

# Microsoft. Mouse Programmer's Reference Guide

for IBM<sup>®</sup> Personal Computers and Compatibles

**Microsoft Corporation** 

Information in this document is subject to change without notice and does not represent a commitment on the part of Microsoft Corporation. The software described in this document is furnished under a license agreement. The software may be used or copied only in accordance with the terms of the agreement.

©Copyright Microsoft Corporation, 1986

If you have comments about this documentation or the software it describes, complete the Problem Report at the back of this manual and return it to Microsoft.

Microsoft<sup>®</sup> and the Microsoft logo are registered trademarks, and  $InPort_{TM}$  is a trademark, of Microsoft Corporation.

IBM<sup>®</sup> is a registered trademark of International Business Machines Corporation.

Turbo Pascal<sup>®</sup> is a registered trademark of Borland International, Inc.

Hercules TM is a trademark of Hercules Computer Technology.

WordStar<sup>®</sup> is a registered trademark of MicroPro International Corporation.

Document Number 990973002-600-R00-1186 Part Number 000-099-158

11

## Contents

#### About This Guide vii PREADME.DOC vii Product Support viii Microsoft Software License Agreement Addendum ix Disclaimer of Warranty ix

#### **Creating Mouse Menus**

1 Creating Your Own Mouse Menu 1-1 Mouse Menu Language 1-1 Statement Format 1-2 Labels 1-3Parameters 1-31-7Comments Mouse Menu Program Structure 1-7 Mouse Event Statements (BEGIN, ASSIGN) 1-8 Menu Subroutine Statements (MENU, OPTION, MEND) 1-9 Popup Subroutine Statements (POPUP, TEXT, SELECT, PEND) 1 - 10Action Statements (EXECUTE, TYPE, NOTHING) 1-14 String Match Statement (MATCH) 1 - 16Creating a Mouse Menu 1-18 Running a Mouse Menu Program 1 - 21

#### 2 Mouse Menu Language Statements 2-1

ASSIGN 2-2 Description 2 - 22-2Parameters BEGIN 2-4 Description 2-42-5Parameters EXECUTE 2-7 2-7Description 2-7Parameters MATCH 2-8 2-8Description 2-8Parameters MENU...MEND 2-11 Description 2 - 112 - 12Parameters NOTHING 2-14Description 2 - 14OPTION 2-15 Description 2 - 15

Parameters	2 - 15
POPUPPEN	D $2-16$
Description	2 - 16
Parameters	2 - 16
SELECT 2-19	
Description	2 - 19
Parameters	2 - 19
TEXT 2-21	
Description	2-21
Parameter	2-21
TYPE 2-22	
Description	2-22
Parameters	2 - 22

#### **3** Sample Mouse Menu Programs 3-1

SIMPLE Mouse Menu Program 3-1 SIMPLE Mouse Menu Source Program 3-2 DOSOVRLY Mouse Menu Program 3-2 DOSOVRLY Mouse Menu Source Program 3-3

#### 4 Mouse Menu Messages 4-1

#### **Designing Mouse Interfaces**

### 5 The Mouse Interface 5-1

Screen Modes 5-1 The Virtual Screen 5-2 Graphics and Text Cursors 5-4 Graphics Cursor 5-4 Software Text Cursor 5-6 Hardware Text Cursor 5-8 Mouse Buttons 5-9 Mouse Unit of Distance: The Mickey 5-9 The Internal Cursor Flag 5-10

#### 6 Mouse Function Descriptions 6-1

Mouse Functions 6-2 Function 0: Mouse Reset and Status 6-4 Function 1: Show Cursor 6-6 Function 2: Hide Cursor 6-7 Function 3: Get Button Status and Mouse Position 6 - 8Function 4: Set Mouse Cursor Position 6-9 Function 5: Get Button Press Information 6 - 10Function 6: Get Button Release Information 6-11 Function 7: Set Minimum and Maximum Horizontal Cursor Position 6-12 Function 8: Set Minimum and Maximum Vertical Cursor Position 6 - 13Function 9: Set Graphics Cursor Block 6-14 Function 10: Set Text Cursor 6-16 Function 11: Read Mouse Motion Counters 6-17

Function 12: Set Interrupt Subroutine Call Mask and Address 6-18

Function 13: Light Pen Emulation Mode On 6 - 21Function 14: Light Pen Emulation Mode Off 6 - 22Function 15: Set Mickey/Pixel Ratio -6-23Function 16: Conditional Off 6-24 Function 19: Set Double-Speed Threshold 6-26Function 20: Swap Interrupt Subroutines 6-27 Function 21: Get Mouse Driver State Storage Requirements 6-30 Function 22: Save Mouse Driver State 6-31 Function 23: Restore Mouse Driver State 6-32 Function 29: Set CRT Page Number 6 - 33Function 30: Get CRT Page Number 6-33 7 Making Mouse Function Calls 7-1 Making Calls from the BASIC Interpreter 7-2 Making Calls from Assembly-Language Programs 7 - 3Making Calls from High-Level-Language Programs 7-6 Making Calls from Microsoft QuickBASIC 7 - 7Making Calls from Microsoft Pascal 7-10 Making Calls from Microsoft FORTRAN 7 - 13Making Calls from Microsoft C 7-14 Piano Program Listing 7 - 16Sample Cursors 7-23 Standard Cursor Shape 7-247 - 25Up Arrow 7 - 26Left Arrow Check Mark 7 - 27Pointing Hand 7 - 287-29Diagonal Cross 7 - 30Rectangular Cross Hourglass 7 - 318 Writing Mouse Programs for IBM EGA Modes 8-1 The EGA Register Interface Library 8-1 How the Interface Library Works 8-1 How to Call the EGA Register Interface Library 8-2 Making Calls from Assembly-Language Programs 8-2 Making Calls from High-Level-Language Programs 8-3 Restrictions on Use of the EGA Register Interface Library 8-6 Calls to BIOS ROM Video Routines 8-6 EGA Register Interface Functions 8-8 Function F0: Read One Register 8-9 Function F1: Write One Register 8-11 Function F2: Read Register Range 8-13 Function F3: Write Register Range 8-15 Function F4: Read Register Set 8-17 Function F5: Write Register Set 8-19 Function F6: Revert to Default Registers 8-21 Function F7: Define Default Register Table 8-22 Function FA: Interrogate Driver 8-24

vi Contents

### Appendix A Mouse Command Line Switches A-1

Control Panel Switches A-1 Mouse Driver Switches A-3 Specifying Mouse Sensitivity A-4 Setting the Interrupt Rate for the InPort Mouse A-4 Specifying the Type and Location of the Mouse A-4 Disabling or Removing the Mouse Driver A-5

Appendix B Linking Existing Mouse Programs with MOUSE.LIB B-1

Appendix C Making Calls from Borland Turbo Pascal Programs C-1

Appendix D Using the Hercules Graphics Card with Mouse Programs D-1

Index I-1

## About This Guide

By now you're probably enjoying the convenience of the Microsoft<sup>®</sup> Mouse with the applications and the Microsoft Expert Mouse Menus that were included in your mouse package. This guide explains how you can create your own Mouse Menu programs for applications, as well as design a mouse interface for applications that you write yourself. It assumes that you have done some programming, understand basic program design concepts, and are familiar with the operation of the Microsoft Mouse.

This guide has two main parts:

- Creating Mouse Menus explains how to create a Mouse Menu program that allows you to use the Microsoft Mouse with an application that doesn't have built-in mouse support.
- **Designing Mouse Interfaces** explains how to build mouse support directly into one of your own applications.

In addition, four appendices give you technical information about the mouse command line switches, linking existing mouse programs with version 6.0 of the Microsoft Mouse Library, using the mouse with Borland Turbo Pascal programs, and using the Hercules Graphics Card with mouse programs.

### PREADME.DOC

The Microsoft Mouse Tools disk that came with this guide may include a file named PREADME.DOC. Read this file for information that became available after this guide was printed.

### **Product Support**

If you have a question about designing a mouse menu or mouse interface and can't find the answer in this guide, call our Product Support staff by dialing the telephone number on the registration card that came with the *Microsoft Mouse Programmer's Reference Guide*. They will be ready to give you the help you need in order to use the Microsoft Mouse with many applications.

When you call, please have the following information at hand:

- The product number on the Microsoft Mouse Tools disk
- The Microsoft Mouse Programmer's Reference Guide
- Your Microsoft Mouse type
- Your system configuration

### Microsoft Software License Agreement Addendum

#### DISTRIBUTION OF MICROSOFT MOUSE LIBRARY

Microsoft grants you the royalty-free right to reproduce and distribute the Mouse Library provided that you (a) distribute the Mouse Library only in conjunction with and as part of your own software product; (b) do not use Microsoft's name, logo, or trademarks to market your software product; (c) include Microsoft's copyright notice for the Library on your product label and as part of the sign-on message for your software product; and (d) otherwise comply with the Microsoft License Agreement and this Addendum. The "Mouse Library" consists of the files described as "MOUSE.LIB", "OLDMOUSE.LIB", and "EGA.LIB".

If you distribute any portion of the Mouse Library, you agree to indemnify, hold harmless, and defend Microsoft from and against any claims or lawsuits, including attorney's fees, that arise or result from such distribution.

### **Disclaimer of Warranty**

#### USE OF MICROSOFT MOUSE EXAMPLE SOURCE CODE

Your compilation of the source code included on the Microsoft Mouse Tools disk and/or described in this guide, and your subsequent use of the resultant programs, constitutes your acceptance of all results, intended or otherwise, of such use. The source code is meant solely as an example, and Microsoft does not warrant, guarantee, or otherwise make any claim concerning the usability or functionality of the programs defined by the source code.

Mouse Menus

## **Creating Mouse Menus**

This section explains how to use the Mouse Menu programming language to create your own mouse menus for applications.

Chapter 1, "Creating Your Own Mouse Menu," gives an overview of the Mouse Menu programming language and explains how to create and run a Mouse Menu program.

Chapter 2, "Mouse Menu Language Statements," explains in detail how to use each of the Mouse Menu language statements.

Chapter 3, "Sample Mouse Menu Programs," provides the listings for two Mouse Menu programs that are both good examples for designing mouse menus and useful programs you may want to use yourself.

Chapter 4, "Mouse Menu Messages," lists the messages that the Mouse Menu programs can display, along with descriptions of possible causes and actions you should take.

## 1 Creating Your Own Mouse Menu

This chapter provides background information that you'll need before you create a Mouse Menu program. It includes:

- An overview of the Mouse Menu programming language
- A description of program statements and their components
- Descriptions of the various types of subroutines
- A discussion of how statements and subroutines are combined to form a Mouse Menu program

Once you're familiar with how a Mouse Menu program is put together, follow the procedure in "Creating a Mouse Menu" at the end of this chapter to create a working mouse menu.

**Note** Mouse menus cannot be used with programs that use graphics display modes or that have built-in mouse support.

### Mouse Menu Language

The Mouse Menu programming language consists of 13 commands. These commands are used in statements, which assign different functions to the mouse, simulate pressing keys, and create menus.

The following table lists the commands in the Mouse Menu programming language:

#### 1-2 Creating Mouse Menus

Command	Purpose
ASSIGN	Assigns new values for the mouse.
BEGIN	Assigns initial values for the mouse.
EXECUTE	Specifies a sequence of statements exe- cuted when the mouse is moved, a mouse button is clicked, or a menu item is chosen.
MATCH	Specifies the action taken when a unique string of characters is displayed at a specific location on the screen.
MENU	Begins a Menu subroutine.
MEND	Ends a Menu subroutine.
NOTHING	Indicates that no action is taken. An alternative to the EXECUTE, TYPE, and MATCH statements.
OPTION	Defines an item in a Menu subroutine and the action taken when the item is selected.
POPUP	Begins a Popup subroutine.
PEND	Ends a Popup subroutine.
SELECT	Defines the action taken when an item is selected in a popup menu.
TEXT	Defines the text for a popup menu title or menu items.
TYPE	Specifies a key or keys typed when the mouse is moved, a mouse button is clicked, or a menu item is chosen.

### **Statement Format**

You can enter statements in the Mouse Menu programming language in uppercase or lowercase letters. Most statements have the following format:

[label:] command [parameters ; comments]

The BEGIN statement and statements within Menu and Popup subroutines don't use this format because they don't require labels—BEGIN doesn't need a label because it's always the first statement in a program; statements within subroutines don't need labels because the program executes them sequentially.

The components of a statement are described next.

### Labels

A *label* is the name you give a statement. For example, in the following statement "mat1" is the label of the MATCH statement:

mat1: MATCH 23,,INVERSE,"FORMAT",exec1,exec2

A label allows the program to execute statements in a different order than the order in which they appear.

When using labels, follow these rules:

- A statement's label must begin with a letter and be followed by a colon (:).
- Put at least one space between the colon and the command.
- Do not use command names or the words BACKSPACE, ENTER, ESCAPE, or TAB for labels.
- Use any printable standard ASCII characters except for a colon.
- Use labels that suggest what the statement does in the program. For example, use "menu1" as the label for the first Menu subroutine.

### Parameters

A parameter is a variable that affects the action of the statement. Generally, when you use the statement, you must substitute an appropriate value for each parameter. All statements except NOTHING, MEND, and PEND have parameters.

Parameters come after the command word in a statement. Put a space between the command word and the first parameter. Commas must separate any parameters after the first one.

The EXECUTE and TYPE statements allow a variable number of parameters. These statements can have from 1 to 31 parameters. Other statements have a set number of parameters. If you don't want to use a parameter but want to use the parameters that follow, include an additional comma to hold the place of the unused parameter.

For example, in the following statement, "23", "INVERSE", "FORMAT", "exec1", and "exec2" are the values of MATCH statement parameters. The two commas (..) indicate that the second parameter is not used: mat1: MATCH 23,, INVERSE, "FORMAT", exec1, exec2 The program automatically uses a specific value (the default value) for any parameter that is left out of a statement that has a set number of parameters. The Mouse Menu programming language uses three types of parameters: numeric parameters, string parameters, and *attribute* parameters. Numeric Numeric parameters are used for numeric data, such as parameters screen coordinates or movement-sensitivity values for the mouse. As the name suggests, you must use a number for a numeric parameter. In the preceding example, "23", the row coordinate for the MATCH statement, is the value of a numeric parameter. String Most string parameters specify text for menus or mesparameters sages. Use a string of digits, letters, special characters, or spaces for a string parameter. In the example above, "FORMAT", the string that the MATCH statement looks for, is the string parameter. Display attribute A display attribute parameter specifies how a menu or parameters message box appears on the screen. This parameter can have one of four values: "normal", "bold", "inverse", or, if your system uses a color display adapter and monitor, a number that designates specific foreground and background colors. Figure 1.1 shows how the values "normal", "bold", and "inverse" affect the text displayed by a popup menu.

1.1 Effects of Attribute BASIC Commands BASIC Commands BASIC Commands Parameters Cancel Menu Cancel Menu Cancel Menu List List List Run Run Run Load Load Load Normal Bold Inverse

If you do not specify an attribute parameter, the default attribute is used. The default attributes are included in the description of each statement in Chapter 2, "Mouse Menu Language Statements."

If your system uses a color display adapter and color monitor, you can use the attribute parameter in a statement to specify particular colors for the background and foreground of a menu or message box. Text is displayed in the foreground color; the rest of the box is displayed in the background color.

The table on the next page lists the background and foreground colors available, and gives a corresponding value for each. (The exact shades of colors may vary somewhat on different equipment.) The value for a particular color differs depending on whether the color is being used for the foreground or background. The display attribute that specifies a particular color combination is the sum of the values for the desired foreground and background colors.

**Note** If you specify a display attribute value greater than 127, the foreground color will blink when the menu or message box is displayed.

Color menus

#### **1–6** Creating Mouse Menus

Foreground and	Color	Foreground	Background
background	Black	0	0
	Blue	1	16
	Green	2	32
	Cyan (blue-green)	3	48
	Red	4	64
	Magenta	5	80
	Brown	6	96
	White	7	112
	Gray	8	128
	Light Blue	9	144
	Light Green	10	160
	Light Cyan	11	176
	Light Red	12	192
	Light Magenta	13	208
	Yellow	14	<b>224</b>
	White (high intensity)	15	240

If you want green text on a blue background, the value of the attribute parameter would be 18. The value for a green foreground is 2, and the value for a blue background is 16; add these two values together to get the final value of 18.

Specifying a value of 7 is equivalent to specifying the attribute parameter "normal". The value 7 is the sum of 0, the value for a black background, and 7, the value for a white foreground. Similarly, you can specify a "bold" menu by specifying the attribute value 15, and an "inverse" menu by specifying the value 112. "Bold" uses high-intensity white for the foreground (15) and black for the background (0); "inverse" uses black for the foreground (0) and white for the background (112).

Note A gray background (128) looks the same as a black background (0).

### Comments

Comments describe what a statement does. Comments have no effect on how the statement is executed. They are used only to help you read and understand the program.

You can insert comments at the end of a statement or on a separate line. Precede a comment with a semicolon (;). If you include comments on the same line as the statement, separate the last parameter of the statement and the semicolon preceding the comments with a space.

### Mouse Menu Program Structure

There are five types of statements in a Mouse Menu source program:

■ Mouse Event Statements: BEGIN, ASSIGN

Define what action is taken when a mouse event occurs (such as clicking a mouse button)

Menu Subroutine Statements: MENU, OPTION, MEND

Create single-column popup menus

Popup Subroutine Statements: POPUP, TEXT, SELECT, PEND

Create multiple-column menus and message boxes

Action Statements: EXECUTE, TYPE, NOTHING

Perform an action as a result of a Mouse Event, Menu Subroutine, or String Match Statement

String Match Statement: MATCH

Executes other statements depending on what is displayed on the screen

The following sections describe how each statement type is used in a Mouse Menu source program. (For specific information about statements and their parameters, see Chapter 2, "Mouse Menu Language Statements.")

### Mouse Event Statements (BEGIN, ASSIGN)

Mouse Event statements specify which statements the program executes when the user clicks a mouse button or moves the mouse.

Use the BEGIN statement to specify the initial statements executed when particular mouse events occur and to set the initial mouse sensitivity. Always use BEGIN as the first statement in your program.

There are three types of parameters in the BEGIN statement:

#### ■ Button Parameters:

lfbtn	Left button
rtbtn	Right button
btbtn	Both buttons

Define the action taken when one or both mouse buttons are pressed

■ Movement Parameters:

lfmov	Mouse left
rtmov	Mouse right
upmov	Mouse up
dnmov	Mouse down

Define the action taken when the mouse is moved. The cursor keys are often assigned to the mouse movement parameters in a TYPE statement.

Movement Sensitivity Parameters:

hsen Horizontal movement sensitivity vsen Vertical movement sensitivity

Define how much the mouse must move (in *mickeys*, the unit of mouse movement) before the cursor moves. This is helpful in tailoring cursor movement to the different column and row widths found in spreadsheet programs. (For more information on mickeys, see Chapter 5, "The Mouse Interface.")

#### ASSIGN statement

Use the ASSIGN statement to assign new values to mouse events or mouse sensitivity. ASSIGN is useful if you want to execute different statements or subroutines depending on the mode of an application program or on other conditions that require the mouse to be used differently.

#### BEGIN statement

### Menu Subroutine Statements (MENU, OPTION, MEND)

Menu subroutines create single-column popup menus. Single-column menus are bordered menus with a single column of menu items. (Figure 1.1, earlier in this chapter, shows examples of single-column menus.) The user chooses items in the menu by moving the mouse pointer to the desired item, then clicking either mouse button. If the user clicks both mouse buttons at once, the equivalent of a NOTHING statement is executed and the menu disappears.

Menu subroutines use this format:

label: MENU ["title",row,column,attribute] OPTION ["text",pointer]

MEND

A Menu subroutine begins with a MENU statement that specifies:

#### MENU statement

- The menu's title, enclosed in double quotation marks
- The row and column of the screen where the upper-left corner of the menu will appear
- The menu's display attribute (for more information, see "Parameters" earlier in this chapter)

OPTION statements specify the menu items and action when an item is chosen. At least one OPTION statement should be included in each Menu subroutine as an exit point from the menu.

The pointer parameter is the label of the statement that is executed when the user chooses that menu item. If no pointer parameter is specified, the equivalent of a NOTHING statement is executed when that item is chosen and the menu disappears. OPTION statement

MEND statement	A MEND (or "menu end") statement always follows the last OPTION statement to end the Menu subroutine.	
Sample Menu subroutine	This sample Menu subroutine produces the "Inverse Attribute" menu shown in Figure 1.1: menul: MENU "BASIC Commands", 5, 20 OPTION "Cancel Menu" OPTION "List", F1 OPTION "Run", F2	
	OFTION "Load",F3 MEND F1: TYPE 0,59 ;simulate pressing the F1 key F2: TYPE 0,60 ;simulate pressing the F2 key F3: TYPE 0,61 ;simulate pressing the F3 key	

The menu produced by this subroutine appears at row 5, column 20. Because no attribute was specified, inverse screen characteristics (the default attribute) are used. When the menu appears on the screen, the cursor bar is always on the first menu item (in this case, "Cancel Menu").

If the user chooses "Cancel Menu", the menu disappears because no pointer parameter is specified for that OPTION statement. If the user chooses any other item, the statement identified in the pointer parameter for that OPTION statement is executed.

### Popup Subroutine Statements (POPUP, TEXT, SELECT, PEND)

Popup subroutines are used to create more complex menus or message boxes.

Multiple-column menus are used in the same way as single-column menus: the user chooses items by moving the mouse pointer to the item, then clicking either mouse button. Clicking both mouse buttons at once removes the menu from the screen. When the menu first appears on the screen, the highlight is always over the first menu item. Figure 1.2 shows a sample multiple-column menu:

CURSOR N	10VEMENT
Cancel menu	Top of screen
Screen up	Bottom of scrn
Screen down	Start of file
Previous place	End of file

**1.2** Multiple-Column Menu

Message boxes are simply popup menus that display messages instead of menu items. You can combine Popup subroutines with MATCH statements so that message boxes appear when the program mode changes, or when other conditions cause the screen display to change.

Figure 1.3 shows a sample message box:

HOUSE HELP Left button - Displays Edit/Block menu Right button - Displays Cursor movement menu Both buttons - Displays Edit/File menu Moving the mouse up,down,left, or right will cause the cursor to move in that direction.

Popup subroutines for multiple-column menus and message boxes use the following format:

label: POPUP [row,column,attribute] [TEXT ["text string"]]

SELECT [row,col,width,pointer]

PEND

**1.3** Message Box

Popup subroutine format

## 1-12 Creating Mouse Menus

POPUP statement	Each Popup subroutine begins with a POPUP statement that specifies:
	<ul> <li>The row and column of the menu's top-left corner</li> <li>The menu's display attribute. For more information on display attributes, see "Parameters" earlier in this chapter.</li> </ul>
TEXT statements	Use TEXT statements to specify the menu title and menu items. Type in the title text, item text, and menu borders exactly as they'll appear on each line of the menu, and enclose them in double quotation marks. You can include ASCII graphics characters, such as "=" or "1", in the borders or item text.
	The text will be located on the screen relative to the coordinates you specify in the POPUP statement.
SELECT	Use SELECT statements to define:
statements	■ The areas in which the user can choose each menu item. Specify the row, column, and width of the selec- tion area, relative to the menu's top-left corner. The relative coordinates of the top-left corner of the popup menu are "1,1".
	■ The statement that is executed when the user chooses an item. As with the OPTION statement for a single- column menu, you specify the label of the statement that is executed.
	You must include at least one SELECT statement in each Popup subroutine as an exit point.
PEND	A PEND (or "popup end") statement always follows the

### เส

last SELECT statement to end the Popup subroutine.

This sample Popup subroutine creates the multiplecolumn menu shown earlier, in Figure 1.2:

```
movmen: popup 2,1
        text " ====== CURSOR MOVEMENT ======= "
        text "| Cancel menu
                                              1 **
                              Top of screen
        text "| Screen up
                                               **
                               Bottom of scrn
        text "| Screen down
                                               11
                              Start of file
        **
                                               **
        select 2,3,15
        select 3,3,15,keyctrlr
        select 4,3,15,keyctrlc
        select 5,3,15,keyctrlqp
select 2,16,15,keyctrlqe
        select 3,16,15,keyctrlqx
        select 4,16,15,keyctrlgr
        select 5,16,15,keyctrlgc
        pend
```

In this example, the top-left corner of the menu will be at row 2, column 1. Because no attribute parameter is specified, the menu will be displayed using the inverse display attribute.

The TEXT statements specify the menu items and their locations relative to the top-left corner. The first item starts at "relative" row 2, column 3 in the menu, but its actual coordinates are row 3, column 3. ASCII graphics characters are used to create solid menu borders.

When the menu appears on the screen, the first item (in this case, "Cancel menu") is highlighted.

The SELECT statements define the item selection areas. In the first item ("Cancel menu"), "2, 13, 15" define the row, column, and width of the selection area, respectively. Because the SELECT statement for "Cancel menu" does not specify a label for the pointer parameter, the menu will be cleared from the screen if the user chooses "Cancel menu." The other SELECT statements execute the statements named in their pointer parameters. Sample Popup subroutines The following sample Popup subroutine creates the message box shown in Figure 1.3:

\_\_\_\_\_ MOUSE HELP \_\_\_\_\_ text " •• text "| Left button - Displays Edit/Block menu text "| Right button - Displays Cursor movement menu \*\* text "| Both buttons - Displays Edit/File menu .. text " ., text "| Moving the mouse up, down, left, or right will .. text "I cause the cursor to move in that direction. text " .. ... select 1,18,10 pend

The POPUP statement defines row 2, column 1 as the top-left-corner coordinates. Because no attribute parameter is specified, "inverse" will be used.

The TEXT statements define the message box border and the message text. The single SELECT statement defines an exit point for the menu. Because the message box has only one SELECT statement, the user cannot move the cursor within the message box.

### Action Statements (EXECUTE, TYPE, NOTHING)

Action statements specify what action is taken when the user chooses a menu item, clicks one or both buttons, or moves the mouse.

Use the EXECUTE statement to define a series of statements that will be executed when:

- The user clicks one or both mouse buttons
- The user chooses a menu item
- The user moves the mouse
- A MATCH statement is executed (see the next section, "String Match Statement")

Use statement labels to specify the statements that the EXECUTE statement will carry out. You can specify up to 31 labels for each EXECUTE statement. An

#### EXECUTE statement

EXECUTE statement can carry out another EXECUTE statement to increase the number of statements that are carried out. You can link up to 31 EXECUTE statements in this manner.

Here is a sample EXECUTE statement with five labels:

exec1: EXECUTE dsk,s,a,s,exec4

This statement executes the statements labeled "dsk", "s", "a", "s", and "exec4".

Use the TYPE statement to simulate pressing keys on the keyboard. For example, the following TYPE statement simulates pressing the a key:

key1: TYPE "a"

The following TYPE statement simulates typing the *diskcopy a: b:* command and pressing the ENTER key:

key15: TYPE "diskcopy a: b:",enter

You can indicate which key or sequence of keys is simulated in one of three ways:

- Use its key name, or a sequence of key names, enclosed in double quotation marks (for example, "A").
- Use the ASCII code for the character on the key (for example, 65 for "A"). You can use extended ASCII codes, ASCII control characters, and extended keyboard scan codes to simulate special keys or key sequences, such as ALT, CONTROL-Q, spacebar, and arrow keys. (See the IBM *BASIC* manual for a list of ASCII codes. For a list of ASCII control characters and extended keyboard scan codes, see "TYPE" in Chapter 2, "Mouse Menu Language Statements.")
- Use its symbolic name. The predefined symbolic keys are "enter", "tab", "backsp", and "esc".

#### TYPE statement

Here are sample TYPE statements. The comments indicate which key(s) each statement simulates.

Label	Code	Comments
dir:	TYPE "dir"	;type the command "dir"
a:	TYPE "a:"	type "a:"
lf:	<b>TYPE 0,75</b>	simulate the left arrow key
rt:	TYPE 0,77	simulate the right arrow key
up:	<b>TYPE 0,72</b>	simulate the up arrow key
dn:	TYPE 0,80	simulate the down arrow key
s:	TYPE $32$	type a space
ent:	TYPE enter	simulate the ENTER key

The statements labeled "dir" and "a" simulate typing a character string by enclosing the characters in double quotation marks.

The next four statements define the arrow keys using extended keyboard scan codes. The statement labeled "s" simulates the spacebar by using the standard ASCII code.

The statement labeled "ent" simulates pressing ENTER by using the symbolic name for the key.

Use the NOTHING statement to specify that no action is taken. Most often, this statement is used with other statements when you want to disable a parameter.

### String Match Statement (MATCH)

MATCH statements permit a Mouse Menu program to take different actions depending on what is displayed on the screen.

A MATCH statement specifies a string of characters, a row and column on the screen, and a display attribute. If a line on the screen matches the specified string, begins at the specified row and column, and appears in the specified display attribute, then the program executes a particular statement. This feature enables a Mouse Menu source program to respond to different operating modes of the application program or screen display.

For example, if an application program always displays "COMMAND" on line 22 of the screen when it is in command mode, and displays "ALPHA" in the same place when it is in alphanumeric mode, you can use a MATCH statement to take a different action depending on which mode the application program is in.

NOTHING statement A MATCH statement uses the following format:

MATCH row, column, attribute, string, match, nomatch

- The "row" and "column" parameters describe where the "string" parameter must be located on the screen for a match.
- The "attribute" parameter indicates how the string must appear on the screen for a match. This parameter can have one of the symbolic values "normal", "bold", or "inverse", or a decimal value that denotes specific foreground and background colors. (For information on the attribute parameter, see "Parameters" earlier in this chapter.) If the attribute parameter is left blank or given the value of 0, all display attributes are matched.
- The "match" and "nomatch" parameters are the labels of the statements to be executed if the match is made or not made.

The following sample Mouse Menu source program shows how a MATCH statement is used: Sample program using MATCH statements

BEGIN menul, chna, ent, lf, rt, up, dn chna: MATCH 4,1,normal, "A",ex20,ex19 chnb: MATCH 4,1,normal, "B",ex21,chna chnc: MATCH 4,1,normal, "C",ex19,chna ana: ASSIGN , chna anb: ASSIGN , chnb anc: ASSIGN , chnc ex19: EXECUTE cls, a, ent, ana ;change to A: ex20: EXECUTE cls,b,ent,anb ;change to B: ex21: EXECUTE cls,c,ent,anc ;change to C: ent: TYPE enter TYPE "cls", enter cls: TYPE "a:" a: TYPE "b:" b: TYPE "c:" **c** :

This program changes the active disk when the user clicks the right mouse button. The program follows this procedure:

- When the user clicks the right mouse button, the MATCH statement labeled "chna" checks row 4, column 1 on the screen. If it finds an "A" in "normal" display, it executes the statement labeled "ex20".
- The "ex20" statement clears the screen, changes the active drive to "B:" and executes the statement labeled "anb", which reassigns the right button parameter to "chnb".
- Now if the user clicks the right mouse button, the MATCH statement labeled "chnb" checks row 4, column 1 on the screen. If it finds a "B" in "normal" display, it executes the statement labeled "ex21".
- The "ex21" statement clears the screen, changes the active drive to "C:" and executes the statement labeled "anc", which reassigns the right button parameter to "chnc".
- Now if the user clicks the right mouse button, the MATCH statement labeled "chnc" checks row 4, column 1 on the screen. If it finds a "C" in "normal" display, it executes the statement labeled "ex19".
- The "ex19" statement clears the screen, changes the active drive to "A:" and executes the statement labeled "ana", which reassigns the right button parameter to "chna". The program is now back to step 1.

### Creating a Mouse Menu

You should now be able to start writing Mouse Menu programs. Follow the procedure below to create a source file and then an executable Mouse Menu program file from the source file.

**Note** The Microsoft Mouse Tools disk that came with this guide includes Mouse Menu source files for some commonly used applications that don't have built-in mouse support (such as WordStar). Use the following procedure to create mouse menus from these source files.

To create a mouse menu:

Write the Mouse Menu program into a source file using a text editor or word processing program. Save the source file with the filename extension ".DEF". This file is used by the MAKEMENU utility program to generate an executable Mouse Menu program (a .MNU file).

Be sure to save the source file as a standard ASCII text file. Most simple editors save files in ASCII by default, but when using a word processing program, such as Microsoft Word, you usually need to select a special "unformatted" option to get ASCII text.

If you want to create a mouse menu from one of the source files included on the Microsoft Mouse Tools disk, you can copy the source file and edit the copy to meet your specific needs.

**Note** When a source file is converted to a .MNU file, it must not exceed 57K.

2 Use the MAKEMENU utility to create an executable menu file from the source file.

To use MAKEMENU, type *makemenu* and press ENTER.

At the prompt, type the name of the source file (without the ".DEF" extension), then press ENTER.

If your file has no errors, MAKEMENU displays this message:

Conversion completed

and returns you to DOS. The mouse menu is ready to be tested following the procedure given below.

If your file has errors, MAKEMENU displays the types of errors and statements containing the errors. (For more information on error messages, see Chapter 4, "Mouse Menu Messages.") Correct the source program and repeat this procedure.

# Testing the mouse menu

When the Mouse Menu source file has been translated into an executable menu file, it is ready to be tested.

**Note** If, when you ran the Mouse Setup program, you did not specify that the mouse driver should be loaded automatically every time you start DOS, make sure you type *mouse* to install the mouse driver before you start your menu file.

To test the mouse menu:

I Type menu <filename> at the DOS prompt and press ENTER to start the Mouse Menu program. In this command, <filename> is the name of the Mouse Menu program file without the .MNU extension.

When the Mouse Menu file has been loaded, this message appears:

Menu installed

2 Start your application program and try out the menu to ensure that it works under all conditions in your program.

If it doesn't work as desired, end the Mouse Menu program by typing *menu off* at the DOS prompt and pressing ENTER.

This message is displayed:

Keyboard emulation off

Correct the source file, then run the MAKEMENU utility program again.

### Running a Mouse Menu Program

Follow these steps to run a Mouse Menu program:

- 1 Use the DOS COPY command to copy the executable Mouse Menu (.MNU) file and the MENU.COM file onto the disk that contains the application program with which you want to use the menu.
- Type menu < filename > to run the Mouse Menu program for the application. In this command, <filename > is the name of the Mouse Menu program.

**Note** To start a Mouse Menu program that is not in the current directory, include the path name of the directory that contains the Mouse Menu file. For more information, see the PATH command in your DOS manual.

When the Mouse Menu file has been loaded, the following message appears:

Menu installed

Run the application program according to the instructions in the program's documentation.

A Mouse Menu program runs independently of the corresponding application program. You should end the Mouse Menu program you're running and begin another whenever you end one application and begin another.

To end the Mouse Menu program:

Type *menu off* and press ENTER.

This message is displayed:

Keyboard emulation off

You can then load and run another Mouse Menu program.

#### Ending a Mouse Menu program

Memory allocation for mouse menus MENU.COM can allocate up to 57K of memory for a Mouse Menu program. (The size of MENU.COM (7K) plus the size of the .MNU file cannot exceed 64K.) If the menu file is less than 6K, MENU.COM allocates 6K of memory. If the menu file is greater than 6K, MENU.COM allocates the exact size of the file.

Every time you start DOS, the first menu file you load determines the amount of memory reserved for a menu file. If you plan to use more than one mouse menu before restarting your system, first load the .MNU file that requires the greatest amount of memory so that MENU.COM will have allocated enough memory to hold each menu file.

## 2 Mouse Menu Language Statements

This chapter describes in alphabetical order each of the statements used by the Mouse Menu programming language. Each statement description includes:

- The statement syntax
- A description of each parameter
- $\blacksquare$  An example of how to use the statement

In the syntax diagram for each statement:

- The command word appears in capital letters.
- Labels appear in small letters. Each label must be separated from the command word by a colon (:) and a space.
- Parameters appear in small letters. Each parameter must be separated from other parameters by a comma (,). If a parameter is not used, the statement must include an additional comma where the parameter would have appeared. (For example, if the second parameter in a statement is not used, the statement would include two commas in a row (,,) after the first parameter.)
- If a parameter appears in brackets ([]), it is optional. If a parameter does not appear in brackets, it is required. If a parameter appears in double quotation marks (""), the double quotation marks are required.
- If a parameter can appear more than once in a statement, the second occurrence of the parameter is enclosed in brackets and followed by an ellipsis (...).

# Statement syntax conventions

### ASSIGN

label: ASSIGN [lfbtn],[rtbtn],[btbtn],[lfmot],[rtmot], [upmot],[dnmot],[hsen],[vsen]

### Description

ASSIGN redefines one or more of the mouse parameters given in the BEGIN statement or most recent ASSIGN statement. If a parameter value isn't specified in an ASSIGN statement, the last parameter value given (in either the BEGIN statement or another ASSIGN statement) is used. Statement labels are used for all parameters except "hsen" and "vsen".

All ASSIGN statements must be labeled.

### Parameters

lfbtn	New label of the first statement executed when the user clicks the left mouse button.
rtbtn	New label of the first statement executed when the user clicks the right mouse but- ton.
btbtn	New label of the first statement executed when the user clicks both mouse buttons at once.
lfmot	New label of the first statement executed when the user moves the mouse to the left.
rtmot	New label of the first statement executed when the user moves the mouse to the right.
upmot	New label of the first statement executed when the user moves the mouse forward.
dnmot	New label of the first statement executed when the user moves the mouse backward.
hsen	New value of the horizontal movement sensitivity parameter.
vsen	New value of the vertical movement sensi- tivity parameter.
### Example

BEGIN esc,ent,mm1,lf,rt,up,dn

reassign: ASSIGN y,not,,,,not,not,16,18

In this example, the BEGIN statement assigns the initial values of all button and movement parameters. Because no values are specified for the sensitivity parameters ("vsen" and "hsen"), the default values are used. The ASSIGN statement changes the values of the left button, right button, and up and down movement parameters. (If "Not" were the label of a NOTHING statement, the ASSIGN statement would disable any response to clicking the right mouse button or moving the mouse forward or backward.) It also changes the value of "hsen" to 16 and the value of "vsen" to 18. Commas are used for the parameters whose values aren't changed.

## BEGIN

BEGIN [lfbtn],[rtbtn],[btbtn],[lfmot],[rtmot],[upmot], [dnmot],[hsen],[vsen]

### Description

BEGIN defines what actions are taken when the mouse is used. Because BEGIN is always the first statement in a menu source file, it doesn't require a statement label.

The parameters for BEGIN define the statements to be executed when the mouse buttons are clicked or the mouse is moved. It also defines the movement sensitivity for the mouse. All parameters are optional. If no value is given for a button or mouse movement parameter, the corresponding function is not used.

**Note** When a Menu subroutine is executed, the parameters for BEGIN do not affect the mouse functions. Either mouse button can be used to choose an item in a menu, and all mouse movement functions are active.

