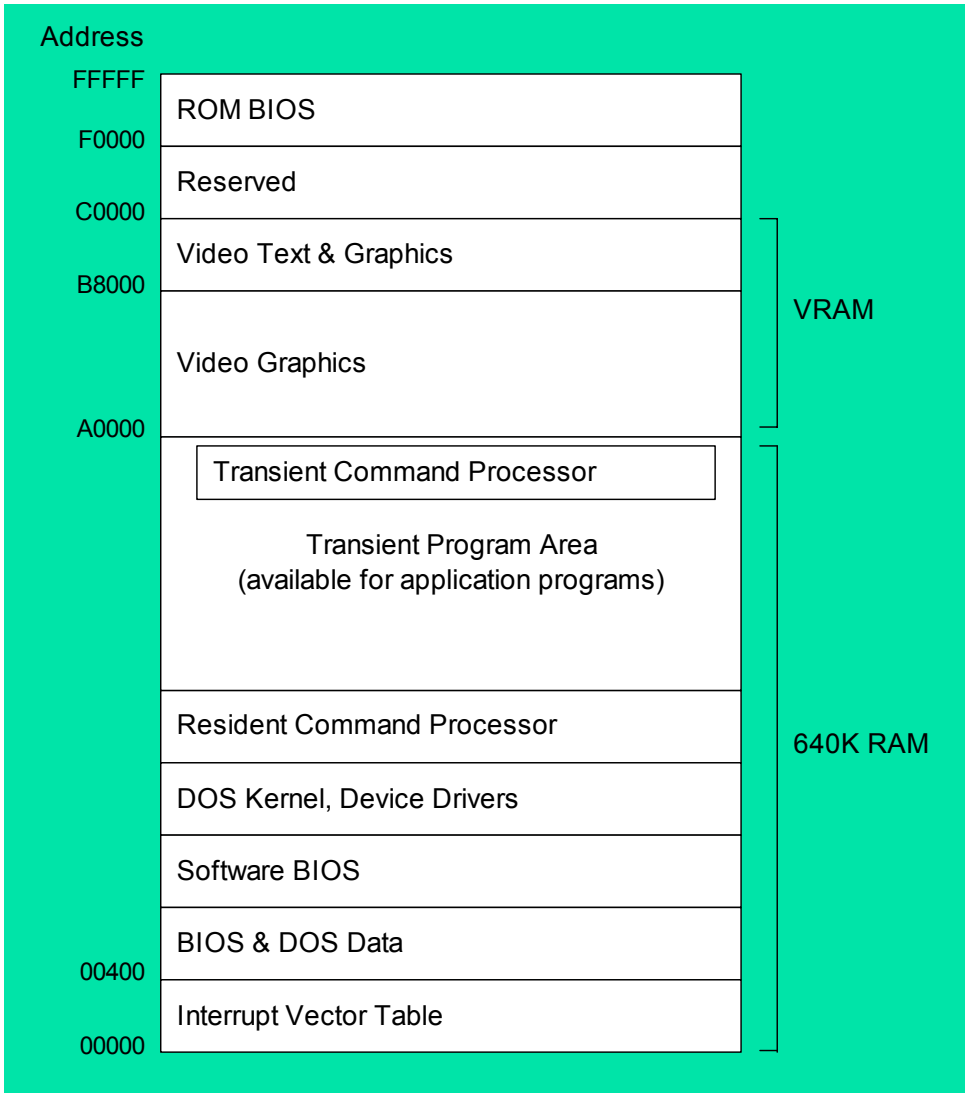
- .model compact
  - code is NEAR, data is FAR
- .model huge & .model large
  - both code and data are FAR
- .model flat
  - both code and data are 32-bit NEAR



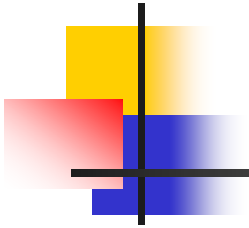
MS-DOS Memory Organization

- Interrupt Vector Table
- BIOS & DOS data
- Software BIOS
- MS-DOS kernel
- Resident command processor
- Transient programs
- Video graphics & text
- Reserved (device controllers)
- ROM BIOS

MS-DOS Memory Map



Redirecting Input-Output (1 of 2)



- Input-output devices and files are interchangeable
- Three primary types of I/O:
  - Standard input (console, keyboard)
  - Standard output (console, display)
  - Standard error (console, display)
- Symbols borrowed from Unix:
  - < symbol: *get input from*
  - > symbol: *send output to*
  - | symbol: pipe output from one process to another
- Predefined device names:
  - PRN, CON, LPT1, LPT2, NUL, COM1, COM2

Redirecting Input-Output (2 of 2)

- Standard input, standard output can both be redirected
- Suppose we have created a program named myprog.exe that reads from standard input and writes to standard output. Following are MS-DOS commands that demonstrate various types of redirection:

```
myprog < infile.txt

myprog > outfile.txt

myprog < infile.txt > outfile.txt
```

