LOGITECH

MOUSE PROGRAMMER'S TOOLKIT

LOGITECH, Inc.
Corporate Headquarters

805 Veterans Blvd.
Redwood City, CA 94063
U.S.A.
(415) 365-9852

LOGITECH, SA
European Headquarters

CH-1111 Romanel-Morges
Switzerland
41/21/87 96 56

ALGOL LOGITECH, Spa

Via Durazzo 2
20134 Milano, MI
Italy
39/2/215-5622

First Edition November 1986

LOGITECH, Inc. makes no warranties with respect to this documentation and disclaims any implied warranties of merchantability and fitness for a particular purpose. The information in this document is subject to change without notice. LOGITECH, Inc. assumes no responsibility for any errors that may appear in this document.

From time to time changes may occur in the filenames and in the files actually included on the distribution disks. LOGITECH, Inc. makes no warranties that such files or facilities as mentioned in this documentation exist on the distribution disks or as part of the materials distributed.

LU-UD004-1

Initial issue: November 1986

Current revision: November 1986

This edition applies to Release 3.0 and above of the software.

Printed: November 1986

TRADEMARKS

LOGIMOUSE is a registered trademark and *CLICK* and *LOGIMENU* are trademarks of LOGITECH, Inc.
IBM is a registered trademark of, and IBM PC/XT and IBM AT are trademarks of International Business Machines Corporation.
*Hercules*TM Graphics Card is a trademark of Hercules Computer Technology.
HP Vectra is a registered trademark of Hewlett Packard Company.
Lotus and *1-2-3* are registered trademarks of Lotus Development Corp.
WordStar is a registered trademark of MicroPro.
AT&T, AT&T 6300 and *AT&T 6300 Plus* are trademarks of AT&T.
Sigma Color 400 is a trademark of Sigma Designs, Inc.
Tecmar Graphics Master is a trademark of Tecmar, Inc.
Microsoft and *MSDOS* are registered trademarks of Microsoft Corp.
Compaq Portable is a trademark of Compaq Computer Corp.

TABLE OF CONTENTS

CHAPTER 1
LOGITECH MOUSE DRIVER FUNCTIONS

This manual describes the functions of the standard mouse driver, provides descriptions of the graphics and text cursors, and gives examples of how to use each of the functions from 8086 assembler and high-level languages.

The software protocols used by the LOGITECH MOUSE driver are compatible with the standard driver protocols of Microsoft Corp. as documented in the installation and operation manual published by Microsoft Corp. entitled *Microsoft (R) Mouse User's Guide for IBM Personal Computers.*

This reference section applies to the LOGITECH Mouse Driver (*MOUSE.COM* and *MOUSE.SYS*) Release 3.00 and above.

Interface modules are available for a variety of languages. The source code of the interfaces is included in the diskette accompanying this manual.

The floppy diskette that comes with the *LOGITECH MOUSE PROGRAMMERS Toolkit* contains subdirectories *(MOD, C, TURBO, MSP)* which contain the source code of programs to interface the Mouse Driver with a Modula-2, C, Turbo Pascal, Microsoft Pascal program respectively.

The examples have been written using the Microsoft 8086 Assembler, the LOGITECH MODULA-2 compiler, the Microsoft C compiler, the Borland Turbo Pascal compiler, the Microsoft Pascal compiler the GWBASIC system and the IBM BASICA system. The interfaces may require some adaptations for other compilers, memory models of the 8086, or environments.

1.1 PROGRAM INTERFACE

1.1.1 Programming Interface for Assembler

The mouse driver functions are called through a software interrupt, number 51 (33H). The parameters and returned values are passed in the registers. Register AX is always used for the function number.

1.1.2 Programming Interface for Modula-2

The module **MOUSE** provides the interface to the mouse driver for a Modula-2 program compiled with the LOGITECH MODULA-2 system. Compile MOUSE.DEF and MOUSE.MOD, and include the necessary IMPORT list.

1.1.3 Programming Interface for C

The C interface is written in Microsoft Assembler for the Microsoft C compiler using the large model of the 8086. The interface is contained in the file MIF.ASM. It needs to be assembled and the resulting .OBJ object file linked to the application. The interface for other compilers will not differ significantly, but it is advisable to look carefully at the source given in MIF.ASM.

1.1.4 Programming Interface for Microsoft Pascal

The Microsoft Pascal interface is written in Assembler for the Microsoft Pascal compiler using the large memory model of the 8086. The interface is contained in the file PASMIF.ASM. It needs to be assembled and the resulting .OBJ object file linked to the application.

1.1.5 Programming Interface for Turbo Pascal

MOUSE.PAS contains the types, constants, variables, procedures needed to interface the mouse driver from a Turbo Pascal Program. The Pascal code should be included at the start of the Turbo Pascal application program.