Statement labels are required for all parameters except the mouse movement parameters. These are the labels of the statements that are executed when the event governed by each parameter occurs.

The movement sensitivity parameters control the horizontal and vertical movement sensitivity of the mouse. Movement sensitivity is the distance the mouse must move (measured in mickeys, the unit of mouse movement) before the on-screen pointer moves. (For more information about mickeys, see Chapter 5, "The Mouse Interface.")

## Parameters

lfbtn	Label of the first statement executed when the user clicks the left mouse button. If you don't specify a label, nothing happens when the user clicks the left mouse button.
rtbtn	Label of the first statement executed when the user clicks the right mouse button. If you don't specify a label, nothing happens when the user clicks the right mouse but- ton.
btbtn	Label of the first statement executed when the user clicks both mouse buttons. If you don't specify a label, nothing happens when the user clicks both mouse buttons.
lfmot	Label of the first statement executed when the user moves the mouse to the left. If you don't specify a label, nothing happens when the user moves the mouse to the left.
rtmot	Label of the first statement executed when the user moves the mouse to the right. If you don't specify a label, nothing happens when the user moves the mouse to the right.
upmot	Label of the first statement executed when the user moves the mouse forward. If you don't specify a label, nothing happens when the user moves the mouse forward.
dnmot	Label of the first statement executed when the user moves the mouse backward. If you don't specify a label, nothing happens when the user moves the mouse backward.
hsen	Number between 0 and 32767 that defines how many mickeys the mouse must move vertically before the on-screen pointer moves. If 0 is specified, the mouse is dis- abled horizontally. If no value is specified, the default value of 4 mickeys is used. (One mickey is approximately 1/200 inch.)
vsen	Number between 0 and 32767 that defines how many mickeys the mouse must move vertically before the on-screen pointer moves. If 0 is specified, the mouse is dis- abled vertically. If no value is specified, the default value of 8 mickeys is used.

### Example

BEGIN ent, esc, , lf, rt, up, dn

lf:	TYPE	0,75	;simulate	the	left cursor key
rt:	TYPE	0,77	;simulate	the	right cursor key
up:	TYPE	0,72	;simulate	the	up cursor key
dn:	TYPE	0,80	;simulate	the	down cursor key
esc:	TYPE	ESC	;simulate	the	Esc key
ent:	TYPE	ENTER	;simulate	the	enter key

The BEGIN statement in this example gives initial values for all parameters except "btbtn", "hsen", and "vsen". Because "btbtn" isn't specified, nothing happens when the user clicks both mouse buttons. Because no values are given for "hsen" and "vsen", the default values are used (4 and 8 mickeys, respectively).

# EXECUTE

label: EXECUTE statement[,statement...]

## Description

EXECUTE can carry out other statements when one of the following events occurs:

- A menu item is selected
- The mouse is moved
- One or both mouse buttons are clicked
- A MATCH statement is executed

Each EXECUTE statement may specify up to 31 other statements to be executed. EXECUTE can call other EXECUTE statements to increase the number even further; up to 31 EXECUTE statements can be linked in this manner. Statements within an EXECUTE statement are executed sequentially, starting with the first statement.

## Parameters

label	Name of the EXECUTE statement. All EXECUTE statements must be labeled.
statement	Name(s) of the statement(s) to be executed. Any labeled statement can be used. (Using the calling statement may cause an endless loop.)

### Example

dir: TYPE "dir" ;type dir s: TYPE 32 ;simulate the spacebar ;" " may also be used a: TYPE "A:" ;type A: ent: TYPE ENTER ;simulate the ENTER key exec4: EXECUTE dir,s,a,ent

The EXECUTE statement labeled "exec4" executes the statements labeled "dir", "s", "a", and "ent". These statements simulate typing dir A: and pressing ENTER.

## MATCH

label: MATCH row, column, attribute, string, match, nomatch

### Description

MATCH executes other statements or subroutines depending on whether or not it finds a specified string in a given screen location.

Values for the row and column parameters are given in absolute screen coordinates. The starting coordinates for the screen are in the upper-left corner of the screen (row 1, column 1).

## Parameters

label	Name of the MATCH statement. All MATCH statements must be labeled.
row	A number that specifies the row of the first character of the match string. If no value is specified, row 1 is assigned.
column	A number that specifies the column of the first character of the match string. If no value is specified, column 1 is assigned.
attribute	A value that specifies how the match string must appear on the screen for a match to occur. This can be one of the symbolic values "normal", "bold", or "inverse", or a decimal value that denotes specific foreground and background colors. (For more information, see "Parameters" in Chapter 1, "Creating Your Own Mouse Menu.") If the attribute parameter is left blank or given the value of 0, the MATCH statement matches any attribute value.

string	The string to match. This can be any string of up to 255 ASCII characters, enclosed in double quotation marks (""). You must specify the string parameter.
match	Label of a statement or subroutine exe- cuted if the string is matched. This label must be present in the program.
nomatch	Label of a statement or subroutine exe- cuted if the string is not matched. This label must be present in the program.

### Example

BEGIN leftb,rightb,bothb,mousel,mouser,mouseu,moused,16,40
leftb: MATCH 1,12,normal,"e",imen,chk33
chk33: MATCH 1,12,,"n",imen,chk1
chk1: MATCH 1,11,,":",emen,not
imen: POPUP 2,1

This sample from the WS.DEF menu source file checks whether WordStar is displaying the BEGINNING MENU or the MAIN MENU.

When the user clicks the left mouse button:

• The MATCH statement labeled "leftb" looks for an "e" at row 1, column 12. This is the first character in the string "editing no file", which is on the screen in that position if WordStar version 3.2 is displaying the BEGINNING MENU. If "leftb" finds the "e" in that position, it executes the statement labeled "imen". (In WS.DEF, "imen" displays the NO-FILE popup menu for WordStar.)

If "leftb" doesn't find the "e" in that position, it executes the statement labeled "chk33".

The "chk33" statement looks for the letter "n" at row 1, column 12. This is the first character in the string "not editing", which is on the screen in that position if WordStar version 3.3 is displaying the BEGINNING MENU.

If "chk33" finds the "n" in that position, it executes the statement labeled "imen". (In WS.DEF, "imen" displays the NO-FILE popup menu for WordStar.)

If "chk33" doesn't find the "n" in that position, it executes the statement labeled "chkl".

The "chkl" statement looks for a colon (:) after the disk drive identifier in the first line of the WordStar MAIN MENU display.

If "chkl" finds a colon, it executes the statement labeled "emen". (In WS.DEF, "emen" displays the EDIT/BLOCK popup menu.)

If "chkl" doesn't find a colon, the menu program does nothing.

## MENU...MEND

label: MENU ["title"], [row], [column], [attribute]

. N (1781)

MEND

## Description

The MENU statement is the first statement in a Menu subroutine. A Menu subroutine creates a single-column popup menu. (For an ex-

ample of the format of a Menu subroutine, see "Menu Sub- routine Statements" in Chapter 1, "Creating Your Own Mouse Menu.")

Menus created with a Menu subroutine are bordered, single-column menus. The specific dimensions of a menu are determined by the number of items in a menu and the largest number of characters in either the longest menu item or the menu title.

When the menu is displayed, the first menu item (if any) is highlighted. The user chooses any menu item by moving the mouse until that item is highlighted, then clicking either mouse button. If the user clicks both mouse buttons, the equivalent of a NOTHING statement is executed and the menu disappears.

The MEND ("menu end") statement indicates the end of a Menu subroutine. Each Menu subroutine must have a MEND statement. MEND statements are not labeled.

### **Parameters**

label	Name of the Menu subroutine. All Menu subroutines must be labeled.
title	Text of the menu title, enclosed in double quotation marks (""). The menu title is limited to one line above the rest of the menu. If you don't specify a title, a blank line is used.
row	A number that specifies the row where the top-left corner of the menu appears. Be sure to specify a value that allows the entire menu to be displayed. (For example, if the menu contains 20 items and you choose a row value greater than 5, then some of the screen items will not be displayed on the 25-row screen.) If you don't specify a row, the top-left corner is in row 1.
column	A number that specifies the column where the top-left corner of the menu appears. If you don't specify a column, the top-left corner is in column 1.
attribute	A value that specifies how the menu is displayed on the screen. This can be "normal", "bold", or "inverse", or a decimal value that specifies particular foreground and background colors. (For more information, see "Parameters" in Chapter 1, "Creating Your Own Mouse Menu.") If you don't specify a value, inverse is used. The colors of the mouse pointer depend on the display attribute value for the menu. (For detailed infor- mation on how the interaction between the mouse pointer and menu display determine the colors of the pointer, see "Graphics Cursor" in Chapter 5, "The Mouse Inter- face.")

,

### Example

menu1:	MENU OPTION OPTION OPTION OPTION MEND	"Display Dire "Cancel" "A:",ex1 "B:",ex2 "C:",ex3	ectory",5,5,normal
ex1: ex2: ent: dir: a: b: c: s:	EXECUTE EXECUTE TYPE 13 TYPE "d TYPE "A TYPE "B TYPE "C TYPE 32	dir,s,a,ent dir,s,b,ent dir,s,c,ent "" ""	;dir A: ;dir B: ;dir C: ;simulate the enter key ;type dir ;type A: ;type B: ;type C: ;type a space

In this example, the MENU statement uses all four parameters. The menu title is "Display Directory". The top-left column of the menu is in row 5, column 5. The menu is displayed with a normal screen attribute.

The OPTION statements specify which statements are executed when the user chooses items from the menu. (For more information about the OPTION statement, see "OPTION" later in this chapter.)

## NOTHING

label: NOTHING

### Description

Use the NOTHING statement to specify that no action is taken when the user clicks a mouse button, moves the mouse, or chooses a menu option, or when a MATCH statement is executed. A NOTHING statement must be labeled.

#### Example

rightb: MATCH 1,11,NORMAL, ":",movmen,nul
.
.
movmen: POPUP 2,1
TEXT "====== CURSOR MOVEMENT =====""""
.
.
.
nul: NOTHING

This example from the WS.DEF Mouse Menu program determines which popup menu is displayed when the user clicks the right mouse button.

- If the MATCH statement finds the specified character, it executes the statement labeled "movmen" to display the CURSOR MOVEMENT popup menu.
- If the MATCH statement doesn't find the specified character, it executes the NOTHING statement labeled "nul", and the Mouse Menu program does nothing.

# OPTION

[label:] OPTION [text], [pointer]

## Description

OPTION statements define each menu item in a Menu subroutine. OPTION parameters define the text of the menu item and what happens when the user chooses the menu item.

OPTION statements are usually not labeled, although they can be. If they are labeled, the MAKEMENU program ignores the labels when assembling the source program.

## Parameters

text	Legend text for the menu item. The legend text must be enclosed in double quotation marks (""). If you don't specify legend text for a menu item, the menu displays a blank line for that item.
pointer	Label of the statement that is executed when the user chooses the menu item. If you don't include a pointer parameter, the menu is cleared from the screen when the user chooses the menu item. (For example, you'd leave out the pointer parameter for a "cancel menu" item.)

### Example

menu5:	MENU	"Format",5,5,normal
	OPTION	"Cancel"
	OPTION	"A:",ex16
	OPTION	"B:",ex17
	OPTION	"C:",ex18
	MEND	

This example shows OPTION statements that define four menu items. If the user chooses the first menu item, the menu is cleared from the screen because the OPTION statement has no pointer parameter. If the user chooses any other menu item, the specified statement is executed.

## POPUP...PEND

label: POPUP [row], [column], [attribute]

•

PEND

### Description

The POPUP statement is the first statement in a Popup subroutine. A Popup subroutine creates a multiplecolumn menu or a message box. (For an example of the format of a Popup subroutine, see "Popup Subroutine Statements" in Chapter 1, "Creating Your Own Mouse Menu.")

The PEND ("popup end") statement indicates the end of a Popup subroutine. Each Popup subroutine must have a PEND statement. PEND statements are not labeled.

### Parameters

label

row

Name of the Popup subroutine. All POPUP statements must be labeled.

A number that specifies the row where the top-left corner of the menu or message box appears. Be sure to specify a value that allows the entire menu or message box to be displayed. (For example, if the menu or message box takes up 20 lines and you choose a row value greater than 5, then some of the screen items will not be displayed on the 25-row screen.) If you don't specify a row, the top-left corner is in row 1.

- column A number that specifies the column where the top-left corner of the menu or message box appears. If you don't specify a column, the top-left corner is in column 1.
- attribute A value that specifies how the menu is displayed on the screen. This can be "normal", "bold", or "inverse", or a decimal value that specifies particular foreground and background colors. (For more information, see "Parameters" in Chapter 1, "Creating Your Own Mouse Menu.") If you don't specify a value, inverse is used. The colors of the mouse pointer depend on the display attribute value for the menu. (For detailed information on how the interaction between the mouse pointer and menu display determine the colors of the pointer, see "Graphics Cursor" in Chapter 5, "The Mouse Interface.")

### Examples

This example from the VC.DEF Mouse Menu program is a Popup subroutine for a multiple-column menu:

- DELETE: POPUP 2,1,inverse TEXT "Delete: Row Column " SELECT 1,9,3,DR SELECT 1,13,6,DC PEND
- DR: TYPE "/dr" DC: TYPE "/dc"

The POPUP statement defines the top-left corner of the menu as row 2, column 1. The menu contains the menu title and the menu items on the same line, as shown by the single TEXT statement. The two SELECT statements define the item selection areas. (For more information about SELECT and TEXT, see "SELECT" and "TEXT" later in this chapter.) This example from the WS.DEF Mouse Menu program is a Popup subroutine for a message box:

.. text "I text " ,, Left button - Displays Edit/Block menu text "| Right button - Displays Cursor movement menu \*\* text "| Both buttons - Displays Edit/File menu \*\* text " \*\* .. text "| Moving the mouse up, down, left, or right will text "| cause the cursor to move in that direction. ... text " ,, " select 1,18,10 pend

In this example, ASCII graphics characters are used to create solid double borders for the menu. The single SELECT statement is used to clear the message box from the screen. (Since the label for an executable statement is not included in the SELECT statement, clicking a mouse button simply clears the message box from the screen.)

# SELECT

SELECT row, column, width [, pointer]

## Description

The SELECT statement is used in Popup subroutines to define selection areas for items on the menu. It also specifies which statement is executed if the cursor is in the defined area. The defined area does not have to contain any text. (For more information about the TEXT statement, see "TEXT" later in this chapter.)

SELECT statements do not have labels.

Note The highlight in a menu or message box jumps from one defined selection area to another when the user moves the mouse. It is a good idea to define each part of a menu with a SELECT statement so that the movement of the highlight and the mouse are visually coordinated. However, make sure you don't define the same screen position with more than one SELECT statement.

### **Parameters**

row	A number that defines the horizontal start- ing point (row) of the item selection area. The defined area is relative to the "row" and "column" coordinates specified in the POPUP statement.
column	A number that defines the vertical starting

A number that defines the vertical starting point (column) of the item selection area. The defined area is relative to the "row" and "column" coordinates specified in the POPUP statement.

width	The number of characters in the item selec- tion area. If you don't specify a number, one character is assumed.
pointer	Label of the statement executed when the user chooses the menu item. If a pointer parameter isn't included, the menu is cleared from the screen.

### Examples

For examples of how to use SELECT statements, see "Popup Subroutine Statements" in Chapter 1, "Creating Your Own Mouse Menu" and "POPUP...PEND" earlier in this chapter.

## TEXT

TEXT "string"

## Description

TEXT is used in Popup subroutines to define the menu title and the legend text for menu items. It is similar to the "title" and "text" parameters in the MENU and OPTION statements, but allows text to be placed anywhere on the screen below and to the right of the upperleft corner specified for the popup menu.

## Parameter

string Defines the popup menu title or the legend text of a menu item. Text may include ASCII graphics characters. All text must be enclosed in double quotation marks (""). Text location on the screen is relative to the top-left corner of the popup menu. Text display attributes are determined by the attribute parameter in the POPUP statement.

### Examples

For examples of how to use TEXT statements, see "Popup Subroutine Statements" in Chapter 1, "Creating Your Own Mouse Menu," and "POPUP...PEND" earlier in this chapter.

## TYPE

label: TYPE key [,key ...]

### Description

A TYPE statement simulates typing one or more keystrokes. Keys are specified by enclosing the keystroke(s) in double quotation marks, using the ASCII code that corresponds to the key(s), using a predefined symbolic key name, or, for certain special-function keys, using the key's extended keyboard scan code.

**Note** All keys specified in the TYPE statement are inserted into a keyboard buffer when the menu program is running and are not output as keystrokes until the menu program becomes inactive.

### Parameters

labelName of the TYPE statement. Every<br/>TYPE statement must be labeled.keyName of the key. It can be:

- A single letter or number enclosed in double quotation marks (""), or a sequence of keystrokes enclosed in double quotation marks (such as "dir")
- A standard ASCII code (characters 0 through 127), or an extended ASCII code (characters 128 through 255)
- An extended keyboard scan code
- Any of the following predefined symbolic keys: "enter", "tab", "backsp", "esc".

**Note** If you want to simulate typing a quotation mark ("), use ASCII code 34.

The ASCII control characters (0 through 31) and extended keyboard scan codes that you can use with the TYPE statement are listed after the examples below. Refer to the IBM *BASIC* manual for a complete list of ASCII character codes.

### Examples

These TYPE statements use character strings to define the keystrokes:

dir: TYPE "dir" ;type the command "dir" a: TYPE "a:" ;type "a:"

This TYPE statement uses an ASCII code to simulate typing a space:

s: TYPE 32 ;type a space

These TYPE statements use extended keyboard scan codes to simulate the arrow keys:

lf:	TYPE	0,75	;simulate	the	left arrow key
rt:	TYPE	0,77	:simulate	the	right arrow key
up:	TYPE	0,72	;simulate;simulate	the	up arrow key
dn:	TYPE	0,80		the	down arrow key

#### ASCII Control Characters and Extended Keyboard Scan Codes

This section lists the functions of the ASCII control characters and the extended keyboard scan codes when used with the TYPE statement. (See the IBM *BASIC* manual for a complete list of ASCII codes.) It also lists the key sequences that cannot be simulated using the TYPE statement.

**Note** The output characteristics listed for particular key functions are for a mouse menu running at the DOS level. A standard application may not interpret all keyboard operations in the same way. Applications that reprogram or directly access the keyboard, or bypass the DOS system facilities for keyboard input, may not function correctly with mouse menus.

**ASCII Control Characters** The following table lists the function of each ASCII control character when used with the TYPE statement:

ASCII code	Key equivalent	ASCII code	Key equivalent
0	none	16	CONTROL-P
1	CONTROL-A	17	$\operatorname{CONTROL}-\mathbf{Q}$
2	CONTROL-B	18	CONTROL-R
3	CONTROL-C	19	CONTROL-S
4	CONTROL-D	20	CONTROL-T
<b>5</b>	CONTROL-E	21	CONTROL-U
6	CONTROL-F	22	CONTROL-V
7	CONTROL-G	<b>23</b>	CONTROL-W
8	backspace	<b>24</b>	CONTROL-X
9	horizontal tab	<b>25</b>	CONTROL-Y
10	line feed	<b>26</b>	CONTROL-Z
11	CONTROL-K	27	ESCAPE
12	CONTROL-L	<b>28</b>	CONTROL-
13	carriage return	29	CONTROL-]
14	CONTROL-N	30	CONTROL-
15	CONTROL-O	31	CONTROL

**Extended Keyboard Scan Codes** Extended keyboard scan codes have two components: a character code (which is always 0) and a scan code (for example, "0,75"). The tables below list the scan codes you can use with the TYPE statement and the character code 0 to simulate specific keystrokes. (Standard or extended ASCII characters cannot be used as extended keyboard scan codes.)

Keystroke(s)	Scan code	
HOME	71	
CONTROL-HOME	119	
up arrow key	72	
down arrow key	80	
left arrow key	75	
CONTROL-left arrow key	115	
right arrow key	77	
CONTROL-right arrow key	116	
END	79	
CONTROL-END	117	
PAGEUP	73	
CONTROL-PAGEUP	132	
PAGEDOWN	81	
CONTROL-PAGEDOWN	118	
CONTROL-PRINTSCREEN	114	
INSERT	82	
DELETE	83	
SHIFT-TAB	15	

Keystroke(s)	Scan code	Keystroke(s)	Scan code
		ILC SUI ORC(S)	100
F1	<b>59</b>	ALT-0	129
F2	60	ALT-1	120
F3	61	ALT-2	121
F4	62	ALT-3	122
F5	63	ALT-4	123
F6	64	ALT-5	124
F7	65	ALT-6	125
F8	66	ALT-7	126
F9	67	ALT-8	127
F10	68	ALT-9	128
shift-F1 (F11)	84	ALT	130
SHIFT-F2 $(F12)$	85	ALT-=	131
SHIFT-F3 (F13)	86	ALT-A	30
shift-F4 (F14)	87	ALT-B	48
SHIFT-F5 (F15)	88	ALT-C	46
SHIFT-F6 (F16)	89	ALT-D	32
SHIFT-F7 (F17)	90	ALT-E	18
SHIFT-F8 (F18)	91	ALT-F	33
SHIFT-F9 (F19)	92	ALT-G	<b>34</b>
SHIFT-F10 (F20)	93	ALT-H	35

### Mouse Menu Language Statements 2-27

Keystroke(s)	Scan code	Keystroke(s)	Scan code
$\frac{2209201010(2)}{CONTROL-F1 (F91)}$	04		03
CONTROL F2 (F22)	94 05		20
CONTROL F2 (F22)	90 06	ALT-J	37
CONTROL F3 $(F23)$	90		07 20
CONTROL - F4 (F24)	97		30 FO
CONTROL = F S (F 25)	90	ALT-IVI	30
CONTROL-F6 (F26)	99	ALT-IN	49
CONTROL-F7 (F27)	100	ALT-O	<b>24</b>
CONTROL-F8 $(F28)$	101	ALT-P	<b>25</b>
CONTROL-F9 (F29)	102	ALT-Q	16
CONTROL-F10 (F30)	103	ALT-R	19
ALT-F1 (F31)	104	ALT-S	31
ALT-F2 (F32)	105	ALT-T	20
alt-F3 (F33)	106	ALT-U	<b>22</b>
alt-F4 (F34)	107	ALT-V	47
ALT-F5 (F35)	108	ALT-W	17
ALT-F6 (F36)	109	ALT-X	45
ALT-F7 (F37)	110	ALT-Y	<b>21</b>
ALT-F8 (F38)	111	ALT-Z	44
ALT-F9 (F39)	112		
ALT-F10 (F40)	113		

#### Key Sequences That Cannot Be Simulated

Some key sequences cannot be simulated using the TYPE statement because they are suppressed in the ROM (Read-Only Memory) BIOS (Basic Input/Output System) keyboard routine. These include the following key combinations:

SHIFT-PRINTSCREEN

ALT-BACKSPACE

ALT-ESCAPE

ALT plus one of the following characters: []; ' ', . / \* ALT plus one of the following keys: ENTER, CONTROL, SHIFT, CAPS LOCK, NUM LOCK, SCROLL LOCK

ALT plus one of the arrow keys

CONTROL plus one of the following characters: 1 3 4 5 7 8 9 0 =; ',', ./

CONTROL plus one of the following keys: TAB, SHIFT, CAPS LOCK, NUM LOCK

CONTROL-BREAK

CONTROL-ALT-DELETE

CONTROL plus one of the arrow keys

CONTROL-INSERT

# 3 Sample Mouse Menu Programs

This chapter includes the source program listings for two basic Mouse Menu programs that simplify some of the tasks commonly performed on an IBM PC or compatible computer:

- The SIMPLE Mouse Menu program allows you to simulate pressing the ENTER, ESCAPE, INSERT, and arrow keys by either clicking or moving the mouse.
- The DOSOVRLY Mouse Menu program allows you to execute simple DOS commands by using the mouse to choose the commands from a menu rather than typing them on the keyboard.

You can type the source program listing for either mouse menu into a source file, run MAKEMENU to generate an executable Mouse Menu file, then start using the mouse menu immediately. Or you may want to use these listings as a basis for designing similar mouse menus that include additional features specific to your needs.

## SIMPLE Mouse Menu Program

The SIMPLE Mouse Menu allows you to use the mouse instead of typing a few commonly used keys. It is most helpful when used with applications that require frequent use of the arrow keys. For example, in many spreadsheet applications you must press the arrow keys on your keyboard to move the cursor on the screen. If the SIMPLE Mouse Menu is installed, you can move the cursor on the screen by simply moving the mouse on your desk top. In addition, clicking the left mouse button is equivalent to pressing the ENTER key, clicking the right mouse button simulates pressing the ESCAPE key, and clicking both mouse buttons at once is the same as pressing the INSERT 1.0

key. If your application does not use one of these keys, and you click the corresponding mouse button(s) by accident, the application will respond as if you had typed the key on the keyboard. You can correct the mistake as you would any typing error.

### SIMPLE Mouse Menu Source Program

; A menu to simulate arrow, enter, escape, ; and insert keys ; ; begin ent,es,ins,lf,rt,up,dn,32,16 ; ent: type enter es: type esc ins: type 0,82 ; lf: type 0,75 rt: type 0,77 up: type 0,72 dn: type 0,80

## **DOSOVRLY Mouse Menu Program**

The DOSOVRLY Mouse Menu allows you to choose several commonly used DOS commands at the DOS command level by simply pointing to an option on a menu and clicking the mouse. In other words, this mouse menu "overlays" DOS.

In addition to a main menu, the DOSOVRLY Mouse Menu program has two submenus, "Directory" and "Change Directory," which each list additional DOS commands. The source program for DOSOVRLY is a good example of how you might create a hierarchy of menus and submenus in one of your own Mouse Menu programs.

The DOSOVRLY Mouse Menu provides several features that are useful at the DOS command level:

- Moving the mouse left and right simulates pressing the left and right arrow keys. This allows you to edit your DOS commands by just moving the mouse.
- Clicking the right mouse button simulates pressing the ENTER key.

- Clicking both mouse buttons at once simulates typing cls, the DOS command for clearing the screen.
- Clicking the left mouse button displays the DOSOVRLY main menu. Options on this menu allow you to clear the screen, execute the DATE or TIME command, or choose "DIRECTORY" or "CHANGE DIRECTORY" to view the corresponding submenus of DOS commands. To select a menu option, move the highlight to the option, then click either mouse button. From within a submenu, you can choose an option to move to the other submenu or click the left mouse button to return to the main menu.

**Note** In the DOSOVRLY source program, the lb, rb, bb, lm, and rm parameters specified in the BEGIN statement are labels for EXECUTE statements. These EXECUTE statements branch off to the appropriate TYPE statements. This format demonstrates how you can use the EXECUTE statement in your Mouse Menu programs. This program could be simplified by branching directly from the BEGIN statement to the TYPE statements using: BEGIN mnu1, return, cls, left, right

### **DOSOVRLY Mouse Menu Source Program**

```
BEGIN 1b,rb,bb,1m,rm
                              ; main menu if left button
lb: execute mnul
rb: execute return ; type enter if right button
bb: execute cls ; type CLS if both buttons
lm: execute left ; left arrow if left motion
rm: execute right ; right arrow if right motion
mnul: MENU "Main Menu", 2,55, NORMAL
    option "cancel ", none
option "clear the screen ", cls
option "date ", date
option "time ", time
option "DIRECTORY ", mnu3
    option "CHANGE DIRECTORY ", mnu2
    MEND
'mnu2: MENU "Change Directory", 2,55, NORMAL
option "cancel ", none
option "cd ... ", cd1
option "cd ... ", cd2
option "DIRECTORY ", mnu3
                                        ", mnul
    option "MAIN MENU
    MEND
mnu3: MENU "Directory", 2, 55, NORMAL
    option "cancel ", none
```

option option option option option option option option	"dir "dir *.exe "dir *.bat "dir *.bak "dir *.sys "dir *.doc "dir *. "CHANGE DIRECTORY "MAIN MENU	",dir ",dire ",dirb ",dirx ",dirs ",dird ",dirz ",mnu2 ",mnul
; MEND		
none:	nothing	; do nothing
;	time onton	
recurn:	type enter	
loft ·	type CIS , encer	· left arrow
rert.	type $0,75$	, rent annow
date:	type "date" enter	, iight allow
time:	type "time" enter	<b>^</b>
cd1 ·	type "cd " ente	ar ar
cd2:	type "cd	
dir:	type "dir" enter	
dire:	type "dir *.exe"	enter
dirb:	type "dir *.bat"	enter
dirx:	type "dir *.bak"	enter
dirs:	type "dir *.svs"	, enter
dird:	type "dir *.doc"	, enter
dirz:	type "dir *."	-

# 4 Mouse Menu Messages

This chapter lists the messages that the MENU and MAKEMENU programs can display, along with descriptions of possible causes and what actions you may take in response to them.

#### **Conversion completed**

 The MAKEMENU program finished creating an executable menu file.
 No action is required. The DOS system prompt appears after MAKEMENU displays this message.

#### Error--Invalid statement: xxxx

Either the statement had no label, the statement's label didn't end with a colon (:), or the statement had an invalid parameter or syntax error.

Make sure that all statements (except the BEGIN statement and statements within Menu and Popup subroutines) are labeled. Make sure that all labels are followed by a colon. Check the statement syntax for correct use of commas and spaces.

#### Error--Label already used: /abe/

The same label was used to name more than one statement.

Make sure that labels are unique for each statement.

#### Error--Label not found: xxxx

A label specified for a parameter did not exist. Make sure that the statements have labels and that the labels are correct.

#### Illegal function call at address xxxxxx

A TYPE or EXECUTE statement had too many parameters, a SELECT statement defined the item selection area outside of the menu, or a SELECT or an OPTION statement had quotation marks placed incorrectly. Use the correct number of parameters, redefine the item selection area, or ensure that double quotation marks are used correctly to designate text strings.

#### Keyboard emulation off

 The Mouse Menu program is no longer running. No action is required.

#### Keyboard emulation on

 The Mouse Menu program is running. No action is required.

#### Menu installed

• You started up a Mouse Menu program, and it is running.

No action is required. Use the mouse menu as usual.

#### Name of file to convert:

You typed "makemenu" to create an executable Mouse Menu file.

Type in a Mouse Menu filename without the ".DEF" extension.

Mouse Interfaces

# **Designing Mouse Interfaces**

This section provides the technical information you need to design a mouse interface for one of your own application programs. To build mouse support into your program, you'll include calls to a set of mouse functions. You can make mouse function calls from the BASIC interpreter, from assembly-language programs, and from programs in high-level languages such as Microsoft QuickBASIC, Pascal, FORTAN, and C.

Chapter 5, "The Mouse Interface," describes the interface between the computer screen and the Microsoft Mouse software.

Chapter 6, "Mouse Function Descriptions," describes the input, output, and operation of each function call your program can make to the mouse driver.

Chapter 7, "Making Mouse Function Calls," describes how to make mouse function calls from interpreted BASIC, assembly, and high-level-language programs. It also includes the source listing for the Piano demonstration program that came in your Microsoft Mouse package, and explains how to specify eight mouse cursor shapes.

Chapter 8, "Writing Mouse Programs for IBM EGA Modes," explains how to use the Microsoft EGA Register Interface when your program includes mouse support for IBM enhanced graphics modes D, E, F, and 10.
# 5 The Mouse Interface

This chapter describes the interface between the mouse software and the IBM PC. In particular, it discusses how the mouse software uses certain features of the computer to create a cursor on the screen and control its movement. This chapter talks about:

- Screen modes
- The virtual screen
- Graphics and text cursors
- Mouse buttons
- The mouse unit of distance: the mickey
- The internal cursor flag

Since many of the mouse functions use the interface, it is important for you to read the following sections carefully and understand them before you use the functions in application programs.

# Screen Modes

The screen mode defines the number of pixels (points of light) on the screen and the types of objects that appear on the screen. The available screen modes depend on the adapter in your computer. These screen modes and the adapters on which they are supported are listed in the table on the next page. (Specific information about each screen mode is given in the documentation for each adapter.)

Screen Mode	Display Adapter	Virtual Screen (X x Y)	Cell Size	Bits/Pixel (Graphics Modes)
0	C,E,3270	640 x 200	16 x 8	
1	C,E,3270	640 x 200	16 x 8	·
<b>2</b>	C,E,3270	640 x 200	8 x 8	
3	C,E,3270	640 x 200	8 x 8	
4	C, E, 3270	640 x 200	2 x 1	2
5	C,E,3270	640 x 200	2 x 1	2
6	C, E, 3270	640 x 200	1 x 1	1
7	M,E,3270	640 x 200	8 x 8	
D	E	640 x 200	16 x 8	2
E	$\mathbf{E}$	640 x 200	1 x 1	1
F	$\mathbf{E}$	640 x 350	1 x 1	1
10	E	640 x 350	1 x 1	1
30	3270	720 x 350	1 x 1	1
	Н	720 x 348	1 x 1	1

Display adapters:

M = IBM Monochrome Display/Printer Adapter

C = IBM Color/Graphics Monitor Adapter

E = IBM Enhanced Graphics Adapter

3270 = IBM All Points Addressable Graphics Adapter (3270 PC)

H = Hercules Monochrome Graphics Card

# The Virtual Screen

The mouse software operates on the computer screen as if it were a *virtual screen* of individual points arranged in a matrix of horizontal and vertical points. The "Virtual Screen" column in the table above gives the number of horizontal and vertical points in the matrix for each supported screen mode. Whenever interrupt 10h is called to change the screen mode, the mouse software intercepts the call and determines which virtual screen to use. The mouse software also reads the screen mode and chooses the appropriate virtual screen whenever mouse function 0 is called to reset default parameter values in the mouse software.

Regardless of the screen mode, the software uses a pair of virtual-screen coordinates to locate an object on the screen. Each pair of coordinates defines a point on the virtual screen. The horizontal coordinate is given first.

Many mouse functions take virtual-screen coordinates as input or return them as output. Whenever you refer to a pixel or character in a mouse function, make sure that the horizontal and vertical coordinates are the correct values for the given screen mode. The mouse functions always return correct values for the given screen mode.

In graphics modes 6, E, F, 10, and 30, and for the Hercules graphics display modes, each point in the virtual screen has a one-to-one correspondence with each pixel on the screen. In these modes, the full range of coordinates in the "Virtual Screen" column is permitted.

In graphics modes 4 and 5, the screen has half the number of pixels that it has in the other graphics modes. To compensate, the mouse software uses only even-numbered horizontal coordinates. This means that every other point in the virtual screen corresponds to a pixel.

In text modes 2, 3, and 7, only characters are permitted on the screen. Each character is an 8-by-8-pixel group (see the "Cell Size" column in the table).

Because the mouse software cannot access the individual pixels in a character, it uses the coordinates of the pixel in the upper-left corner of the cell for the character's location. Since each character is an 8-by-8-pixel group, both the horizontal and vertical coordinates are multiples of eight.

For example, the character in the upper-left corner of the screen has the coordinates (0,0), and the character immediately to the right of it has the coordinates (8,0).

In text modes 0 and 1, as in text modes 2, 3, and 7, only characters are permitted on the screen. Each character is a 16-by-8-pixel group (see "Cell Size" in the table).

The mouse software uses the coordinates of just one pixel in a character for the character's location. However, the screen has only half as many pixels as in modes 2, 3, and 7. To compensate, the mouse software uses horizontal coordinates that are multiples of 16. Graphics modes 6, E, F, 10, and 30

#### Graphics modes 4 and 5

Text modes 2, 3, and 7

#### Text modes 0 and 1

For example, the character in the upper-left corner of the screen has the coordinates (0,0), and the character immediately to the right of it has the coordinates (16,0).

## **Graphics and Text Cursors**

The mouse has three different cursors:

- The graphics cursor is a shape (for example, an arrow) that moves over the images on the screen.
- The software text cursor is a character attribute (for example, an underscore) that moves from character to character on the screen.
- The hardware text cursor is a flashing block, halfblock, or underscore that moves from character to character on the screen.

The mouse software supports only one of these cursors on the screen at any time. In your application program, you can choose which cursor is on the screen, and even switch back and forth between cursors.

Mouse functions 9 and 10 permit you to define the characteristics of these cursors in your application program. You can define the characteristics yourself, or use the characteristics of the sample cursors provided. For more information about the sample cursors, see Chapter 7, "Making Mouse Function Calls."

The following paragraphs describe the cursors in detail.

## **Graphics Cursor**

The graphics cursor, used when the computer is in one of the graphics modes, is a block of pixels.

- In modes 6, E, F, 10, and 30, and for the Hercules Monochrome Graphics Card, it is 256 pixels in a 16by-16-pixel square.
- In modes 4 and 5, it is 128 pixels in an 8-by-16-pixel square.

As the mouse moves, the block moves over the screen and interacts with the pixels directly under it. This interaction creates the cursor shape and background.

The interaction between the cursor points and screen pixels is defined by two 16-by-16-bit arrays: the screen mask and the cursor mask.

- The screen mask determines whether the cursor pixel is part of the shape or background.
- The cursor mask determines how the pixel under the cursor contributes to the color of the cursor.

In your application program, you can specify the shapes of the screen mask and cursor mask by defining them as arrays and passing these arrays as parameters in a call to mouse function 9. (For more information, see the description of function 9 in Chapter 6, "Mouse Function Descriptions.")

The interactions between the screen mask and the cursor mask differ somewhat between graphics modes 4, 5, 6, F, and 30 and the IBM Enhanced Graphics Adapter graphics modes E and 10.

In modes 6, F, and 30, and for the Hercules Graphics Card, each bit in the masks corresponds to a pixel in the cursor block. In modes 4 and 5, each pair of bits corresponds to a pixel.