Interrupt Vector Table

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

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

Software Interrupts

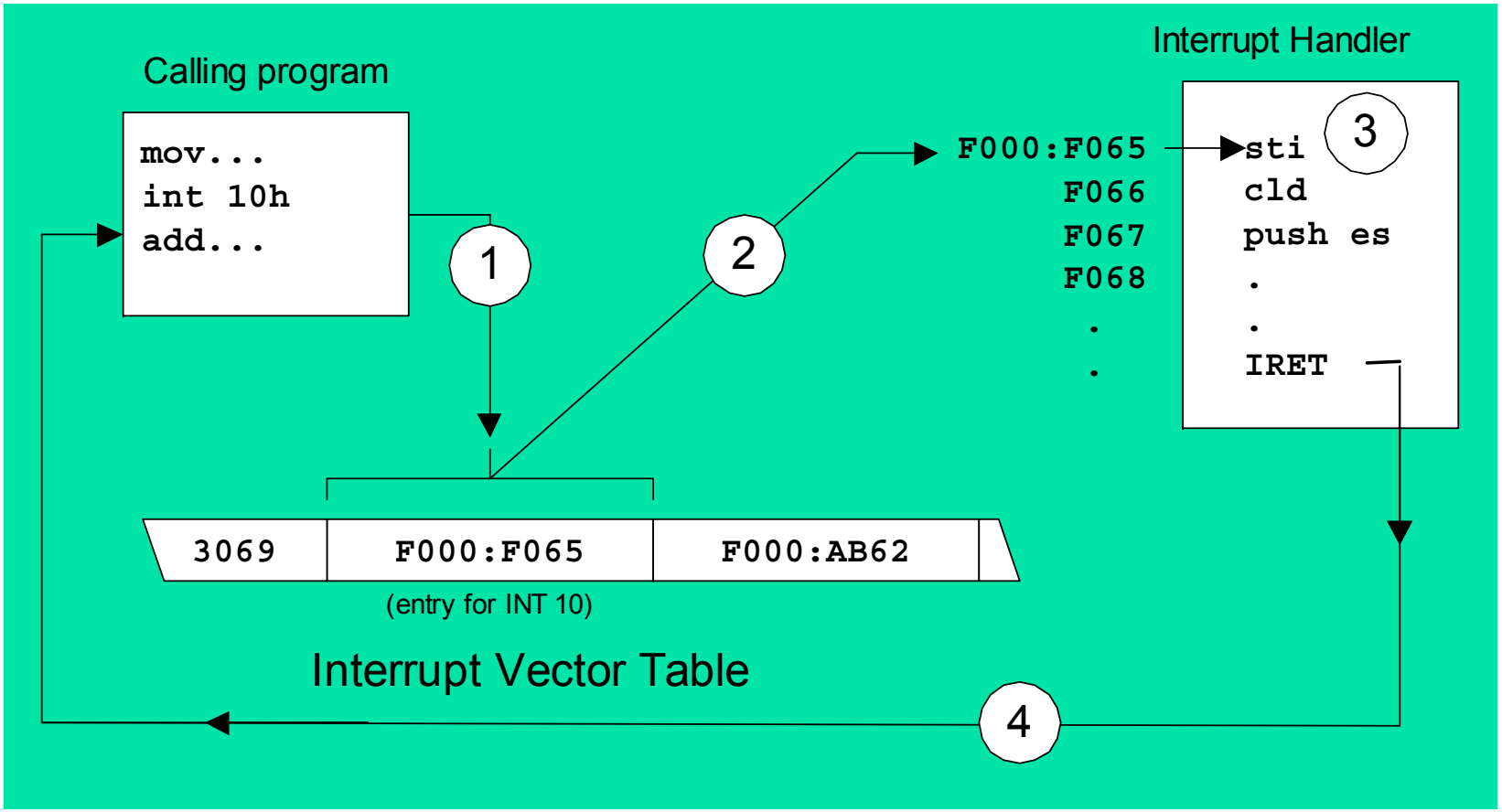
- The INT instruction executes a software interrupt.
- The code that handles the interrupt is called an interrupt handler.
- Syntax:

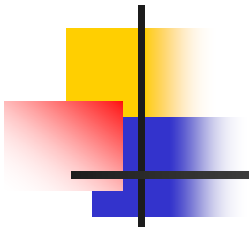
```
INT number
(number = 0..FFh)
```

The Interrupt Vector Table (IVT) holds a 32-bit segment-offset address for each possible interrupt handler.

Interrupt Service Routine (ISR) is another name for interrupt handler.

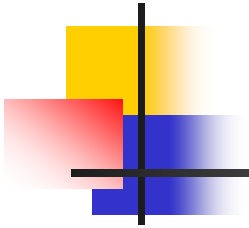
Interrupt Vectoring Process





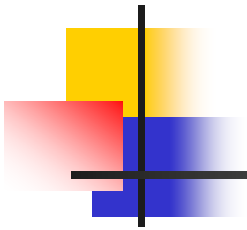
Common Interrupts

- INT 10h Video Services
- INT 16h Keyboard Services
- INT 17h Printer Services
- INT 1Ah Time of Day
- INT 1Ch User Timer Interrupt
- INT 21h MS-DOS Services



Hardware Interrupts

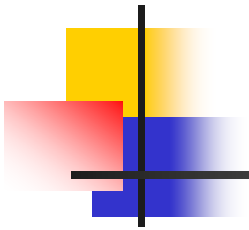
- Generated by the Intel 8259 Programmable Interrupt Contoller (PIC)
  - in response to a hardware signal
- Interrupt Request Levels (IRQ)
  - priority-based interrupt scheduler
  - brokers simultaneous interrupt requests
  - prevents low-priority interrupt from interrupting a high-priority interrupt



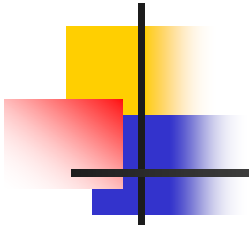
Common IRQ Assignments

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

Common IRQ Assignments

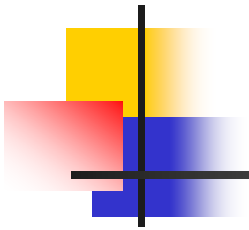


- 8 CMOS real-time clock
- 9 modem, video, network, sound, and USB controllers
- 10 (available)
- 11 (available)
- 12 mouse
- 13 Math coprocessor
- 14 Hard disk controller
- 15 (available)



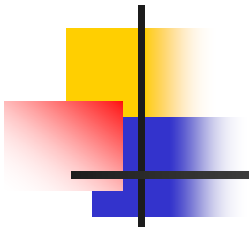
Interrupt Control Instructions

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



What's Next

- MS-DOS and the IBM-PC
- **MS-DOS Function Calls (INT 21h)**
- Standard MS-DOS File I/O Services



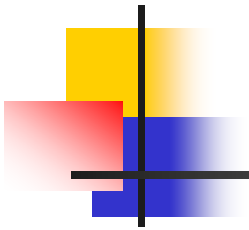
MS-DOS Function Calls (INT 21h)

- ASCII Control Characters
- Selected Output Functions
- Selected Input Functions
- Example: String Encryption
- Date/Time Functions