1.1.6 Programming Interface From the BASIC Interpreter

It is possible to make a mouse driver function call from a BASIC program running under the IBM BASICA system or the GWBASIC interpreter. The following shows how to make these calls.

Insert an initialization sequence as follows:

```
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      DEF SEG=MSEG
```

Be sure that the statements appear at the beginning of the program before the first call to a mouse function. Then use the CALL statement to make the call.

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

Where MOUSE is the variable containing the BASIC entry offset of the mouse software, and M1%, M2%, M3%, and M4% are the names of the integer variables you have chosen for parameters in this call. They correspond to the values for AX, BX, CX, and DX which are described in this document. As an example:

```
100     'Set minimum and maximum horizontal position to
        '(320,100)
200     M1%=7   'function number is 7
300     M3%=0   'minimum coordinate
400     M4%=639 'maximum coordinate
500     CALL MOUSE(M1%,M2%,M3%,M4%)
```

Note : There are two entry points in the driver, the first is the interrupt entry point (which is the address found in the interrupt vector table entry for interrupt 51) and the second is the BASIC (or FAR CALL) entry point, which is 2 bytes after the interrupt entry point. The second entry point is necessary so that the mouse driver functions can be called as BASIC subroutines using the CALL statement or using a far call from another language.

1.1.7 Notes

Programs which intercept the video i/o software interrupt (int 10h) should exercise utmost care, and the following guidelines should be observed:

- The program should guarantee that video mode changes (function 0) are intercepted by the mouse driver so that the driver can display the correct type of cursor (text or graphics). The mouse driver guarantees consistency of the screen contents to programs that write on the screen via int 10h (or via DOS).

- Some programs write directly into the video memory. If this is the case the program should use function 2 (hide cursor) or 16 (conditional hide cursor) to turn off the cursor before writing to the screen. When finished writing, function 1 (show cursor) should be called to redisplay the cursor.

- To check for the presence in memory of the LOGITECH mouse driver one should check for a "signature" at offset 16 (10H) from the entry point of software interrupt 51 (33H). The signature is the string

  "LOGITECH MOUSE DRIVER".

- The presence of the mouse driver must be checked for by examining the interrupt vector entry for interrupt 51 (33H). Use DOS function 35H (Get Vector) or check directly the offset and segment value at address 0000:00CCH. The vector may be uninitialized (value is zero), in which case the mouse driver has not been loaded and must not be called. If initialized, function 0 of the driver will return a flag if the mouse (or driver) is present.

1.2 THE STANDARD MOUSE DRIVER FUNCTIONS

The following is a list of the mouse driver functions and their assigned number. The number is to be passed in register AX when calling interrupt 51 (33H). Each is described in its own section further in this chapter.

Function Name	Number	
Flag Reset (Mouse Initialization)	0	(00H)
Show Cursor	1	(01H)
Hide Cursor	2	(02H)
Get Mouse Position & Button Status	3	(03H)
Set Mouse Cursor Position	4	(04H)
Get Button Press Information	5	(05H)
Get Button Release Information	6	(06H)
Set Minimum & Maximum X Position	7	(07H)
Set Minimum & Maximum Y Position	8	(08H)
Define Graphic Cursor	9	(09H)
Define Text Cursor	10	(0AH)
Read Mouse Motion Counters	11	(0BH)
Define Event Handler	12	(0CH)
Light Pen Emulation Mode On	13	(0DH)
Light Pen Emulation Mode Off	14	(0EH)
Set Mouse Motion/Pixel Ratio	15	(0FH)
Conditional Hide Cursor	16	(10H)
Set Speed Threshold	19	(13H)

1.2.0 FUNCTION 0 - RESET

This function checks the presence of the mouse, resets it, and initializes the driver. All current features are reset to their initial values. Return parameters are mouse status (0 if the mouse is not connected, -1 if connected) and the number of buttons on the mouse.