To create the cursor, the mouse software operates on the data in the computer's screen memory that defines the color of each pixel on the screen. First, the software logically ANDs the screen mask with the 256 bits of data that define the pixels under the cursor. Then, it logically XORs the cursor mask with the result of the AND operation. The following table shows how these operations affect the individual screen bits:

If the screen mask bit is	And the cursor mask bit is	The resulting screen bit is
0	0	0
0	1	1
1	0	unchanged
1	1	inverted

In modes 6, F, and 30, and for the Hercules Graphics Card, each screen bit defines the color of a single pixel. Therefore, one bit in the screen mask and one bit in the Screen mask and cursor mask

Modes 4, 5, 6, F, and 30, and the Hercules Graphics Card

cursor mask define the pixel's color when the cursor is over it. For example, if the first bit in the screen mask is 1 and the first bit in the cursor mask is 0, then the upper-left corner of the cursor block is transparent. In modes 4 and 5, each *pair* of screen bits defines the color of a pixel. Therefore, a pair of bits in the screen mask and a pair in the cursor mask define a pixel's color. Modes E and 10 In EGA four-plane modes E and 10, as in modes 6, F, and 30, each bit in the screen mask and cursor mask corresponds to a pixel in the cursor block. The cursor mask and screen mask are stored in off-screen memory. Each plane has its own cursor mask and screen mask. Therefore, for each plane, the "resulting screen" bit" in the table on the previous page is actually the bit used in the color table lookup on the EGA. In modes E and 10, the mouse driver automatically sets the write mask register on the EGA to all 1s. Therefore, when your application program calls mouse function 9 to set the cursor shape, the cursor pixels may be either black, white, transparent, or inverted. The mouse driver does not support color cursors in modes E and 10. Whenever a mouse function refers to the graphics cursor Graphics cursor "hot spot" location, it gives the point on the virtual screen that lies directly under the cursor's hot spot. The hot spot is the point in the cursor block that the mouse software uses to determine the cursor coordinates. You can define the hot spot in the cursor block by passing the horizontal and vertical coordinates of the point to mouse function 9. The coordinates can be within the range -16 to 16; however, in modes 4 and 5, the horizontal coordinate must be an even number. In all graphics modes, the coordinates are relative to the upper-left corner of the cursor block. Software Text Cursor The software text cursor is used when the computer is in one of the text modes. The text cursor affects how characters appear on the screen. Unlike the graphics cursor, the text cursor usually does not have a shape of its own. Instead, it changes the

character attributes (such as foreground and background color, intensity, and underscoring) of the character directly under it. If the cursor does have a shape of its own, it is one of the 256 characters in the ASCII character set.

The effect of the text cursor on the character under it is defined by two 16-bit values called the *screen mask* and the *cursor mask*.

- The screen mask determines which of the character's attributes are preserved.
- The cursor mask determines how these attributes are changed to yield the cursor.

To create the cursor, the mouse software operates on the data that defines each character on the screen. The software first logically ANDs the screen mask and the 16 bits of screen data for the character under the cursor. It then logically XORs the cursor mask and the result of the AND operation.

The 16 bits of screen data for each character take the following form:

15	14	12	11	10	8	7	0
b	bckgd		i	foregd		char	
						L	

odd address (M + 1)

even address (M)

In Figure 5.1:

Bit(s)	Purpose
15 (b)	Sets blinking or nonblinking character
12-14 (bckgd)	Set the background color
11 (i)	Sets high intensity or medium intensity
8–10 (foregd)	Set the foreground color
0-7 (char)	ASCII value of the character

The range of values for each field depends on the display adapter in the computer. (See the display adapter documentation for details.) Screen mask and cursor mask

5.1 Screen Data for Character The screen mask and cursor mask are divided into the same fields as those shown in Figure 5.1. The values of these fields in the screen mask and cursor mask define the character's new attributes when the cursor is over the character.

For example, to invert the foreground and background colors, the screen mask and cursor mask should have the values shown in Figure 5.2:

5.2 Sample		b	bckgd	i	foregd	char	=
Values	screen mask	0	111	0	111	11111111	&H77FF
	cursor mask	0	111	0	111	00000000	&H7700

In your application program, you can define the values of the screen mask and cursor mask by passing their values as parameters to mouse function 10. (For more information, see the description of function 10 in Chapter 6, "Mouse Function Descriptions.")

Whenever a mouse function refers to the text cursor location, it gives the coordinates of the character under the cursor. The text cursor does not have a hot spot.

## Hardware Text Cursor

The hardware text cursor is another cursor that can be used when the computer is in one of the text modes.

The hardware text cursor is actually the computer's cursor (the one you see on the screen after the DOS systemlevel prompt). The mouse software allows you to adapt this cursor to your needs.

The hardware cursor is 8 pixels wide and 8 to 14 pixels tall. Each horizontal set of pixels forms a line called a scan line. There are 8 to 14 scan lines.

A scan line can be on or off. If a scan line is on, it appears as a flashing bar on the screen. If a scan line is off, it has no effect on the screen. Your program can define which lines are on and which are off by passing the numbers of the first and last lines in the cursor to mouse function 10.

The number of lines in the cursor depends on the display adapter in the computer.

Scan lines

- If the computer has an IBM Color/Graphics Monitor Adapter, the cursor has 8 lines, numbered 0 to 7.
- If the computer has an IBM Monochrome Display and Printer Adapter, the cursor has 14 lines, numbered 0 to 13.
- If the computer has an IBM Enhanced Graphics Adapter and an IBM Color Display, the cursor has 8 lines, numbered 0 to 7. If the computer has an IBM Enhanced Graphics Adapter and an IBM Enhanced Color Display, the cursor has 14 lines, numbered 0 to 13.

Note Only block cursors are supported on the 3270 PC.

# **Mouse Buttons**

Mouse functions 5 and 6 read the state of the mouse buttons and keep a count of the number of times the buttons are pressed and released.

A button state is *pressed* if the button is down, and *released* if the button is up. When a mouse function returns the state of the buttons, it returns an integer value in which the first 2 bits are set or cleared. Bit 0 represents the state of the left button, and bit 1 represents the state of the right button. If a bit is set (equal to 1), the button is down. If a bit is clear (equal to 0), the button is up.

The mouse software increments a counter each time the corresponding button is pressed or released. The software sets a counter to 0 after a reset (mouse function 0) or after a counter's contents are read.

# Mouse Unit of Distance: The Mickey

The motion of the mouse track ball is translated into values that express the direction and duration of the motion. The values are given in a unit of distance called a *mickey*, which is approximately 1/200 inch.

When the user slides the mouse across a desk top, the mouse hardware passes to the mouse software a horizontal and vertical *mickey count*; that is, the number of mickeys the mouse ball has rolled in the horizontal and vertical directions. The software uses the mickey count to move the cursor a certain number of pixels on the screen.

The number of pixels that the cursor moves does not have to correspond one-to-one with the number of mickeys the track ball rolls. The mouse software defines a *sensitivity* for the mouse—the number of mickeys required to move the cursor 8 pixels on the screen. The sensitivity determines the rate at which the cursor moves on the screen.

In your application program, you can define the mouse's sensitivity by passing a mickey count to mouse function 15. The mickey count can be any value from 1 to 32767. For example, if you pass a count of 8, the sensitivity is 8 mickeys per 8 pixels. That is, the cursor will move 1 pixel for each mickey the ball rolls, or one character for every 8 mickeys the ball rolls.

## The Internal Cursor Flag

The mouse software maintains an internal flag that determines when the cursor should be displayed on the screen. The value of this flag is always 0 or less.

- When the flag is 0, the cursor is displayed.
- When the flag is any other value, the cursor is hidden.

Application programs cannot access this flag directly. To change the flag's value, the program must call mouse functions 1 and 2. Function 1 *increments* the flag; function 2 *decrements* it. Initially, the flag's value is -1, so a call to function 1 displays the cursor.

Your program can call either function 1 or function 2 any number of times, but if it calls function 2, it must call function 1 later to restore the flag's previous value. For example, if the cursor is on the screen and your program calls function 2 five times, it must also call function 1 five times to return the cursor to the screen.

#### Mouse sensitivity

If the cursor is displayed, any additional calls to function 1 have no effect on the internal cursor flag, so one call to function 2 will always hide the cursor. In addition, your program can call mouse function 0 or change screen modes to reset the flag to -1 and hide the cursor.

# 6 Mouse Function Descriptions

This chapter describes the input, output, and operation of each mouse function call. The actual statements required to make the function calls depend on the programming language you're using. For specific instructions on making calls from the BASIC interpreter, assemblylanguage programs, and high-level-language programs, refer to Chapter 7, "Making Mouse Function Calls."

**Note** If you are designing an application program with mouse support that uses extended graphics modes D, E, F, and 10 on the IBM Enhanced Graphics Adapter (EGA), the program must interact with the adapter through the Microsoft EGA Register Interface. For instructions on using the EGA Register Interface, see Chapter 8, "Writing Mouse Programs for IBM EGA Modes."

# **Mouse Functions**

The following table lists the mouse functions by function number:

Number	Function
0	Mouse Reset and Status
1	Show Cursor
2	Hide Cursor
3	Get Button Status and Mouse Position
4	Set Mouse Cursor Position
5	Get Button Press Information
6	Get Button Release Information
7	Set Minimum and Maximum Horizontal Cursor Position
8	Set Minimum and Maximum Vertical Cursor Position
9	Set Graphics Cursor Block
10	Set Text Cursor
11	Read Mouse Motion Counters
12	Set Interrupt Subroutine Call Mask and Address
13	Light Pen Emulation Mode On
14	Light Pen Emulation Mode Off
15	Set Mickey/Pixel Ratio
16	Conditional Off
19	Set Double-Speed Threshold
20	Swap Interrupt Subroutines
21	Get Mouse Driver State Storage Requirements
<b>22</b>	Save Mouse Driver State
23	Restore Mouse Driver State
29	Set CRT Page Number
30	Get CRT Page Number

Each function description in this chapter specifies the following:

- The parameters required to make the call (input) and the expected return values (output)
- Any special considerations regarding the function
- A BASIC interpreter program fragment that illustrates how to use the call. (For more information about calling mouse functions from the BASIC interpreter, see Chapter 7, "Making Mouse Function Calls.")

In the descriptions of all functions, the parameter names M1%, M2%, M3%, and M4% are dummy variable names. When making a call, use the names of the variables that you want to pass.

The dummy variable names include the percent sign (%) to emphasize that only integer variables can be used as parameters. Constants, single-precision variables, and double-precision variables are not allowed.

If the function description does not specify an input for a parameter, you don't need to supply a value for that parameter before making the call. If the function description does not specify an output value for a parameter, the parameter's value is the same before and after the call.

**Caution** All function calls require four parameters. The mouse software does not check input values, so be sure that the values you assign to the parameters are correct for the given function and screen mode. If you pass the wrong number of parameters or assign incorrect values, you will get unpredictable results.

# Function 0: Mouse Reset and Status

Input	Output
M1%=0	M1%=mouse status
	M2%=number of buttons
	(2  if  M1% = -1,  otherwise  0)

Function 0 returns the current status of the mouse hardware and software. The mouse status is 0 if the mouse hardware and software are not installed or -1 if the hardware and software are installed.

This function also resets the mouse driver to the following default values:

Parameter	Value
cursor position	screen center
internal cursor flag	-1 (cursor hidden)
graphics cursor	arrow
text cursor	reverse video
interrupt call mask	all 0 (no interrupt subroutine specified)
light pen emulation mode	enabled
horizontal mickey/pixel ratio	8 to 8
vertical mickey/pixel ratio	16 to 8
horizontal min/max cursor position	0/current display-mode virtual screen x-value minus 1
vertical min/max cur- sor position CRT page number	0/current display-mode virtual screen y-value minus 1 0
1 0	

## Example

This program fragment verifies that the mouse is installed, and returns an error message if it is not.

100 200 ٠ Is mouse present? If not, error. ٠ 300 400 DEF SEG=0 500 MSEG=256\*PEEK(51\*4+3)+PEEK(51\*4+2) 600 MOUSE=256 \* PEEK (51 \* 4 + 1) + PEEK (51 \* 4) + 2 IF MSEG OR (MOUSE-2) THEN 60 PRINT "Mouse not found":END 700 800 900 DEF SEG=MSEG 1000 IF PEEK (MOUSE-2)=207 then 700 M1%=0 1100 CALL MOUSE (M1%, M2%, M3%, M4%) 1200 CALL MOUSE (M1%, M2%, 1300 IF NOT (M1%) THEN 700

# **Function 1: Show Cursor**

Input Output

M1% = 1

Function 1 increments the internal cursor flag and, if the flag is 0, displays the cursor on the screen. The cursor tracks the motion of the mouse, changing position as the mouse changes position.

The current value of the internal cursor flag depends on the number of calls that have been made to functions 1 and 2. (For more information, see "The Internal Cursor Flag" in Chapter 5, "The Mouse Interface.") The default flag value is -1; therefore, after a reset (function 0), the program must call function 1 to redisplay the cursor.

If the internal cursor flag is already 0, this function does nothing.

#### Example

100 ٠ 200 Show the cursor 300 400 M1%=1 CALL MOUSE (M1%, M2%, M3%, M4%) 500

# **Function 2: Hide Cursor**

Input Output

M1% = 2

Function 2 removes the cursor from the screen and decrements the internal cursor flag. When the cursor is hidden, it continues to track the motion of the mouse, changing position as the mouse changes position.

Use this function before you change any area of the screen that contains the cursor. This will ensure that the cursor won't affect the data written to the screen.

**Note** If your program changes the screen mode, function 2 is called automatically so that the cursor will be drawn correctly the next time it is displayed.

Remember that each time your program calls function 2, it must call function 1 later to restore the internal cursor flag to its previous value. (For more information, see "The Internal Cursor Flag" in Chapter 5, "The Mouse Interface.")

**Note** At the end of your program, call function 2 to hide the mouse cursor. If the internal cursor flag is 0 when the program ends, the mouse cursor will remain on the screen.

### Example

```
100 '
200 ' Hide the cursor
300 '
400 M1%=2
500 CALL MOUSE(M1%, M2%, M3%, M4%)
```

# Function 3: Get Button Status and Mouse Position

Input	Output
M1%=3	M2%=button status
	M3%=cursor position (horizontal)
	M4%=cursor position (vertical)

Function 3 returns the state of the left and right mouse buttons and the horizontal and vertical coordinates of the cursor.

The button status is a single integer value. Bit 0 represents the left button; bit 1 represents the right button. These bits are 1 if the corresponding button is down, and 0 if it is up.

The cursor coordinates are always within the range of minimum and maximum values for the virtual screen. (For more information, see "The Virtual Screen" in Chapter 5, "The Mouse Interface.")

#### Example

100 '
200 ' Check button status
300 '
400 M1%=3
500 CALL MOUSE (M1%, M2%, M3%, M4%)
600 IF M2% AND 1 THEN PRINT "Left button down."
700 IF M2% AND 2 THEN PRINT "Right button down."

## Function 4: Set Mouse Cursor Position

#### Input

Output

M1%=4 M3%=new cursor coordinate (horizontal) M4%=new cursor coordinate (vertical)

Function 4 sets the cursor to the specified horizontal and vertical screen coordinates. The parameter values must be within the horizontal and vertical coordinate ranges for the virtual screen.

If the screen is not in a mode with a cell size of  $1 \ge 1$ , the parameter values are rounded to the nearest horizontal or vertical coordinate values permitted for the current screen mode. (For more information, see "The Virtual Screen" in Chapter 5, "The Mouse Interface.")

## Example

Assume that HMAX and VMAX are the maximum horizontal and vertical coordinate values for the virtual screen. This call to function 4 sets the cursor to the center of the screen:

100 '
200 ' Put cursor in center of screen
300 '
400 M1%=4
500 M3%=INT(HMAX/2)
600 M4%=INT(VMAX/2)
700 CALL MOUSE(M1%, M2%, M3%, M4%)

# Function 5: Get Button Press Information

Input	Output			
M1% = 5	M1%=button status			
M2%=button	M2%=number of button presses			
	M3%=cursor (horizontal) at last press			
	M4%=cursor (vertical) at last press			

Function 5 returns the following:

- The current status of the specified button
- The number of times the specified button was pressed since the last call to this function
- The horizontal and vertical coordinates of the cursor the last time the specified button was pressed

The parameter M2% specifies which button is checked. If this parameter is 0, the left button is checked. If this parameter is 1, the right button is checked.

The button status is a single integer value. Bit 0 represents the left button and bit 1 represents the right button. These bits are 1 if the corresponding button is down, and 0 if it is up.

The number of button presses is always in the range 0 to 32767. Overflow is not detected. The count is set to 0 after the call.

The values for the horizontal and vertical coordinates are in the ranges defined by the virtual screen. These values represent the cursor position when the button was last pressed, not the cursor's current position.

### Example

100 '
200 ' Get button press information
300 '
400 M1%=5
500 M2%=0 ' left button
600 CALL MOUSE(M1%, M2%, M3%, M4%)
700 IF(M1% AND 1) THEN PRINT "Left button down."

# Function 6: Get Button Release Information

Input	Output			
M1% = 6	M1%=button status			
M2%=button	M2%=number of button releases			
	M3%=cursor (horizontal) at last release			
	M4%=cursor (vertical) at last release			

Function 6 returns the following:

- The current status of the specified button
- The number of times the specified button was released since the last call to this function
- The horizontal and vertical coordinates of the cursor the last time the specified button was released

The parameter M2% specifies which button is checked. If this parameter is 0, the left button is checked. If this parameter is 1, the right button is checked.

The button status is a single integer value. Bit 0 represents the left button and bit 1 represents the right button. These bits are 1 if the corresponding button is down, and 0 if it is up.

The number of button releases is always in the range 0 to 32767. Overflow is not detected. The count is set to 0 after the call.

The values for the horizontal and vertical coordinates are in the ranges defined by the virtual screen. These values represent the cursor position when the button was last released, not the cursor's current position.

## Example

```
100 '
200 ' Get button release information
300 '
400 M1%=6
500 M2%=1 ' right button
600 CALL MOUSE(M1%, M2%, M3%, M4%)
700 IF (M1% AND 2) THEN PRINT "Right button down."
```

# Function 7: Set Minimum and Maximum Horizontal Cursor Position

#### Input

Output

M1% = 7

M3%=minimum position

M4%=maximum position

Function 7 sets the minimum and maximum horizontal cursor coordinates on the screen. All cursor movement after the call to function 7 is restricted to the specified area. The minimum and maximum values are defined by the virtual screen. (For more information, see "The Virtual Screen" in Chapter 5, "The Mouse Interface.")

If the minimum value is greater than the maximum value, the two values are swapped.

#### Example

100 '
200 ' Limit cursor to horizontal positions below 150
300 '
400 M1%=7
500 M3%=0
600 M4%=150
700 CALL MOUSE(M1%, M2%, M3%, M4%)

# Function 8: Set Minimum and Maximum Vertical Cursor Position

Input

Output

M1%=8 M3%=minimum position M4%=maximum position

Function 8 sets the minimum and maximum vertical cursor coordinates on the screen. After function 8 is called, cursor movement is restricted to the specified area. The minimum and maximum values are defined by the virtual screen. (For more information, see "The Virtual Screen" in Chapter 5, "The Mouse Interface.")

If the minimum value is greater than the maximum value, the two values are swapped.

## Example

100 '
200 ' Limit cursor to vertical positions between
300 ' 100 and 150
400 '
500 M1%=8
600 M3%=100
700 M4%=150
800 CALL MOUSE(M1%, M2%, M3%, M4%)

## Function 9: Set Graphics Cursor Block

Input	Output
 M1%=9	
M2%=cursor hot spot (horizontal)	
M3%=cursor hot spot (vertical)	

M4%=pointer to screen and cursor masks

Function 9 defines the shape, color, and center of the graphics cursor (the cursor used when the computer is in graphics mode). Your program must call function 1 to display the graphics cursor.

Function 9 uses the values found in the screen mask and cursor mask to build the cursor shape and color.

To pass the screen mask and cursor mask, assign their values to an integer array (packed 2 bytes per integer) and use the first element of the array as the parameter M4% in the call (see the example on the next page).

The cursor hot spot values must define one pixel within the cursor. The values must be within the range -16 to 16.

For more information about the screen mask, cursor mask, and the graphics cursor hot spot, see "Graphics and Text Cursors" in Chapter 5, "The Mouse Interface."

**Note** For more information about calling function 9 from programs in assembly language, see the section "Making Calls from Assembly-Language Programs" in Chapter 7, "Making Mouse Function Calls."

## Example

To define a cursor in high-resolution graphics mode, first assign the values to the cursor array, then make the call:

100 200	' Define the screen mas	k
300	1	
400	CURSOR(0,0) = & HFFFF	1111111111111111111
500	CURSOR(1,0) = & HFFFF	111111111111111111
600	CUBSOR(2,0) = HEFEFE	1111111111111111111111
700	CURSOR(3,0) = & HFFFF	11111111111111111111111
800	CURSOR(4, 0) = CHEFEE	111111111111111111111111
900	CURSOR(5, 0) = CHEFFE	111111111111111111111111
1000	CUPSOR(5, 0) = CUFFFF	111111111111111111111111
1100	CUPSOR(7, 0) - CUFFFFF	111111111111111111111111
1 200	CURSOR(7,0) = CUFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF	* 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 200	CURSOR(0,0) = CUREEEE	***********
1400	$CURSOR(9,0) = \alpha HFFFF$	
1400	$CURSOR(10,0) = \alpha HFFFF$	
1500	CURSOR(11,0) = & HFFFF	
1600	CURSOR(12,0) = & HEFFF	
1700	CURSOR(13,0) = & HEFFF	
1800	CURSOR(14,0) = & HEFFFF	
1900	CURSOR(15,0) = & HFFFF	,1111111111111111111
2000		
2100	Define the cursor ma	sk
2200		
2300	CURSOR(0, 1) = & H8000	100000000000000000000000000000000000000
2400	CURSOR(1,1) = & HEOOO	111000000000000000000000000000000000000
2500	CURSOR(2,1) = & HF800	1111100000000000000
2600	CURSOR(3,1) = & HFEOO	1111111000000000
2700	CURSOR(4,1) = & HD800	11011000000000000
2800	CURSOR(5,1) = &HOCOO	0000110000000000
2900	CURSOR(6, 1) = &HO6OO	0000011000000000
3000	CURSOR(7, 1) = & HO300	0000001100000000
3100	CURSOR(8,1) = &HOOOO	000000000000000000000000000000000000000
3200	CURSOR(9,1) = &HOOOO	000000000000000000000000000000000000000
3300	CURSOR(10, 1) = & HOOOO	000000000000000000000000000000000000000
3400	CURSOR(11, 1) = & HOOOO	000000000000000000000000000000000000000
3500	CURSOR(12, 1) = & HOOOO	000000000000000000000000000000000000000
3600	CURSOR(13, 1) = & HOOOO	000000000000000000000000000000000000000
3700	CURSOR(14, 1) = & HOOOO	1000000000000000000
3800	CURSOR(15, 1) = & HOOOO	'0000000000000000000
3900		
4000	' Set the mouse cursor	shape, color, and
4050	' hot spot	
4100	t	
4200	M1%=9	
4300	M2%=0 ' horizontal hot	spot
4400	M3%=0 ' vertical hot s	pot
4500	CALL MOUSE (M1%, M2%, M3	%, CURSOR (0,0))

# Function 10: Set Text Cursor

#### Input

Output

M1% = 10

M2%=cursor select

M3%=screen mask value/scan line start

M4%=cursor mask value/scan line stop

Function 10 selects the software or hardware text cursor. Your program must call function 1 to display the text cursor.

The value of the parameter M2% specifies which cursor is selected. If M2% is 0, the software text cursor is selected. If M2% is 1, the hardware text cursor is selected.

If the software text cursor is selected, parameters M3%and M4% must specify the screen mask and cursor mask. These masks define the attributes of a character when the cursor is over it. The mask values depend on the display adapter in the computer. (For more information, see "Software Text Cursor" in Chapter 5, "The Mouse Interface.")

If the hardware text cursor is selected, parameters M3% and M4% must specify the line numbers of the first and last scan lines in the cursor. These line numbers depend on the display adapter in the computer. (For more information, see "Hardware Text Cursor" in Chapter 5, "The Mouse Interface.")

100	1
200	' Create text cursor that inverts foreground
300	' and background colors
400	1
500	M1%=10
600	M2%=0 'select software text cursor
700	M3%=&HFFFF ' screen mask
800	M4 <sup>°</sup> / <sub>2</sub> =&H7700 ' cursor mask
900	CALL MOUSE (M1%, M2%, M3%, M4%)

# Function 11: Read Mouse Motion Counters

Inpu	t	0	u	t	p	u	t

M1%=11 M3%=count (horizontal) M4%=count (vertical)

Function 11 returns the horizontal and vertical mickey count since the last call to this function. The mickey count is the distance that the mouse has moved, in 1/200 inch increments. (For more information, see "Mouse Unit of Distance: The Mickey" in Chapter 5, "The Mouse Interface.")

The mickey count is always within the range -32768 to 32767.

- A positive horizontal count indicates motion to the right. A negative horizontal count indicates motion to the left.
- A positive vertical count indicates motion to the bottom of the screen. A negative vertical count indicates motion to the top of the screen.

Overflow is ignored. The mickey count is set to 0 after the call is completed.

## Example

```
100 '
200 ' Get the mickey count
300 '
400 M1%=11
500 CALL MOUSE (M1%, M2%, M3%, M4%)
```

# Function 12: Set Interrupt Subroutine Call Mask and Address

Inj	pu	t
-----	----	---

Output

M1%=12 M3%=call mask M4%=subroutine address

Function 12 sets the call mask and subroutine address for mouse hardware interrupts.

The mouse hardware interrupts automatically stop execution of your program and call the specified subroutine whenever one or more of the conditions defined by the call mask occur. When the subroutine finishes, your program continues execution at the point of interruption.

The call mask is a single integer value that defines which conditions cause an interrupt. Each bit in the call mask corresponds to a specific condition, as shown in the following table:

Mask bit	Condition	
0	Cursor position changes	
1	Left button pressed	
2	Left button released	
3	Right button pressed	
4	Right button released	
5 - 15	Not used	

To enable the subroutine for a given condition, set the corresponding call mask bit to 1 and pass the mask as parameter M3%.

To disable the subroutine for a given condition, set the corresponding bit to 0 and pass the mask as parameter M3%.

A call to function 0 automatically sets the call mask to 0.

**Note** Before your program ends, be sure to set the interrupt call mask to 0. If the call mask and subroutine remain defined when the program is no longer running, the subroutine will still be executed if one of the conditions defined by the call mask occurs.

When the mouse software makes a call to the subroutine, it loads the following information into the CPU registers:

Information
Condition mask (similar to the call mask except a bit is set only if the condition has occurred)
Button state
Cursor coordinate (horizontal)
Cursor coordinate (vertical)
Horizontal mouse counts (mickeys)
Vertical mouse counts (mickeys)

**Note** The DS register contains the mouse driver data segments. The subroutine is responsible for setting DS as needed.

To use function 12 with the BASIC interpreter:

- Load an assembly-language subroutine into the BASIC interpreter's data segment. All exits from the subroutine must use a far return instruction.
- Assign the entry address of the subroutine to an integer variable.
- Pass this variable to function 12 as the fourth parameter.

To use function 12 in a high-level-language program:

Link an assembly-language subroutine with the program's object file(s). All exits from the subroutine must use a far return instruction.

#### Calling from interpreted BASIC programs

Calling from high-levellanguage programs 2 Assign the entry address of the subroutine to an integer variable.

In QuickBASIC programs and C small- and compactmodel programs, the address is an offset only.

In FORTRAN programs, Pascal programs, and C medium-, large-, and huge-model programs, the address consists of both a segment and an offset.

3 Pass this variable to function 12 as the fourth parameter.

Calling from assemblylanguage programs For information on using function 12 in assemblylanguage programs, see "Making Calls from Assembly-Language Programs" in Chapter 7, "Making Mouse Function Calls."

#### Example

The following example calls function 12 from the BASIC interpreter. Assume that an assembly-language subroutine has been loaded into the BASIC interpreter's data segment and that the integer variable SKETCH% has been assigned the subroutine's entry address. The following BASIC statements set up calls to SKETCH% any time the user presses the left mouse button.

100 '
200 ' Call subroutine SKETCH on left button press
300 '
400 M1%=12
500 M3%=&H01
600 M4%=SKETCH%
700 CALL MOUSE(M1%, M2%, M3%, M4%)

## Function 13: Light Pen Emulation Mode On

Input Output

M1%=13

Function 13 allows the mouse to emulate a light pen. When the mouse emulates a light pen, calls to the PEN function (described in the IBM *BASIC* manual) return the cursor position at the last "pen down."

The mouse buttons control the "pen down" and "pen off the screen" states. The pen is down when both mouse buttons are down. The pen is off the screen when both mouse buttons are up.

The mouse software enables light pen emulation mode after each reset (function 0).

## Example

100 ' 200 ' Enable light pen 300 ' 400 M1%=13 500 CALL MOUSE (M1%, M2%, M3%, M4%) 6-22 Designing a Mouse Interface

## Function 14: Light Pen Emulation Mode Off

Input Output

M1% = 14

Function 14 disables light pen emulation. When light pen emulation is disabled, calls to the PEN function (described in the IBM *BASIC* manual) return information about the light pen only.

If a program uses both a light pen and the mouse, the program must disable the mouse light pen emulation mode to work correctly.

#### Example

100 ' 200 ' Disable light pen 300 ' 400 M1%=14 500 CALL MOUSE(M1%, M2%, M3%, M4%)

# Function 15: Set Mickey/Pixel Ratio

#### Input

Output

M1% = 15

- ---<u>F</u> --

M3%=mickey/pixel ratio (horizontal)

M4%=mickey/pixel ratio (vertical)

Function 15 sets the mickey-to-pixel ratio for horizontal and vertical mouse motion. The ratios specify the number of mickeys per 8 pixels. The values must be in the range 1 to 32767. (For more information, see "Mouse Unit of Distance: The Mickey" in Chapter 5, "The Mouse Interface.")

The default value for the horizontal ratio is 8 mickeys to 8 pixels. With this ratio, the mouse must travel 6.4 inches to move the cursor horizontally across the screen.

The default value for the vertical ratio is 16 mickeys to 8 pixels. With this ratio, the mouse must travel 4 inches to move the cursor vertically across the screen.

## Example

100 '
200 ' Set mickey/pixel ratio at 16 to 8 and 32 to 8
300 '
400 M1%=15
500 M3%=16 ' horizontal ratio
600 M4%=32 ' vertical ratio
700 CALL MOUSE(M1%, M2%, M3%, M4%)

# Function 16: Conditional Off

#### Input

Output

M1% = 16M4% = address of the region

array

Function 16 defines a region on the screen for updating. If the mouse pointer is in the defined region or moves into it, function 16 hides the mouse cursor while the region is being updated. After your program calls function 16, the program must call function 1 to show the cursor again.

The region is defined by placing the screen coordinate values in a four-element array. The elements of the array are defined as follows:

Array Offset	Value		
1	Left x screen coordinate		
2	Upper y screen coordinate		
3	Right x screen coordinate		
4	Lower y screen coordinate		

Function 16 is similar to function 2, but is intended for advanced applications that require quicker screen updates.

**Note** For information on calling function 16 from assembly-language programs, see "Making Calls from Assembly-Language Programs" in Chapter 7, "Making Mouse Function Calls."
### Example

100 ۲ . 200 Define screen region for conditional off ٠ 300 OFF% (1) =10 OFF% (2) =30 OFF% (3) =40 OFF% (4) =80 ' left x value of region 400 ' upper y value of region 500 ' right x value of region 600 ' lower y value of region 700 800 M1%=16 900 CALL MOUSE (M1%, M2%, M3%, OFF% (0)) 1000 ' 1100 ' Screen update routine . . 2200 ' 2300 M1%=1 2400 CALL MOUSE (M1%, M2%, M3%, M4%)

## Function 19: Set Double-Speed Threshold

#### Input

Output

M1% = 19

M4%=threshold speed in mickeys/second

Function 19 sets the threshold speed for doubling the cursor's motion on the screen. Using function 19 makes it easier to point at images widely separated on the screen.

Parameter M4% defines the mouse's threshold speed. If no value is given, or if the mouse is reset by a call to function 0, a default value of 64 mickeys per second is assigned. If the mouse moves faster than the value of M4%, cursor motion doubles in speed. The threshold speed remains set until function 19 is called again or until the mouse is reset by function 0.

Once your program turns on the speed-doubling feature, this feature is always on. However, the program can effectively turn off this feature by setting M4% to a speed faster than the mouse can physically move (for example, 10,000) and then calling function 19.

#### Example

100 ٠ Set threshold to 32 mickeys/sec 110 120 130 M1%=19 ' mickeys/second 140 M4% = 32CALL MOUSE (M1%, M2%, M3%, M4%) 150 . 1000 ٠ Turn off speed doubling M1%=19 1010 M4%=10000 ' mickeys/second 1020 1030 CALL MOUSE (M1%, M2%, M3%, M4%)

# Function 20: Swap Interrupt Subroutines

Input	Output
M1%=20 M2%=segment of new subroutine M3%=new call mask M4%=offset of new subroutine	M2%=segment of old subroutine M3%=old call mask M4%=offset of old subroutine

Function 20 sets new values for the call mask and subroutine address for mouse hardware interrupts and returns the values that were previously specified.

The mouse hardware interrupts automatically stop execution of your program and call the specified subroutine whenever one or more of the conditions defined by the call mask occur. When the subroutine finishes, your program continues execution at the point of interruption.

The call mask is an integer value that defines which conditions cause an interrupt. Each bit in the call mask corresponds to a specific condition, as shown in the following table:

Mask bit	Condition				
0	Cursor position changes				
1	Left button pressed				
2	Left button released				
3	Right button pressed				
4	Right button released				
5 - 15	Not used				

To enable the subroutine for a given condition, set the corresponding call mask bit to 1 and pass the mask as parameter M3%.

To disable the subroutine for a given condition, set the corresponding bit to 0 and pass the mask as parameter M3%. Function 0 automatically disables all interrupts.

**Note** Before your program ends, be sure to restore the initial values of the call mask and subroutine address.

When the mouse software makes a call to the subroutine, it loads the following information into the CPU registers:

Register	Information			
AX	Condition mask (similar to the call mask except a bit is set only if the condition has occurred)			
BX	Button state			
CX	Cursor coordinate (horizontal)			
DX	Cursor coordinate (vertical)			
DI	Horizontal mouse counts (mickeys)			
SI	Vertical mouse counts (mickeys)			

Note The DS register contains the mouse driver data segments. The subroutine is responsible for setting DS as needed.

Calling from interpreted BASIC programs	<ul> <li>To use function 20 with the BASIC interpreter:</li> <li>1 Load an assembly-language subroutine into the BASIC interpreter's data segment. All exits from the subroutine must use a far return instruction.</li> <li>2 Assign the entry offset of the subroutine to an integer variable.</li> <li>3 Pass this variable to function 20 as the fourth parameter.</li> </ul>
	If M2% is set to 0 in an interpreted BASIC program, the segment of the new subroutine is assumed to be identical to the BASIC data segment.
Calling from high-level- language programs	<ul> <li>To use function 20 in a high-level-language program:</li> <li>1 Link an assembly-language subroutine with the program's object file(s). All exits from the subroutine must use a far return instruction.</li> <li>2 In FORTRAN programs, Pascal programs, and C medium-, large-, and huge-model programs, assign the subroutine's segment to an integer variable. In Quick-BASIC programs, and C small- and compact-model programs, set this variable to 0.</li> </ul>

3 Assign the subroutine's offset to an integer variable.

Pass the variable containing either the subroutine's segment or 0 to function 20 as the second parameter. Pass the variable containing the subroutine's offset as the fourth parameter.

If M2% is set to 0 in a high-level-language program, the segment of the new subroutine is assumed to be identical to the program's data segment.

For information on using function 20 in assemblylanguage programs, see "Making Calls from Assembly-Language Programs" in Chapter 7, "Making Mouse Function Calls."

Calling from assemblylanguage programs

## Example

The following example calls function 20 from the BASIC interpreter. Assume that an assembly-language subroutine has been loaded into the BASIC interpreter's data segment and that the integer variable SKETCH% has been assigned the subroutine's entry offset. The following BASIC statements set up calls to SKETCH% any time the user presses the left mouse button.

100 '
200 ' Call subroutine SKETCH on left button press
300 '
400 M1%=20
500 M2%=0
600 M3%=&H01
700 M4%=SKETCH%
800 CALL MOUSE(M1%, M2%, M3%, M4%)

If your program does not change the return values of M1%, M2%, M3%, and M4%, you can restore the previous interrupt subroutine call mask and address by adding the following statement after the initial call to function 20.

CALL MOUSE (M1%, M2%, M3%, M4%)

## Function 21: Get Mouse Driver State Storage Requirements

Input	Output
M1% = 21	M2%=buffer size required for mouse
	driver state

Function 21 returns the size of the buffer required to store the current state of the mouse driver. It is used with functions 22 and 23 when you want to temporarily interrupt a program that is using the mouse and execute another program that also uses the mouse, such as one of the Microsoft Expert Mouse Menu programs.

#### Example

100 '
200 ' Get required storage size
300 '
400 M1%=21
500 CALL MOUSE(M1%, M2%, M3%, M4%)
600 BufSIZE%=M2%

## Function 22: Save Mouse Driver State

#### Input

Output



M1% = 22

M4%=pointer to the buffer

Function 22 saves the current mouse driver state in a buffer allocated by your program. It is used with functions 21 and 23 when you want to temporarily interrupt a program that is using the mouse and execute another program that also uses the mouse.

Before your program calls function 22, the program should call function 21 to determine the buffer size required for saving the mouse driver state, then allocate the appropriate amount of memory.

**Note** For information on calling function 22 from an assembly-language program, see "Making Calls from Assembly-Language Programs" in Chapter 7, "Making Mouse Function Calls."

## Example

Assume that the buffer size was obtained by calling function 21 and has been allocated in the BASIC interpreter's data segment. Assume also that BUFPTR contains the address of the buffer.

100 '
200 ' Save the mouse driver state
300 '
400 M1%=22
500 M4%=BUFPTR
600 CALL MOUSE(M1%, M2%, M3%, M4%)

## Function 23: Restore Mouse Driver State

#### Input

Output

M1%=23

M4%=pointer to the buffer

Function 23 restores the last mouse driver state saved by function 22. It is used with functions 21 and 22 when you want to temporarily interrupt a program that is using the mouse and execute another program that also uses the mouse. To restore the mouse driver state saved by function 22, call function 23 at the end of the interrupt program.

**Note** For information on calling function 23 from an assembly-language program, see "Making Calls from Assembly-Language Programs" in Chapter 7, "Making Mouse Function Calls."

#### Example

Assume that function 22 saved the mouse driver state in a buffer allocated by the program. Assume also that BUFPTR contains the address of the buffer.

100 '
200 ' Restore the mouse state
300 '
400 M1%=23
500 M4%=BUFPTR
600 CALL MOUSE(M1%, M2%, M3%, M4%)

## Function 29: Set CRT Page Number

#### Input

Output

0

M1%=29 M2%=CRT page for mouse cursor display

Function 29 specifies the CRT page on which the mouse cursor will be displayed.

For information on the number of CRT pages available in each display mode your adapter supports, see the documentation that came with the graphics adapter.

### Example

```
100 '
200 ' Display mouse cursor on page 3
300 '
400 M1%=29
500 M2%=3
600 CALL MOUSE(M1%, M2%, M3%, M4%)
```

## Function 30: Get CRT Page Number

Output

M1%=30

Input

M2%=CRT page number of current

cursor display

Function 30 returns the number of the CRT page on which the mouse cursor is displayed.