AH=4Ch: Terminate Process

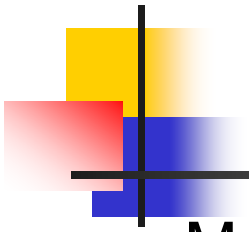
- Ends the current process (program), returns an optional 8-bit return code to the calling process.
- A return code of 0 usually indicates successful completion.

```
mov ah,4Ch              ; terminate process
mov al,0                ; return code
int 21h

; Same as:

.EXIT 0
```



Selected Output Functions

- ASCII control characters
- 02h, 06h - Write character to standard output
- 05h - Write character to default printer
- 09h - Write string to standard output
- 40h - Write string to file or device



ASCII Control Characters

Many INT 21h functions act upon the following control characters:

- 08h - Backspace (moves one column to the left)
- 09h - Horizontal tab (skips forward n columns)
- 0Ah - Line feed (moves to next output line)
- 0Ch - Form feed (moves to next printer page)
- 0Dh - Carriage return (moves to leftmost output column)
- 1Bh - Escape character

INT 21h Functions 02h and 06h:
Write Character to Standard Output

Write the letter 'A' to standard output:

```
mov ah,02h
mov dl,'A'
int 21h
```

Write a backspace to standard output:

```
mov ah,06h
mov dl,08h
int 21h
```

INT 21h Function 05h:
Write Character to Default Printer

Write the letter 'A':

```
mov ah,05h
mov dl,65
int 21h
```

Write a horizontal tab:

```
mov ah,05h
mov dl,09h
int 21h
```

INT 21h Function 09h:
Write String to Standard Output

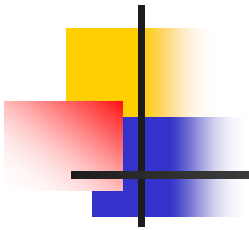
- The string must be terminated by a '$' character.
- DS must point to the string's segment, and DX must contain the string's offset:

```
.data
string BYTE "This is a string$"

.code
mov  ah,9
mov  dx,OFFSET string
int  21h
```

INT 21h Function 40h:
Write String to File or Device

Input: BX = file or device handle (console = 1), CX = number of bytes to write, DS:DX = address of array

```
.data
message "Writing a string to the console"
bytesWritten WORD ?

.code
    mov ah,40h
    mov bx,1
    mov cx,LENGTHOF message
    mov dx,OFFSET message
    int 21h
    mov bytesWritten,ax
```



Selected Input Functions

- 01h, 06h - Read character from standard input
- 0Ah - Read array of buffered characters from standard input
- 0Bh - Get status of the standard input buffer
- 3Fh - Read from file or device

INT 21h Function 01h:
Read single character from standard input

- Echoes the input character
- Waits for input if the buffer is empty
- Checks for Ctrl-Break (^C)
- Acts on control codes such as horizontal Tab

```
.data
char BYTE ?
.code
mov ah,01h
int 21h
mov char,al
```

INT 21h Function 06h:
Read character from standard input without waiting

- Does not echo the input character
- Does not wait for input (use the Zero flag to check for an input character)
- Example: repeats loop until a character is pressed.

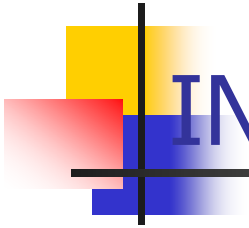
```
.data
char BYTE ?
.code
L1: mov  ah,06h          ; keyboard input
    mov  dl,0FFh         ; don't wait for input
    int  21h
    jz   L1              ; no character? repeat loop
    mov  char,al         ; character pressed: save it
    call DumpRegs        ; display registers
```

INT 21h Function 0Ah:
Read buffered array from standard input (1 of 2)

- Requires a predefined structure to be set up that describes the maximum input size and holds the input characters.
- Example:

```
count = 80

KEYBOARD STRUCT
    maxInput BYTE count             ; max chars to input
    inputCount BYTE ?               ; actual input count
    buffer BYTE count DUP(?)        ; holds input chars
KEYBOARD ENDS
```

Directives: STRUCT, ENDS, ALIGN (Chap10)



INT 21h Function 0Ah (2 of 2)

Executing the interrupt:

```
.data
kybdData KEYBOARD <>

.code
    mov ah,0Ah
    mov dx,OFFSET kybdData
    int 21h
```

INT 21h Function 0Bh:
Get status of standard input buffer

- Can be interrupted by Ctrl-Break (^C)
- Example: loop until a key is pressed. Save the key in a variable:

```
L1: mov ah,0Bh        ; get buffer status
    int 21h
    cmp al,0          ; buffer empty?
    je  L1            ; yes: loop again
    mov ah,1          ; no: input the key
    int 21h
    mov char,al       ; and save it
```

Example: String Encryption

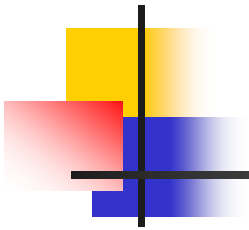
Reads from standard input, encrypts each byte, writes to standard output.

```
XORVAL = 239                ; any value between 0-255
.code
main PROC
     mov  ax,@data
     mov  ds,ax
L1:  mov  ah,6              ; direct console input
     mov  dl,0FFh           ; don't wait for character
     int  21h               ; AL = character
     jz   L2                ; quit if ZF = 1 (EOF)
     xor  al,XORVAL
     mov  ah,6              ; write to output
     mov  dl,al
     int  21h
     jmp  L1                ; repeat the loop
L2:  exit
```

INT 21h Function 3Fh:
Read from file or device

- Reads a block of bytes.
- Can be interrupted by Ctrl-Break (^C)
- Example: Read string from keyboard:

```
.data
inputBuffer BYTE 127 dup(0)
bytesRead WORD ?
.code
mov  ah,3Fh
mov  bx,0                       ; keyboard handle
mov  cx,127                     ; max bytes to read
mov  dx,OFFSET inputBuffer      ; target location
int  21h
mov  bytesRead,ax               ; save character count
```



Date/Time Functions

- 2Ah - Get system date
- 2Bh - Set system date *
- 2Ch - Get system time
- 2Dh - Set system time *

INT 21h Function 2Ah:
Get system date

- Returns year in CX, month in DH, day in DL, and day of week in AL

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

INT 21h Function 2Bh:
Set system date

- Sets the system date. AL = 0 if the function was not successful in modifying the date.