Input

AX = 0

Output

AX = mouse status
BX = number of buttons

The initial values are:

Cursor:

invisible
arrow in graphics mode
software cursor in text mode

Event Handler:	NIL
Minimum X Position:	0
Maximum X Position:	639
Minimum Y Position:	0
Maximum Y Position:	199

1.2.0.1 Calling Reset (0) from Assembly Language

After checking that the mouse driver is loaded (interrupt vector not zero), call function 0 and check for the returned value. It is recommended to check for the returned value zero, which indicates that no mouse or driver is present (AX remains zero).

```
;
; Check if the interrupt vector is set correctly
;
        MOV             AL, 33H
        MOV             AH, 35H         ; Get Vector
        INT             21H                     ; From DOS
        CMP             BX, 0           ; Is offset in BX zero?
        JNZ             MOUSE_OK
        MOV             AX, ES
        CMP             AX, 0           ; Is segment in ES zero?
        JNZ             MOUSE_OK
;
; You get here if the mouse interrupt is not initialized
;
        JMP             NO_MOUSE
;
; Now check for the presence of the mouse (and driver), and reset
; the driver and the mouse
;
MOUSE_OK:
        MOV             AX, 0
        INT             33H
        TEST            AX, AX                  ; Zero in AX means no mouse
        JNZ             MOUSE_READY
        JMP             NO_MOUSE
```

1.2.0.2 Calling Reset (0) from Modula-2

Mouse.DriverInstalled already indicates whether the interrupt vector has been initialized.

```
MODULE test;
IMPORT Mouse;
VAR
   mouseStatus : INTEGER;
   noButtons : CARDINAL;

BEGIN
   mouseStatus := 0;
   IF Mouse.DriverInstalled THEN
      Mouse.FlagReset ( mouseStatus, noButtons );
      IF mouseStatus = 0 THEN
        (* Mouse not available *)
      END
   END
END test.
```

1.2.0.3 Calling Reset (0) from C

```
test()
{
   int mouseStatus = 0;
   unsigned noButtons;

   if DriverInstalled()
   {
      FlagReset ( &mouseStatus, &noButtons );
      if ( mouseStatus == 0 )
         { /* mouse not available */ }
   }
}
```

1.2.0.4 Calling Reset (0) from Microsoft Pascal

```
program test;

function DriverInstalled: boolean; EXTERN;
procedure FlagReset( VARS mouseStatus,
                                numberOfButtons: Integer); EXTERN;

var
    mouseStatus, noButtons : Integer;
begin
  if DriverInstalled () then
  begin
     FlagReset ( mouseStatus, noButtons );
     if mouseStatus = 0 then
        (* mouse not available *)
  end
end.
```

1.2.0.5 Calling Reset (0) from Turbo Pascal

```
program test;
{$i mouse.pas}
var
    mouseStatus, noButtons : Integer;

begin
  if DriverInstalled() then
  begin
     FlagReset ( mouseStatus, noButtons );
     if mouseStatus = 0 then
        (* mouse not available *)
  end
end.
```

1.2.0.6 Calling Reset (0) from Basic

```
10  DEF SEG = 0
20  MSEG = 256*PEEK(51*4+3)+PEEK(51*4+2)
30  IF MSEG = 0 THEN 90
40  ENTRY = 256*PEEK(51*4+1)+PEEK(51*4)
50  DEF SEG = MSEG

60  IF PEEK(ENTRY) = 207 THEN 90
70  GOTO 100
90  PRINT "Mouse Driver Not Loaded" : END
100  PRINT "Mouse Driver Loaded OK"

200  MOUSE = ENTRY+2  ' basic entry point
210  M1% = 0
220  CALL MOUSE (M1%,M2%,M3%,M4%)
230  IF M1% = -1 THEN 290
240  PRINT "Mouse not responding" : END
290  ' Mouse is connected and responding - continue
500  ' the rest of your program
```

1.2.1 FUNCTION 1 - SHOW CURSOR

Show Cursor makes the mouse driver cursor visible. The cursor shape is as defined by function 9 (Define Graphics Cursor) or 10 (Define Text Cursor) according to the current video mode.

Refer to function 2 (Hide Cursor) for information on how many calls to Show Cursor are needed to make the cursor visible after calling Hide Cursor.

Input

AX = 1

Output

none

1.2.1.1 Calling Show Cursor (1) from Assembler

```
SHOW_CURSOR:
        MOV     AX, 1
        INT     33H
```

1.2.1.2 Calling Show Cursor (1) from Modula-2

```
MODULE test;
IMPORT Mouse;
BEGIN
  Mouse.ShowCursor;
END test.
```

1.2.1.3 Calling Show Cursor (1) from C

```
test ()
{
  ShowCursor ();
}
```

1.2.1.4 Calling Show Cursor (1) from Microsoft Pascal

```
program test;

procedure ShowCursor; EXTERN;

begin
  ShowCursor;
end.
```

1.2.1.5 Calling Show Cursor (1) from Turbo Pascal

```
program test;
{$i mouse.pas}

begin
  ShowCursor;
end.
```

1.2.1.6 Calling Show Cursor (1) from Basic

```
200  ' assuming set up as in A.2.1
210  M1% = 1
220  CALL MOUSE (M1%, M2%, M3%, M4%)
```

1.2.2 FUNCTION 2 - HIDE CURSOR

This function makes the cursor invisible. Although the cursor is hidden it still tracks the motion of the mouse, changing position as the mouse changes position.