#### Example

100 ' 200 ' Get CRT page number 300 ' 400 M1%=30 500 CALL MOUSE (M1%, M2%, M3%, M4%)



## 7 Making Mouse Function Calls

The statements and instructions required to call the mouse functions depend on the language you're using for your application program. This chapter explains how to make mouse function calls from the following types of programs:

- BASIC programs running under the BASIC interpreter
- Assembly-language programs
- Programs in Microsoft high-level languages

This chapter also includes the BASIC source program listing for the Piano demonstration program that came in your Microsoft Mouse package.

The last section in this chapter describes eight sample mouse cursors you can use in high-resolution graphics mode. Additional examples

## Making Calls from the BASIC Interpreter

To make a mouse function call from a BASIC program running under the BASIC interpreter:

- Assign the offset and segment of the BASIC entry point into the mouse driver to a pair of integer variables in your program. The mouse entry offset and segment are in memory. To get these values, insert the following statements into your program:
  - 10 DEF SEG=0
    20 MSEG=256\*PEEK(51\*4+3)+PEEK(51\*4+2)
    30 MOUSE=256\*PEEK(51\*4+1)+PEEK(51\*4)+2
    40 IF MSEG OR (MOUSE-2) THEN 60
    50 PRINT "Mouse Driver not found":END
    60 DEF SEG=MSEG
    70 IF PEEK(MOUSE-2)=207 then 50

Be sure that these statements appear before any calls to mouse functions.

2 Use the CALL statement to make the call. The statement should have the form

CALL MOUSE (M1%, M2%, M3%, M4%)

where MOUSE is the variable containing the offset of the BASIC entry point into the mouse driver, and M1%, M2%, M3%, and M4% are the names of the integer variables you have chosen for parameters in this call. (Constants and noninteger variables are not allowed.) All of the parameters must appear in the CALL statement even if no value is assigned to one or more of them.

To ensure that the variables are integer variables, use the percent sign (%) as part of all the variable names. You may also use the DEFINT statement at the beginning of your program. For example, the statement

10 DEFINT A-Z

defines all variables as integer variables. If this statement appears at the beginning of the program, the variable names don't need to include the percent sign.

## Example

Assuming that the variable MOUSE has the offset of the BASIC entry point into the mouse driver, use the following statements to set the cursor position to 320 (horizontal) and 100 (vertical):

```
      100
      '

      200
      ' Set cursor position to (320,100)

      300
      '

      400
      M1%=4
      'Function number is 4

      500
      M3%=320
      'Horizontal coordinate

      600
      M4%=100
      'Vertical coordinate

      700
      CALL MOUSE (M1%, M2%, M3%, M4%)
```

**Note** For additional examples of making calls from the BASIC interpreter, see the sample source code after the description of each function in Chapter 6, "Mouse Function Descriptions," and the section "Piano Program Listing" later in this chapter.



## Making Calls from Assembly-Language Programs

To make a mouse function call from an assemblylanguage program:

 Include statements in your program that check if the mouse driver is installed. These statements must appear before any calls to mouse functions. (See the assembly-language program example on the next page.)

Load the appropriate CPU registers (AX, BX, CX, DX, SI, DI, and/or ES) with the parameter values.

Execute software interrupt 51 (33H).

For all mouse functions except functions 9, 12, 16, 20, 22, and 23, the AX, BX, CX, and DX registers correspond to the M1%, M2%, M3%, and M4% parameters defined for the BASIC interpreter in Chapter 6, "Mouse Function Descriptions."

The parameter definitions for functions 9, 12, 16, 20, 22, and 23 are given in the following table:

Function	Input	Output
9	AX = 9 BX = cursor hot spot (horizontal) CX = cursor hot spot (vertical) ES:DX = address of first element in screen and cursor masks array	- 0
12	AX = 12 CX = call mask ES:DX = subroutine address	
16	AX = 16 CX = upper x screen coordinate DX = left y screen coordinate SI = lower x screen coordinate DI = right y screen coordinate	
20	AX = 20 CX = new call mask ES:DX = new subroutine address	CX = old call mask ES:DX = old subroutine address
22	AX = 22 ES:DX = start of buffer address	
23	AX = 23 ES:DX = start of buffer address	

#### Example

Assemblylanguage program The following assembly-language program puts an IBM Color/Graphics Adapter into 640 x 200 graphics mode and displays the default mouse cursor (the standard cursor shape described under "Sample Cursors" later in this chapter). Clicking the left mouse button returns the video display to 80-column, black-and-white text mode, and ends the program.

stack	segment	stack db	'stack' 256 dup(?)		
stack	ends		100 aap(.)		ſ
, data	segme msgl	nt publ db	ic 'data' "Microsoft	Mouse not	found","\$"

db "Press the left mouse button to EXIT", "\$" msa2 data ends . 'code' code segment public assume cs:code, ds:data, es:data; ss:stack start: push bp mov bp,sp mov ax, seg data ;Set DS to the ds,ax mov ;data segment ;Save PSP segment address push es ax,03533h ;Get int 33h vector mov int 21h ;by calling int 21 ;Check segment and mov ax,es ;offset of int 33 ax,bx or ;vector. If 0 or pointing to jnz begin mov bl,es:[bx] ;IRET, driver not installed bl,Ocfh cmp jne begin :Exit mov dx,offset msql ;Get not found message offset mov ah,09h ;Output message to screen int 21h pop es jmp short exit :Exit begin: ax,0 ;Initialize mouse mov int 33h cmp ax:0 ; Is mouse installed? jz exit ~ ;No, exit ax,0006h mov ;Set up for 640x200 resolution 10h ; graphics mode (CGA mode 6) ax,4 Set Carson Vos ; Function 4-set cursor position int mov cx,200 ;M3 = 200mov dx.100 :M4 = 100mov 33h int ; Function 7 Setmin Ermax hon cursor pes ax,7 mov cx,150 :M3 = 150mov dx,450 :M4 = 450mov 33h int ; Function 8 Set mint for ax vert cursor po. ax,8 mov :M3 = 50mov cx,50 dx,150 :M4 = 150mov int 33h ax,1 ;Show the mouse cursor mov int 33h dx,offset msq2 mov ;Get exit message ah,09h ;Output message to screen mov 21h int xor ax,ax

#### 7-6 Designing a Mouse Interface

around:			
	mov int	ax,3 33h	;Get mouse status
	cmp jne	bx,0001h around	;Left mouse button ;pressed? Branch if not
	mov	ax,0	
	int	33h	;Reset mouse
	mov	ax,0003h	;Set up 80x25
	int	10h	character text mode
exit:			
	mov	sp, bp	
	qoq	qd	
	mov	ax,04c00h	:Terminate
	int	21h	,
;			
code	ends start		
enu	Scarc		

## Making Calls from High-Level-Language Programs

Mouse function calls from high-level-languages can be included as ordinary procedure calls in the source program. After the program is compiled, it must be linked with the Microsoft Mouse Library (MOUSE.LIB), which is included on the Microsoft Mouse Tools disk. MOUSE.LIB contains procedures that give access to all the mouse functions.

This section describes how to make function calls from the following high-level languages:

- Microsoft QuickBASIC
- Microsoft Pascal (version 3.30 or later)
- Microsoft FORTRAN (version 3.30 or later)
- Microsoft C (version 3.0 or later)

For information about linking programs written for earlier versions of the Microsoft Mouse Library, see Appendix B, "Linking Existing Mouse Programs with MOUSE.LIB (Version 6.0)."

For information on accessing the mouse functions from a program written in Borland Turbo Pascal, see Appendix C, "Making Calls from Borland Turbo Pascal Programs."

## Making Calls from Microsoft QuickBASIC

To make a mouse function call from a program in Microsoft QuickBASIC (version 1.0 or later):

1 Include statements in your program that check if the mouse driver is installed. These statements must appear before any calls to mouse functions. (See the QuickBASIC program example on the next page.)

Call the mouse library procedure "MOUSE" as a regular QuickBASIC external subroutine.

Compile the program and link it with MOUSE.LIB.

If you are using version 2.0 of the QuickBASIC compiler, you can compile and link in one step from within the QuickBASIC editor following the procedure given below. (For instructions on linking with MOUSE.LIB outside of the QuickBASIC editor, see your documentation on Microsoft QuickBASIC.)

## Linking with MOUSE.LIB within QuickBASIC

To simultaneously compile a QuickBASIC program (version 2.0) and link the program with MOUSE.LIB, first set up a special library subroutine called "USERLIB.EXE". To set up USERLIB.EXE:

1 Create a QuickBASIC source file that contains the single statement:

CALL MOUSE

Compile this source file outside of the QuickBASIC editor. To do this, type the following at the DOS prompt:

qb <filename>;

and press the ENTER key. (<filename> is the name of the source file.)

Make sure the compiled QuickBASIC file, MOUSE.LIB, and the QuickBASIC utilities BUILDLIB.EXE and BRUN20.LIB are in the current directory.

#### Setting up USERLIB.EXE

4	Τo	create	the	USER	LIB	.EXE	file,	type	the	folle	owing	$\mathbf{at}$
	$_{\mathrm{the}}$	DOS	prom	ipt:			,	• 1			0	

*buildlib* <filename>,*userlib*,,*mouse*;

and press the ENTER key. (<filename> is the name of the compiled QuickBASIC file.)

#### Linking within QuickBASIC

To compile a QuickBASIC program and link it with MOUSE.LIB in one step:

- Make sure the USERLIB.EXE file is in the same directory as the QuickBASIC compiler (QB.EXE) before you start QuickBASIC.
- 2 To start QuickBASIC, type qb/l (not qb) and press the ENTER key.
- Compile the mouse application program within the QuickBASIC editor. This also automatically links the program with MOUSE.LIB.

#### Example

#### QuickBASIC program

The following program puts an IBM Color/Graphics Adapter into 640 x 200 graphics mode and displays the default mouse cursor (the standard cursor shape described under "Sample Cursors" later in this chapter). This program calls the subroutine "chkdrv", which is shown in the assembly-language program that follows.

```
' Mouse library call test in QuickBASIC V2.0
 call chkdrv
 screen O
 m1%=0
                        ' function O
 call mouse (m1%,m2%,m3%,m4%)
  if (m1\% = 0) then
     print "Microsoft Mouse not found"
     end
  end if
 m1%=4
                          function 4
 m3\% = 200
 m4\% = 100
  call mouse (m1%, m2%, m3%, m4%)
                        ' function 7
  m1%=7
  m3\% = 150
  m4\% = 450
  call mouse (m1%, m2%, m3%, m4%)
```

```
' function 8
m1%=8
m3\% = 50
m4%=150
call mouse (m1%, m2%, m3%, m4%)
screen 2
print "Cursor limited to the center of the screen."
print "Press the left mouse button to EXIT."
m1\% = 1
call mouse (m1%, m2%, m3%, m4%)
m2\% = 99
while ( m2% <> 1 )
  m1\% = 3
  call mouse (m1%,m2%,m3%,m4%)
wend
screen 0
end
```

In the following assembly-language source program, "chkdrv" checks if the mouse driver is installed.

#### chkdrv subroutine

Making Mouse Function Calls 7-9

; mdata mdata	segment msg ends	byte public 'da db ''Mouse Dr:	ata' iver not installed","\$"
mcode ;	segment assume public	para public 'Co cs:mcode chkdrv	DDE '
; chkdrv	proc push push mov int mov or jnz mov cmp jne mov mov	far bp es ax,03533h 21h ax,es ax,bx back bl,es:[bx] bl,Ocfh back ax,seg mdata ds,ax	;Get int 33h by ;calling int 21 ;Check segment and offset ;of int 33 vector. If ;O or pointing to IRET, ;driver not installed ;Exit ;Set up DS to ;point to data seg
	mov mov int pop pop	dx,offset msg ah,O9h 21h es bp	;Get message ;output to screen

mov int	ax,04c00h 21h	;Terminate
pop	es	
pop	bp	
ret		
endp		
	· ·	
ends		
end		
	mov int pop pop ret endp ends end	mov ax,04c00h int 21h pop es pop bp ret endp ends end

## Making Calls from Microsoft Pascal

To make a mouse function call from a program in Microsoft Pascal (version 3.30 or later):

- 1 Include statements in your program that check if the mouse driver is installed. These statements must appear before any calls to mouse functions. (See the Pascal program example below.)
- Declare the mouse library procedure "MOUSES" as an external procedure. The parameters can be declared as either INTEGER or WORD. Use one of the following statements to declare MOUSES as an external procedure:

PROCEDURE MOUSES (VARS m1, m2, m3, m4: INTEGER) ; EXTERN;

or

PROCEDURE MOUSES (VARS m1, m2, m3, m4: WORD) ; EXTERN;

- 3 Use Microsoft Pascal calling conventions to make the call.
- Link the compiled program with MOUSE.LIB. (For details about using the LINK command, see your documentation on Microsoft Pascal.)

#### Example

#### Pascal program

The following program puts an IBM Color/Graphics Adapter into 640 x 200 graphics mode and displays the default mouse cursor (the standard cursor shape described under "Sample Cursors" later in this chapter). This program calls two procedures, "graf" and "chkdrv". These procedures are shown in the assembly-language listing that follows this program.

```
program mtest (output);
procedure mouses (vars m1, m2, m3, m4:word); extern;
procedure chkdrv; extern;
procedure graf; extern;
var
  m1, m2, m3, m4: word;
begin {demo}
  chkdrv:
                        {Mouse driver installed?
                                                      }
                        {No, exit
                                                      3
  M1:=0:
                                                      ž
                        {Yes, initialize mouse
  mouses (m1, m2, m3, m4);
  if ( ml = 0 ) then
writeln('Microsoft Mouse not found')
  else
     begin
       m1 := 4;
                        {function call 4, set mouse }
       m3 := 200;
                        {horizontal cursor position }
       m4 := 100:
                        {vertical cursor position
                                                      }
       mouses( m1, m2, m3, m4 );
       m1 := 7;
                        {function call 7, set mouse }
       m3 := 150;
                        {minimum horizontal position}
       m4 := 450;
                        {maximum horizontal position}
       mouses( m1, m2, m3, m4 );
                        {function call 8, set mouse }
       m1 := 8;
       m3 := 50:
                        {minimum vertical position
                                                      ì
       m4 := 150;
                        {maximum vertical position
       mouses( m1, m2, m3, m4 );
       graf;
                        {change into graphics mode
                                                      }
       writeln('Cursor limited to the center of the screen.');
       writeln ('Press the left mouse button to EXIT.');
       m1 :=1:
                        {function call 1
       mouses(m1,m2,m3,m4);
                               {show mouse cursor
                                                      }
       m2 := 999;
                        {dummy value for loop
                                                      }
                        {until ....
       repeat
         m1 := 3;
                        {function call 3
                                                      7
         mouses( m1, m2, m3, m4 ); {Get current mouse status}
       until m2 = 1; {left mouse button pressed }
     end
```

0

end. {demo}

graf and chkdrv procedures	In the following assembly-language source program, "graf" changes the display mode to 640 x 200 graphics mode (CGA mode 6) and "chkdrv" checks if the mouse driver is installed.					
	; mdata	segment	byte public 'd	lata'	C	
		msg	db "Mouse Dr	'iver not installed","\$"		
	mdata	ends				
	mcode	segment assume	para public 'O cs:mcode	CODE '		
	;	public	graf			
	, graf	proc push mov int pop ret	far bp ax,06h 10h bp	;Change to graphics ;mode by calling ;int 10 service		
	graf ;	endp				
	;	public	chkdrv			
	chkdrv	proc push push	far bp es		0	
		mov int mov or jnz mov cmp jne	ax,03533h 21h ax,es ax,bx back b1,es:[bx] b1,Ocfh back	;Get int 33h by ;calling int 21 ;Check segment, offset ;of int 33 vector. If ;O or pointing to IRET, ;driver not installed ;Exit		
		mov mov mov int pop pop mov	ax, seg mdata ds, ax dx, offset msg ah, O9h 21h es bp ax, O4cOOh 21b	;Set up DS to ;point to data seg ;Get message ;output to screen ;Terminate		
	back	IIIC	2111			
	chkdrv	pop pop ret endp	es bp		C	
	; mcode	ends end				

## Making Calls from Microsoft FORTRAN

To make a mouse function call from a program in Microsoft FORTRAN (version 3.30 or later):

- 1 Include statements in your program that check if the mouse driver is installed. These statements must appear before any calls to mouse functions. (See the FORTRAN program example below.)
- Call the mouse library procedure "MOUSES" as a regular FORTRAN external subroutine.
- Link the compiled program with MOUSE.LIB. (For details about using the LINK command, see your documentation on Microsoft FORTRAN.)

### Example

The following program puts an IBM Color/Graphics Adapter into 640 x 200 graphics mode and displays the default mouse cursor (the standard cursor shape described under "Sample Cursors" later in this chapter). This program calls two assembly-language subroutines, "graf" and "chkdrv". (See the assembly-language program after the Pascal program example earlier in this chapter for the "graf" and "chkdrv" subroutine listings.) FORTRAN program

PROGRAM MTEST

С С -- Mouse Library calls test in MS FORTRAN V3.31 --С INTEGER\*2 M1, M2, M3, M4 EXTERNAL GRAF, CHKDRV С -- Call driver checking routine CALL CHKDRV() С -- Mouse init call --M1 = 0CALL MOUSES (M1, M2, M3, M4) IF ( M1 .EQ. O) THEN ' WRITE(\*,\*)' Microsoft Mouse not found' STOP ENDIF -- Place cursor in the center of the screen --С M1 = 4M3 = 200M4 = 100CALL MOUSES (M1, M2, M3, M4) С -- Set minimum and maximum horizontal position --M1 = 7

```
M3 = 150
        M4 = 450
        CALL MOUSES (M1, M2, M3, M4)
С
        -- Set minimum and maximum vertical position --
        M1 = 8
        M3 = 50
        M4 = 150
        CALL MOUSES (M1, M2, M3, M4)
        CALL GRAF()
        WRITE (*,*) ' Cursor is limited to the center of the screen.' WRITE (*,*) ' Press the left mouse button to EXIT.'
        M1 = 1
        CALL MOUSES (M1, M2, M3, M4)
С
        -- Loop for left mouse button pressed --
        M2 = 9999
 100
           M1 = 3
           CALL MOUSES (M1, M2, M3, M4)
        IF ( M2 .NE. 1 ) GOTO 100
        STOP
        END
```

## Making Calls from Microsoft C

To make a mouse function call from a program in Microsoft C (version 3.0 or later):

- 1 Include statements in your program that check if the mouse driver is installed. These statements must appear before any calls to mouse functions. (See the C program example on the next page.)
- Call one of the following mouse library procedures as a regular C external routine:

Use this procedure	To call from a
CMOUSES	Small-model program
CMOUSEC	Compact-model program
CMOUSEM	Medium-model program
CMOUSEL	Large- or huge-model program

Parameters are declared as signed or unsigned integers. Since MOUSE.LIB requires that all parameters be passed by reference, precede each parameter name with "&" (address of). I Link the compiled program with MOUSE.LIB. (For details about using the LINK command, see your documentation on Microsoft C.)

## Example

The following program puts an IBM Color/Graphics C program Adapter into 640 x 200 graphics mode and displays the default mouse cursor (the standard cursor shape described under "Sample Cursors" later in this chapter).

```
#include <stdio.h>
#include <dos.h>
void chkdrv();
void graf();
main()
{
                          502 PREADME, 00C
   int m1, m2, m3, m4;
                                                    */
                        /* check for mouse driver
   chkdrv();
         m1=z\phi
     <del>z</del> 0:
   m 1
                        /* initialize mouse
                                                    */
   cmouses(&m1, &m2, &m3, &m4);
   if(m1 = 0) \{
      printf("Microsoft Mouse not found");
                       /* exit, if mouse not found */
       exit (-1);
       }
  m1 = 4:
                        /* function call 4
  m3 = 200;
                        /* set mouse position at
   m4 = 100; /* center of the screen
cmouses(&m1, &m2, &m3, &m4);
  m4 = 100:
  m1 = 7;
                        /* function call 7
                        /* minimum horizontal value */
  m3 = 150;
                        /* maximum horizontal value */
   m4 = 450;
   cmouses(&m1, &m2, &m3, &m4);
  m1 = 8:
                        /* function call 8
   m3 = 50;
                        /* minimum vertical value
  graf();
   printf("Cursor limited to the center of the screen.\n");
   printf("Press the left mouse button to EXIT.");
                        /* Function 1, show cursor */
   m1 = 1;
   cmouses(&m1, &m2, &m3, &m4);
   m2 = 0;
                         /* Loop until left mouse
                                                    */
```

```
while (m2 != 1 ) { /* button is pressed
                                                                                 */
                               m1 = 3;
                               cmouses (&m1, &m2, &m3, &m4);
                                }
                           m1 = 2;
                           cmouses(&m1, &m2, &m3, &m4);
                        }
                        void chkdrv()
                        Ł
                           union REGS inregs, outregs;
                           struct SREGS segregs;
                           long address;
        UNSIENEN
                           unsign char first_byte;
SLE PRIMOME, NOC
                           inregs.x.ax = 0x3533;
                           intdosx ( &inregs, &outregs, &segregs );
                           address=(((long)segregs.es) <<16) + (long) outregs.x.bx;
                           first_byte=*(long far *)address;
If ((address == 0) || (first_byte == 0xcf)){
                               printf("Mouse driver not installed");
                               exit();
                                }
                        }
                        void graf()
                        {
                           union REGS cpuregs;
                           cpuregs.x.ax = 0x0006;
                           int86 ( 0x10, &cpuregs, &cpuregs );
                        }
```

## **Piano Program Listing**

This section presents the complete source code for the Piano demonstration program that came in your Microsoft Mouse package. The program is written in BASIC for the IBM Personal Computer's BASIC Interpreter. (The Piano source program listing is also in the file PIANO.BAS on the Microsoft Mouse Tools Disk.)

The following is an explanation of the program details:

Line numbers	Comments	
1000-1090	Copyright message	(
1100-1160	Set up music, clear graphics screen to blue.	
1170–1250	Read in the frequencies for the various piano keys.	

ł

Line numbers	Comments
1260-1380	Link the mouse software and the pro- gram.
1390–1430	Function 15 sets the mouse sensitivity. With this setting, a horizontal move- ment of 1.6 inches moves the cursor across the entire screen. This rela- tively high sensitivity permits songs to be played rapidly. Accuracy is no problem since the piano keys are large.
1440-1620	The integer array CURSOR contains the screen and cursor masks, which define the shape and color of the cur- sor. These statements define the screen mask; the mask is set to all ones. The mask is logically ANDed with the screen under the cursor.
1630–1810	Define the cursor mask. The values are XORed with the result of the AND operation to create the cursor shape and color. In this case, the cursor shape is a north-pointing arrowhead. Its color is the inverse of whatever is under it.
1820–1860	Function 9 sets the cursor shape. It also defines the cursor hot spot. In this case, the hot spot is the tip of the arrowhead. The mouse software automatically prevents the cursor hot spot from leaving the screen.
1870–1930	Read in the Microsoft logo from pre- calculated data and place the data on the screen.
1940 - 2150	Draw the white and black piano keys.
2160-2200	Draw the QUIT box in the lower-right corner.
2210 - 2240	Function 4 centers the cursor to just under the piano keys.
2250	Function 1 turns on the cursor. The cursor appears on the screen and can be moved using the mouse.

## 7-18 Designing a Mouse Interface

Line numbers	Comments
2260-2290	Function 3 gives the status of the two mouse buttons and the location of the cursor. This is probably the most com- mon mouse function used in applica- tions.
2300-2370	Some decisions are made. If both mouse buttons are up, or if the cursor is not on the piano keyboard, then any sound that might be playing is turned off.
2380-2430	At this point, the mouse button is down when the cursor is over the QUIT box. The program turns off the cursor, clears the screen, then quits.
2440-2510	The program has determined that a button is down and the cursor is over the piano keyboard. These statements determine which key the mouse cursor is over.
2520-2570	The note is played by the SOUND statement set with the correct fre- quency. This note is played in the background as the program loops back to line 2090.
2580-2630	This data contains the correct fre- quencies to play the musical notes.
2640-3050	Data to draw the Microsoft logo using the PUT statement.

1000	1
1010	' THE VIRTUAL PIANO
1020	1
1030	' COPYRIGHT (C) 1983 BY MICROSOFT CORPORATION
1040	' WRITTEN BY CHRIS PETERS
1050	1
1060	·
1070	1
1080	' INITIALIZE
1090	1
1100	DEFINT A-Z
1110	DIM CURSOR (15, 1), FREQ (27, 2), MICROSOFT (839)
1120	KEY OFF
1130	PLAY"MF"
1140	SCREEN 1
1150	COLOR 1,1
1160	CLS
1170	1

```
1180 ' Read in the flat, normal, and sharp note frequencies
1190
1200 FOR J=0 TO 2
1210 FOR I=O TO 6
1220 READ K
1230 FREQ(I,J)=K : FREQ(I+7,J)=K*2 : FREQ(I+14,J)=K*4 :
     FREQ(I+21, J) = K*8
1240 NEXT
1250 NEXT
1260
1270 '
       Determine mouse driver location; if not found, quit.
1280 '
1290 DEF SEG=0
1300 MSEG=256*PEEK(51*4+3)+PEEK(51*4+2)
                                            'Get mouse segment
1310 MOUSE=256*PEEK(51*4+1)+PEEK(51*4)+2 'Get mouse offset
1320 IF MSEG OR (MOUSE-2) THEN 1370
1330 PRINT"Mouse: Microsoft Mouse driver not found"
1340 PRINT
1350 PRINT"Press any key to return to system"
1360 I$=INKEY$ : IF I$="" THEN 1360 ELSE SYSTEM
1370 DEF SEG=MSEG
                                           'Set mouse segment
1375 IF PEEK (MOUSE-2) = 207 THEN 1330
1380 M1=0 : CALL MOUSE (M1, M2, M3, M4)
                                            'Initialize mouse
1390
1400 ' Set Mouse sensitivity
1410 '
1420 M1 = 15 : M3=4 : M4=8
1430 CALL MOUSE (M1, M2, M3, M4)
1440
1450 '
       Define the "logical and" cursor mask
1460 '
1470 CURSOR( 0,0) = & HFFFF ' Binary 111111111111111
1480 CURSOR ( 1,0) = & HFFFF ' Binary 111111111111111
1490 CURSOR ( 2,0) = & HFFFF ' Binary 11111111111111
1500 CURSOR (3,0) = & \text{HFFFF}
                          ' Binary 11111111111111111
1510 CURSOR (4,0) = \& HFFFF
                           .
                             Binary 1111111111111111
1520 CURSOR (5,0) = & \text{HFFFF}
                           1
                             Binary 1111111111111111
1530 CURSOR (6,0) = & HFFFF
1540 CURSOR (7,0) = & HFFFF
                           •
                             Binary 11111111111111111
                           .
                             Binary 1111111111111111
1550 CURSOR (8,0) = & HFFFF
                           ÷
                             Binary 11111111111111111
                           ' Binary 11111111111111111
1560 CURSOR (9,0) = \& HFFFF
                          ' Binary 1111111111111111
1570 CURSOR (10, 0) = \& HFFFF
1580 CURSOR (11,0) = & HFFFF ' Binary 111111111111111
                           ' Binary 11111111111111111
1590 CURSOR (12, 0) = \& HFFFF
1600 CURSOR (13, 0) = \& HFFFF
                           .
                             Binary 1111111111111111
                           .
                             Binary 1111111111111111
1610 CURSOR (14, 0) = \& HFFFF
1620 CURSOR (15,0) = & HFFFF ' Binary 1111111111111111
1630
1640 ' Define the "exclusive or" cursor mask
1650 '
1660 CURSOR ( 0,1)=&H0300 ' Binary 0000001100000000
1670 CURSOR ( 1,1) = & HO300 ' Binary 0000001100000000
1680 CURSOR ( 2,1)=&HOFCO '
                             Binary 0000111111000000
             3,1)=&HOFCO '
1690 CURSOR
                             Binary 0000111111000000
             4,1)=&H3FFO
                           1
1700 CURSOR (
                             Binary 0011111111110000
1710 CURSOR (5,1) = & H3FFO
                           1
                             Binary 0011111111110000
1720 CURSOR (6,1) = \& HFCFC
                           ' Binary 1111110011111100
1730 CURSOR ( 7,1) = & HCOOC ' Binary 110000000001100
1740 CURSOR ( 8,1) = & HOOOO ' Binary 0000000000000000
```

```
1750 CURSOR( 9,1)=&H0000 ' Binary 000000000000000
1760 CURSOR (10, 1) = & HOOOO ' Binary 000000000000000
1770 CURSOR (11, 1) = & HOOOO ' Binary 000000000000000
1780 CURSOR (12, 1) = & HOOOO '
                             Binary 0000000000000000
1790 CURSOR (13, 1) =&H0000 '
                             Binary 0000000000000000
1800 CURSOR (14, 1) = & H0000 ' Binary 0000000000000000
1810 CURSOR (15, 1) = & H0000 ' Binary 000000000000000
1820
1830 ' Set the mouse cursor shape
1840 '
1850 M1 = 9 : M2 = 6 : M3 = 0
1860 CALL MOUSE (M1, M2, M3, CURSOR (0, 0))
1870
1880 '
       Draw the MICROSOFT logo from pre-calculated data
1890 '
1900 FOR I=0 TO 779
1910 READ MICROSOFT(I)
1920 NEXT
1930 PUT (62, 0), MICROSOFT, PSET
1940
1950 ' Initialize keyboard size parameters
1960
1970 YL = 60 : WKL = 80 : BKL = 45 : KW = 15 :
     WKN = 21
1980 XL = 320-KW*WKN : YH = YL + WKL : XH = 319 :
     BKW2=KW3
1990 QX = 272 : QY = 176
2000
2010 ' Draw the white keys
2020 '
2030 LINE (XL, YL) - (XH, YH), 3, BF
2040 FOR I=XL TO XH STEP KW
2050 LINE (I,YL)-(I,YH),0
2060 NEXT
2070
2080 ' Draw the "black" keys
2090 '
2100 C=6
2110 FOR X=XL TO XH STEP KW
2120 C=C+1 : IF C=7 THEN C=0
2130 IF C=0 OR C=3 THEN 2150
2140 LINE (X-BKW2,YL) - (X+BKW2,YL+BKL), 2, BF
2150 NEXT
2160
2170 ' Draw the quit box
2180
2190 LINE (QX,QY) - (319,199),3,B
2200 LOCATE 24,36 : PRINT"Quit";
2210
2220 '
       Set mouse cursor location, then turn on cursor
2230 '
2240 \text{ M1} = 4 : \text{M3} = 320 : \text{M4} = 160 : \text{CALL MOUSE}(\text{M1}, \text{M2}, \text{M3}, \text{M4})
2250 \text{ M1} = 1 : CALL MOUSE (M1, M2, M3, M4)
2260
2270 ' MAIN LOOP
     . 1
2280
2290 M1=3 : CALL MOUSE (M1, BT, MX, MY)
                                          'Get mouse location
                                          and button status
2300 IF (BT AND 2) THEN OTV=7 : GOTO 2340
```

'If right button down, set high octave 2310 IF (BT AND 1) THEN OTV=0 : GOTO 2340 'If left button down, set lower octave 'If both buttons up, turn off sound 2320 SOUND 442,0 2330 GOTO 2290 'Keep looping... 'Correct for medium resolution screen 2340 MX = MX22350 IF MX <= XL OR MY < YL THEN 2320 'If above keyboard, turn off sound 'If on keyboard, 2360 IF MY <= YH THEN 2470 play sound 'If above quit box, 2370 IF MY < QY OR MX < QX THEN 2320' turn off sound 2380 ' 2390 ' Button down inside the quit box 2400 ' 2410 M1=2 : CALL MOUSE (M1, M2, M3, M4) 'Turn off mouse cursor 2420 CLS ' Clear screen 2430 END ' Quit 2440 2450 ' Button down over keyboard, determine which key 2460 2470 WKY = (MX-XL) kW+OTV : R = 1'Get which white key cursor is over 2480 IF MY > YL+BKL THEN 2560 'Is it lower than the black keys? 2490 MK=(MX-XL) MOD KW ' No, get which side of key 2500 IF MK <= BKW2 THEN R=0 : GOTO 2560 'Is it the left black key? 2510 IF MK >= KW-BKW2 THEN R=2 ' Is it the right black key? 2520 2530 ' Play the note. For BASIC interpreter duration = 2 2540 ' For BASIC compiler duration = 12550 ' 2560 SOUND FREQ(WKY,R),2 2570 GOTO 2290 ' Continue looping 2580 2590 ' Musical note frequencies 2600 ' 2610 DATA 131,139,156,175,185,208,233 2620 DATA 131,147,165,175,196,220,247 2630 DATA 139,156,165,185,208,233,247 2640 2650 ' Data to draw the MICROSOFT logo 2660 ' 2690 DATA 0,0,0,-193,240,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 2700 DATA 0,0,0,0,0,768,-1,0,0,0,0,3840,-1,-16129,0, -253,0,0,-193,240 2710 DATA 0,0,0,0,0,0,0,0,0,-193,0,16128,4095,252, 16128,-1,240, -256,-769,0 2720 DATA 0,0,0,0,-193,240,768,-1,255,768,-1,1023,-1, -1,240,0,0,0,-193,1922730 DATA -256,4095,252,-253,-1,255,-256,-1,240,-253, -1, -1, 768, -1, 255, 16128, -1, -3841, 768, -1 2740 DATA 1023,-1,-1,240,0,0,0,-193,192,-256,4095,252, -193,-1,-3841,-256,-1, 252,-1009,0 2750 DATA -256,4032,-1,-16129,-253,-1,-1,768,-1,1023, -1, -1, 240, 0, 0, 0, -193, 240, -253, 4095

2760 DATA 252,-3841,0,-961,-256,-1,255,0,0,0,3840,-1, -16129, -241, 0, -253, 960, -1, 1023, -1 2770 DATA -1,240,0,0,0,-193,240,-253,4095,1020,255,0, -253,-256,4032,-16129,-1,-1,-1,4092 2780 DATA 4095, -16129, -4033, 0, 16128, 1008, -1, 1023, -1, -1, 240,0,0,0,-193,252,-241,4095,1020,252 2790 DATA 0,-256,-256,960,-15361,252,0,0,4095,1023, -16129, -16321, 0, 3840, 1008, 255, 0, 3840, 252, 0 2800 DATA 0,0,0,-193,252,-241,4095,4092,240,0,16128, -64,192,-16129,0,0,0,3840,255,0 2810 DATA 255,0,768,1020,255,0,3840,252,0,0,0,0,-193, 255, -193, 4095, 4092, 240, 0, 16128 2820 DATA -64,192,-12289,-1,192,-241,-12289,-3841,0, 255,0,768,1020,255,0,3840,252,0,0,0 2830 DATA 0,-193,255,-193,4095,16380,192,0,3840,-16, 960.-12289,240,0,0,-15553,-1,768,252,0 2840 DATA 0,1023,255,0,3840,252,0,0,0,0,-193,-16129, -1,4095,16380,192,0,0,-256,4032 2850 DATA -16129,0,0,0,768,-1,1008,252,0,0,1023,-1, 255,3840,252,0,0,0,0,-3265 2860 DATA -16129,-3073,4095,16380,192,0,0,-256,-1,4095, -1,0,-253,-16129,-1,1020,252,0,0,1023 2870 DATA -1,255,3840,252,0,0,0,0,-3265,-3073,-3073, 4095,16380,192,0,0,-256,-1,4095,240 2880 DATA 0,0,-16321,-241,1023,252,0,0,1023,-1,255, 3840,252,0,0,0,0,-4033,-3073,-15361 2890 DATA 4095,16380,192,0,0,-256,-1,252,0,0,0,0,16128, -15361,252,0,0,1023,-1,255 2900 DATA 3840,252,0,0,0,0,-4033,-1,-15361,4095,16380,192, 0,0,-256,-1,4092,240,0,0 2910 DATA -16321,768,-3073,252,0,0,1023,255,0,3840,252,0, 0,0,0,-4033,-193,1023,4095,4092 2920 DATA 240,0,0,-256,-64,4092,-1,192,-241,-16129,0, -3841,255,0,768,1020,255,0,3840,252 2930 DATA 0,0,0,0,-4033,-193,1023,4095,4092,240,0,16128, -64,4032,255,0,0,0,16128,252 2940 DATA -3841,255,0,768,1020,255,0,3840,252,0,0,0,0, -4033, -241, 1020, 4095, 1020, 252, 0 2950 DATA -256, -256, 960, 1023, 252, 0, 0, 16383, 1023, -3841, -16321,0,3840,1008,255,0,3840,252,0,0 2960 DATA 0,0,-4033,-241,1020,4095,1020,255,0,-253,-256, 960,-16129,-1,-1,-1,16380,-1,-3841,-4033 2970 DATA 0,16128,1008,255,0,3840,252,0,0,0,0,-4033,-253, 1008,4095,252,-3841,0,-961,-256 2980 DATA 192,-16129,0,0,0,3840,-1,-16129,-241,0,-253,960, 255,0,3840,252,0,0,0,0 2990 DATA -4033,-253,1008,4095,252,-193,768,-3841,-256, 192, -16129, -1009, 0, -256, 4032, -1, 255, -253, 240, -193 3000 DATA 768,255,0,3840,252,0,0,0,0,-4033,-256,960,4095, 252,-253,-1,255,-256,192,-16129 3010 DATA -253,-1,-1,768,-1,252,16128,-1,-3841,768,255,0, 3840,252,0,0,0,0,-4033,-256 3020 DATA 960,4095,252,16128,-1,240,-256,192,-16129,0,0, 0,0,-193,192,768,-1,255,768,255 3030 DATA 0,3840,252,0,0,0,0,0,0,0,0,0,768,-1,0,0,0,0,3840,-1 3040 DATA -16129,0,0,0,0,-193,240,0,0,0,0,0,0,0,0,0,0,0,0,0,0 3050 DATA 0,0,0,0,0,0,0,0,-193,240,0,0,0,0,0,0,0,0,0,0,0

## Sample Cursors

This section describes the following sample graphics cursors:

- Standard Cursor Shape
- Up Arrow
- Left Arrow
- Check Mark
- Pointing Hand
- Diagonal Cross
- Rectangular Cross
- Hourglass

These sample cursors illustrate the wide variety of cursor shapes that can be defined for use in application programs.

The sample cursors are designed for high-resolution graphics mode. Each cursor is a white shape with a black outline on a transparent field. The shape typically suggests the type of action you may take with the mouse. For example, an arrow usually means "make a selection by pointing at an item."

To use a sample cursor in an interpretive BASIC program, copy the BASIC statements presented for the cursor directly to your program. Type the statements exactly as shown, using line numbers that are consistent with your program's numbering scheme.

To use a sample cursor in an assembly- or high-levellanguage program, define an array in your program and assign the values given for each cursor to the array elements. Assign the values in a way that will make their storage order identical to their storage order in a BASIC program.

The statements in this section define only the cursor's shape. It is up to you to define the action associated with a cursor by including the necessary statements in your program.