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

INT 21h Function 2Ch:
Get system time

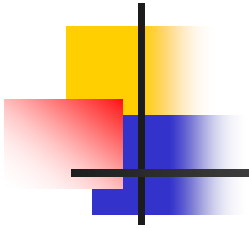
- Returns hours (0-23) in CH, minutes (0-59) in CL, and seconds (0-59) in DH, and hundredths (0-99) in DL.

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

INT 21h Function 2Dh:
Set system time

- Sets the system date. AL = 0 if the function was not successful in modifying the time.

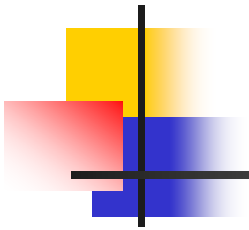
```
mov   ah,2Dh
mov   ch,hours
mov   cl,minutes
mov   dh,seconds
int   21h
cmp   al,0
jne   failed
```



Example: Displaying Date and Time

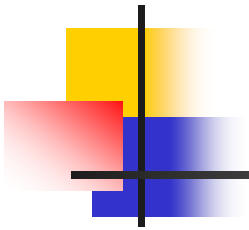
- Displays the system date and time, using INT 21h Functions 2Ah, 2Ch, and 2h.
- Demonstrates simple date formatting
- Sample output:

```
Date: 12-8-2001,    Time: 23:01:23
```



What's Next

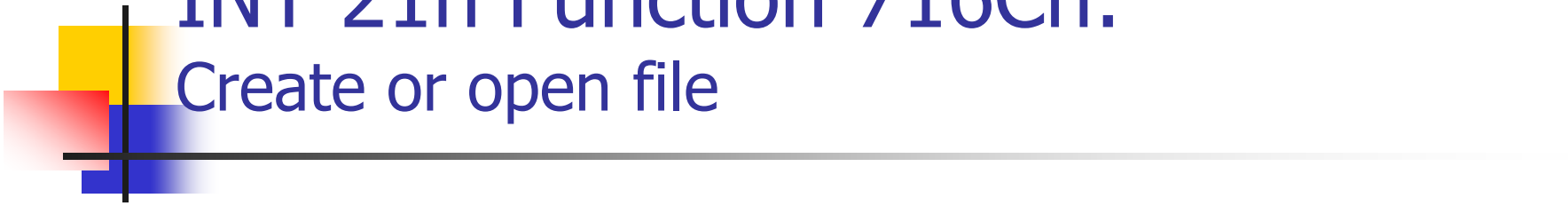
- MS-DOS and the IBM-PC
- MS-DOS Function Calls (INT 21h)
- **Standard MS-DOS File I/O Services**



Standard MS-DOS File I/O Services

- 716Ch - Create or open file
- 3Eh - Close file handle
- 42h - Move file pointer
- 5706h - Get file creation date and time
- Selected Irvine16 Library Procedures
- Example: Read and Copy a Text File
- Reading the MS-DOS Command Tail
- Example: Creating a Binary File

INT 21h Function 716Ch:
Create or open file



- AX = 716Ch
- BX = access mode (0 = read, 1 = write, 2 = read/write)
- CX = attributes (0 = normal, 1 = read only, 2 = hidden,
  3 = system, 8 = volume ID, 20h = archive)
- DX = action (1 = open, 2 = truncate, 10h = create)
- DS:SI = segment/offset of filename
- DI = alias hint (optional)

Example: Create a New File

```
mov   ax,716Ch                  ; extended open/create
mov   bx,2                      ; read-write
mov   cx,0                      ; normal attribute
mov   dx,10h + 02h              ; action: create + truncate
mov   si,OFFSET Filename
int   21h
jc    failed
mov   handle,ax                 ; file handle
mov   actionTaken,cx            ; action taken to open file
```

Example: Open an Existing File

```
mov   ax,716Ch                    ; extended open/create
mov   bx,0                        ; read-only
mov   cx,0                        ; normal attribute
mov   dx,1                        ; open existing file
mov   si,OFFSET Filename
int   21h
jc    failed
mov   handle,ax                   ; file handle
mov   actionTaken,cx              ; action taken to open file
```

INT 21h Function 3Eh:
Close file handle

- Use the same file handle that was returned by INT 21h when the file was opened.
- Example:

```
.data
filehandle WORD ?
.code
    mov  ah,3Eh
    mov  bx,filehandle
    int  21h
    jc   failed
```

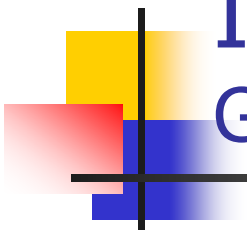
INT 21h Function 42h:
Move file pointer

Permits random access to a file (text or binary).

```
mov   ah,42h
mov   al,0              ; offset from beginning
mov   bx,handle
mov   cx,offsetHi
mov   dx,offsetLo
int   21h
```

AL indicates how the pointer's offset is calculated:

0: Offset from the beginning of the file
1: Offset from the current pointer location
2: Offset from the end of the file



INT 21h Function 5706h:
Get file creation date and time

- Obtains the date and time when a file was created (not necessarily the same date and time when the file was last modified or accessed.)

```
mov ax,5706h
mov bx,handle          ; handle of open file
int 21h
jc  error
mov date,dx
mov time,cx
mov milliseconds,si
```

ReadString Procedure

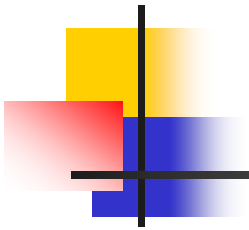
The ReadString procedure from the Irvine16 library reads a string from standard input and returns a null-terminated string. When calling it, pass a pointer to a buffer in DX. Pass a count of the maximum number of characters to input, plus 1, in CX. Writestring inputs the string from the user, returning when either of the following events occurs:

1. CX –1 characters were entered.
2. The user pressed the Enter key.

```
.data
buffer BYTE 20 DUP(?)
.code
mov  dx,OFFSET buffer
mov  cx,LENGTHOF buffer
call ReadString
```

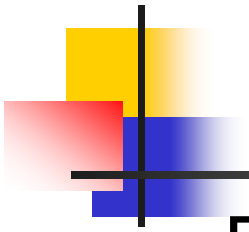
ReadString Implementation

```
ReadString PROC
    push cx                 ; save registers
    push si
    push cx                 ; save character count
    mov  si,dx              ; point to input buffer
    dec  cx                 ; save room for null byte
L1:mov  ah,1                ; function: keyboard input
    int  21h                ; returns character in AL
    cmp  al,0Dh             ; end of line?
    je   L2                 ; yes: exit
    mov  [si],al            ; no: store the character
    inc  si                 ; increment buffer pointer
    loop L1                 ; loop until CX=0
L2:mov  BYTE PTR [si],0   ; insert null byte
    pop  ax                 ; original digit count
    sub  ax,cx              ; AX = size of input string
    pop  si                 ; restore registers
    pop  cx
    ret
ReadString ENDP             ; returns AX = size of string
```



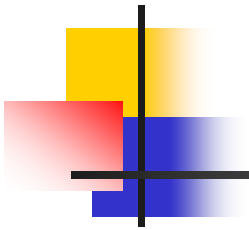
PC-BIOS

- The BIOS (Basic Input-Output System) provides low-level hardware drivers for the operating system.
  - accessible to 16-bit applications
  - written in assembly language, of course
  - source code published by IBM in early 1980's
- Advantages over MS-DOS:
  - permits graphics and color programming
  - faster I/O speeds
  - read mouse, serial port, parallel port
  - low-level disk access



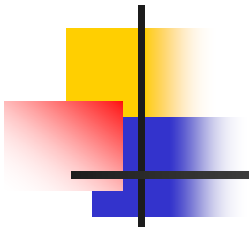
BIOS Data Area

- Fixed-location data area at address 00400h
  - this area is also used by MS-DOS
  - this area is accessible under Windows 98 & Windows Me, but not under Windows NT, 2000, or XP.
- Contents:
  - Serial and parallel port addresses
  - Hardware list, memory size
  - Keyboard status flags, keyboard buffer pointers, keyboard buffer data
  - Video hardware configuration
  - Timer data



What's Next

- Introduction
- **Keyboard Input with INT 16h**
- VIDEO Programming with INT 10h
- Drawing Graphics Using INT 10h
- Memory-Mapped Graphics
- Mouse Programming

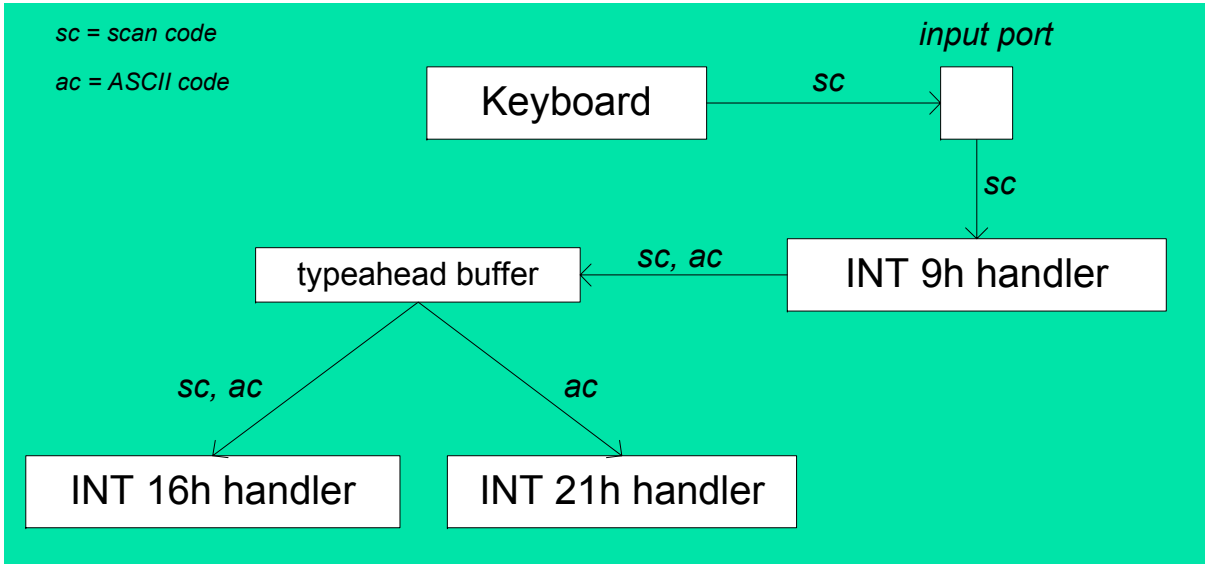


Keyboard Input with INT 16h

- How the Keyboard Works
- INT 16h Functions

How the Keyboard Works

- Keystroke sends a scan code to the keyboard serial input port
- Interrupt triggered: INT 9h service routine executes
- Scan code and ASCII code inserted into keyboard typeahead buffer

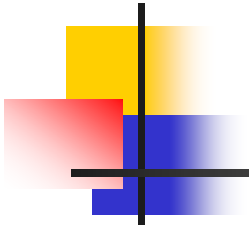


Keyboard Flags

16-bits, located at 0040:0017h – 0018h.