## **Standard Cursor Shape**

The standard cursor shape is a solid arrow that points up and to the left. The hot spot is just beyond the arrow's tip, so you can point to an item without covering it. The standard cursor is the most convenient shape when using the mouse to choose or select items from the screen.

100	•		
200	' Define the screen m	nask	
300	1		
400	CURSOR(0,0) = &H3FFF'	Binary 001111111111111	
500	CURSOR(1,0) = & H1FFF'	Binary 000111111111111	
600	CURSOR(2,0) = & HOFFF'	Binary 0000111111111111	
700	CURSOR(3,0) = & HO7FF'	Binary 0000011111111111	
800	CURSOR(4,0) = & HO3FF'	Binary 0000001111111111	
900	CURSOR(5,0) = & HO1FF'	Binary 0000000111111111	
1000	CURSOR(6,0) = & HOOFF'	Binary 0000000011111111	
1100	CURSOR(7,0) = & HOO7F'	Binary 0000000001111111	
1200	CURSOR(8,0) = &HOO3F'	Binary 0000000000111111	
1300	CURSOR(9,0) = & HOO1F'	Binary 0000000000011111	
1400	CURSOR(10,0) = & HO1FF'	Binary 0000000111111111	
1500	CURSOR(11,0) = &H10FF	Binary 0001000011111111	
1600	CURSOR(12.0) = & H30FF'	Binary 0011000011111111	
1700	CURSOR(13.0) = & HF87F'	Binary 1111100001111111	
1800	CURSOR(14.0) = & HF87F'	Binary 1111100001111111	
1900	CURSOR(15,0) = &HFC3F	Binary 1111110000111111	
2000	1		
2100	' Define the cursor m	nask	
2200	1		
2300	CURSOR(0,1) = &HOOOO'	Binary 0000000000000000	
2400	CURSOR(1,1) = &H4000'	Binary 010000000000000	
2500	CURSOR(2,1) = & H6000'	Binary 011000000000000	
2600	CURSOR(3,1) = & H7000'	Binary 011100000000000	
2700	CURSOR(4,1) = & H7800'	Binary 0111100000000000	
2800	CURSOR(5,1) = & H7COO'	Binary 0111110000000000	
2900	CURSOR(6,1) = &H7EOO'	'Binary 0111111000000000	
3000	CURSOR ( 7, 1) = & H7F00 '	'Binary 0111111100000000	
3100	CURSOR ( 8, 1) = & H7F80 '	'Binary 0111111110000000	
3200	CURSOR ( 9,1)=&H78CO '	'Binary 0111111111000000	
3300	CURSOR (10, 1) = & H7COO '	'Binary 0111110000000000	
3400	CURSOR (11, 1) =&H4600	'Binary 0100011000000000	
3500	CURSOR (12, 1) = & HO6OO	'Binary 0000011000000000	
3600	CURSOR (13, 1) = & HO300	'Binary 0000001100000000	
3700	CURSOR (14, 1) = & HO3OO	'Binary 0000001100000000	
3800	CURSOR (15, 1) = & HO180	'Binary 0000000110000000	
3900			
4000	' Set the mouse curs	sor shape, color, and hot	
4050	' spot		
4100	j I <sup>−</sup>		
4200	M1% = 9		
4300	M2% = -1 ' Horizonta	al hot spot	
4400	M3% = -1 ' Vertical	hot spot	
4500	4500 CALL MOUSE (M1%, M2%, M3%, CURSOR (0, 0))		

#### **Up Arrow**

The up arrow is a solid, up-directed arrow with the hot spot at the tip. This shape is useful when directing a motion on the screen with the mouse.

```
100
 200 '
        Define the screen mask
 300 '
                            'Binary 1111100111111111
 400 CURSOR (0,0) = & HF 9FF
 500 CURSOR ( 1,0) = & HFOFF
                            'Binary 1111000011111111
600 CURSOR ( 2,0) =&HE07F
700 CURSOR ( 3,0) =&HE07F
800 CURSOR ( 4,0) =&HC03F
                            'Binary 1110000001111111
                             'Binary 1110000001111111
                             'Binary 110000000111111
                             'Binary 110000000111111
 900 CURSOR ( 5,0) = & HCO3F
                             'Binary 100000000011111
1000 \text{ CURSOR}(6,0) = & H801F
                            'Binary 100000000011111
1100 CURSOR (7, 0) = & H801F
1200 CURSOR (8,0) = \& HOOOF
                            'Binary 000000000001111
1300 CURSOR ( 9, 0) = & HOOOF
                             'Binary 000000000001111
                             'Binary 1111000011111111
1400 CURSOR (10, 0) = \& HFOFF
                             'Binary 1111000011111111
1500 CURSOR (11, 0) = \& HFOFF
1600 CURSOR (12, 0) = \& HFOFF
                             'Binary 1111000011111111
                             'Binary 1111000011111111
1700 CURSOR (13, 0) = \& HFOFF
                            'Binary 1111000011111111
1800 CURSOR (14, 0) = \& HFOFF
1900 CURSOR (15,0) = & HFOFF 'Binary 1111000011111111
2000
     . .
2100
         Define the cursor mask
2200 '
2300 CURSOR( 0,1)=&H0000 'Binary 000000000000000
2400 CURSOR ( 1, 1) = & H0600 'Binary 000001100000000
2500 CURSOR ( 2,1) = & HOFOO 'Binary 0000111100000000
2600 CURSOR ( 3, 1) = & HOFOO 'Binary 0000111100000000
2700 CURSOR ( 4, 1) = & H1F80 'Binary 0001111110000000
2800 CURSOR ( 5, 1) = & H1F80 'Binary 0001111110000000
2900 CURSOR ( 6, 1) = & H3FCO 'Binary 0011111111000000
              7,1)=&H3FC0 'Binary 0011111111000000
3000 CURSOR (7,1)=&H3FC0 'Binary 001111111000000
3100 CURSOR (8,1)=&H7FE0 'Binary 01111111100000
3200 CURSOR ( 9,1) = & H0600 'Binary 000001100000000
3300 CURSOR (10, 1) = & H0600 'Binary 000001100000000
3400 CURSOR (11, 1) = & H0600 'Binary 000001100000000
3500 CURSOR (12, 1) = & HO600 'Binary 000001100000000
3600 CURSOR (13, 1) = & HO600 'Binary 000001100000000
3700 CURSOR (14, 1) = & H0600 'Binary 000001100000000
3800 CURSOR (15, 1) = & HOOOO 'Binary 00000000000000000
3900
4000 '
         Set the mouse cursor shape, color, and hot
4050 '
         spot
     . 1
4100
4200 M1 = 9
4300 M2 = 5
               ' Horizontal hot spot
4400 M3 = 0
               ' Vertical hot spot
4500 CALL MOUSE (M1, M2, M3, CURSOR (0, 0))
```

### Left Arrow

The left arrow is a solid, left-directed arrow with the hot spot at the tip. This shape is useful when directing a motion on the screen with the mouse. To generate a right arrow, just reverse the binary bit pattern for each array element and move the hot spot to the new tip. For example, the first element, Binary 111111000011111 (&HFE1F), becomes Binary 111110000111111 (&HF87F).

100	•		
200	' Define the screen	mask	
300	1		
400	CURSOR(0,0) = &HFF1F	'Binary	1111111000011111
500	CURSOR(1,0) = CHEOIE	Binary	111100000011111
600	CURSOR(1,0) = CURSOR(1,0)	Binary	000000000000000000000000000000000000000
700	CURSOR(2,0) = CUOOOO	'Dinary	000000000000000000000000000000000000000
700	CURSOR(3,0) = 2H00000	Dinary	000000000000000000000000000000000000000
800	$CURSUR(4,0) = \alpha H0000$	binary	111100000000000000000000000000000000000
900	CURSUR(5,0) = a HEOIE	Binary	111100000011111
1000	CURSOR(6,0) = & HEETE	Binary	1111111000011111
1100	CURSOR(7,0) = & HFFFF	'Binary	1111111111111111
1200	CURSOR(8,0) = &HFFFF	Binary	11111111111111111
1300	CURSOR(9,0) = & HFFFF	Binary	11111111111111111
1400	CURSOR(10,0) = & HFFFF	'Binary	11111111111111111
1500	CURSOR(11,0) = & HFFFF	'Binary	11111111111111111
1600	CURSOR(12,0) = & HFFFF	'Binary	11111111111111111
1700	CURSOR(13,0) = & HFFFF	'Binary	11111111111111111
1800	CURSOR(14,0) = & HFFFF	'Binary	11111111111111111
1900	CURSOR(15,0) = & HFFFF	'Binary	11111111111111111
2000		-	
2100	' Define the cursor	mask	
2200	1		
2300	CURSOR(0,1) = &HOOOO	'Binarv	000000000000000000000000000000000000000
2400	CURSOR(1,1) = &HOOCO	'Binary	000000011000000
2500	CURSOR(2,1) = &HO7CO	'Binary	0000011111000000
2600	CURSOR(3,1) = &H7FFE	'Binary	0111111111111110
2700	CURSOR(4,1) = &HO7CO	'Binary	0000011111000000
2800	CURSOR(5,1) = &HO7CO	'Binary	000000011000000
2900	CURSOR(61) = &HOOOO	'Binary	000000000000000000000000000000000000000
3000	CURSOR(7,1) = &HOOOO	'Binary	000000000000000000000000000000000000000
3100	CURSOR(8,1) = HOOOO	'Binary	000000000000000000000000000000000000000
3200	CURSOR(9,1) = &HOOOO	'Binary	000000000000000000000000000000000000000
3200	CURSOR(10, 1) = CHOOOO	'Binory	000000000000000000000000000000000000000
3300	CURSOR(10, 1) = CHOOOO	'Binomy	000000000000000000000000000000000000000
3400	CURSOR(11, 1) = CHOOOO	'Binowy	000000000000000000000000000000000000000
3500	CURSOR(12,1) = aH00000	binary	000000000000000000000000000000000000000
3600	CORSOR(13,1) = & HOOOOO	Binary	000000000000000000000000000000000000000
3700	CURSOR(14, 1) = & HOOOO	Binary	000000000000000000000000000000000000000
3800	CURSOR(15,1) = & HOOOO	'Binary	000000000000000000000000000000000000000
3900			
4000	' Set the mouse cur	rsor shap	pe, color, and hot
4050	' spot		
4100	1		
4200	M1 = 9		
4300	M2 = O ' Horizonta	al hot sp	pot
4400	M3 = 3 ' Vertical	hot spot	t
4500	CALL MOUSE (M1, M2, M3	, CURSOR (O	D,O))
# **Check Mark**

The check mark is a solid figure with the hot spot in the center of the "V" formed by the check. This shape can be used when checking off items in a list with the mouse or while a program is checking some aspect of its operation.

```
100
 200 '
         Define the screen mask
 300 '
 400 CURSOR ( 0,0) = & HFFF0 'Binary 1111111111110000
 500 CURSOR ( 1,0) =&HFFE0 'Binary 11111111100000
600 CURSOR ( 2,0) =&HFFC0 'Binary 111111111000000
 700 CURSOR (3,0) = \& HFF81
                             'Binary 111111110000001
 800 CURSOR (4,0) = \& HFFO3
                             'Binary 111111110000011
 900 CURSOR (
                             'Binary 0000011000000111
               5,0) = & HO607
1000 CURSOR (
               6,0) = \& HOOOF
                             'Binary 000000000001111
1100 CURSOR (
               7,0) = & HOO1F
                             'Binary 000000000011111
1200 CURSOR (
               (8, 0) = \& HCO3F
                             'Binary 1100000000111111
1300 CURSOR ( 9,0) = & HF07F
                             'Binary 1111000001111111
1400 CURSOR (10,0) = \& HFFFF
                             'Binary 1111111111111111
1500 CURSOR (11, 0) = \& HFFFF
                             'Binary 1111111111111111
1600 CURSOR (12,0) =&HFFFF
1700 CURSOR (13,0) =&HFFFF
                             'Binary 111111111111111
                             'Binary 1111111111111111
1800 CURSOR (14, 0) = \& HFFFF
                             'Binary 1111111111111111
1900 CURSOR (15,0) =&HFFFF 'Binary 111111111111111
2000
     1
2100
         Define the cursor mask
2200
2300 CURSOR ( 0,1) = & HOOOO 'Binary 000000000000000
2400 \text{ CURSOR}(1,1) = \& HOOO6
                             'Binary 000000000000110
2600 CURSOR (3,1)=&H0018 'Binary 00000000001100
2700 CURSOR (4,1)=&H0030 'Binary 000000000110000
2800 CURSOR ( 5,1) = & HOO60 'Binary 000000001100000
2900 CURSOR ( 6,1) = & H70C0 'Binary 0111000011000000
3000 CURSOR ( 7,1) = & H1D80 'Binary 0001110110000000
3100 CURSOR ( 8,1) = & H0700 'Binary 0000011100000000
3200 CURSOR ( 9,1) = & HOOOO 'Binary 0000000000000000
3300 CURSOR (10, 1) = & HOOOO 'Binary 00000000000000000
3400 CURSOR (11,1) = & HOOOO 'Binary 0000000000000000
3500 CURSOR (12,1) = & HOOOO 'Binary 0000000000000000
                             'Binary 0000000000000000
3600 \text{ CURSOR}(13, 1) = \& H0000
3700 CURSOR (14, 1) = & H0000 'Binary 0000000000000000
3800 CURSOR (15, 1) = & HOOOO 'Binary 0000000000000000
3900
4000
         Set the mouse cursor shape, color, and hot
     1
4050
         spot
     .
4100
4200 M1 = 9
                ' Horizontal hot spot
4300 M2 = 6
                ' Vertical hot spot
4400 M3 = 7
4500 CALL MOUSE (M1, M2, M3, CURSOR (0, 0))
```

# **Pointing Hand**

The pointing hand is a solid figure with the hot spot at the tip of the extended finger. The pointing hand is another convenient shape to use when choosing or selecting items from the screen, especially if the items are represented by icons or symbols such as the keys of a piano keyboard or a calculator.

100	•	
200	' Define the scree	n mask
300	•	
400	CURSOR(0,0) = &HE1FF	'Binary 1110000111111111
500	CURSOR(1,0) = EHE1EE	'Bipary 11100001111111111
600	CURSOR(1,0) = CUE1EE	'Binany 1110000111111111
800	$CORSOR(2,0) = \alpha HEIFF$	
700	CURSOR(3,0) = & HEIFF	Binary 1110000111111111
800	CURSOR(4,0) = & HEIFE	Binary 1110000111111111
900	CURSOR(5,0) = & HEOOO	'Binary 11100000000000000
1000	CURSOR(6,0) = & HEOOO	'Binary 11100000000000000
1100	CURSOR(7,0) = & HEOOO	'Binary 11100000000000000
1200	CURSOR(8,0) = &HOOOO	'Binary 000000000000000000
1300	CURSOR(9,0) = &HOOOO	'Binary 000000000000000000
1400	CURSOR(10,0) = &HOOOO	'Binary 0000000000000000
1500	CURSOR(11, 0) = HOOOO	Binary 000000000000000000
1600	CURSOR(12,0) = CHOOOO	'Binary 000000000000000000000000000000000000
1700	CURSOR(12,0) = CHOOOO	'Binary 000000000000000000000000000000000000
1000	CURSOR(13,0) = & HOOOO	Binary 000000000000000000000000000000000000
1800	CURSUR(14,0) = & H0000	Binary 000000000000000000000000000000000000
1900	CURSOR(15,0) = & HOOOO	Binary 000000000000000000000000000000000000
2000		
2100	' Define the curso	r mask
2200	•	
2300	CURSOR(0,1) = &H1EOO	'Binary 0001111000000000
2400	CURSOR(1,1) = & H1200	'Binary 0001001000000000
2500	CURSOR(2,1) = & H1200	'Binary 0001001000000000
2600	CURSOR(3,1) = & H1200	'Binary 0001001000000000
2700	CURSOR(41) = & H1200	'Binary 0001001000000000
2800	CURSOR(5,1) = H13FF	'Binary 0001001111111111
2000	CURSOP(6,1) = CH1249	'Binary 0001001001001001
2000	CURSOR(0,1) = CU1249	'Binary 0001001001001001
3000	CURSOR(7,1) =	Binary 0001001001001001
3100	$CURSOR(8,1) - \alpha He 249$	
3200	CORSOR(9,1) = & H9001	Binary 1001000000000000000
3300	CORSOR(10, 1) = & H9001	Binary 100100000000000
3400	CURSOR(11, 1) = & H9001	'Binary 1001000000000000
3500	CURSOR(12,1) = & H8001	'Binary 10000000000000000
3600	CURSOR(13, 1) = & H8001	'Binary 1000000000000000
3700	CURSOR(14, 1) = & H8001	'Binary 100000000000000000
3800	CURSOR(15,1) = & HFFFE	'Binary 1111111111111111
3900	1	
4000	' Set the mouse of	rsor shape color and bot
4050	' cnot	i soi snape, coror, and not
4100	spor	
4100	N1 - 0	
4200	M1 = 9	
4300	MZ = 5 Horizont	al not spot
4400	M3 = 0 'Vertical	hot spot
4500	CALL MOUSE (M1.M2.M3	CURSOR (0,0))

# **Diagonal Cross**

The diagonal cross is a solid figure with the hot spot at the center of the cross. This shape is useful as a pointer in a game, or when canceling an operation or deleting an item from a list.

```
100
 200 '
        Define the screen mask
 300
 400 CURSOR( 0,0)=&H07E0 'Binary 0000011111100000
500 CURSOR( 1,0)=&H0180 'Binary 0000000110000000
 600 CURSOR ( 2,0) = & HOOOO 'Binary 000000000000000
 700 CURSOR ( 3,0) = & HCOO3 'Binary 110000000000011
 800 CURSOR ( 4,0) = & HFOOF 'Binary 1111000000001111
 900 CURSOR ( 5,0) = & HCOO3 'Binary 110000000000011
1000 CURSOR ( 6,0) = & H0000 'Binary 0000000000000000
1100 CURSOR ( 7,0) = & H0180 'Binary 0000000110000000
1200 CURSOR ( 8,0)=&H07E0 'Binary 000001111000000
1300 CURSOR ( 9,0)=&HFFFF 'Binary 111111111111
                           'Binary 1111111111111111
1400 CURSOR (10, 0) = \& HFFFF
                           'Binary 1111111111111111
1500 CURSOR (11, 0) = \& HFFFF
                           'Binary 1111111111111111
1600 CURSOR (12, 0) = \& HFFFF
1700 CURSOR (13,0) = & HFFFF 'Binary 1111111111111111
1800 CURSOR (14,0) = & HFFFF 'Binary 111111111111111
1900 CURSOR (15,0) = & HFFFF 'Binary 1111111111111111
2000
2100 '
        Define the cursor mask
2200
2300 CURSOR( 0,1)=&H0000 'Binary 000000000000000
2400 CURSOR(1,1)=&H700E 'Binary 011100000001110
2500 CURSOR(2,1)=&H1C38 'Binary 00011000011100
2600 CURSOR ( 3,1) = & H0660 'Binary 0000011001100000
2700 CURSOR ( 4,1) = & HO3CO 'Binary 0000001111000000
2800 CURSOR ( 5,1) = & H0660 'Binary 0000011001100000
2900 CURSOR ( 6,1) = & H1C38 'Binary 0001110000111000
3000 CURSOR (7,1) = & H700E 'Binary 0111000000001110
3100 CURSOR (
              8,1)=&H0000 'Binary 0000000000000000
3400 CURSOR (11, 1) = & H0000 'Binary 000000000000000
3500 CURSOR (12,1) = & HOOOO 'Binary 0000000000000000
3600 CURSOR (13, 1) = & H0000 'Binary 0000000000000000
3700 CURSOR (14, 1) = & H0000 'Binary 0000000000000000
3800 CURSOR (15,1) = & HOOOO 'Binary 00000000000000000
3900
4000
         Set the mouse cursor shape, color, and hot
     .
4050
         spot
4100
4200 M1 = 9
4300 M2 = 7
                ' Horizontal hot spot
                ' Vertical hot spot
4400 M3 = 4
4500 CALL MOUSE (M1, M2, M3, CURSOR (0, 0))
```

# **Rectangular Cross**

The rectangular cross is a solid figure with the hot spot at the center of the cross. This shape is useful as a pointer in a game, or when inserting items into a list.