Bit	Description
0	Right Shift key is down
1	Left Shift key is down
2	Either Ctrl key is down
3	Either Alt key is down
4	Scroll Lock toggle is on
5	Num Lock toggle is on
6	Caps Lock toggle is on
7	Insert toggle is on
8	Left Ctrl key is down

Bit	Description
9	Left Alt key is down
10	Right Ctrl key is down
11	Right Alt key is down
12	Scroll key is down
13	Num Lock key is down
14	Caps Lock key is down
15	SysReq key is down



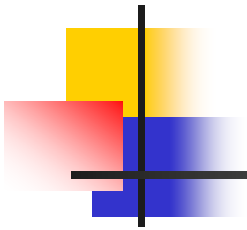
INT 16h Functions

- Provide low-level access to the keyboard, more so than MS-DOS.
- Input-output cannot be redirected at the command prompt.
- Function number is always in the AH register
- Important functions:
  - set typematic rate
  - push key into buffer
  - wait for key
  - check keyboard buffer
  - get keyboard flags

Function 10h: Wait for Key

If a key is waiting in the buffer, the function returns it immediately. If no key is waiting, the program pauses (blocks), waiting for user input.

```
.data
scanCode  BYTE ?
ASCIICode BYTE ?

.code
mov ah,10h
int 16h
mov scanCode,ah
mov ASCIICode,al
```



Function 12h: Get Keyboard Flags

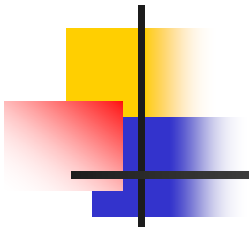
Retrieves a copy of the keyboard status flags from the BIOS data area.

```
.data
keyFlags WORD ?

.code
mov ah,12h
int 16h
mov keyFlags,ax
```

Clearing the Keyboard Buffer

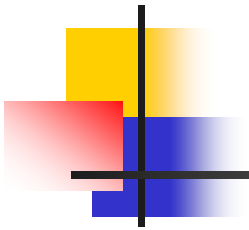
Function 11h clears the Zero flag if a key is waiting in the keyboard typeahead buffer.

```
L1: mov ah,11h                  ; check keyboard buffer
    int 16h                     ; any key pressed?
    jz  noKey                   ; no: exit now
    mov ah,10h                  ; yes: remove from buffer
    int 16h
    cmp ah,scanCode             ; was it the exit key?
    je  quit                    ; yes: exit now (ZF=1)
    jmp L1                      ; no: check buffer again

noKey:                          ; no key pressed
    or  al,1                    ; clear zero flag
quit:
```



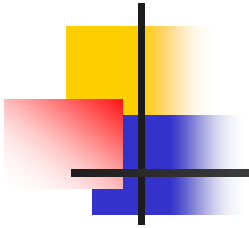
What's Next

- Introduction
- Keyboard Input with INT 16h
- **VIDEO Programming with INT 10h**
- Drawing Graphics Using INT 10h
- Memory-Mapped Graphics
- Mouse Programming



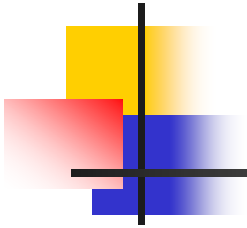
VIDEO Programming with INT 10h

- Basic Background
- Controlling the Color
- INT 10h Video Functions
- Library Procedure Examples



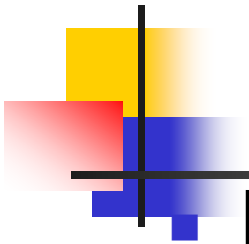
Video Modes

- Graphics video modes
  - draw pixel by pixel
  - multiple colors
- Text video modes
  - character output, using hardware or software-based font table
  - mode 3 (color text) is the default
  - default range of 80 columns by 25 rows.
  - color attribute byte contains foreground and background colors



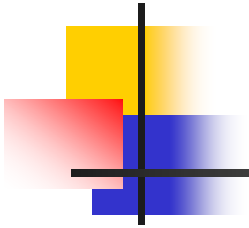
Three Levels of Video Access

- MS-DOS function calls
  - slow, but they work on any MS-DOS machine
  - I/O can be redirected
- BIOS function calls
  - medium-fast, work on nearly all MS-DOS-based machines
  - I/O cannot be redirected
- Direct memory-mapped video
  - fast – works only on 100% IBM-compatible computers
  - cannot be redirected
  - does not work under Windows NT, 2000, or XP



Controlling the Color

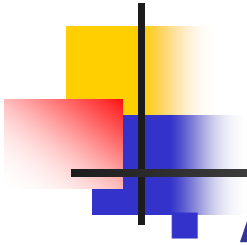
- Mix primary colors: red, yellow, blue
  - called subtractive mixing
  - add the intensity bit for 4th channel
- Examples:
  - red + green + blue = light gray (0111)
  - intensity + green + blue = white (1111)
  - green + blue = cyan (0011)
  - red + blue = magenta (0101)
- Attribute byte:
  - 4 MSB bits = background
  - 4 LSB bits = foreground



Constructing Attribute Bytes

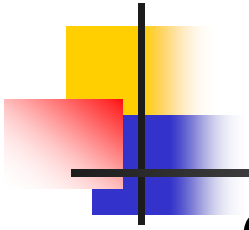
- Color constants defined in Irvine32.inc and Irvine16.inc:

- Examples:
  - Light gray text on a blue background:
    - (blue SHL 4) OR lightGray
  - White text on a red background:
    - (red SHL 4) OR white



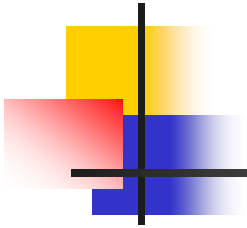
INT 10h Video Functions

- AH register contains the function number
- 00h: Set video mode
  - text modes listed in Table 15-6
  - graphics modes listed in Table 15-6
- 01h: Set cursor lines
- 02h: Set cursor position
- 03h: Get cursor position and size
- 06h: Scroll window up
- 07h: Scroll window down
- 08h: Read character and attribute



INT 10h Video Functions *(cont)*

- 09h: Write character and attribute
- 0Ah: Write character
- 10h (AL = 03h): Toggle blinking/intensity bit
- 0Fh: Get video mode
- 13h: Write string in teletype mode



Displaying a Color String

Write one character and attribute:

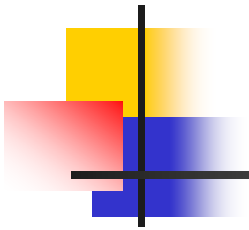
```
mov  si,OFFSET string
. . .
mov  ah,9                  ; write character/attribute
mov  al,[si]               ; character to display
mov  bh,0                  ; video page 0
mov  bl,color              ; attribute
or   bl,10000000b          ; set blink/intensity bit
mov  cx,1                  ; display it one time
int  10h
```

Gotoxy Procedure

```
;-----------------------------------------
   --
Gotoxy PROC
;
; Sets the cursor position on video page 0.
; Receives: DH,DL = row, column
; Returns: nothing
;-----------------------------------------
   ---
   pusha
   mov ah,2
   mov bh,0
   int 10h
   popa
   ret
Gotoxy ENDP
```

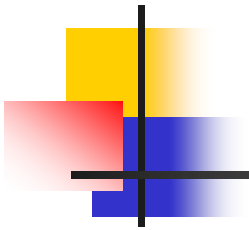
Clrscr Procedure

```
Clrscr PROC
   pusha
   mov      ax,0600h        ; scroll window up
   mov      cx,0            ; upper left corner (0,0)
   mov      dx,184Fh        ; lower right corner
   (24,79)
   mov      bh,7            ; normal attribute
   int      10h             ; call BIOS
   mov      ah,2            ; locate cursor at 0,0
   mov      bh,0            ; video page 0
   mov      dx,0            ; row 0, column 0
   int      10h
   popa
   ret
Clrscr ENDP
```



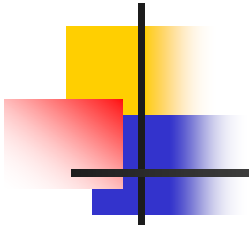
What's Next

- Introduction
- Keyboard Input with INT 16h
- VIDEO Programming with INT 10h
- **Drawing Graphics Using INT 10h**
- Memory-Mapped Graphics
- Mouse Programming



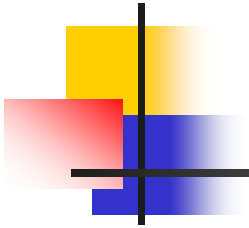
Drawing Graphics Using INT 10h

- INT 10h Pixel-Related Functions
- DrawLine Program
- Cartesian Coordinates Program
- Converting Cartesian Coordinates to Screen Coordinates



INT 10h Pixel-Related Functions

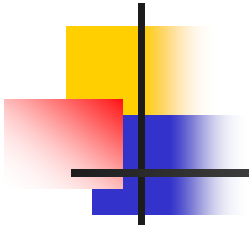
- Slow performance
- Easy to program
- 0Ch: Write graphics pixel
- 0Dh: Read graphics pixel



DrawLine Program

- Draws a straight line, using INT 10h function calls
- Saves and restores current video mode
- Excerpt from the *DrawLine* program (DrawLine.asm):

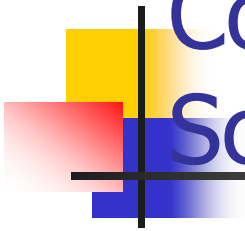
```
mov ah,0Ch       ; write pixel
mov al,color           ; pixel color
mov bh,0         ; video page 0
mov cx,currentX
int 10h
```



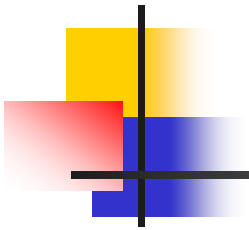
Cartesian Coordinates Program

- Draws the X and Y axes of a Cartesian coordinate system
- Uses video mode 6A (800 x 600, 16 colors)
- Name: Pixel2.asm
- Important procedures:
  - DrawHorizLine
  - DrawVerticalLine

Converting Cartesian Coordinates to Screen Coordinates

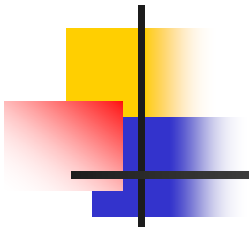


- Screen coordinates place the origin (0,0) at the upper-left corner of the screen
- Graphing functions often need to display negative values
  - move origin point to the middle of the screen
- For Cartesian coordinates X, Y and origin points *sOrigX* and *sOrigY*, screen X and screen Y are calculated as:
  - $sx = (sOrigX + X)$
  - $sy = (sOrigY - Y)$



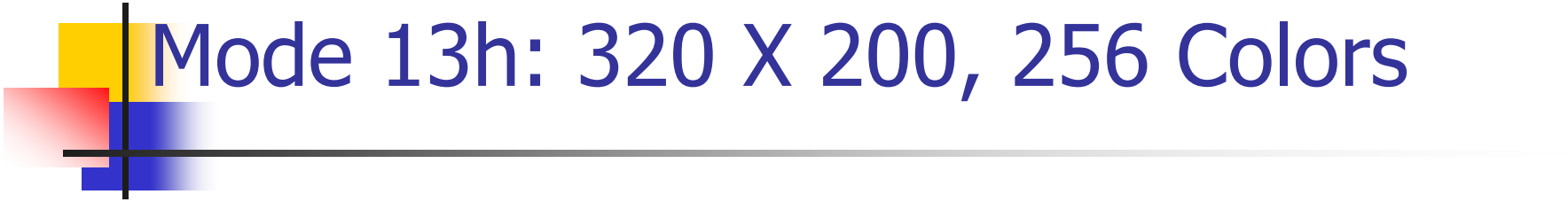
What's Next

- Introduction
- Keyboard Input with INT 16h
- VIDEO Programming with INT 10h
- Drawing Graphics Using INT 10h
- **Memory-Mapped Graphics**
- Mouse Programming



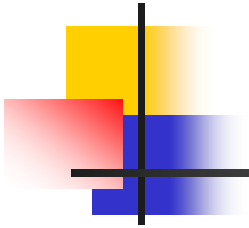
Memory-Mapped Graphics

- Binary values are written to video RAM
  - video adapter must use standard address
- Very fast performance
  - no BIOS or DOS routines to get in the way



Mode 13h: 320 X 200, 256 Colors

- Mode 13h graphics (320 X 200, 256 colors)
  - Fairly easy to program
  - read and write video adapter via IN and OUT instructions
  - pixel-mapping scheme (1 byte per pixel)