<pre>200 ' Define the screen mask 300 ' 400 CURSOR( 0,0)=&amp;HFC3F 'Binary 11111000011111 500 CURSOR( 1,0)=&amp;HFC3F 'Binary 11111000011111 700 CURSOR( 2,0)=&amp;HFC3F 'Binary 11111000011111 700 CURSOR( 4,0)=&amp;HO000 'Binary 00000000000000 900 CURSOR( 4,0)=&amp;HO000 'Binary 000000000000000 1000 CURSOR( 5,0)=&amp;HO000 'Binary 11111000011111 1100 CURSOR( 7,0)=&amp;HFC3F 'Binary 11111000011111 1200 CURSOR( 7,0)=&amp;HFC3F 'Binary 11111000011111 1300 CURSOR( 8,0)=&amp;HFC3F 'Binary 111111000011111 1300 CURSOR( 1,0)=&amp;HFFFF 'Binary 111111111111 1400 CURSOR( 1,0)=&amp;HFFFF 'Binary 1111111111111 1500 CURSOR( 1,0)=&amp;HFFFF 'Binary 1111111111111 1500 CURSOR( 1,0)=&amp;HFFFF 'Binary 11111111111111 1600 CURSOR( 1,0)=&amp;HFFFF 'Binary 11111111111111 1700 CURSOR( 1,0)=&amp;HFFFF 'Binary 11111111111111 1800 CURSOR( 1,0)=&amp;HFFFF 'Binary 11111111111111 1900 CURSOR( 1,0)=&amp;HFFFF 'Binary 11111111111111 1900 CURSOR( 1,0)=&amp;HFFFF 'Binary 11111111111111 1900 CURSOR( 1,0)=&amp;HFFFF 'Binary 111111111111111 1900 CURSOR( 1,0)=&amp;HFFFF 'Binary 00000000000000 2400 CURSOR( 1,1)=&amp;H0000 'Binary 0000000110000000 2500 CURSOR( 2,1)=&amp;H0180 'Binary 0000000110000000 2500 CURSOR( 3,1)=&amp;H0180 'Binary 0000000110000000 2500 CURSOR( 4,1)=&amp;H7FFF 'Binary 01111111111110 2800 CURSOR( 4,1)=&amp;H7FFF 'Binary 01111111111110 2800 CURSOR( 4,1)=&amp;H7FFF 'Binary 01111111111110 2800 CURSOR( 5,1)=&amp;H0180 'Binary 0000000110000000 2900 CURSOR( 6,1)=&amp;H0180 'Binary 0000000110000000 2900 CURSOR( 6,1)=&amp;H0180 'Binary 0000000110000000 3000 CURSOR( 7,1)=&amp;H0000 'Binary 00000000000000 3000 CURSOR( 1,1)=&amp;H0000 'Binary 00000000000000000 3000 CURSOR( 1,1)=&amp;H0000 'Binary 000000000000000 3000 CURSOR( 1,1)=&amp;H0000 'Binary 00000000000000 3000 CURSOR( 1,1)=&amp;H0000 'Binary 00000000000000 3000 CURSOR( 1,1)=&amp;H0000 'Binary 000000000000000 3000 CURSOR( 1,1)=&amp;H0000 'Binary 00000000000000000000000 3000 CURSOR( 1,1)=&amp;H0000 'Binary 000000000000000000000000000000000000</pre>	100	1
<pre>300 ' Define the cursor mask 300 ' South of the set of the s</pre>	200	' Define the screen mask
400 CURSOR( 0,0) =&HFC3F 'Binary 1111110000111111 500 CURSOR( 1,0) =&HFC3F 'Binary 1111110000111111 600 CURSOR( 2,0) =&HFC3F 'Binary 111110000111111 700 CURSOR( 3,0) =&H0000 'Binary 000000000000000 900 CURSOR( 4,0) =&H0000 'Binary 000000000000000 900 CURSOR( 5,0) =&H0000 'Binary 000000000000000 900 CURSOR( 5,0) =&HFC3F 'Binary 111110000111111 1000 CURSOR( 7,0) =&HFC3F 'Binary 1111110000111111 1100 CURSOR( 7,0) =&HFC3F 'Binary 1111110000111111 1200 CURSOR( 9,0) =&HFC3F 'Binary 11111110000111111 1300 CURSOR( 10,0) =&HFFFF 'Binary 11111111111111 1400 CURSOR( 10,0) =&HFFFF 'Binary 111111111111111 1500 CURSOR( 12,0) =&HFFFF 'Binary 111111111111111 1600 CURSOR( 12,0) =&HFFFF 'Binary 111111111111111 1700 CURSOR( 12,0) =&HFFFF 'Binary 11111111111111 1800 CURSOR( 13,0) =&HFFFF 'Binary 11111111111111 1900 CURSOR( 13,0) =&HFFFF 'Binary 111111111111111 2000 ' 2100 ' Define the cursor mask 2200 ' 2100 CURSOR( 0,1) =&H0000 'Binary 000000000000000 2500 CURSOR( 1,1) =&H0180 'Binary 0000000110000000 2500 CURSOR( 2,1) =&H0180 'Binary 0000000110000000 2500 CURSOR( 3,1) =&H0180 'Binary 0000000110000000 2700 CURSOR( 4,1) =&H7FFE 'Binary 011111111111111 2800 CURSOR( 5,1) =&H0180 'Binary 0000000110000000 2900 CURSOR( 4,1) =&H0180 'Binary 0000000110000000 2000 CURSOR( 5,1) =&H0180 'Binary 0000000110000000 3000 CURSOR( 5,1) =&H0180 'Binary 0000000110000000 3000 CURSOR( 7,1) =&H0000 'Binary 000000010000000 3000 CURSOR( 12,1) =&H0000 'Binary 0000000000000 3000 CURSOR( 12,1) =&H0000 'Binary 00000000000000 3000 CURSOR( 12,1) =&H0000 'Binary 0000000000000000000 3000 CURSOR( 12,1) =&H0000 'Binary 000000000000000000000000000000000000	300	1
<pre>500 CURSOR( 1,0) =&amp;HFC3F 'Binary 1111110000111111 600 CURSOR( 2,0) =&amp;HFC3F 'Binary 1111110000111111 700 CURSOR( 3,0) =&amp;HO000 'Binary 00000000000000 900 CURSOR( 4,0) =&amp;HO000 'Binary 000000000000000 900 CURSOR( 5,0) =&amp;HO000 'Binary 000000000000000 900 CURSOR( 7,0) =&amp;HFC3F 'Binary 111110000111111 1200 CURSOR( 7,0) =&amp;HFC3F 'Binary 111110000111111 1200 CURSOR( 1,0) =&amp;HFFFF 'Binary 1111110000111111 1300 CURSOR( 1,0) =&amp;HFFFF 'Binary 1111110000111111 1400 CURSOR( 1,0) =&amp;HFFFF 'Binary 1111111111111 1500 CURSOR( 1,0) =&amp;HFFFF 'Binary 11111111111111 1600 CURSOR( 1,0) =&amp;HFFFF 'Binary 11111111111111 1700 CURSOR( 1,0) =&amp;HFFFF 'Binary 11111111111111 1600 CURSOR( 1,0) =&amp;HFFFF 'Binary 11111111111111 1700 CURSOR( 1,0) =&amp;HFFFF 'Binary 11111111111111 1800 CURSOR( 1,0) =&amp;HFFFF 'Binary 11111111111111 1900 CURSOR( 1,0) =&amp;HFFFF 'Binary 11111111111111 1900 CURSOR( 1,1) =&amp;HO000 'Binary 000000000000000 2400 CURSOR( 0,1) =&amp;HO000 'Binary 000000110000000 2500 CURSOR( 1,1) =&amp;HO180 'Binary 000000110000000 2600 CURSOR( 1,1) =&amp;HO180 'Binary 000000110000000 2600 CURSOR( 1,1) =&amp;HO180 'Binary 000000110000000 2600 CURSOR( 5,1) =&amp;HO180 'Binary 000000110000000 2600 CURSOR( 5,1) =&amp;HO180 'Binary 0000000110000000 2600 CURSOR( 5,1) =&amp;HO180 'Binary 00000001000000 2600 CURSOR( 5,1) =&amp;HO000 'Binary 000000010000000 2600 CURSOR( 5,1) =&amp;HO000 'Binary 00000000000000 2600 CURSOR( 5,1) =&amp;HO000 'Binary 0000000000000 2600 CURSOR( 1,1) =&amp;HO000 'Binary 000000000000000 2600 CURSOR( 1,1) =&amp;HO000 'Binary 000000000000000 2600 CURSOR( 1,1) =&amp;HO000 'Binary 000000000000000 2600 CURSOR( 1,1) =&amp;HO000 'Binary 0000000000000000 2600 CURSOR( 1,1) =&amp;HO000 'Binary 0000000000000000 2600 CURSOR( 1,1) =&amp;HO000 'Binary 000000000000000 2600 CURSOR( 1,1) =&amp;HO000 'Binary 00000000000000000 2600 CURSOR( 1,1) =&amp;HO000 'Binary 000000000000000 2600 CURSOR( 1,1) =&amp;HO000 'Binary 0000000000000000 2600 CURSOR( 1,1) =&amp;HO000 'Binary 00000000000000</pre>	400	CURSOR(0,0) = & HFC3F 'Binary 1111110000111111
<pre>600 CURSOR( 2,0) =&amp;HFC3F 'Binary 111111000011111 700 CURSOR( 3,0) =&amp;H0000 'Binary 00000000000000 800 CURSOR( 4,0) =&amp;H0000 'Binary 00000000000000 900 CURSOR( 6,0) =&amp;HFC3F 'Binary 11111000011111 1100 CURSOR( 6,0) =&amp;HFC3F 'Binary 11111000011111 1200 CURSOR( 8,0) =&amp;HFC3F 'Binary 11111000011111 1300 CURSOR( 9,0) =&amp;HFC3F 'Binary 111111000011111 1300 CURSOR( 9,0) =&amp;HFC3F 'Binary 111111000011111 1400 CURSOR( 10,0) =&amp;HFC3F 'Binary 1111111111111 1500 CURSOR( 11,0) =&amp;HFC3F 'Binary 11111111111111 1500 CURSOR( 12,0) =&amp;HFFFF 'Binary 11111111111111 1600 CURSOR( 12,0) =&amp;HFFFF 'Binary 11111111111111 1700 CURSOR( 13,0) =&amp;HFFFF 'Binary 11111111111111 1800 CURSOR( 14,0) =&amp;HFFFF 'Binary 1111111111111 1900 CURSOR( 14,0) =&amp;HFFFF 'Binary 11111111111111 2000 ' 2100 ' Define the cursor mask 2200 ' 2300 CURSOR( 0,1) =&amp;H0000 'Binary 000000000000000 2400 CURSOR( 1,1) =&amp;H0180 'Binary 000000110000000 2500 CURSOR( 2,1) =&amp;H0180 'Binary 000000110000000 2500 CURSOR( 3,1) =&amp;H0180 'Binary 000000110000000 2600 CURSOR( 5,1) =&amp;H0180 'Binary 000000110000000 2700 CURSOR( 6,1) =&amp;H0180 'Binary 0000000110000000 2000 CURSOR( 6,1) =&amp;H0180 'Binary 0000000000000 2000 CURSOR( 6,1) =&amp;H0000 'Binary 00000000000000 2000 CURSOR( 12,1) =&amp;H0000 'Binary 000000000000000 2000 CURSOR( 12,1) =&amp;H0000 'Binary 00000000000000 2000 CURSOR( 12,1) =&amp;H0000 'Binary 00000000000000 2000 CURSOR( 12,1) =&amp;H0000 'Binary 000000000000000 2000 CURSOR( 12,1) =&amp;H0000 'Binary 000000000000000 2000 CURSOR( 12,1) =&amp;H0000 'Binary 000000000000000 2000 CURSOR( 13,1) =&amp;H0000 'Binary 000000000000000000000000000000000000</pre>	500	CURSOR(1,0) = & HFC3F 'Binary 1111110000111111
700       CURSOR( 3,0) =&HOOOO       'Binary 000000000000000000000000000000000000	600	CURSOR(2,0) = & HFC3F 'Binary 1111110000111111
800 CURSOR( 4,0) =&H0000 'Binary 000000000000000000000000000000000000	700	CURSOR(3,0) = cHOOOO Binary 000000000000000000000000000000000000
000       CURSOR ( 1,0) =&HOOOO       'Binary       COOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO	800	CURSOR(4,0) = cHOOOO Binary 000000000000000000000000000000000000
<pre>1000 CURSOR( 5,0)=&amp;HFC3F 'Binary 111110000111111 1100 CURSOR( 7,0)=&amp;HFC3F 'Binary 111110000111111 1200 CURSOR( 8,0)=&amp;HFC3F 'Binary 111110000111111 1300 CURSOR( 10,0)=&amp;HFFFF 'Binary 111111111111111 1400 CURSOR(10,0)=&amp;HFFFF 'Binary 11111111111111 1500 CURSOR(11,0)=&amp;HFFFF 'Binary 11111111111111 1500 CURSOR(12,0)=&amp;HFFFF 'Binary 11111111111111 1600 CURSOR(14,0)=&amp;HFFFF 'Binary 11111111111111 1700 CURSOR(14,0)=&amp;HFFFF 'Binary 11111111111111 1800 CURSOR(14,0)=&amp;HFFFF 'Binary 1111111111111 1900 CURSOR(15,0)=&amp;HFFFF 'Binary 1111111111111 1900 CURSOR(15,0)=&amp;HFFFF 'Binary 1111111111111 1900 CURSOR(15,0)=&amp;HFFFF 'Binary 000000000000000 2100 CURSOR( 0,1)=&amp;H0180 'Binary 0000000110000000 2500 CURSOR( 1,1)=&amp;H0180 'Binary 000000110000000 2500 CURSOR( 3,1)=&amp;H0180 'Binary 000000110000000 2700 CURSOR( 4,1)=&amp;H7FFE 'Binary 0111111111110 2800 CURSOR( 5,1)=&amp;H0180 'Binary 000000110000000 2900 CURSOR( 6,1)=&amp;H0180 'Binary 000000110000000 2000 CURSOR( 6,1)=&amp;H0180 'Binary 0000000110000000 2000 CURSOR( 6,1)=&amp;H0180 'Binary 0000000110000000 2000 CURSOR( 1,1)=&amp;H0000 'Binary 000000010000000 2000 CURSOR( 1,1)=&amp;H0000 'Binary 0000000000000000 2000 CURSOR( 1,1)=&amp;H0000 'Binary 0000000000000000 2000 CURSOR( 1,1)=&amp;H0000 'Binary 0000000000000000 2000 CURSOR( 1,1)=&amp;H0000 'Binary 000000000000000 2000 CURSOR( 11,1)=&amp;H0000 'Binary 000000000000000 2000 CURSOR( 11,1)=&amp;H0000 'Binary 000000000000000 2000 CURSOR( 11,1)=&amp;H0000 'Binary 00000000000000 2000 CURSOR( 11,1)=&amp;H0000 'Binary 00000000000000 2000 CURSOR( 11,1)=&amp;H0000 'Binary 000000000000000 2000 'A000 CURSOR( 11,1)=&amp;H0000 'Binary 00000000000000000 2000 CURSOR( 11,1)=&amp;H0000 'Binary 000000000000000000000000000000000000</pre>	000	CURSOR(-4,0) = ano coord Binary 000000000000000000000000000000000000
<pre>1000 CURSOR( 0,0)-AHECSF 'Binary lllllloodollllll 1100 CURSOR( 8,0)=&amp;HECSF 'Binary lllllloodollllll 1200 CURSOR( 9,0)=&amp;HECSF 'Binary lllllloodollllll 1300 CURSOR( 10,0)=&amp;HEFFF 'Binary lllllllllllllllllll 1400 CURSOR(11,0)=&amp;HEFFF 'Binary llllllllllllllllll 1500 CURSOR(12,0)=&amp;HEFFF 'Binary lllllllllllllllllll 1500 CURSOR(12,0)=&amp;HFFFF 'Binary lllllllllllllllllll 1500 CURSOR(13,0)=&amp;HFFFF 'Binary lllllllllllllllllllll 1500 CURSOR(14,0)=&amp;HFFFF 'Binary lllllllllllllllllllll 1500 CURSOR(15,0)=&amp;HFFFF 'Binary lllllllllllllllllllllllll 1500 CURSOR(15,0)=&amp;HFFFF 'Binary llllllllllllllllllllllllllllllllllll</pre>	1000	CURSOR((5,0) = CUECCE / Binary 000000000000000000000000000000000000
<pre>1100 CURSOR( 7,0)-AHECSF 'Binary 11111000011111 1200 CURSOR( 8,0)=&amp;HECSF 'Binary 111111000011111 1300 CURSOR(10,0)=&amp;HEFFF 'Binary 111111111111111 1500 CURSOR(11,0)=&amp;HEFFF 'Binary 111111111111111 1600 CURSOR(12,0)=&amp;HEFFF 'Binary 111111111111111 1700 CURSOR(12,0)=&amp;HEFFF 'Binary 111111111111111 1800 CURSOR(14,0)=&amp;HEFFF 'Binary 111111111111111 1900 CURSOR(15,0)=&amp;HEFFF 'Binary 111111111111111 1900 CURSOR(15,0)=&amp;HEFFF 'Binary 111111111111111 2000 ' 2100 ' Define the cursor mask 2200 ' 2300 CURSOR( 0,1)=&amp;HOOOO 'Binary 0000000000000000 2400 CURSOR( 1,1)=&amp;HO18O 'Binary 000000110000000 2500 CURSOR( 2,1)=&amp;H018O 'Binary 000000110000000 2600 CURSOR( 3,1)=&amp;H018O 'Binary 000000110000000 2700 CURSOR( 4,1)=&amp;H018O 'Binary 000000110000000 2900 CURSOR( 5,1)=&amp;H018O 'Binary 000000110000000 3000 CURSOR( 6,1)=&amp;H018O 'Binary 000000110000000 3000 CURSOR( 7,1)=&amp;H018O 'Binary 000000110000000 3000 CURSOR( 8,1)=&amp;H018O 'Binary 0000000110000000 3000 CURSOR( 1,1)=&amp;H0000 'Binary 000000010000000 3000 CURSOR( 1,1)=&amp;H0000 'Binary 0000000000000000 3000 CURSOR( 11,1)=&amp;H0000 'Binary 000000000000000 3000 CURSOR( 11,1)=&amp;H0000 'Binary 000000000000000 3000 CURSOR( 11,1)=&amp;H0000 'Binary 0000000000000000 3000 CURSOR( 11,1)=&amp;H0000 'Binary 000000000000000 3000 CURSOR( 11,1)=&amp;H0000 'Binary 000000000000000 3000 CURSOR( 11,1)=&amp;H0000 'Binary 0000000000000000 3000 CURSOR( 11,1)=&amp;H0000 'Binary 000000000000000 3000 CURSOR( 11,1)=&amp;H0000 'Binary 000000000000000 3000 CURSOR( 11,1)=&amp;H0000 'Binary 000000000000000 3000 CURSOR( 11,1)=&amp;H0000 'Binary 000000000000000 3000 CURSOR( 11,1)=&amp;H0000 'Binary 0000000000000000000 3000 CURSOR( 15,1)=&amp;H0000 'Binary 000000000000000000000000000000000000</pre>	11000	CURSOR(0,0) = are CSF Binary IIIIIOOOOIIIIII
<pre>1200 CURSOR( 8,0)=&amp;HFCSF 'Binary 11111000011111 1300 CURSOR(10,0)=&amp;HFFFF 'Binary 111111111111111 1400 CURSOR(10,0)=&amp;HFFFF 'Binary 111111111111111 1500 CURSOR(11,0)=&amp;HFFFF 'Binary 111111111111111 1600 CURSOR(12,0)=&amp;HFFFF 'Binary 111111111111111 1700 CURSOR(13,0)=&amp;HFFFF 'Binary 111111111111111 1800 CURSOR(14,0)=&amp;HFFFF 'Binary 111111111111111 1900 CURSOR(15,0)=&amp;HFFFF 'Binary 111111111111111 2000 ' 2100 ' Define the cursor mask 2200 ' 2300 CURSOR( 0,1)=&amp;H0180 'Binary 0000000110000000 2400 CURSOR( 1,1)=&amp;H0180 'Binary 000000110000000 2500 CURSOR( 2,1)=&amp;H0180 'Binary 000000110000000 2700 CURSOR( 4,1)=&amp;H0180 'Binary 000000110000000 2700 CURSOR( 5,1)=&amp;H0180 'Binary 000000110000000 2900 CURSOR( 6,1)=&amp;H0180 'Binary 000000110000000 3000 CURSOR( 7,1)=&amp;H0180 'Binary 000000110000000 3000 CURSOR( 6,1)=&amp;H0180 'Binary 0000000110000000 3000 CURSOR( 7,1)=&amp;H0180 'Binary 0000000110000000 3000 CURSOR( 1,1)=&amp;H0000 'Binary 000000000000000 3000 CURSOR( 1,1)=&amp;H0000 'Binary 000000000000000 3000 CURSOR( 10,1)=&amp;H0000 'Binary 000000000000000 3000 CURSOR( 11,1)=&amp;H0000 'Binary 000000000000000 3000 CURSOR( 12,1)=&amp;H0000 'Binary 000000000000000 3000 CURSOR( 11,1)=&amp;H0000 'Binary 00000000000000000 3000 CURSOR( 12,1)=&amp;H0000 'Binary 0000000000000000 3000 CURSOR( 12,1)=&amp;H0000 'Binary 000000000000000 3000 CURSOR( 11,1)=&amp;H0000 'Binary 000000000000000 3000 CURSOR( 12,1)=&amp;H0000 'Binary 000000000000000 3000 CURSOR( 12,1)=&amp;H0000 'Binary 000000000000000 3000 CURSOR( 12,1)=&amp;H0000 'Binary 000000000000000 3000 CURSOR( 12,1)=&amp;H0000 'Binary 000000000000000000000000000000000000</pre>	1200	$CURSOR(7,0) = \alpha HeCSE Binary IIIIIO000111111$
<pre>1300 CURSOR( 9,0) =&amp;HFFFF 'Binary 111111111111111 1400 CURSOR(10,0) =&amp;HFFFF 'Binary 1111111111111111 1500 CURSOR(11,0) =&amp;HFFFF 'Binary 111111111111111111111111111111111111</pre>	1200	$CURSOR(8,0) = \alpha HFC3F$ Binary IIIIIIOOOOIIIIII
1400 CURSOR (10,0) = & HFFFF 'Binary 111111111111111 1500 CURSOR (11,0) = & HFFFF 'Binary 11111111111111111 1600 CURSOR (13,0) = & HFFFF 'Binary 111111111111111111111111111111111111	1300	CURSOR(9,0) = & HFFFF Binary IIIIIIIIIIIII
<pre>1500 CURSOR(11,0) =&amp;HFFF 'Binary 11111111111111 1600 CURSOR(12,0) =&amp;HFFFF 'Binary 11111111111111 1700 CURSOR(13,0) =&amp;HFFFF 'Binary 11111111111111 1800 CURSOR(14,0) =&amp;HFFFF 'Binary 11111111111111 1900 CURSOR(15,0) =&amp;HFFFF 'Binary 11111111111111 2000 ' 2100 ' Define the cursor mask 2200 ' 2300 CURSOR(0,1) =&amp;H0000 'Binary 00000000000000 2400 CURSOR(1,1) =&amp;H0180 'Binary 000000110000000 2500 CURSOR(2,1) =&amp;H0180 'Binary 0000000110000000 2600 CURSOR(3,1) =&amp;H0180 'Binary 0000000110000000 2700 CURSOR(4,1) =&amp;H0180 'Binary 000000110000000 2900 CURSOR(5,1) =&amp;H0180 'Binary 000000110000000 3000 CURSOR(6,1) =&amp;H0180 'Binary 0000000110000000 3100 CURSOR(7,1) =&amp;H0180 'Binary 0000000110000000 3200 CURSOR(8,1) =&amp;H0000 'Binary 000000000000000 3300 CURSOR(10,1) =&amp;H0000 'Binary 000000000000000 3400 CURSOR(12,1) =&amp;H0000 'Binary 000000000000000 3500 CURSOR(13,1) =&amp;H0000 'Binary 0000000000000000 3600 CURSOR(14,1) =&amp;H0000 'Binary 0000000000000000 3600 CURSOR(14,1) =&amp;H0000 'Binary 000000000000000 3600 CURSOR(15,1) =&amp;H0000 'Binary 000000000000000 3600 CURSOR(14,1) =&amp;H0000 'Binary 000000000000000 3600 CURSOR(14,1) =&amp;H0000 'Binary 000000000000000000000000000000000000</pre>	1400	CORSOR(10,0) = & HEFFF Binary IIIIIIIIIIIIIII
<pre>1600 CURSOR(12,0)=&amp;HFFFF 'Binary llllllllllllllllllllllllllll 1700 CURSOR(13,0)=&amp;HFFFF 'Binary lllllllllllllllllllll 1800 CURSOR(14,0)=&amp;HFFFF 'Binary lllllllllllllllllllll 1900 CURSOR(15,0)=&amp;HFFFF 'Binary llllllllllllllllllll 2000 ' 2100 ' Define the cursor mask 2200 ' 2300 CURSOR(0,1)=&amp;H0000 'Binary 00000000000000 2400 CURSOR(1,1)=&amp;H0180 'Binary 0000000110000000 2500 CURSOR(2,1)=&amp;H0180 'Binary 0000000110000000 2600 CURSOR(3,1)=&amp;H0180 'Binary 0000000110000000 2700 CURSOR(4,1)=&amp;H7FFE 'Binary 0111111111110 2800 CURSOR(5,1)=&amp;H0180 'Binary 0000000110000000 3000 CURSOR(6,1)=&amp;H0180 'Binary 0000000110000000 3000 CURSOR(6,1)=&amp;H0180 'Binary 0000000110000000 3100 CURSOR(6,1)=&amp;H0180 'Binary 000000000000000 3200 CURSOR(9,1)=&amp;H0000 'Binary 000000000000000 3300 CURSOR(10,1)=&amp;H0000 'Binary 0000000000000000 3400 CURSOR(12,1)=&amp;H0000 'Binary 000000000000000 3500 CURSOR(12,1)=&amp;H0000 'Binary 000000000000000 3600 CURSOR(13,1)=&amp;H0000 'Binary 000000000000000 3600 CURSOR(14,1)=&amp;H0000 'Binary 0000000000000000 3600 CURSOR(15,1)=&amp;H0000 'Binary 000000000000000000000000000000000000</pre>	1500	CURSOR(11,0) = & HFFFF Binary 111111111111111111111111111111111111
<pre>1700 CURSOR(13,0) =&amp;HFFFF 'Binary 11111111111111 1800 CURSOR(14,0) =&amp;HFFFF 'Binary 11111111111111 1900 CURSOR(15,0) =&amp;HFFFF 'Binary 11111111111111 2000 ' Define the cursor mask 2200 ' 2300 CURSOR( 0,1) =&amp;H0000 'Binary 00000000000000 2400 CURSOR( 1,1) =&amp;H0180 'Binary 000000110000000 2500 CURSOR( 2,1) =&amp;H0180 'Binary 000000110000000 2600 CURSOR( 3,1) =&amp;H0180 'Binary 000000110000000 2700 CURSOR( 4,1) =&amp;H7FFE 'Binary 0111111111111 2800 CURSOR( 5,1) =&amp;H0180 'Binary 000000110000000 2900 CURSOR( 6,1) =&amp;H0180 'Binary 000000110000000 3000 CURSOR( 6,1) =&amp;H0180 'Binary 000000110000000 3100 CURSOR( 7,1) =&amp;H0180 'Binary 0000000110000000 3200 CURSOR( 8,1) =&amp;H0180 'Binary 00000000000000 3300 CURSOR( 10,1) =&amp;H0000 'Binary 000000000000000 3400 CURSOR( 10,1) =&amp;H0000 'Binary 000000000000000 3500 CURSOR( 11,1) =&amp;H0000 'Binary 00000000000000 3500 CURSOR( 12,1) =&amp;H0000 'Binary 00000000000000 3600 CURSOR( 12,1) =&amp;H0000 'Binary 000000000000000 3600 CURSOR( 12,1) =&amp;H0000 'Binary 00000000000000 3600 CURSOR( 15,1) =&amp;H0000 'Binary 000000000000000 3600 CURSOR( 15,1) =&amp;H0000 'Binary 000000000000000 3600 CURSOR( 15,1) =&amp;H0000 'Binary 0000000000000000 3600 CURSOR( 15,1) =&amp;H0000 'Binary 000000000000000000000000000000000000</pre>	1600	CURSOR (12,0) = & HFFFF 'Binary 111111111111111
<pre>1800 CURSOR (14,0) =&amp;HFFFF 'Binary 111111111111111 1900 CURSOR (15,0) =&amp;HFFFF 'Binary 11111111111111 2000 ' 2100 ' Define the cursor mask 2200 ' 2300 CURSOR ( 0,1) =&amp;H0000 'Binary 00000000000000 2400 CURSOR ( 1,1) =&amp;H0180 'Binary 000000110000000 2600 CURSOR ( 2,1) =&amp;H0180 'Binary 000000110000000 2700 CURSOR ( 3,1) =&amp;H0180 'Binary 0000000110000000 2700 CURSOR ( 4,1) =&amp;H7FFE 'Binary 0111111111111 2800 CURSOR ( 5,1) =&amp;H0180 'Binary 000000110000000 2900 CURSOR ( 6,1) =&amp;H0180 'Binary 000000110000000 3000 CURSOR ( 6,1) =&amp;H0180 'Binary 0000000110000000 3100 CURSOR ( 8,1) =&amp;H0180 'Binary 00000000000000 3200 CURSOR ( 8,1) =&amp;H0000 'Binary 00000000000000 3400 CURSOR ( 10,1) =&amp;H0000 'Binary 00000000000000 3500 CURSOR (12,1) =&amp;H0000 'Binary 000000000000000 3600 CURSOR (12,1) =&amp;H0000 'Binary 000000000000000 3600 CURSOR (12,1) =&amp;H0000 'Binary 000000000000000 3600 CURSOR (12,1) =&amp;H0000 'Binary 0000000000000000 3600 CURSOR (12,1) =&amp;H0000 'Binary 000000000000000000000000000000000000</pre>	1700	CURSOR (13,0) = & HFFFF 'Binary 1111111111111111
<pre>1900 CURSOR (15,0) = &amp; HFFFF 'Binary 11111111111111111111 2000 ' 2100 ' Define the cursor mask 2200 ' 2300 CURSOR ( 0,1) = &amp; H0000 'Binary 0000000000000 2400 CURSOR ( 1,1) = &amp; H0180 'Binary 000000110000000 2600 CURSOR ( 2,1) = &amp; H0180 'Binary 000000110000000 2700 CURSOR ( 3,1) = &amp; H0180 'Binary 0111111111110 2800 CURSOR ( 4,1) = &amp; H7FFE 'Binary 01111111111110 2800 CURSOR ( 5,1) = &amp; H0180 'Binary 000000110000000 2900 CURSOR ( 6,1) = &amp; H0180 'Binary 0000000110000000 3000 CURSOR ( 6,1) = &amp; H0180 'Binary 0000000110000000 3100 CURSOR ( 8,1) = &amp; H0180 'Binary 00000000000000 3200 CURSOR ( 8,1) = &amp; H0000 'Binary 00000000000000 3300 CURSOR ( 10,1) = &amp; H0000 'Binary 000000000000000 3400 CURSOR (11,1) = &amp; H0000 'Binary 000000000000000 3500 CURSOR (12,1) = &amp; H0000 'Binary 000000000000000 3600 CURSOR (13,1) = &amp; H0000 'Binary 000000000000000 3600 CURSOR (14,1) = &amp; H0000 'Binary 0000000000000000 3600 CURSOR (15,1) = &amp; H0000 'Binary 00000000000000000 3600 CURSOR (15,1) = &amp; H0000 'Binary 000000000000000000000000000000000000</pre>	1800	CURSOR (14,0) = & HFFFF 'Binary 111111111111111
2000 ' 2100 ' Define the cursor mask 2200 ' 2300 CURSOR( 0,1)=&H0000 'Binary 0000000000000 2400 CURSOR( 1,1)=&H0180 'Binary 00000011000000 2500 CURSOR( 2,1)=&H0180 'Binary 000000110000000 2600 CURSOR( 3,1)=&H0180 'Binary 000000110000000 2700 CURSOR( 4,1)=&H7FFE 'Binary 0111111111111 2800 CURSOR( 5,1)=&H0180 'Binary 000000110000000 3000 CURSOR( 6,1)=&H0180 'Binary 0000000110000000 3000 CURSOR( 7,1)=&H0180 'Binary 00000000000000 3000 CURSOR( 8,1)=&H0180 'Binary 00000000000000 3000 CURSOR( 8,1)=&H0000 'Binary 00000000000000 3000 CURSOR(10,1)=&H0000 'Binary 000000000000000 3400 CURSOR(11,1)=&H0000 'Binary 00000000000000 3500 CURSOR(12,1)=&H0000 'Binary 00000000000000 3600 CURSOR(13,1)=&H0000 'Binary 00000000000000 3700 CURSOR(15,1)=&H0000 'Binary 00000000000000 3600 CURSOR(15,1)=&H0000 'Binary 000000000000000 3600 CURSOR(15,1)=&H0000 'Binary 000000000000000 3600 CURSOR(15,1)=&H0000 'Binary 000000000000000000000000000000000000	1900	CURSOR (15,0) = & HFFFF 'Binary 111111111111111
<pre>2100 ' Define the cursor mask 2200 ' 2300 CURSOR( 0,1)=&amp;H0000 'Binary 0000000000000000 2400 CURSOR( 1,1)=&amp;H0180 'Binary 000000110000000 2500 CURSOR( 2,1)=&amp;H0180 'Binary 000000110000000 2600 CURSOR( 3,1)=&amp;H0180 'Binary 000000110000000 2700 CURSOR( 4,1)=&amp;H7FFE 'Binary 0111111111110 2800 CURSOR( 5,1)=&amp;H0180 'Binary 000000110000000 3000 CURSOR( 6,1)=&amp;H0180 'Binary 000000110000000 3100 CURSOR( 8,1)=&amp;H0180 'Binary 000000000000000 3200 CURSOR( 8,1)=&amp;H0000 'Binary 000000000000000 3300 CURSOR( 10,1)=&amp;H0000 'Binary 000000000000000 3400 CURSOR(10,1)=&amp;H0000 'Binary 000000000000000 3500 CURSOR(11,1)=&amp;H0000 'Binary 000000000000000 3600 CURSOR(11,1)=&amp;H0000 'Binary 000000000000000 3600 CURSOR(14,1)=&amp;H0000 'Binary 000000000000000 3600 CURSOR(15,1)=&amp;H0000 'Binary 000000000000000 3600 CURSOR(15,1)=&amp;H0000 'Binary 0000000000000000 3700 CURSOR(15,1)=&amp;H0000 'Binary 000000000000000000000000000000000000</pre>	2000	•
2200 ' 2300 CURSOR( 0,1)=&HOOOO 'Binary 000000000000000 2400 CURSOR( 1,1)=&HO180 'Binary 000000110000000 2500 CURSOR( 2,1)=&HO180 'Binary 000000110000000 2600 CURSOR( 2,1)=&HO180 'Binary 000000110000000 2700 CURSOR( 4,1)=&H7FFE 'Binary 0111111111111 2800 CURSOR( 5,1)=&H0180 'Binary 000000110000000 2900 CURSOR( 6,1)=&H0180 'Binary 000000110000000 3000 CURSOR( 7,1)=&H0180 'Binary 0000000110000000 3100 CURSOR( 8,1)=&H0180 'Binary 00000000000000 3200 CURSOR( 9,1)=&H0000 'Binary 00000000000000 3300 CURSOR(10,1)=&H0000 'Binary 000000000000000 3400 CURSOR(11,1)=&H0000 'Binary 000000000000000 3500 CURSOR(12,1)=&H0000 'Binary 00000000000000 3600 CURSOR(12,1)=&H0000 'Binary 00000000000000 3600 CURSOR(14,1)=&H0000 'Binary 000000000000000 3600 CURSOR(15,1)=&H0000 'Binary 000000000000000 3600 CURSOR(15,1)=&H0000 'Binary 0000000000000000 3600 CURSOR(15,1)=&H0000 'Binary 000000000000000000 3600 CURSOR(15,1)=&H0000 'Binary 000000000000000000000000000000000000	2100	' Define the cursor mask
2300 CURSOR( 0,1)=&H0000 'Binary 000000000000000 2400 CURSOR( 1,1)=&H0180 'Binary 000000110000000 2600 CURSOR( 2,1)=&H0180 'Binary 000000110000000 2600 CURSOR( 3,1)=&H0180 'Binary 000000110000000 2700 CURSOR( 4,1)=&H7FFE 'Binary 01111111111110 2800 CURSOR( 4,1)=&H7FFE 'Binary 000000110000000 2900 CURSOR( 6,1)=&H0180 'Binary 0000000110000000 3000 CURSOR( 6,1)=&H0180 'Binary 0000000110000000 3100 CURSOR( 8,1)=&H0180 'Binary 00000000000000 3200 CURSOR( 9,1)=&H0000 'Binary 000000000000000 3400 CURSOR(10,1)=&H0000 'Binary 000000000000000 3400 CURSOR(11,1)=&H0000 'Binary 000000000000000 3500 CURSOR(12,1)=&H0000 'Binary 000000000000000 3600 CURSOR(13,1)=&H0000 'Binary 000000000000000 3600 CURSOR(15,1)=&H0000 'Binary 000000000000000 3600 CURSOR(15,1)=&H0000 'Binary 0000000000000000 3600 CURSOR(15,1)=&H0000 'Binary 000000000000000 3600 CURSOR(15,1)=&H0000 'Binary 000000000000000 3600 CURSOR(15,1)=&H0000 'Binary 0000000000000000 3600 CURSOR(15,1)=&H0000 'Binary 0000000000000000 3600 CURSOR(15,1)=&H0000 'Binary 000000000000000000000000000000000000	2200	<b>'</b>
2400 CURSOR (1,1)=&HO180 'Binary 00000011000000 2500 CURSOR (2,1)=&HO180 'Binary 000000110000000 2600 CURSOR (3,1)=&HO180 'Binary 000000110000000 2700 CURSOR (4,1)=&H7FFE 'Binary 0111111111110 2800 CURSOR (5,1)=&H0180 'Binary 000000110000000 2900 CURSOR (6,1)=&H0180 'Binary 000000110000000 3000 CURSOR (6,1)=&H0180 'Binary 00000000000000 3000 CURSOR (7,1)=&H0180 'Binary 00000000000000 3200 CURSOR (8,1)=&H0000 'Binary 00000000000000 3300 CURSOR (10,1)=&H0000 'Binary 00000000000000 3400 CURSOR (11,1)=&H0000 'Binary 00000000000000 3500 CURSOR (12,1)=&H0000 'Binary 00000000000000 3600 CURSOR (13,1)=&H0000 'Binary 00000000000000 3600 CURSOR (15,1)=&H0000 'Binary 00000000000000 3600 CURSOR (15,1)=&H0000 'Binary 000000000000000 3600 CURSOR (15,1)=&H0000 'Binary 00000000000000000 3600 CURSOR (15,1)=&H0000 'Binary 000000000000000000000000000000000000	2300	CURSOR( 0,1)=&H0000 'Binary 000000000000000
2500 CURSOR( 2,1)=&H0180 'Binary 00000011000000 2600 CURSOR( 3,1)=&H0180 'Binary 000000110000000 2700 CURSOR( 4,1)=&H7FFE 'Binary 0111111111111 2800 CURSOR( 5,1)=&H0180 'Binary 000000110000000 2900 CURSOR( 6,1)=&H0180 'Binary 0000000110000000 3000 CURSOR( 7,1)=&H0180 'Binary 00000000000000 3000 CURSOR( 8,1)=&H0000 'Binary 00000000000000 3200 CURSOR( 10,1)=&H0000 'Binary 00000000000000 3300 CURSOR(10,1)=&H0000 'Binary 00000000000000 3400 CURSOR(11,1)=&H0000 'Binary 00000000000000 3500 CURSOR(12,1)=&H0000 'Binary 00000000000000 3600 CURSOR(13,1)=&H0000 'Binary 00000000000000 3700 CURSOR(14,1)=&H0000 'Binary 00000000000000 3800 CURSOR(15,1)=&H0000 'Binary 00000000000000 3900 ' 4000 ' Set the mouse cursor shape, color, and hot 4000 ' Set the mouse cursor shape, color, and hot 4000 M1 = 9 4300 M2 = 7 ' Horizontal hot spot 4400 M3 = 4 ' Vertical hot spot 4500 CALL MOUSE(M1,M2,M3,CURSOR(0,0))	2400	CURSOR(1,1)=&H0180 'Binary 0000000110000000
2600 CURSOR( 3,1)=&H0180 'Binary 000000011000000 2700 CURSOR( 4,1)=&H7FFE 'Binary 0111111111110 2800 CURSOR( 5,1)=&H0180 'Binary 000000110000000 2900 CURSOR( 6,1)=&H0180 'Binary 000000110000000 3000 CURSOR( 7,1)=&H0180 'Binary 00000000000000 3100 CURSOR( 7,1)=&H0180 'Binary 00000000000000 3200 CURSOR( 8,1)=&H0000 'Binary 000000000000000 3300 CURSOR(10,1)=&H0000 'Binary 000000000000000 3400 CURSOR(11,1)=&H0000 'Binary 000000000000000 3500 CURSOR(11,1)=&H0000 'Binary 000000000000000 3600 CURSOR(13,1)=&H0000 'Binary 000000000000000 3700 CURSOR(14,1)=&H0000 'Binary 000000000000000 3800 CURSOR(15,1)=&H0000 'Binary 0000000000000000 3900 ' 4000 ' Set the mouse cursor shape, color, and hot 4000 ' Set the mouse cursor shape, color, and hot 4000 M1 = 9 4300 M2 = 7 ' Horizontal hot spot 4400 M3 = 4 ' Vertical hot spot 4500 CALL MOUSE(M1,M2,M3,CURSOR(0,0))	2500	CURSOR(2,1)=&H0180 'Binary 0000000110000000
2700 CURSOR( 4,1)=&H7FFE 'Binary Oll11111111110 2800 CURSOR( 5,1)=&H0180 'Binary O000000110000000 2900 CURSOR( 6,1)=&H0180 'Binary O00000110000000 3000 CURSOR( 7,1)=&H0180 'Binary O0000000000000 3100 CURSOR( 8,1)=&H0000 'Binary O0000000000000 3200 CURSOR( 9,1)=&H0000 'Binary O0000000000000 3400 CURSOR(10,1)=&H0000 'Binary O0000000000000 3500 CURSOR(11,1)=&H0000 'Binary O0000000000000 3600 CURSOR(12,1)=&H0000 'Binary O0000000000000 3600 CURSOR(12,1)=&H0000 'Binary O0000000000000 3600 CURSOR(14,1)=&H0000 'Binary O0000000000000 3600 CURSOR(15,1)=&H0000 'Binary O00000000000000 3600 CURSOR(15,1)=&H0000 'Binary O00000000000000 3600 CURSOR(15,1)=&H0000 'Binary O000000000000000000 3600 CURSOR(15,1)=&H0000 'Binary O000000000000000000000000000000000000	2600	CURSOR ( 3, 1) = & H0180 'Binary 0000000110000000
2800 CURSOR( 5,1)=&H0180 'Binary 00000011000000 2900 CURSOR( 6,1)=&H0180 'Binary 00000011000000 3000 CURSOR( 7,1)=&H0180 'Binary 000000011000000 3100 CURSOR( 8,1)=&H0180 'Binary 0000000000000 3200 CURSOR( 9,1)=&H0000 'Binary 00000000000000 3300 CURSOR(10,1)=&H0000 'Binary 00000000000000 3400 CURSOR(11,1)=&H0000 'Binary 00000000000000 3500 CURSOR(12,1)=&H0000 'Binary 000000000000000 3600 CURSOR(13,1)=&H0000 'Binary 000000000000000 3600 CURSOR(14,1)=&H0000 'Binary 000000000000000 3600 CURSOR(15,1)=&H0000 'Binary 000000000000000 3600 CURSOR(15,1)=&H0000 'Binary 0000000000000000 3900 ' 4000 ' Set the mouse cursor shape, color, and hot 4000 ' Set the mouse cursor shape, color, and hot 4000 M1 = 9 4300 M2 = 7 ' Horizontal hot spot 4500 CALL MOUSE(M1,M2,M3,CURSOR(0,0))	2700	CURSOR ( 4,1) = & H7FFE 'Binary 011111111111110
2900 CURSOR( 6,1)=&HO18O 'Binary OOOOOOO11000000 3000 CURSOR( 7,1)=&HO18O 'Binary OOOOOOO110000000 3100 CURSOR( 8,1)=&HO18O 'Binary OOOOOOOOOOOOOOO 3200 CURSOR( 9,1)=&HO00O 'Binary OOOOOOOOOOOOOOOO 3400 CURSOR(10,1)=&HO00O 'Binary OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO	2800	CURSOR( 5,1)=&H0180 'Binary 0000000110000000
3000 CURSOR( 7,1) =&HO18O 'Binary 0000000110000000 3100 CURSOR( 8,1) =&HO18O 'Binary 00000000000000 3200 CURSOR( 9,1) =&HO00O 'Binary 00000000000000 3400 CURSOR(10,1) =&HO00O 'Binary 00000000000000 3500 CURSOR(11,1) =&HO00O 'Binary 00000000000000 3600 CURSOR(12,1) =&HO00O 'Binary 00000000000000 3600 CURSOR(14,1) =&HO00O 'Binary 00000000000000 3600 CURSOR(15,1) =&HO00O 'Binary 00000000000000 3600 CURSOR(15,1) =&HO00O 'Binary 000000000000000 3600 CURSOR(15,1) =&HO00O 'Binary 000000000000000000000000000000000000	2900	CURSOR(6,1) = &HO180 'Binary 000000110000000
3100 CURSOR( 8,1) =&HOOOO 'Binary OOOOOOOOOOOOOO 3200 CURSOR( 9,1) =&HOOOO 'Binary OOOOOOOOOOOOOOO 3300 CURSOR(10,1) =&HOOOO 'Binary OOOOOOOOOOOOOOO 3400 CURSOR(11,1) =&HOOOO 'Binary OOOOOOOOOOOOOOOO 3500 CURSOR(12,1) =&HOOOO 'Binary OOOOOOOOOOOOOOO 3600 CURSOR(13,1) =&HOOOO 'Binary OOOOOOOOOOOOOOO 3700 CURSOR(14,1) =&HOOOO 'Binary OOOOOOOOOOOOOOO 3800 CURSOR(15,1) =&HOOOO 'Binary OOOOOOOOOOOOOOO 3800 CURSOR(15,1) =&HOOOO 'Binary OOOOOOOOOOOOOOO 3900 ' 4000 ' Set the mouse cursor shape, color, and hot 4000 ' Set the mouse cursor shape, color, and hot 4100 ' 4200 M1 = 9 4300 M2 = 7 ' Horizontal hot spot 4400 M3 = 4 ' Vertical hot spot 4500 CALL MOUSE(M1,M2,M3,CURSOR(0,0))	3000	CURSOR(7,1) = &HO180 'Binary 000000110000000
3200 CURSOR( 9,1)=&H0000 'Binary 00000000000000 3300 CURSOR(10,1)=&H0000 'Binary 00000000000000 3400 CURSOR(11,1)=&H0000 'Binary 00000000000000 3500 CURSOR(12,1)=&H0000 'Binary 00000000000000 3600 CURSOR(13,1)=&H0000 'Binary 00000000000000 3700 CURSOR(14,1)=&H0000 'Binary 00000000000000 3800 CURSOR(15,1)=&H0000 'Binary 00000000000000 3900 ' 4000 ' Set the mouse cursor shape, color, and hot 4000 ' Set the mouse cursor shape, color, and hot 4000 ' Set the mouse cursor shape, color, and hot 4000 ' Set the mouse cursor shape, color, and hot 4000 M1 = 9 4300 M2 = 7 ' Horizontal hot spot 4400 M3 = 4 ' Vertical hot spot 4500 CALL MOUSE(M1,M2,M3,CURSOR(0,0))	3100	CURSOR(81) = &HOOOO Binary OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
<pre>3300 CURSOR(10,1)=&amp;H0000 'Binary 00000000000000 3400 CURSOR(11,1)=&amp;H0000 'Binary 00000000000000 3500 CURSOR(12,1)=&amp;H0000 'Binary 00000000000000 3600 CURSOR(13,1)=&amp;H0000 'Binary 00000000000000 3700 CURSOR(14,1)=&amp;H0000 'Binary 00000000000000 3800 CURSOR(15,1)=&amp;H0000 'Binary 00000000000000 3900 ' 4000 ' Set the mouse cursor shape, color, and hot 4000 ' Set the mouse cursor shape, color, and hot 4000 ' Set the mouse cursor shape, color, and hot 4000 ' Set the mouse cursor shape, color, and hot 4000 M1 = 9 4300 M2 = 7 ' Horizontal hot spot 4400 M3 = 4 ' Vertical hot spot 4500 CALL MOUSE(M1,M2,M3,CURSOR(0,0))</pre>	3200	CURSOR(9,1) = &HOOOO Binary OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
3400 CURSOR(11,1)=&H0000 'Binary 00000000000000 3500 CURSOR(12,1)=&H0000 'Binary 00000000000000 3600 CURSOR(13,1)=&H0000 'Binary 00000000000000 3700 CURSOR(14,1)=&H0000 'Binary 00000000000000 3800 CURSOR(15,1)=&H0000 'Binary 00000000000000 3900 ' 4000 ' Set the mouse cursor shape, color, and hot 4000 ' Set the mouse cursor shape, color, and hot 4000 ' Set the mouse cursor shape, color, and hot 4000 ' Set the mouse cursor shape, color, and hot 4000 ' Set the mouse cursor shape, color, and hot 4000 ' Set the mouse cursor shape, color, and hot 4000 M1 = 9 4300 M2 = 7 ' Horizontal hot spot 4400 M3 = 4 ' Vertical hot spot 4500 CALL MOUSE(M1,M2,M3,CURSOR(0,0))	3300	CURSOR(10, 1) = &HOOOO  Binary 000000000000000000000000000000000000
3500 CURSOR (12,1) =&H0000 'Binary 000000000000000 3600 CURSOR (12,1) =&H0000 'Binary 000000000000000 3700 CURSOR (14,1) =&H0000 'Binary 00000000000000 3800 CURSOR (15,1) =&H0000 'Binary 00000000000000 3900 ' 40000 ' Set the mouse cursor shape, color, and hot 4050 ' spot 4100 ' 4200 M1 = 9 4300 M2 = 7 ' Horizontal hot spot 4400 M3 = 4 ' Vertical hot spot 4500 CALL MOUSE (M1,M2,M3,CURSOR (0,0))	3400	CURSOR(11, 1) = HOOOO Binary 000000000000000000000000000000000000
3600 CURSOR(13,1)=&H0000 'Binary 000000000000000 3700 CURSOR(14,1)=&H0000 'Binary 000000000000000 3800 CURSOR(15,1)=&H0000 'Binary 00000000000000 3900 ' 4000 ' Set the mouse cursor shape, color, and hot 4050 ' spot 4100 ' 4200 M1 = 9 4300 M2 = 7 ' Horizontal hot spot 4400 M3 = 4 ' Vertical hot spot 4500 CALL MOUSE(M1,M2,M3,CURSOR(0,0))	3500	CURSOR(12, 1) = anocoo Binary COOCOOCOOCOOCOO
3700 CURSOR(14,1) =&H0000 'Binary 00000000000000 3700 CURSOR(14,1) =&H0000 'Binary 00000000000000 3800 CURSOR(15,1) =&H0000 'Binary 00000000000000 3900 ' 4000 ' Set the mouse cursor shape, color, and hot 4000 ' Set the mouse cursor shape, color, and hot 4000 ' Set the mouse cursor shape, color, and hot 4000 M1 = 9 4300 M2 = 7 ' Horizontal hot spot 4400 M3 = 4 ' Vertical hot spot 4500 CALL MOUSE(M1,M2,M3,CURSOR(0,0))	3600	CURSOR(12,1) = anocoo Binary COOOCOOCOOCOOCOO
3800 CURSOR (14,1) - AROOOD 'Binary 00000000000000 3800 CURSOR (15,1) = &H0000 'Binary 00000000000000 4000 ' Set the mouse cursor shape, color, and hot 4000 ' spot 4100 ' 4200 M1 = 9 4300 M2 = 7 ' Horizontal hot spot 4400 M3 = 4 ' Vertical hot spot 4500 CALL MOUSE (M1,M2,M3,CURSOR (0,0))	3000	CURSOR(13,1) = aHOOOO BIHATY OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
<pre>3900 'Set the mouse cursor shape, color, and hot 4000 ' Set the mouse cursor shape, color, and hot 4000 ' spot 4100 ' 4200 M1 = 9 4300 M2 = 7 ' Horizontal hot spot 4400 M3 = 4 ' Vertical hot spot 4500 CALL MOUSE(M1,M2,M3,CURSOR(0,0))</pre>	3700	CURSOR(14,1) = CHOOOO Binary 000000000000000000000000000000000000
4000 ' Set the mouse cursor shape, color, and hot 4050 ' spot 4100 ' 4200 M1 = 9 4300 M2 = 7 ' Horizontal hot spot 4400 M3 = 4 ' Vertical hot spot 4500 CALL MOUSE (M1,M2,M3,CURSOR(0,0))	3800	CURSOR (15,1) = & HOUDO BINARY 000000000000000000000000000000000000
4000 ' Set the mouse cursor shape, color, and not 4050 ' spot 4100 ' 4200 M1 = 9 4300 M2 = 7 ' Horizontal hot spot 4400 M3 = 4 ' Vertical hot spot 4500 CALL MOUSE (M1,M2,M3,CURSOR(0,0))	3900	
4050 ' spot 4100 ' 4200 M1 = 9 4300 M2 = 7 ' Horizontal hot spot 4400 M3 = 4 ' Vertical hot spot 4500 CALL MOUSE (M1,M2,M3,CURSOR(0,0))	4000	Set the mouse cursor shape, color, and not
4100 ' 4200 M1 = 9 4300 M2 = 7 ' Horizontal hot spot 4400 M3 = 4 ' Vertical hot spot 4500 CALL MOUSE (M1,M2,M3,CURSOR(0,0))	4050	spot
4200 M1 = 9 4300 M2 = 7 ' Horizontal hot spot 4400 M3 = 4 ' Vertical hot spot 4500 CALL MOUSE (M1,M2,M3,CURSOR(0,0))	4100	
4300 M2 = 7 ' Horizontal hot spot 4400 M3 = 4 ' Vertical hot spot 4500 CALL MOUSE (M1, M2, M3, CURSOR $(0, 0)$ )	4200	M1 = 9
4400 M3 = 4 ' Vertical hot spot 4500 CALL MOUSE (M1, M2, M3, CURSOR (0, 0))	4300	M2 = 7 ' Horizontal hot spot
4500 CALL MOUSE (M1, M2, M3, CURSOR (0, 0))	4400	M3 = 4 ' Vertical hot spot
	4500	CALL MOUSE (M1, M2, M3, CURSOR (0, 0))

# Hourglass

The hourglass is a solid figure with the hot spot at the center of the glass. This shape can be used to show that the operation in progress will take some time to complete.

```
100
 200 '
         Define the screen mask
 300
 700 CURSOR ( 3,0) = & H0000 'Binary 0000000000000000
 800 \text{ CURSOR}(4,0) = \& H8001
                            'Binary 1000000000000000
 900 CURSOR (5,0) = \& HCOO3
                            'Binary 1100000000000011
1000 \text{ CURSOR}(6,0) = \& \text{HE}007
                             'Binary 1110000000000111
1100 CURSOR (7,0) = \& HFOOF
                             'Binary 1111000000001111
                             'Binary 1110000000000111
1200 \text{ CURSOR}(8,0) = & \text{HEOO7}
1300 \text{ CURSOR}(9,0) = & \text{HCOO3}
                             'Binary 110000000000011
1400 CURSOR (10, 0) = \& H8001
                             'Binary 10000000000000000
                             'Binary 0000000000000000
1500 \text{ CURSOR}(11,0) = \& H0000
1600 \text{ CURSOR}(12,0) = \& H0000
                             'Binary 0000000000000000
1700 CURSOR (13,0) = & H0000 'Binary 0000000000000000
1800 CURSOR (14,0) = & H0000 'Binary 0000000000000000
1900 CURSOR (15,0) = & HFFFF 'Binary 111111111111111
2000
2100 '
         Define the cursor mask
2200
2300 CURSOR( 0,1)=&H0000 'Binary 00000000000000
2400 CURSOR( 1,1)=&H7FFE 'Binary 0111111111110
2500 CURSOR( 2,1)=&H6006 'Binary 011000000000110
2600 CURSOR ( 3,1) = & H300C
                             'Binary 0011000000001100
2700 \text{ CURSOR}(4,1) = & \text{H1818}
                            'Binary 0001100000011000
2800 CURSOR ( 5,1) = & HOC30 'Binary 0000110000110000
2900 CURSOR (
               6, 1) = & HO660
                            'Binary 0000011001100000
                             'Binary 0000001111000000
3000 \text{ CURSOR}(7,1) = \& HO3CO
3100 CURSOR ( 8,1) = & HO660
                             'Binary 0000011001100000
                            'Binary 0000110000110000
3200 CURSOR ( 9,1) = & HOC30
                             'Binary 0001100110011000
3300 CURSOR (10, 1) = & H1998
                             'Binary 0011001111001100
3400 \text{ CURSOR}(11, 1) = \& H33CC
3500 \text{ CURSOR}(12,1) = & H67E6
                             'Binary 0110011111100110
                            'Binary 0111111111111110
3600 CURSOR (13, 1) = & H7FFE
3800 CURSOR (15,1) = & HOOOO 'Binary 0000000000000000
3900
4000
         Set the mouse cursor shape, color, and hot
4050
         spot
4100
4200 M1 = 9
4300 M2 = 7
                ' Horizontal hot spot
                ' Vertical hot spot
4400 M3 = 7
4500 CALL MOUSE (M1, M2, M3, CURSOR (0, 0))
```

C

# 8 Writing Mouse Programs for IBM EGA Modes

If your application program includes mouse support for IBM enhanced graphics modes D, E, F, and 10, your program must interact with the IBM Enhanced Graphics Adapter (EGA) through the Microsoft EGA Register Interface Library (EGA.LIB). EGA.LIB is included on the Microsoft Mouse Tools disk. If your program tries to set the EGA registers directly, rather than through this interface, the mouse cursor will not be drawn correctly.