Mode 13h Details

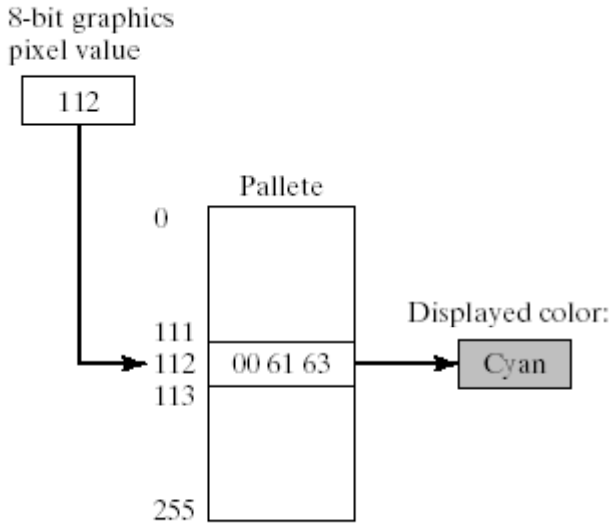
- OUT Instruction
  - 16-bit port address assigned to DX register
  - output value in AL, AX, or EAX
  - Example:

```
mov   dx,3c8h              ; port address
mov   al,20h               ; value to be sent
out   dx,al                ; send to the port
```

- Color Indexes
  - color integer value is an index into a table of colors called a palette

Color Indexes in Mode 13h

Converting Pixel Color Indexes to Display Colors.



RGB Colors

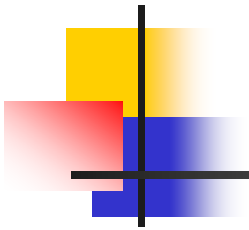
Additive mixing of light (red, green, blue). Intensities vary from 0 to 255.

Examples:

Red	Green	Blue	Color
0	30	30	cyan
30	30	0	yellow
30	0	30	magenta
40	0	63	lavender

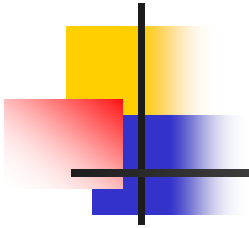
Red	Green	Blue	Color
0	0	0	black
20	20	20	dark gray
35	35	35	medium gray
50	50	50	light gray
63	63	63	white

Red	Green	Blue	Color
63	0	0	bright red
10	0	0	dark red
30	0	0	medium red
63	40	40	pink



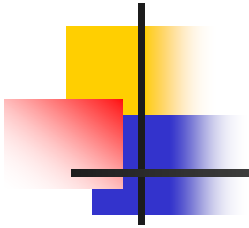
What's Next

- Introduction
- Keyboard Input with INT 16h
- VIDEO Programming with INT 10h
- Drawing Graphics Using INT 10h
- Memory-Mapped Graphics
- **Mouse Programming**



Mouse Programming

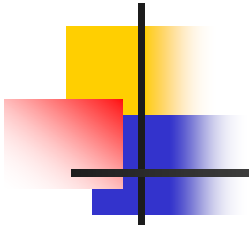
- MS-DOS functions for reading the mouse
- Mickey – unit of measurement (200th of an inch)
  - mickeys-to-pixels ratio (8 x 16) is variable
- INT 33h functions
- Mouse Tracking Program Example



Reset Mouse and Get Status

- INT 33h, AX = 0
- Example:

```
mov   ax,0
int   33h
cmp   ax,0
je    MouseNotAvailable
mov   numberOfButtons,bx
```



Show/Hide Mouse

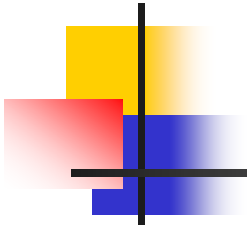
- INT 33h, AX = 1 (show), AX = 2 (hide)
- Example:

```
mov   ax,1          ; show
int   33h
mov   ax,2          ; hide
int   33h
```

Get Mouse Position & Status

- INT 33h, AX = 4
- Example:

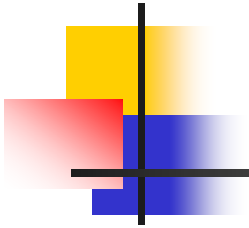
```
mov   ax,4
mov   cx,200        ; X-position
mov   dx,100        ; Y-position
int   33h
```



Get Button Press Information

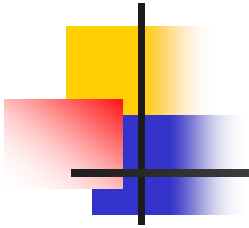
- INT 33h, AX = 5
- Example:

```
mov   ax,5
mov   bx,0          ; button ID
int   33h
test  ax,1          ; left button down?
jz    skip          ; no - skip
mov   X_coord,cx    ; yes: save coordinates
mov   Y_coord,dx
```



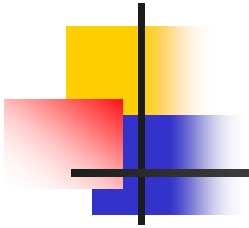
Other Mouse Functions

- AX = 6: Get Button Release Information
- AX = 7: Set Horizontal Limits
- AX = 8: Set Vertical Limits



Mouse Tracking Program

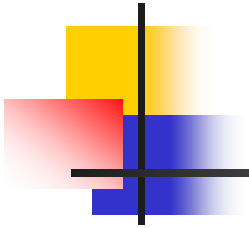
- Tracks the movement of the text mouse cursor
- X and Y coordinates are continually updated in the lower-right corner of the screen
- When the user presses the left button, the mouse's position is displayed in the lower left corner of the screen
- Source code (c:\Irvine\Examples\ch15\mouse.asm)



Set Mouse Position

- INT 33h, AX = 3
- Example:

```
mov   ax,3
int   33h
test  bx,1
jne   Left_Button_Down
test  bx,2
jne   Right_Button_Down
test  bx,4
jne   Center_Button_Down
mov   Xcoord,cx
mov   yCoord,dx
```



Summary

- Working at the BIOS level gives you a high level of control over hardware
- Use INT 16h for keyboard control
- Use INT 10h for video text
- Use memory-mapped I/O for graphics
- Use INT 33h for the mouse