The EGA Register Interface allows your program to write to and read from write-only registers on the EGA. You need this capability to use interrupt-driven graphics, such as the cursor update code.

# The EGA Register Interface Library

The Microsoft EGA Register Interface Library consists of nine functions that can be called from assembly-language programs or from programs written in high-level languages such as Microsoft QuickBASIC, Pascal, FOR-TRAN, and C. These functions:

- Read from or write to one or more of the EGA writeonly registers
- Define default values for EGA write-only registers or reset the registers to these default values
- Check whether the EGA Register Interface is present and, if so, return its version number

# How the Interface Library Works

The mouse driver loads the EGA Register Interface Library if it detects an EGA installed in the system. The interface maintains *shadow maps* (memory images) of the EGA write-only registers, which allow application programs to read these registers. The shadow maps are updated whenever your program calls one of the interface functions to set a register; therefore, the shadow maps always contain the last values written to the registers. When your program calls one of the interface functions to read a register, the function call returns the value stored in the shadow map.

The code in the interface intercepts mode-change calls to the BIOS ROM (INT 10h with AH = 0) and updates the shadow maps and default register tables accordingly.

# How to Call the EGA Register Interface Library

This section shows how to call functions in the EGA Register Interface Library from programs written in assembly language and high-level languages.

#### Making Calls from Assembly-Language Programs

To call EGA Register Interface functions from an assembly-language program:

Load the AX, BX, CX, DX, and ES registers (as required) with the parameter values.

Execute software interrupt 16 (10h).

Values returned by the EGA Register Interface functions are placed in the registers.

When called from assembly-language programs, functions F2, F3, F4, F5, and F7 expect ES:BX to be a table pointer.

## Example

Use the following instructions to set the palette registers to the values in the array "mytable":

## Making Calls from High-Level-Language Programs

You can include EGA Register Interface function calls in QuickBASIC, Pascal, FORTRAN, and C programs as ordinary procedure calls.

To make an EGA function call from a high-levellanguage program:

Declare the appropriate procedure as an external procedure:

For compiled BASIC and Pascal programs, use the procedure "EGA" if the argument addresses are in the program's data segment (short addresses), or the procedure "EGAS" if the arguments are in another segment (long addresses).

For FORTRAN programs, use the procedure "EGAS".

For C programs, use the procedure "cegas" for smallmodel programs, the procedure "cegam" for mediummodel programs, or the procedure "cegal" for largemodel programs.

Your program must pass the addresses (not the values) of five integer arguments to these procedures, so be sure to include an appropriate parameter list in the declarations.

Use the normal calling conventions to make the calls.Ink the compiled program with EGA.LIB.

All functions require five parameters: E1, E2, E3, E4, and E5. The following table shows how these parameters correspond to the registers listed in the function descriptions:

Parameter	Register	0
E1	AH	
E2	BX	
E3	$\mathbf{C}\mathbf{X}$	
E4	DX	
E5	ES	

E5 is a dummy parameter for all functions except function FA (Interrogate Driver). For function FA, the value returned for ES is placed in E5.

Use the following conventions when calling functions F2, F3, F4, F5, and F7 from a high-level-language program:

Procedures that use short argument addresses ("EGA", "cegas", and "cegam") set register ES to the value in register DS when they are called.

Procedures that use long argument addresses ("EGAS" and "cegal") set register ES to the value of the segment passed as part of parameter E2.

■ In BASIC, FORTRAN, and Pascal programs, fill an integer array (packed 2 bytes per integer) with the table values required by the function. Pass the first element of the array as parameter E2.

In C programs, fill a character array with the table values required by the function. Pass either the name of the array or a pointer to the array as parameter E2.

#### Examples

In a Pascal program with long argument addresses, use the following statement to declare "EGAS" as an external procedure:

```
0
```

PROCEDURE EGAS (VARS E1, E2, E3, E4, E5:INTEGER); EXTRN;

Once the procedure has been declared, use the following statements to restore the default settings for the EGA registers:

E1 := 246 (\*Function number is 246 = F6 (hexadecimal)\*) EGAS(E1, E2, E3, E4, E5)

In a small-model C program (version 3.0 or later), the following example restores the default settings for the EGA registers:

In a QuickBASIC program, the following example prints the version number of the EGA Register Interface:

Interrogate driver, get version number. e1% = &hOOFA e2% = 0 call ega(e1%, e2%, e3%, e4%, e5%) if (e2% <> 0) then 100 print "EGA Register Interface not found" end 100 print "EGA Register Interface found, version " def seg = e5% majver = peek(e2%) minver = peek(e2% + 1) def seg print " = "; print majver; print "."; print minver def seg

# Restrictions on Use of the EGA Register Interface Library

This section describes restrictions on the ways that application programs can use the EGA Register Interface Library.

# **Calls to BIOS ROM Video Routines**

The EGA Register Interface Library only intercepts calls to the BIOS ROM video routines (INT 10h, AH = 13h or less) that change the screen mode (AH = 0). It does not intercept any other BIOS ROM video routine calls. However, any other BIOS ROM video routine calls should restore all registers, so there is no problem in using them.

A call to interrupt 10h to set the color palette (AH = Bh) is an exception to this rule. Use EGA Register Interface function F5 to set the color palette. (For more information about function F5, see "EGA Register Interface Functions" later in this chapter.)

#### Attribute Controller Registers

Before your application program uses the Attribute Controller registers (I/O address 3C0h) in an extended interrupt 10h call, the program must set the Address or Data register flip-flop to the Address register (by doing an input from I/O port 3BAh or 3DAh). The flip-flop is always reset to this state when the program returns from the interrupt 10h call.

An interrupt routine that accesses the attribute chip always leaves the flip-flop set to the Address register when the program returns from the interrupt call. Therefore, if your application program sets the flip-flop to the Data register and expects the flip-flop to remain in this state, the program must disable interrupts between the time it sets the flip-flop to the Data register state and the last time the flip-flop is assumed to be in this state.

#### Sequencer Memory Mode Register

When the Sequencer Memory Mode register (I/O address 3C5h, data register 4) is accessed, the sequencer produces a glitch on the CAS lines that may cause problems with video random-access memory (VRAM). As a result, your application program cannot use the EGA Register Interface to read from or write to this register. Instead, use

the following procedure to safely alter this register:

- 1 Disable interrupts.
- Set Synchronous Reset (bit 1) in the Sequencer Reset register to 0.
- Read/modify/write the Sequencer Memory Mode register.
- Set Synchronous Reset (bit 1) in the Sequencer Reset register to 1.
- **5** Enable interrupts.

# Input Status Registers

Your application program cannot use the EGA Register Interface to read Input Status registers 0 (I/O address 3C2h) and 1 (I/O address 3BAh or 3DAh). If the program must read these registers, it should do so directly.

## Graphics Controller Miscellaneous Register

When the Graphics Controller Miscellaneous register (I/O address 3CFh, data register 6) is accessed, a glitch on the CAS lines occurs that may cause problems with video random-access memory (VRAM). As a result, your application program should not use the EGA Register Interface to read from or write to this register.

EGA Register Interface function F6 does not alter the state of the Graphics Controller Miscellaneous register. Use the following procedure to safely alter this register:

1 Disable interrupts.

- Set Synchronous Reset (bit 1) in the Sequencer Reset register to 0.
- Read/modify/write the Graphics Controller Miscellaneous register.
- Set Synchronous Reset (bit 1) in the Sequencer Reset register to 1.
- **5** Enable interrupts.

# EGA Register Interface Functions

This section describes each EGA Register Interface function in detail. The following list shows these functions by function number:

Number (Hex)	Function
FO	Read one register
F1	Write one register
F2	Read register range
F3	Write register range
F4	Read register set
F5	Write register set
F6	Revert to default registers
F7	Define default register table
FA	Interrogate driver

Note Calls F8h, F9h, and FBh through FFh are reserved.

Each function description includes:

- The parameters required to make the call (input) and the expected return values (output)
- Any special considerations regarding the function

If the function description does not specify an input for a parameter, you don't need to supply a value for that parameter before making the call. If the function description does not specify an output value for a parameter, the parameter's value is the same before and after the call.

**Caution** The EGA Register Interface does not check input values, so be sure that the values you load into the registers are correct before making a call.

# Function F0: Read One Register

Function F0 reads data from a specified register on the EGA.

## Input:

AH = FOh

BX = Pointer for pointer/data chips:

BH = 0BL = pointer

Ignored for single registers

DX = Port number:

Pointer/data chips

```
0h: CRT Controller (3?4h)
8h: Sequencer (3C4h)
10h: Graphics Controller (3CEh)
18h: Attribute Controller (3C0h)
```

Single registers

20h: Miscellaneous Output register (3C2h) 28h: Feature Control register (3?Ah) 30h: Graphics 1 Position register (3CCh) 38h: Graphics 2 Position register (3CAh)

? = B for monochrome modes or D for color modes

# Output:

AX: Restored

BH: Restored BL: Data

O

DX: Restored

All other registers restored

#### Examples

The following example saves the contents of the Sequencer Map Mask register in "myvalue":

myvalue db ?

mov mov	ah, bx,	0f0h 0002h	<pre>;f0 = read one register ;bh = 0 / bl = map mask ;index</pre>
mov int mov	dx, 10h myva	0008h alue, bl	;dx = sequencer ;get it! ;save it!

The following example saves the contents of the Miscellaneous Output register in "myvalue":

myvalue db ?

mov	ah, OfOh	;f0 = read one register
mov	dx, 0020h	;dx = miscellaneous output
		; register
int	10h	;get it!
mov	myvalue, bl	;save it!

# Function F1: Write One Register

Function F1 writes data to a specified register on the EGA.

When your application program returns from a call to function F1, the contents of registers BH and DX are not restored. Your program must save and restore these registers itself if this is desired.

## Input:

AH = F1h

BL = Pointer for pointer/data chips

or

Data for single registers

BH = Data for pointer/data chips (ignored for single registers)

DX = Port number:

Pointer/data chips

0h: CRT Controller (3?4h) 8h: Sequencer (3C4h) 10h: Graphics Controller (3CEh) 18h: Attribute Controller (3C0h)

Single registers

20h: Miscellaneous Output register (3C2h) 28h: Feature Control register (3?Ah) 30h: Graphics 1 Position register (3CCh) 38h: Graphics 2 Position register (3CAh)

? = B for monochrome modes or D for color modes

#### **Output:**

AX: Restored

BL: Restored BH: Not restored

DX: Not restored

All other registers restored

#### Examples

The following example writes the contents of "myvalue" into the CRT Controller Cursor Start register:

myvalue db 3h

mov	ah,	Oflh	;	fl = writ	e one	register
mov	bh,	myvalue	;	bh = data	a from	myvalue
mov	bl,	000ah	;	bl = curs	sor sta	art index
mov	dx,	0000h	÷	dx = crt	contr	oller
int	TOU		;	write it:		

The following example writes the contents of "myvalue" into the Feature Control register:

myvalue db 2h

mov	ah,	Oflh	;	f1	=	write	one	register
mov	bl,	myvalue	;	bl	=	data	from	myvalue
mov	dx,	0028h	;	dx	=	featu	re co	ontrol
			;	ree	gis	ster		
int	10h		;	wr	ίtε	a it!		

# Function F2: Read Register Range

Function F2 reads data from a specified range of registers on the EGA. A range of registers is defined to be several registers on a single chip that have consecutive indexes. This call makes sense only for the pointer/data chips.

## Input:

AH = F2h

CH = Starting pointer valueCL = Number of registers (must be > 1)

DX = Port number:

0h: CRT Controller (3?4h)

- 8h: Sequencer (3C4h)
- 10h: Graphics Controller (3CEh)
- 18h: Attribute Controller (3C0h)



ES:BX = Points to table of one-byte entries (length = value in CL). On return, each entry is set to the contents of the corresponding register.

## Output:

AX: Restored

BX: Restored

CX: Not restored

DX: Restored

ES: Restored

All other registers restored

8-14 Designing a Mouse Interface

#### Example

The following example saves the contents of the Attribute Controller Palette registers in "paltable":

paltable db 16 dup (?) mov ax, ds ; assume paltable in data segment ; ; es = data segment mov es, ax mov bx, offset paltable ; es:bx = paltable ; address mov ah, Of2h ; f2 = read register ; range mov cx, 0010h ; ch = start index of 0 ; cl = 16 registers ; to read ; dx = attribute mov dx, 0018h ; controller ; read them! int 10h

# Function F3: Write Register Range

Function F3 writes data to a specified range of registers on the EGA. A range of registers is defined to be several registers on a single chip that have consecutive indexes. This call only makes sense for the pointer/data chips.

## Input:

AH = F3h

CH = Starting pointer valueCL = Number of registers (must be > 1)

- DX = Port number
  - 0h: CRT Controller (3?4h) 8h: Sequencer (3C4h)
  - 10h: Graphics Controller (3CEh)
  - 18h: Attribute Controller (3C0h)
  - ? = B for monochrome modes or D for color modes
- ES:BX = Points to table of one-byte entries (length = value in CL). Each entry contains the value to be written to the corresponding register.

## Output:

AX: Restored

BX: Not restored

CX: Not restored

DX: Not restored

ES: Restored

All other registers restored

8-16 Designing a Mouse Interface

## Example

The following example writes the contents of "cursloc" into the CRT Controller Cursor Location High and Cursor Location Low registers.

cursloc	db	01h,	OOh	; ;	cursor at page offset O100h
	mov	ax,	ds	;;	assume cursloc in data segment
	mov	es,	ax	;	es=data segment
	mov	bx,	offset cursloc	;	es:bx=cursloc address
	mov	ah,	Of3h	;;	f3=write register range
	mov	cx,	0e02h	;;;;	ch=start index of 14 cl=2 registers to write
	mov int	dx, 10h	0000h	;	dx=crt controller write them!

# Function F4: Read Register Set

Function F4 reads data from a set of registers on the EGA. A set of registers is defined to be several registers that may or may not have consecutive indexes, and that may or may not be on the same chip.

## Input:

AH = F4h

CX = Number of registers (must be > 1)

ES:BX = Points to table of records with each entry in this format:

Bytes 1-2: Port number

Pointer/data chips

0h: CRT Controller (3?4h) 8h: Sequencer (3C4h) 10h: Graphics Controller (3CEh) 18h: Attribute Controller (3C0h)

Single registers

20h: Miscellaneous Output register (3C2h) 28h: Feature Control register (3?Ah) 30h: Graphics 1 Position register (3CCh) 38h: Graphics 2 Position register (3CAh)

P = B for monochrome modes or D for color modes

Byte 3: Pointer value (0 for single registers)

Byte 4: EGA Register Interface fills in data read from register specified in bytes 1-3.

#### **Output:**

AX: Restored

BX: Restored

CX: Not restored

ES: Restored

All other registers restored

#### Example

The following example saves the contents of the Miscellaneous Output register, Sequencer Memory Mode register, and CRT Controller Mode Control register in "results":

outvals dw 0020h ; miscellaneous output register ; O for single registers db 0 : returned value db ? dw 0008h ; sequencer db O4h ; memory mode register index db ? ; returned value 0000h ; crt controller dw 17h ; mode control register index db db ? ; returned value results db 3 dup (?) ; assume outvals in mov ax, ds ; data segment mov es, ax ; es=data segment mov bx, offset outvals ; es:bx=outvals address mov ah, Of4h ; f4=read register set mov cx, 3 ; number of entries in outvals int 10h ; get values into outvals! mov si, 3 ; move the returned values from add si, offset outvals ; outvals mov di, offset results ; to results mov cx, 3 3 values to move ; move one value from outvals movloop: mov ax, [si] mov [di], ax ; to results add si, 4 ; skip to next source byte ; point to next destination inc di loop movloop ; byte

# Function F5: Write Register Set

Function F5 writes data to a set of registers on the EGA. A set of registers is defined to be several registers that may or may not have consecutive indexes, and that may or may not be on the same chip.

## Input:

AH = F5h

CX = Number of registers (must be > 1)

ES:BX = Points to table of values with each entry in this format:

Bytes 1-2: Port number

Pointer/data chips

0h: CRT Controller (3?4h) 8h: Sequencer (3C4h) 10h: Graphics Controller (3CEh) 18h: Attribute Controller (3C0h)

Single registers

20h: Miscellaneous Output register (3C2h) 28h: Feature Control register (3?Ah) 30h: Graphics 1 Position register (3CCh) 38h: Graphics 2 Position register (3CAh)

P = B for monochrome modes or D for color modes

Byte 3: Pointer value (0 for single registers)

Byte 4: Data to be written to register specified in bytes 1-3

#### Output

AX: Restored

BX: Restored

CX: Not restored

ES: Restored

All other registers restored

#### Example

The following example writes the contents of "outvals" to the Miscellaneous Output register, Sequencer Memory Mode register, and CRT Controller Mode Control register:

dw db db	0020 0 0a7h	)h ; ; 1 ;	miscellaneous output register O for single registers output value
dw db db	0008 04h 03h	Sh ; ; ;	sequencer memory mode register index output value
dw db db	0000 17h 0a3h	)h ; ; n ;	crt controller mode control register index output value
mov	ax,	ds	; assume outvals in ; data segment
mov	es,	ax	; es=data segment
mov	bx,	off	set outvals ; es:bx=outvals address
mov	ah,	Of 51	n ; f5=write register set
mov	cx,	3	; number of entries in ; outvals
int	10h		; write the registers!
	dw db db db db db db db db mov mov mov mov int	dw 0020 db 0 db 0a7h dw 0008 db 04h db 03h dw 0000 db 17h db 0a3h mov ax, mov es, mov es, mov bx, mov cx, int 10h	<pre>dw 0020h ; db 0 ; db 0a7h ; dw 0008h ; db 04h ; db 03h ; dw 0000h ; db 17h ; db 0a3h ; mov ax, ds mov es, ax mov bx, offs mov ah, 0f5h mov cx, 3 int 10h</pre>

# Function F6: Revert to Default Registers

Function F6 restores the default settings of any registers that your application program has changed through the EGA Register Interface. The default settings are defined in a call to function F7.

## Input:

AH = F6h

## Output:

All registers restored

**Note** If your program makes an interrupt 10h (video display adapter) call to function 0 to set the display mode, the default register values will change to the BIOS values for the selected mode.

## Example

The following example restores the default settings of the EGA registers:

mov	ah,	Of6h	;	f6	= :	revert	to	default
			;	re	egi	sters		
int	10h		;	do	īt	now!		

# Function F7: Define Default Register Table

Function F7 defines a table containing default values for any pointer/data chip or single register. If you define default values for a pointer/data chip, you must define them for *all* registers within that chip.

#### Input:

AH = F7h

DX = Port number:

Pointer/data chips

0h: CRT Controller (3?4h)

8h: Sequencer (3C4h)

10h: Graphics Controller (3CEh)

18h: Attribute Controller (3C0h)

Single registers

20h: Miscellaneous Output register (3C2h)

28h: Feature Control register (3?Ah)

30h: Graphics 1 Position register (3CCh)

38h: Graphics 2 Position register (3CAh)

? = B for monochrome modes or D for color modes

ES:BX = Points to table of one-byte entries. Each entry contains the default value for the corresponding register. The table must contain entries for all registers.

Output:

AX: Restored

BX: Not restored

DX: Not restored

ES: Restored

All other registers restored

#### Examples

The following example defines default values for the Attribute Controller:

attrdflt db 00h, 01h, 02h, 03h, 04h, 05h, 06h, 07h db 10h, 11h, 12h, 13h, 14h, 15h, 16h, 17h db 08h, 00h, 0fh, 00h mov ax, ds ; assume attrdflt in data segment : mov es, ax ; es = data segment mov bx, offset attrdflt ; es:bx = attrdflt address mov ah, Of7h ; f7 = define default ; register table mov dx, 0018h ; dx = attributecontroller int 10h ; do it!

The following example defines a default value for the Feature Control register:

featdflt db 00h ; assume featdflt in mov ax, ds data segment mov es, ax ; es = data segment mov bx, offset featdflt ; es:bx = featdflt address mov ah, Of7h ; f7 = define defaultregister table mov dx, 0028h; dx = feature control register int 10h ; do it!

# **Function FA: Interrogate Driver**

Function FA interrogates the mouse driver and returns a value specifying whether or not the mouse driver is present.

#### Input:

AH = FAh

BX = 0

#### Output:

AX: Restored

BX = 0 if mouse driver is not present

ES:BX: Pointer to EGA Register Interface version number, if present:

Byte 1: Major release number Byte 2: Minor release number (in 1/100ths)

#### Example

The following example interrogates the mouse driver and displays the results:

aotmsg db "mouse driver found", Odh, Oah, 24h "mouse driver not found", Odh, Oah, 24h nopmsg db "revision \$" revmsg db crlf db Odh, Oah, 24h ten db 10 mov bx, 0 ; must be 0 for this call mov ah, 0fah ; fa = interrogate driver int 10h ; interrogate! or bx, bx ; bx = 0 ? jnz found ; branch if driver present mov dx, offset nopmsg ; assume nopmsg in data ; segment 9 = print string
; output not found message mov ah, O9h int 21h : that all for now jmp continue found: mov dx, offset gotmsg ; assume gotmsg in data ; segment ; 9 = print string mov ah. O9h ; output found message int 21h mov dx, offset revmsg ; assume revmsg in data ; segment ; 9 = print string ; output "revision " ; dl = major release number ; convert to ascii ; 2 = display character ; output major release ; number segment mov ah, O9h int 21h mov dl, es:[bx] add dl, "O" mov ah, 2 int 21h ; number ; dl = "." ; 2 = display character mov dl, "." mov ah, 2 mov ah, 2 ; 2 = display character int 21h ; output a period mov al, es:[bx+1] ; al = minor release number xor ah, ah ; ah = 0 idiv ten ; al = 10ths, ah = 100ths mov bx, ax ; save ax in bx mov dl, al ; dl = 10ths add dl, "0" ; convert to ascii mov ah, 2 ; 2 = display character int 21h ; dl = 100ths add dl, "0" ; convert to ascii mov ah, 2 ; 2 = display character int 21h ; dl = 100ths add dl, "0" ; convert to ascii mov ah, 2 ; 2 = display character int 21h ; dl = 100ths add dl, "0" ; convert to ascii mov ah, 2 ; 2 = display character int 21h ; 100ths ; 100ths mov dx, offset crlf ; assume crlf in data ; segment ; 9 = print string mov ah, O9h ; output end of line int 21h ; the end continue:

Appendices

# Appendix A Mouse Command Line Switches

This appendix describes the mouse command line switches you can use to customize the operation of the Control Panel and the mouse driver.

# **Control Panel Switches**

The Control Panel (CPANEL.EXE) is a memory-resident program that allows you to adjust the mouse sensitivity level—the ratio of cursor movement to actual mouse movement. (For information on using the Control Panel, see Chapter 4, "Moving the Mouse," in your *Microsoft Mouse User's Guide*.)

Whenever you invoke the Control Panel, the program reserves memory for the area of the screen the Control Panel overlays. The amount of memory needed depends on the type of display adapter used and the complexity of the image the Control Panel overlays. The Control Panel has a default size for the overlay buffer, but you can use a command line switch to change the amount of memory reserved by the Control Panel. If your system beeps when you activate the Control Panel, the screen buffer is not large enough. Use one of the following command line switches to change the size of the buffer, depending on the type of display adapter installed in your system:

Use this switch	For this display adapter
/C <n></n>	IBM Color/Graphics Adapter
/E <n></n>	IBM Enhanced Graphics Adapter
/H <n></n>	Hercules display adapter
$/M \le n >$	IBM Monochrome Adapter
/A <n></n>	AT&T 6300 display adapter

where  $\langle n \rangle$  is a number in the range 0 to 9. The larger the number, the larger the screen overlay buffer. If no switch is specified, the default value is /E7.

The size of the buffer required depends on the mode of the screen the Control Panel overlays. For example, screens displayed in the enhanced graphics modes require a larger Control Panel overlay buffer than screens displayed in text modes.

In general, use a value in the range 0 to 4 if the Control Panel will overlay only text screens; use a value in the range 5 to 9 if the Control Panel will overlay graphics screens.

The following table shows how many bytes of memory are occupied by the Control Panel and buffer for each possible switch setting:

Setting	/M	$/\mathrm{H}$	${f Switch}/{f A}$	/C	$/\mathrm{E}$
0	9712	14240	14992	9360	9360
1	9760	14288	15040	9456	9456
2	9808	14336	15088	9552	9552
3	9856	14384	15136	9744	9744
4	9904	14432	15184	10128	10128
5	9952	14480	15232	11872	19088
6	10000	14528	15280	12128	19344
7	10048	14576	15328	14768	29168
8	10096	14624	15376	15024	29424
9	10144	14672	15424	15280	29680

Use a Control Panel switch to specify the size of the overlay buffer when you load the Control Panel into memory. If the Control Panel is already in memory, you must first to remove the Control Panel from memory.

To remove the Control Panel from memory:

Type cpanel off

To specify a screen buffer size when you load the Control Panel:

Type *cpanel* followed by the appropriate switch.

For example, to specify the largest possible screen buffer for the area the Control Panel overlays on a CGA system, you would type cpanel/C9

# Mouse Driver Switches

Use mouse driver command line switches to:

- Specify the sensitivity of the mouse
- Set the interrupt rate (for the InPort<sup>®</sup> Mouse only)
- Tell the mouse driver the type and location of the Microsoft mouse installed in your system so the driver can bypass its usual procedure for determining mouse hardware configuration
- Disable the mouse driver or remove it from memory

You can add mouse driver command line switches to the mouse command lines in the AUTOEXEC.BAT or CONFIG.SYS file, or you can type *mouse* and the command line switches at the DOS prompt. If you type one or more switches at the DOS prompt, there must be a space between *mouse* and each switch.

The following sections describe how to use the mouse driver command line switches.

Using a mouse driver switch

Using a Control Panel switch

# **Specifying Mouse Sensitivity**

Use the following command line switches to set mouse sensitivity levels:

Use this switch	$\mathbf{To} \ \mathbf{set}$		
/S <nnn></nnn>	Horizontal and vertical sensitivity		
/H <nnn></nnn>	Horizontal sensitivity only		
/V <nnn></nnn>	Vertical sensitivity only		
/D <nnn></nnn>	Double-speed threshold		

where  $\langle nnn \rangle$  is a number in the range 0 to 100.

The switches for the horizontal and vertical sensitivity are interpreted in the same manner as a Control Panel setting. The double-speed-threshold switch determines the threshold speed for doubling the cursor's motion on the screen. Setting a double-speed threshold makes it easier to move the cursor to widely-separated images on the screen. (You can also use mouse function 19 to build this feature into an application program. For more information, see the description of function 19 in Chapter 6, "Mouse Function Descriptions.")

## Setting the Interrupt Rate for the InPort Mouse

If you are using an InPort Mouse, you can use one of the following command line switch settings to specify the interrupt rate for the mouse:

Switch Setting	Interrupt Rate		
/R0	disabled		
/R1	30Hz (default)		
/R2	50Hz		
/R3	100Hz		
/R4	200Hz		

# Specifying the Type and Location of the Mouse

The command line switches described in this section direct the mouse driver to bypass its usual search to determine the mouse hardware configuration and to look for a particular type of Microsoft Mouse at a particular I/O port.

A-4
This feature is useful if:

- The mouse driver has trouble determining which port the mouse is connected to, given your system's configuration
- More than one InPort device is connected to your computer
- You want to decrease the time required to load the mouse driver

The following table lists each switch you can use to tell the mouse driver to look for a particular mouse hardware configuration:

Use this switch	To look for
/В	Bus or InPort Mouse at primary InPort address
/I1	InPort Mouse at primary InPort address
/I2	InPort Mouse at secondary InPort address
/C1	Serial mouse on COM1
/C2	Serial Mouse on COM2

#### Disabling or Removing the Mouse Driver

If necessary, you can disable the mouse driver or remove it from memory. Before you disable or remove the mouse driver, you need to remove the Control Panel from memory and end any Mouse Menu program you are using.

To remove the Control Panel from memory:

Type cpanel off

To end a Microsoft Expert Mouse Menu program:

■ Type <filename> off

where <filename> is the name of the Expert Mouse Menu program.

To end a Mouse Menu program that you wrote yourself:

Type menu off

To disable or remove the mouse driver from memory:

Type mouse off

If the mouse driver is MOUSE.SYS, it is disabled; if the mouse driver is MOUSE.COM, it is removed from memory.

### Appendix B Linking Existing Mouse Programs with MOUSE.LIB (Version 6.0)

If you have a high-level language program that links with an earlier version of the Microsoft Mouse Library, you may have to modify the program to link it with the new MOUSE.LIB (version 6.0) on the Microsoft Mouse Tools disk.

Version 6.0 of MOUSE.LIB functions the same as the previous mouse library (version 5.03), except that version 6.0 has the following new features:

- New mouse functions 20, 21, 22, 23, 29, and 30
- The fourth parameter (M4%) of mouse function 9 must be passed by reference (instead of by value).
- Mouse function 16 requires four parameters (instead of five).

If your program doesn't call function 9 or 16, you can link it with MOUSE.LIB (version 6.0) without modification.

If your program calls function 9 or 16, you must modify the program so that it conforms with the new interface definitions before you can link it with MOUSE.LIB (version 6.0). If you do not plan to call any of the new mouse functions in your program, you may want to link the program with a previous version of the mouse library.

**Note** Version 5.03 of the Microsoft Mouse Library is included on the Microsoft Mouse Tools disk in the file OLDMOUSE.LIB.

### Appendix C Making Calls from Borland Turbo Pascal Programs

To call mouse functions from a program in Borland Turbo Pascal, use the procedure shown below to pass the correct parameters to the mouse driver. Include this procedure in your code, then call the mouse functions by passing values into this procedure.

Procedure Mouse ( Var m1, m2, m3, m4, m5 : integer ); Var CpuReg: record of AX, BX, CX, DX, BP, SI, DI, DS, ES, FLAGS: integer; end . begin {mouse} if m1 >= 0 then begin CpuReg.AX := m1; {Load parameters CpuReg.BX := m2;{ into appropriate} CpuReq.CX := m3;{ registers if (m1 = 9) or (m1 = 12) or (m1 = 20)or (m1 = 22) or (m1 = 23) then begin CpuReg.ES := seg (m4); { the address of } l; { the user array } { or subroutine } end: else if m1 = 16begin CpuReg.CX := m2; {Left x coordinate} CpuReg.DX := m3; {Upper y coordinate} CpuReg.SI := m4; {Right x coordinate} CpuReg.DI := m5; {Lower y coordinate} end; else CpuReg.DX := m4; intr (\$33, CpuReg); {Call mouse driver } { at interrupt 33h }

C-2

ĺĹ	(m1 = 20) then m2:= CpuReg.ES;	{Special returns }
m1 m2 m3 m4	<pre>:= CpuReg.AX; := CpuReg.BX; := CpuReg.CX; := CpuReg.DX;</pre>	{Return values back} { to parameters }

end;

end; {mouse}

## Appendix D Using the Hercules Graphics Card with Mouse Programs

Before you use the Hercules Monochrome Graphics Card with a program that has built-in mouse support, you must do the following:

- 1 Put the Hercules card into graphics mode (if necessary, see the documentation that came with your Hercules card).
- Store a 6 in memory location 40h:49h if the Hercules card is using CRT page 0. Store a 5 in memory location 40h:49h if the Hercules card is using CRT page 1.
- Call mouse function 0 to set the mouse cursor boundaries and CRT page number to the appropriate values.

Index

### Index

Action statement 1-7, 1-14-1-16 Adapter See specific adapter Address entry 6-20 first element in screen 7-4 register 8-6 restoring previous 6-29 AH register 8-4 ALT-F1(F31) 2-27 ALT-F2(F32) 2-27 ALT-F3(F33) 2-27 ALT-F4(F34) 2-27 ALT-F5(F35) 2-27 ALT-F6(F36) 2-27 ALT-F7(F37) 2-27 ALT-F8(F38) 2-27 ALT-F9(F39) 2-27 ALT-F10(F40) 2-27 Ampersand (&) 7-14 AND operation graphics cursor 5-5 Piano program 7-17 software text cursor 5-7 Argument address 8-3 Array, four-element 6-24 Arrow keys down arrow key 2-25 frequent use 3-1 left arrow key 2-25 right arrow key 2-25 simulating with TYPE statement 2-23 with mouse 3-2 up arrow key 2-25 ASĈII code character set 5-7 control characters 2-23-2-24 extended 2-22 graphics characters 1-12, 1-13, 2-18, 2-21 Ĭist 2-24 use to specify keys 2-22-2-28 value of character 5-7

Assembly-language program calling EGA Register Interface Library 8-2-8-3 cursor 7-23 making function calls 7-3-7-6 use with EGA.LIB 8-1-8-5 Assembly-language subroutine 6-19, 6-20 ASSIGN command 1-2 ASSIGN statement described 2-2 labels 2-2 mouse event value 1-8 sensitivity value 1-8 parameter 2-2 use 1-8, 1-17, 2-3 AT&T 6300 display adapter A-2 Attribute Controller Palette register 8-14 Attribute Controller register 8-6, 8-23 Attribute parameter See also Parameter MATCH statement 2-8 MEND statement 2-12 MENU statement 2-12 POPUP statement 2-17, 2-21 AUTOEXEC.BAT file A-3 AX register 6-19, 6-28, 7-3, 7-4, 8-2 Background color See Color Backspace ASCII code 2-24 use with TYPE statement 2-22 BACKSPACE, prohibited use 1-3 BASIC calling conventions 6-19, 6-28, 8-4 cursor use 7-23 making mouse function calls 7-2-7-3 Basic Input/Output System See BIOS

ROM BASIC interpreter entry point 7-2 BASIC interpreter (continued) fragments 6-3 making function calls 7-2-7-3 parameters 7-3 sample use 6-20 use with function 12 6-19 use with function 20 6-28, 6-29 use with function 22 6-31 bb parameter 3-3 **BEGIN** command 1-2 **BEGIN** statement described 2-4-2-5 format 1-2 initial mouse sensitivity 1-8 labels 2-5 parameters 1-8, 2-5 redefining parameter with ASSIGN statement 2-2 use 1-8, 2-6, 3-2, 3-3 BEGINNING MENU 2-9 BH register, saving and restoring 8-11 BIOS ŘOM 2-28, 8-2, 8-6, 8-21 Bold menu attribute value 1-6 Bold symbolic value 1-17 Borland Turbo Pascal program 7-6, C-1 Brackets ([]), use in statements 2-1 BRUN20.LIB 7-7 btbtn label ASSIGN statement 2-2 **BEGIN statement 2-5** Buffer address 6-32, 7-4 changing size A-2 keyboard 2-22 saving mouse driver state 6-31 size, specifying A-3 storage requirements 6-30 storing state of mouse driver 6-30 BUILDLIB.EXE 7-7 Button bit value 6-8 counter 5-9 double click 2-2, 2-11, 3-1-3-2, 3-3 left 2-2, 3-1, 3-3, 7-4 no action statement (NOTHING) 2-14 number of times pressed 6-10 number of times released 6-11 removing menu with 1-10 returning state 6-8 right 1-17-1-18, 2-2, 3-1, 3-2 state 5-9, 6-19

Button (continued) statement labels 2-5 status 6-8, 6-10, 6-11, 7-18 BX register 6-19, 6-28, 7-3, 7-4, 8-2, 8-4

C model programs 6-20, 6-28 C program calling conventions 8-4 EGA Register Interface Library 8-3-8-5 external routine 7-14 linking with MOUSE.LIB 7-15 making function calls 7-6, 7-14-7-16 mouse library procedures 7-14 parameters 7-14 restoring default setting 8-5 Call mask new values 6-27 parameter definition 7-4 restoring initial values 6-28 restoring previous values 6-29 setting 6-18-6-20Call, mouse function See Mouse CALL statement 7-2 Carriage return 2-24 CAS lines 8-7 cegal procedure 8-3, 8-4 cegam procedure 8-3, 8-4 cegas procedure 8-3, 8-4 CGA mode 7-12 Change Directory submenu 3-2, 3-3--3-4 Character array 8-4 attributes changing with cursor mask 5-7 changing with text cursor 5-6 defined by mask 6-16 preserving with screen mask 5-7 blinking/nonblinking 5-7 screen data 5-7 Check mark cursor 7-27 chkdrv subroutine FORTRAN 7-13-7-14 Pascal 7-10-7-12 QuickBASIC 7-8-7-10 cls command 3-3 CMOUSEC procedure 7-14 CMOUSEL procedure 7-14 CMOUSEM procedure 7-14 CMOUSES procedure 7-14

Colon (:) missing in label 4-1 use in labels 1-3 use in statements 2-1 Color background inverting 5-8 parameter value 1-5-1-6 setting 5-7 foreground inverting 5-8 parameter value 1-5-1-6 setting 5-7 palette 8-6 values for foreground, background 1-6 Column parameter MATCH statement 2-8 MEND statement 2-12 MENU statement 2-12 POPUP statement 2-17 SELECT statement 2-19 COM1, serial mouse on A-5 COM2, serial mouse on A-5 Comma (,) use in parameters 1-3 use in statements 2-1 Command ASSIGN 1-2 **BEGIN 1-2** cls 3-3 **DATE 3-3** DOS editing with DOSOVRLY 3-2 executing with DOSOVRLY 3-2—3-4 DOS COPY 1-21 EXECUTE 1-2 LINK FORTRAN 7-13 Microsoft C 7-15 Pascal 7-10 **MATCH 1-2 MEND 1-2 MENU 1-2** NOTHING 1-2 **OPTION 1-2** PATH 1-21 **PEND 1-2** POPUP 1-2 prohibited use of names 1-3 SELECT 1-2 **TEXT 1-2** 

Command (continued) TIME 3-3 **TYPE 1-2** word, syntax conventions 2-1 Comment, statement 1-7 Compact-model program 7-14 Complex menu, creating 1-10-1-14 Condition mask 6-19 Conditional off function 6-2, 6-24-6-25 CONFIG.SYS file A-3 Constant not allowed in mouse functions 6 - 3Control characters See ASCII code Control Panel program A-1—A-3, A-5 Control Panel switches A-1—A-3 CONTROL- 2-24 CONTROL-<sup>^</sup> 2-24 CONTROL-\_2-24 CONTROL-A 2-24 CONTROL-B 2-24 CONTROL-C 2-24 CONTROL-D 2-24 CONTROL-E 2-24 CONTROL-END 2-25 CONTROL-F 2-24 CONTROL-F1(F21) 2-27 CONTROL-F2(F22) 2-27 CONTROL-F3(F23) 2-27 CONTROL-F4(F24) 2-27 CONTROL-F5(F25) 2-27 CONTROL-F6(F26) 2-27 CONTROL-F7(F27) 2-27 CONTROL-F8(F28) 2-27 CONTROL-F9(F29) 2-27 CONTROL-F10(F30) 2-27 CONTROL-G 2-24 CONTROL-HOME 2-25 CONTROL-K 2-24 CONTROL-L 2-24 **CONTROL-left arrow 2-25** CONTROL-N 2-24 CONTROL-O 2-24 CONTROL-P 2-24 CONTROL-PAGEDOWN 2-25 CONTROL-PAGEUP 2-25 CONTROL-PRINTSCREEN 2-25 CONTROL-Q 2-24 CONTROL-R 2-24 **CONTROL-right arrow 2-25** CONTROL-S 2-24 CONTROL-T 2-24

CONTROL-U 2-24 CONTROL-V 2-24 CONTROL-W 2-24 CONTROL-X 2-24 CONTROL-Y 2-24 CONTROL-Z 2-24 Copying .MŇŬ file 1-21 Mouse Menu file 1-21 Copyright message, Piano program 7-16 Corner, top-left See Column parameter; Row parameter CPANEL.EXE A-1 CPU register 6-19, 7-3 CRT Controller Cursor Location High register 8-16 CRT Controller Cursor Location Low register 8-16 CRT Controller Cursor Start register 8-12 CRT Controller Mode Control register 8-18, 8-20 CRT page 6-4, 6-33 Cursor assembly-language program 7-23 background 5-5 BASIC program 7-23-7-31 block 5-9 check mark 7-27 color 5-5, 7-17 computer, adapting 5-8 coordinates 6-8, 6-9, 6-10, 6-11, 6-12, 6-13, 6-19 default FORTRAN 7-13-7-14 Microsoft C 7-15-7-16 mouse 7-4-7-6 Pascal 7-10-7-12 QuickBASIC 7-8-7-9 diagonal cross 7-29 displaying 6-6 graphics defined 5-4-5-6 defining characteristics 6-14-6-15 hot spot, defined 5-6 parameter 6-4 use with different modes 5-4-5-6 hardware text defined 5-4-5-6 described 5-8-5-9 selecting 6-16 hiding 6-7, 6-24

Cursor (continued) high-level-language program use 7-23 horizontal min/max position 6-4 hot spot check mark 7-27 defined in Piano program 7-17 diagonal cross 7-29 hourglass 7-31 left arrow 7-26 pointing hand 7-28 rectangular cross 7-30 standard shape 7-24 up arrow 7-25 hourglass 7-31 internal flag decrementing 6-7 described 5-10-5-11 incrementing 6-6 parameter 6-4 restoring to initial value 6-7 left arrow 7-26 mask array 7-4 field values 5-8 graphics 5-5-5-6 Piano program 7-17 specifying 6-16 text 5-7 used to build cursor 6-14 minimum/maximum horizontal coordinates 6-12 minimum/maximum vertical coordinates 6-13 movement **BEGIN statement parameters 1-8** double-speed-threshold A-4 help message 1-14 mickey count 5-10 minimum/maximum values 6-12 ratio A-1 SIMPLE mouse menu 6-12 pixel 5-5 pointing hand 7-28 position parameter 6-4 rectangular cross 7-30 removing from screen 6-7 returning CRT page 6-33 samples 7-23-7-31 scan line 6-16 setting position 7-3 shapes 5-5, 7-17, 7-23-7-31

Cursor (continued) software text creating 5-7 defined 5-4-5-6 described 5-6-5-8 selecting 6-16 specifying CRT page 6-33 speed, setting 6-26 standard shape 7-24 text parameter 6-4 setting 6-16 turning on/off 7-17, 7-18 up arrow 7-25 update code 8-1 vertical min/max position 6-4 CURSOR integer array 7-17 CX register Assembly-language program call 7-3 EGA function call 8-4 EGA Register Interface Library 8-2 set interrupt subroutine 6-19 swap interrupt subroutine 6-28 Data register 8-6 DATE command 3-3 .DEF extension 4-2 .DEF source file 1-19 Default settings restoring 8-21 size A-3 values Attribute Controller 8-23 EGA write-only registers 8-1 pointer/data chip 8-22 single régister 8-22 DEFINT statement 7-2 DELETE key 2-25 DI register 6-19, 6-28, 7-4 Diagonal cross cursor 7-29 Directory submenu 3-2, 3-3-3-4 Disk active, changing with MATCH statement 1-17-1-18 Mouse Tools 1-18, 7-6, 7-16, 8-1 Display adapter 5-2, 5-7, 5-8 **Display** attribute specifying with MENU statement 1-9 parameter See Parameter value 2-17

Display mode, changing with graf 7-12 dnmot label ASSIGN statement 2-2 **BEGIN statement 2-5** DOS commands editing with DOSOVRLY 3-2 executing with DOSOVRLY 3-1, 3-2-3-4executing with mouse 3-1, 3-2-3-4 DOS COPY command 1-21 DOS system applications bypassing 2-23 prompt 4-1, 5-8 DOSOVRLY Mouse Menu program 3-1, 3 - 2 - 3 - 4Double-precision variables 6-3 Down arrow key 2-25 DS register 6-19, 6-28 Dummy variables defined 6-3 DX register Assembly-language program call 7-3 EGA function call 8-4 EGA Register Interface Library 8-2 saving and restoring 8-11 set interrupt subroutine 6-19 swap interrupt subroutine 6-28 EGA four-plane mode 5-6 EGA procedure 8-3, 8-4 EGA Register Interface BIOS ROM calls 8-6 calling 8-2-8-5 calling from assembly-language program 8-2-8-3calling from high-level language programs 8 described 8-1-8-2 function call 8-2, 8-3-8-5 listed 8-8-8-25 input values not checked 8-8 restoring default settings 8-21 restrictions on use 8-6 EGA register, restoring default setting 8-5 EGA.LIB license agreement iii linking with 8-3 EGAS procedure 8-3, 8-4, 8-5 Ellipsis (...), use in statements 2-1 END key 2-25

Enhanced graphics modes A-2 ENTER key 3-1, 3-2 ENTER, prohíbited use 1-3 Entry address 6-20 Error messages 4-1 ES register EGĂ function call 8-4 EGA Register Interface Library 8-2 ES:DX register 7-4 ESCAPE key 2-24, 3-1 ESCAPE, prohibited use 1-3 EXECUTE command 1-2 EXECUTE statement described 1-14-1-15, 2-7 error 4-1 parameter 2-7 use 1-17, 3-3 variable number of parameters 1-3 Expert Mouse Menu program A-5 Extended graphics mode 6-1 Extended keyboard scan codes 2-25-2-27 F1 key 2-26 F2 key 2-26 F3 key 2-26 F4 key 2-26 F5 key 2-26 F6 key 2-26 F7 key 2-26 F8 key 2-26 F9 key 2-26 F10 key 2-26 Far return instruction 6-19, 6-28 Feature Control register 8-12, 8-23 File AUTOEXEC.BAT A-3 **CONFIG.SYS A-3** creating in QuickBASIC 7-7 MENU.COM copying 1-21 memory allocation 1-22 .MNU 1-19, 1-21, 1-22 object 6-19, 6-28 PIANO.BÁS 7-16 PREADME.DOC i source See Source file standard ASCII text 1-19 Filename menu 4-2 Foreground color See Color

Format menu subroutine statement 1-9 Popup subroutine 1-11 statement See Statement FORTRAN address 6-20 calling conventions 8-4 chkdry subroutine 7-13-7-14 EGA Register Interface Library 8-3-8-5 external subroutine 7-13 graf subroutine 7-13-7-14 IBM Color/Graphics Adapter 7-13-7-14 linking with MOUSE.LIB 7-13 making function calls 7-6, 7-13-7-14 use with function 20 6-28 Frequency setting, Piano program 7-16, 7-18 Function call, making from assembly-language program 7-3-7-6 define default register table 8-22-8-23 F0 8-9-8-10, 8-21 F1 8-11-8-12 F2described 8-13-8-14 high-level-language program 8-4 use 8-2 F3 described 8-15-8-16 high-level-language program 8-4 use 8-2 F4 described 8-17-8-18 high-level-language program 8-4 use 8-2 F5described 8-19-8-20 high-level-language program 8-4 setting the color palette 8-6 use 8- $\overline{2}$ F6 8-7, 8-21 F7described 8-22-8-23 high-level-language program 8-4 use 8-2, 8-21 F8h, reserved 8-8 F9h, reserved 8-8 FA 8-4, 8-24-8-25 FBh-FFh, reserved 8-8 interface 8-2

Function (continued) interrogate driver 8-24-8-25 mouse See Mouse read one register 8-9-8-10 read register 8-17-8-18 read register range 8-13-8-14 revert to default registers 8-21 write one register 8-11-8-12 write register range 8-15-8-16 write register set 8-19-8-20

Get button press information function 6-2, 6 - 10Get button release information function

6-2, 6-11Get button status and mouse position function 6-8

Get button status function 6-2

Get CRT page number function 6-2, 6-33

Get mouse driver state storage requirements function 6-2, 6-30 Glitch

while accessing Graphics Controller Miscellaneous register 8-7

while accessing Sequencer Memory Mode register 8-6 graf subroutine

FORTRAN 7-13-7-14

Pascal 7-10-7-12

Graphics

characters See ASCII code

cursor See Cursor

interrupt-driven 8-1

Graphics Controller Miscellaneous register 8-7 Graphics mode

changing with graf subroutine 7-12 described 5-3 high-resolution 7-23 IBM Enhanced Graphics Adapter 5-5

menu, restrictions 1-1 mode4 5-3, 5-4, 5-5-5-6

mode5 5-3, 5-4, 5-5-5-6

mode6 5-3, 5-4, 5-5-5-6 mode10 5-3, 5-4, 5-5

mode30 5-3, 5-4, 5-5-5-6

modeE 5-3, 5-4, 5-5

modeF 5-3, 5-4, 5-5-5-6

Hardware test cursor See Cursor Hercules card 5-2, 5-3, 5-4, 5-5, A-2, D-1 Hide cursor function 6-2, 6-7 High-level-language program calling conventions 8-4 calling from 6-19-6-20 cursor 7-23 EGA Register Interface Library 8-3-8-5 linking with MOUSE.LIB 7-6, B-1 making function calls 7-6-7-16 use with EGA.LIB 8-1-8-5 use with function 20 6-28 High-resolution graphics mode 7-23 Highlight position in menu 1-10 use with DOSOVRLY 3-3 **HMAX 6-9** HOME key 2-25 Horizontal movement, sensitivity parameter 1-8, 2-2 Horizontal tab 2-24 Hot spot See Cursor Hourglass cursor 7-31 hsen label ASSIGN statement 2-2 **BEGIN statement 2-5** Huge-model program 7-14

IBM 3270 PC 5-9 IBM All Points Addressable Graphics Adapter 5-2 IBM Color Display 5-9 IBM Color/Graphics Adapter changing buffer size A-2 use 7-4-7-6 use with Microsoft C 7-15-7-16 use with Pascal 7-10 use with QuickBASIC 7-8-7-9 IBM Enhanced Color Display 5-9 IBM Enhanced Graphics Adapter changing buffer size A-2 interacting with 8-1 use with extended graphics modes 6-1 IBM Monochrome Display and Printer Adapter 5-2, 5-9, A-2 IBM PC 5-1-5-11, 7-16 Illegal function call 4-1 Indexes, consecutive 8-13 InPort mouse A-4—A-5 Input Status register 8-7

Input value, function description 6-3 INSERT key 2-25, 3-1-3-2 Integer array 6-14, 8-4 signed or unsigned 7-14 **INTEGER** parameter 7-10 Intensity setting 5-7 Internal cursor flag decrementing 6-7 described 5-10-5-11 incrementing 6-6 parameter 6-4 restoring to initial value 6-7 Interrupt 10h 5-3, 8-6, 8-21 Interrupt call mask parameter 6-4 Interrupt-driven graphics 8-1 Interrupt rate A-3, A-4 Invalid parameter See Parameter Invalid statement 4-1 Inverse menu, attribute value 1-6 Inverse symbolic value 1-17

#### Key

See also specific key ASCII code 1-15 simulation with TYPE statement 1-15, 2-22-2-28 special-function 2-22 specifying 2-22-2-2-28 symbolic name 1-15, 2-22 Key parameter, TYPE statement 2-22 Keyboard buffer 2-22 direct access applications 2-23 emulation 4-2 scan code 1-16, 2-22

#### Label

colon used with 1-3 described 1-3 error 4-1 ignored by MAKEMENU program 2-15 menu subroutine statement 1-9 mouse movement 2-5 not required for BEGIN statement 2-4 parameter See Label parameter prohibited words 1-3 rules for use 1-3 syntax conventions 2-1 Label (continued) unique 4-1 when used 1-2 Label parameter MATCH statement 2-8 MEND statement 2-12 MENU statement 2-12 POPUP statement 2-16 TYPE statement 2-22 Large-model program 7-14, 8-3 lb parameter 3-3 Left arrow cursor 7-26 Left arrow key 2-25 Legend text defining with TEXT statement 2-21 menu for OPTION statement 2-15 placement on screen 2-21 lfbtn label ASSIGN statement 2-2 BEGIN statement 2-5 lfmot label ASSIGN statement 2-2 **BEGIN statement 2-5** License agreement iii Light pen emulating 6-21, 6-22 Light pen emulation mode off function 6-2, 6-22 on function 6-2, 6-21 parameter 6-4 Line feed 2-24 Line number specifying 6-16 LINK command FORTRAN 7-13 Microsoft C 7-15 Pascal 7-10 lm parameter 3-3 Long argument address 8-4, 8-5

M1% 6-3 M2% 6-3 M3% 6-3 M4% 6-3 MAIN MENU, in WordStar 2-9 MAKEMENU program ignoring labels for OPTION statement 2-15 messages 4-1--4-2 use 1-19, 1-20, 3-1 Mask See also Cursor mask; Screen mask Mask (continued) bit 6-27 interrupt call, parameter 6-4 MATCH command 1-2 Match parameter 2-9 MATCH statement combining with Popup subroutines 1-11 described 1-16-1-18, 2-8-2-10 parameters 2-8-2-9 use 1-17-1-18, 2-14 Maximum cursor coordinates See Cursor Medium-model program 7-14, 8-3 Memory-resident program A-1 MEND command 1-2 MEND statement described 2-11-2-13 lack of parameters 1-3 use 1-10 Menu borders 1-13, 2-18 choosing item 1-14-1-16 clearing after item choice 2-15 command list 1-2 complex, creating 1-10-1-14 corner column specified 2-12 row specified 2-12 corner coordinate, specifying 1-9, 1-12 creating 1-1-1-22 dimension determination 2-11 display attribute value 2-12 specifying with MENU statement 2-12 specifying with POPUP statement 2-17display attribute, specifying 1-9, 1-12 DOSOVRLY program 3-1, 3-2-3-4 ending A-6 exit from with OPTION statement 1-9 file 1-19-1-20 filename 4-2 hierarchy, creating 3-2 highlight 1-10, 2-11, 2-19 item location 1-13 item selection area character number specified 2-19 column specified 2-19 defined 2-19-2-20 error 4-1 row specified 2-19 language statements 2-1-2-28

Menu (continued) legend text 2-15 memory allocation 1-22 multiple-column creating 1-10, 1-13, 2-16 sample 1-11 popup defining title 2-21 single-column created with MENU statement 2-11 program ending 1-21 not in directory 1-21 running 1-21 sample 3-1-3-4 structure 1-7-1-8 programming language 1-1-1-7 removing with mouse buttons 1-10 sample source program 1-17-1-18 screen cleared 2-20 SIMPLE program 3-1-3-2 source files 1-18 subroutine beginning statement 1-9 end statement 1-10 ending with MEND statement 2-11 exit statement 1-9 MENU statement used in 2-11 sample 1-10 testing 1-20 title defining with TEXT statement 2-21 specifying 1-9, 2-12 top-left corner See Column parameter; Row parameter use prohibition 1-1 WS.DEF 2-14 MENU command 1-2 MENU program messages 4-1-4-2 MENU statement described 2-11-2-13 use 1-9, 1-10, 2-13 Menu subroutine statement described 1-7, 1-9-1-10 format 1-9 label 1-9 MENU.COM file copying 1-21 memory allocation 1-22 Message box creating 1-14

Message box (continued) creating with Popup subroutine 2-16 Popup subroutine example 2-18 sample 1-11 top-left corner See Column parameter; Row parameter Messages 4-1-4-2 Mickey count 5-10, 6-17, 6-19, 6-28 default value 2-5, 2-6 described 2-4, 5-9-5-10 use 1-8 Mickey/pixel ratio horizontal, parameter 6-4 setting 6-23 vertical, parameter 6-4 Microsoft copyright notice iii license agreement iii logo 7-17, 7-18 Product Support ii Microsoft C See C program Microsoft EGA Register Interface Library See EGA Register Interface Microsoft Expert Mouse Menu program 6 - 30Microsoft FORTRAN See FORTRAN Microsoft Mouse Library 7-6 Microsoft Mouse Tools disk 7-6, 7-16 Microsoft Pascal See Pascal Microsoft QuickBASIC See QuickBASIC Microsoft Word 1-19 Minimum cursor coordinates See Cursor Miscellaneous Output register 8-10, 8-18, 8-20 .MNU extension 1-20 .MNU file copying 1-21 described 1-19 size limitation 1-22 Mode display, changing with graf 7-12 graphics See Graphics Hercules graphics display 5-3 light pen emulation 6-4 screen 5-1-5-2, 6-7 text See Text mode Mode-change calls 8-2 Mouse button See Button disabling vertically, horizontally 2-5

Mouse (continued) driver checking installation 7-3 data segments 6-19 disabling A-5—A-6 FORTRAN 7-13—7-14 interrogating 8-24-8-25 loading automatically 1-20 loading EGA.LIB 8-1 MOUSE.COM A-6 MOUSE.SYS A-6 Pascal 7-10-7-12 QuickBASIC 7-7, 7-9-7-10 removing A-5—A-6 resetting 6-4 restoring state 6-32 saving state 6-31 state, restoring 6-32 storing state in buffer 6-30 switches A-3—A-6 emulating light pen 6-21, 6-22 entry offset 7-2 segment 7-2 function 0 5-3, 5-9, 6-4-6-5 function 1 5-10-5-11, 6-6, 7-17 function 2 5-10—5-11, 6-7 function 3 6-8, 7-18 function 4 6-9, 7-17 function 5 5-9, 6-10 function 6 5-9, 6-11 function 7 6-12function 8 6-13 function 9 5-4, 5-5, 5-6, 6-14-6-15, 7-3-7-4, 7-17 function 10 5-4, 5-8, 6-16 function 11 6-17 function 12 6-18-6-20, 7-3-7-4 function 13 6-21 function 14 6-22, 6-23 function 15 7-17 function 16 6-24-6-25, 7-3-7-4 function 19 6-26, A-4 function 20 6-20-6-29, 7-3-7-4 function 21 6-30 function 22 6-31, 7-3-7-4 function 23 6-32, 7-3-7-4 function 29 6-33 function 30 6-33 function list 6-2 hardware configuration A-3

Mouse (continued) hardware interrupts 6-18-6-20, 6-27 hardware status 6-4 InPort A-4—A-5 motion, reading 6-17 movement parameters 2-2 sensitivity 7-17 setting ratio 6-23 statement labels 2-5 parameter See Parameter program existing program, linking with MOUSE.LIB B-1 temporarily interrupting 6-30, 6-31, 6 - 32reset and status function 6-2, 6-4-6-5 sensitivity adjusting A-1 defining 5-10 double-speed threshold A-4 horizontal A-4 initial 1-8 setting A-4 specifying A-3 vertical A-4 Setup program 1-20 software linking to program 7-16 status 6-4 specifying A-4—A-5 support, built-in 1-1 threshold speed 6-26 tracking 6-6, 6-7 unit of distance 5-9 Mouse count See Mickey Mouse Event Statement 1-7-1-8 Mouse function See Mouse Mouse Library license agreement iii Mouse Menu See Menu Mouse Tools disk 1-18, 8-1 MOUSE.LIB license agreement iii linking with existing mouse programs **B-1** linking with FORTRAN 7-13 linking with high-level-languages 7-6, B-1 linking with Microsoft C 7-14-7-15 linking with Pascal 7-10 linking with QuickBASIC 7-7-7-8

MOUSE.LIB (continued) parameter requirements 7-14 MOUSES use with FORTRAN 7-13 use with Pascal 7-10 use with QuickBASIC 7-7 Movement sensitivity parameter 2-4 values 1-4 Multiple-column menu See Menu Nomatch in MATCH statement 2-9 Normal symbolic value 1-17 NOTHING command 1-2 NOTHING statement described 1-16, 2-14 equivalent 1-9, 2-11 lack of parameters 1-3 Numeric parameter See Parameter OLDMOUSE.LIB iii, B-1 Operation AND graphics cursor 5-5 Piano program 7-17 software text cursor 5-7 ORgraphics cursor 5-5 Piano program 7-17 **OPTION** command 1-2 **OPTION** statement described 2-15 error 4-1 use 1-9, 1-10, 2-13 OR operation 5-5, 7-17 Output value 6-3 Page number, CRT, parameter 6-4 PAGEDOWN key 2-25 PAGEUP key 2-25

Palette register 8-3 Parameter ASSIGN statement 2-2 attribute MATCH statement 1-17 MEND statement 2-12 MENU statement 2-12 POPUP statement 2-17, 2-21 Parameter (continued) BASIC call 7-2 bb 3-3 before and after call 6-3 **BEGIN statement 2-5** bold 1-4, 1-6 button 1-8 changing value with ASSIGN statement 2-3color specification 1-5-1-6 column MATCH statement 1-17 MEND statement 2-12 MENU statement 2-12 POPUP statement 2-17 SELECT statement 2-19 comma used with 1-3-1-4 default attribute 1-5 default value 1-4, 2-3 definitions 7-4 described 1-3-1-6 disabling with NOTHING statement 1 - 16display attribute 1-4-1-5 dummy variable names 6-3 E1 8-4 E2 8-4 E3 8-4 E4 8-4 E5 8-4 excess 4-1 EXECUTE statement 2-7 function call requirements 6-3, 8-4 function F0 8-9-8-10 function F1 8-11-8-12 function F2 8-13-8-14 function F3 8-15-8-16 function F4 8-17-8-18 function F5 8-19-8-20 function F6 8-21 function F7 8-22-8-23 function FA 8-24-8-25 horizontal movement sensitivity 2-2 incorrect for function calls 6-3 initial value assigned 2-3 input requirement 6-3 INTEGER 7-10 invalid 4-1 inverse 1-4, 1-6 key 2-22

Parameter (continued) label MEND statement 2-12 MENU statement 2-12 POPUP statement 2-16 TYPE statement 2-22 lb 3-3 limit to value 6-9 lm 3-3 M1%, M2%, M3%, M4% 6-3 match 1-17 MATCH statement 2-8-2-9 MEND statement 2-12 MENU statement 2-12 mouse, redefining with ASSIGN statement 2-2 mouse function 10 5-8 movement 1-8 nonmatch 1-17 normal 1-4 numeric 1-4 **OPTION** statement 2-15 optional 2-1 output requirement 6-3 pointer 1-9, 1-13, 2-20 POPUP statement 2-16-2-17 preceded with ampersand (&) 7-14 rb 3-3 redefining with ASSIGN statement 2-2 register for EGA Register Interface Library 8-2 required 2-1 required for EGA Register Interface Functions 8-8 resetting default values 5-3 rm 3-3 row MATCH statement 1-17 MEND statement 2-12 MENU statement 2-12 POPUP statement 2-16 SELECT statement 2-19 signed integer 7-14 statements without parameters 1-3 string 1-4, 2-21 syntax conventions 2-1 title MEND statement 2-12 MENU statement 2-12 TYPE statement 2-22 unsigned integer 7-14

Parameter (continued) unused 1-3-1-4 value not required 8-8 value not specified 2-2 vertical movement sensitivity 2-2 width, SELECT statement 2-20 WORD 7-10 Pascal address 6-20 calling conventions 7-10, 8-4 chkdrv subroutine 7-10-7-12 EGA Register Interface Library 8-3-8-5 graf subroutine 7-10-7-12 IBM Color/Graphics Adapter used with 7-10 long argument address 8-5 making function calls from 7-6, 7-10-7-12 use with function 20 6-28 PATH command 1-21 PEN function 6-21, 6-22 PEND command 1-2 **PEND** statement described 1-12, 2-16-2-18 lack of parameters 1-3 use 1-11, 1-13-1-14 Percent sign (%) in dummy variables 6-3 use with variable names 7-2 Piano keys 7-17 Piano program, source code 7-16-7-22 PIANO BAS file 7-16 Pixel 8-by-8 5-3 8-by-16 5-4 16-by-8 5-3-5-4 16-by-16 5-4 cursor 5-5 even-numbered correspondence 5-3 movement 5-10 number on screen 5-1 one-to-one correspondence 5-3 setting ratio to mickey 6-23 Pointer/data chip 8-13, 8-15 Pointer parameter See Parameter Pointing hand cursor 7-28 POPUP command 1-2 Popup menu defining title 2-21 single-column, created with MENU statement 2-11

**POPUP** statement attribute parameter 2-21 described 1-12, 2-16-2-18 use 1-11, 1-13-1-14 Popup subroutine combining with MATCH statements 1 - 11defining legend text with TEXT statement 2-21 defining menu title with TEXT statement 2-21 ending 1-12, 2-16 first statement 2-16 format 1-11 sample 1-13-1-14 statement 1-10-1-14 types 1-7 PREADME.DOC file i Product Support ii Program See specific program Programming language for Mouse Menu 1-1-1-7 PUT statement 7-18 QuickBASIC address 6-20 EGA Register Interface Library 8-3-8-5 external subroutine 7-7 linking with MOUSE.LIB 7-8 making function calls 7-6, 7-7-7-10 use with function 20 6-28 QUIT box 7-17, 7-18 Quotation marks ("") incorrect placement 4-1 simulating 2-22 specifying keys with 2-22 use in statements 2-1 rb parameter 3-3 Read mouse motion counter function 6-2, 6-7 Read one register function 8-9-8-10 Read-Only Memory See BIOS ROM Rectangular cross cursor 7-30 Register **Äddress 8-6** AH 8-4 Attribute Controller 8-6, 8-23 Attribute Controller Palette 8-14

Register (continued) BH 8-11 **BX 8-4** CPU 6-19, 6-28 CRT Controller Cursor Location High CRT Controller Cursor Location Low 8-16 CRT Controller Cursor Start 8-12 CRT Controller Mode Control 8-18, 8-20 CX 8-4 Data 8-6 DX 8-4, 8-11 Feature Control 8-12, 8-23 Graphics Controller Miscellaneous 8-7 Input Status 8-7 Miscellaneous Output 8-10, 8-18, 8-20 palette 8-3 range, defined 8-13 Sequencer Map Mask 8-10 Sequencer Memory Mode 8-6-8-7, 8-18, 8-20 Sequencer Reset 8-7 set, defined 8-17 write-only 8-1 Restore mouse driver state function 6-2, 6 - 32Right arrow key 2-25 rm parameter 3-3 ROM BIOS See BIOS ROM Routine, external, use with Microsoft C 7-14 Row parameter MATCH statement 2-8 MEND statement 2-12 MENU statement 2-12 POPUP statement 2-16 SELECT statement 2-19 rtbtn label ASSIGN statement 2-2 **BEGIN statement 2-5** rtmot label ASSIGN statement 2-2 **BEGIN statement 2-5** 

Save mouse driver state function 6-2, 6-31 Scan code, keyboard See Keyboard Scan line 5-8, 6-16 Screen bit, resulting 5-5 Screen (continued) buffer A-1, A-3 coordinate 6-24, 7-4 data, character 5-7 defining region for updating 6-24 graphics, clearing 7-16 legend text placement 2-21 mask field values 5-8 graphics 5-5-5-6 specifying 6-16 text 5-7 used to build cursor 6-14 mode 5-1-5-2, 6-3, 6-7 overlay buffer A-2 removing cursor from 6-7 virtual 5-2-5-4, 6-9, 6-10, 6-11, 6-13 Screen coordinates 1-4, 2-9 Screen mask 7-17 SELECT command 1-2 SELECT statement described 1-12, 2-19-2-20 error 4-1 use 1-11, 1-13-1-14, 2-17, 2-18 Sensitivity, mouse See Mouse Sequencer Map Mask register 8-10 Sequencer Memory Mode register 8-6-8-7, 8-18, 8-20 Sequencer Reset register 8-7 Set CRT page number function 6-2, 6-33 Set double-speed threshold function 6-2, 6 - 26Set graphics cursor block function 6-2, 6 - 14 - 6 - 15Set interrupt subroutine call mask and address function 6-2, 6-18-6-20 Set mickey/pixel ratio function 6-2, 6-23 Set minimum/maximum vertical cursor function 6-2, 6-13 Set minimum/maximum horizontal cursor function 6-2, 6-12 Set mouse cursor position function 6-9 Set text cursor function 6-2, 6-16 Setup program for mouse 1-20 Shadow maps 8-1-8-2 SHIFT-F1(F11) 2-26 SHIFT-F2(F12) 2-26 SHIFT-F3(F13) 2-26 SHIFT-F4(F14) 2-26 SHIFT-F5(F15) 2-26 SHIFT-F6(F16) 2-26

SHIFT-F7(F17) 2-26 SHIFT-F8(F18) 2-26 SHIFT-F9(F19) 2-26 SHIFT-F10(F20) 2-26 SHIFT-TAB 2-25 Short argument address 8-4 Show cursor function 6-2, 6-6 SI register 6-19, 6-28, 7-4 Signed integer 7-14 SIMPLE Mouse Menu program 3-1-3-2 Simulating arrow keys See TYPE statement keystrokes See TYPE statement Single-column menu See Menu Single-precision variables 6-3 Small-model program 7-14, 8-3 Software interrupt 16(10h) 8-2 Software interrupt 51(33H) 7-3 Software text cursor See Cursor SOUND statement 7-18 Source code, Piano program 7-16-7-22 Source file .DEF 1-19 errors 1-19 on Mouse Tools disk 1-18 saving with word processing program 1-19 size limitation 1-19 use with menu programs 1-19 WS.DEF 2-9-2-10 Space, simulating 2-23 Speed-doubling, cursor 6-26 Spread sheet applications 3-1 Standard cursor 7-24 Statement action, described 1-7 ASSIGN described 2-2 use 1-17, 2-3 BEGIN described 2-4-2-5 **DOSOVRLY 3-3** format 1-2 initial mouse sensitivity 1-8 parameters 1-8 redefining parameter with ASSIGN statement 2-2 use 1-8, 2-6, 3-2 CALL 7-2 calling, use 2-7 comment 1-7

Statement (continued) DEFINT 7-2 EXECUTE described 1-14-1-15, 2-7 error 4-1 use 1-17, 3-3 variable number of parameters 1-3 format 1-2-1-7 invalid 4-1 label mouse movement 2-5 use with EXECUTE statement 1-14-1-15 MATCH described 1-16-1-18, 2-8-2-10 format 1-17 use 1-17-1-18, 2-14 MEND described 2-11-2-13 lack of parameters 1-3 use 1-10 MENU described 2-11-2-13 use 1-9, 1-10, 2-13 menu subroutine, described 1-7 Mouse Event, described 1-7-1-8 NOTHING described 1-16, 2-14 equivalent 1-9 lack of parameters 1-3 OPTION described 2-15 error 4-1 use 1-10, 2-13 order of appearance 1-3 parameter See Parameter PEND described 1-12, 2-16-2-18 lack of parameters 1-3 use 1-11, 1-13-1-14 POPUP described 1-12, 2-16-2-18 use 1-11, 1-13-1-14 Popup subroutine 1-7, 1-10—1-14 PÚT 7-18 SELECT described 1-12, 2-19-2-20 error 4-1 use 1-11, 1-13-1-14, 2-17, 2-18 SOUND 1-16-1-18, 7-18 string match, described 1-7, 1-16-1-18 Statement (continued) syntax conventions 2-1 ŤEXT described 1-12, 2-21 use 1-11, 1-13-1-14, 2-17 TYPE described 1-15-1-16, 2-22 error 4-1 sample 1-16 use 1-17, 3-2, 3-3 variable number of parameters 1-3 types described 1-7 within EXECUTE statement 2-7 within Menu subroutines 1-2 within Popup subroutines 1-2 Status mouse hardware 6-4 mouse software 6-4 String match statement 1-7, 1-16-1-18 String parameter See also Parameter MATCH statement 2-9 **TEXT** statement 2-21 Submenu Change Directory 3-2, 3-3-3-4 Directory 3-2, 3-3-3-4 hierarchy, creating 3-2 Subroutine See also specific subroutine address 6-27, 6-28, 7-4 assembly-language 6-19, 6-28 calling 6-18, 6-19, 6-28 chkdrv in FORTRAN 7-13-7-14 QuickBASIC 7-8-7-10 with Pascal 7-10-7-12 disabling for certain condition 6-27 enabling for certain condition 6-18, 6-27 external, use with FORTRAN 7-13 external, use with QuickBASIC 7-7 graf in FORTRAN 7-13-7-14 with Pascal 7-10-7-12 menu ending with MEND statement 2-11 MENÜ statement used in 2-11 offset 6-29 Popup subroutine, first statement 2-16 segment 6-28-6-29 USERLIB.EXE 7-7-7-8 Swap interrupt subroutine function 6-2, 6-27-6-29

Switch Control Panel A-1-A-3 mouse driver A-3-A-6 settings for Control Panel A-2 Symbolic name for key 2-22 Synchronous Reset 8-7 Syntax conventions, statement 2-1 error 4-1 TAB, prohibited use 1-3 TEXT command 1-2 Text cursor See Cursor Text mode described 5-3 mode 0 5-3-5-4 mode 1 5-3-5-4 mode 2 5-3 mode 3 5-3 mode 7 5-3 overlay buffer size A-2 Text parameter 2-15 TEXT statement described 1-12, 2-21 parameter 2-21 use 1-11, 1-13-1-14, 2-17 Text string designation 4-1 Threshold speed, mouse 6-26 TIME command 3-3 Title parameter MEND statement 2-12 MENU statement 2-12 TYPE command 1-2 **TYPE** statement ASCII control characters 2-23-2-24 described 1-15-1-16, 2-22 error 4-1 key sequences not simulated with 2-28 keyboard scan codes 2-25-2-27 sample 1-16 use 1-17, 3-2, 3-3 variable number of parameters 1-3 Unsigned integer 7-14 Up arrow cursor 7-25

- Up arrow key 2-25 upmot label
- ASSIGN statement 2-2 BEGIN statement 2-5

#### USERLIB.EXE, setting up 7-7-7-8 Utility program, MAKEMENU 1-19-1-20

Variable double-precision 6-3 single-precision 6-3 VC.DEF program, use with POPUP statement 2-17 Version number, returning 8-1 Vertical movement, sensitivity parameter 1-8, 2-2 Video random access memory 8-6, 8-7 Virtual screen See Screen VMAX 6-9 vsen label ASSIGN statement 2-2 BEGIN statement 2-5

Width parameter, SELECT statement 2-20
WORD parameter 7-10
Word processing program, saving source files 1-19
WordStar sample menu 2-9
Write mask register 5-6
Write-only registers 8-1
WS.DEF program 2-9-2-10, 2-14, 2-17

XOR operation graphics cursor 5-5 software text cursor 5-7

# MICROSOFT.

16011 NE 36th Way, Box 97017, Redmond, WA 98073

Problem Report

Street	· · · · · · · · · · · · · · · · · · ·		
City	t	State	Zip
Phone			Date
Instructions			
Use this form to rementation errors,	port problems wit or suggested enha	h Microsoft ncements. M	hardware or software, do fail the form to Microsoft.
Category			
Software Pr	oblem	Hard	ware Enhancement
Hardware H	Problem	Docu	mentation Problem
		(Doc	cument #)
Sottware Er	nhancement		
Software Er	nhancement	Othe	r
Product Desci	ription	Othe	r
Software Er Product Desci Microsoft Product	ription	Othe	r
Software Ef     Product Desci Microsoft Produce Rev	ription ct	Othe	r
Software En     Product Desci Microsoft Produce Rev Operating System	ription ct Registration	Othe	r
Software Er Product Descr Microsoft Product Rev Operating System Rev	ription ct Registration Supplier	Othe	r
Software Ef     Product Descr     Microsoft Produce     Rev Operating System     Rev Other Software Us	ription ct Registration Supplier	Othe	r
Software Er Product Descr Microsoft Produc Rev Operating System Rev Other Software Us Rev	ription ct Registration Supplier sed Supplier	Othe	r
Software Eff Product Description Microsoft Product Rev Operating System Rev Other Software Us Rev Hardware Description	ription ct Registration Supplier sed Supplier	Othe	۲ 
Software Ef     Product Descri Microsoft Product Rev Operating System Rev Other Software Us Rev Hardware Descri Manufacturer	ription ctRegistrationSupplier sedSupplier iptionCPU	Othe	r 
Software Ef Product Descri Microsoft Product Rev Operating System Rev Other Software Us Rev Hardware Descri Manufacturer Disk Size	nhancement ription ctRegistrationSupplier redSupplier iptionCPU	Othe	r Memory K
Software En	ription  ctRegistration  ctRegistration  ctSupplier  cedSupplier  iption CPUTDensity: Single	# Othe	r MemoryK

Describe the problem. (Also describe how to reproduce it, and your diagnosis and suggested correction.) Attach a listing if available.