The Hide Cursor function is cumulative. The cursor visibility status can be seen as a counter which is 1 when the cursor is visible, and 0 or negative when the cursor is invisible. Hide Cursor decrements the status, making the cursor invisible when going from 1 to 0. Show Cursor increments the status, with a maximum value of 1. When the status reaches 1, the cursor is made visible. This means that two calls to Hide Cursor will be canceled by two calls to Show Cursor.

Input

AX = 2

Output

none

1.2.2.1 Calling Function 2 from Assembler

```
HIDE_CURSOR:
        MOV     AX, 2
        INT     33H
```

1.2.2.2 Calling Function 2 from Modula-2

```
MODULE test;
IMPORT Mouse;
BEGIN
  Mouse.HideCursor;
END test.
```

1.2.2.3 Calling Function 2 from C

```
test()
{
  HideCursor();
}
```

1.2.2.4 Calling Function 2 from Microsoft Pascal

```
program test;
procedure HideCursor;EXTERN;
begin
  HideCursor;
end.
```

1.2.2.5 Calling Function 2 from Turbo Pascal

```
program test;
{$i mouse.pas}

begin
  HideCursor;
end.
```

1.2.2.6 Calling Function 2 from Basic

```
200  ' assuming set up as in A.2.1
210  M1% = 2
220  CALL MOUSE (M1%, M2%, M3%, M4%)
```

1.2.3 FUNCTION 3 - GET MOUSE POSITION & BUTTON STATUS

This function reports the status of the buttons and the position of the cursor horizontally and vertically. The button status is a single integer value with bit 0 representing the left button, bit 1 the right and bit 2 the middle. When a button is down the corresponding bit is 1 and when a button is up the bit is 0.

Input

AX = 3

Output

BX = button status
CX = horizontal cursor position
DX = vertical cursor position

1.2.3.1 Calling Function 3 from Assembler

```
GET_POS_BUT:
        MOV     AX, 3
        INT     33H
```

1.2.3.2 Calling Function 3 from Modula-2

```
MODULE test;
IMPORT Mouse;
VAR
   buttonSet : Mouse.ButtonSet;
   horizontal, vertical : INTEGER;

BEGIN
   Mouse.GetPosBut (buttonSet, horizontal, vertical);
END test.
```

1.2.3.3 Calling Function 3 from C

```
test()
{
  struct {
     unsigned leftButton   : 1;
     unsigned rightButton  : 1;
     unsigned middleButton : 1;
         } *buttonStatus;

  int *horizontal, *vertical;

  GetPosBut (buttonStatus, horizontal, vertical);
}
```

1.2.3.4 Calling Function 3 from Microsoft Pascal

```
program test;
procedure GetPosBut ( VARS buttonStatus, horizontal, vertical :
                                                    Integer );EXTERN;
var
   buttonStatus, horizontal, vertical : Integer;

begin
  GetPosBut (buttonStatus, horizontal, vertical);
end.
```

1.2.3.5 Calling Function 3 from Turbo Pascal

```
program test;
{$i mouse.pas}
var
  buttonStatus, horizontal, vertical : Integer;
begin
  GetPosBut (buttonStatus, horizontal, vertical);
end.
```

1.2.3.6 Calling Function 3 from Basic

```
200  ' assuming set up as in A.2.1
210  M1% = 3
220  CALL MOUSE (M1%, M2%, M3%, M4%)
230  IF M2% AND 1 THEN PRINT "Left Button Pressed"
240  IF M2% AND 2 THEN PRINT "Right Button Pressed"
250  IF M2% AND 4 THEN PRINT "Middle Button Pressed"
```

1.2.4 FUNCTION 4 - SET MOUSE CURSOR POSITION

This function sets the cursor position to the specified horizontal and vertical location on the screen. The new values must be within the specified ranges for the mode of the screen (see Chapter 3, section 5 for table of coordinates). If you are using a text screen mode, the values are rounded to the nearest values permitted by the screen for horizontal and vertical positions (ie to character boundaries).

Input

AX = 4
CX = new horizontal cursor position
DX = new vertical cursor position

Output

none

1.2.4.1 Calling Function 4 from Assembler

```
SET_CURSOR_POS:
        MOV     AX, 4
        MOV     CX, HORIZ
        MOV     DX, VERT
        INT     33H
```

1.2.4.2 Calling Function 4 from Modula-2

```
MODULE test;
IMPORT Mouse;
VAR
   horizontal, vertical : INTEGER;

BEGIN
   Mouse.SetCursorPos (horizontal, vertical);
END test.
```

1.2.4.3 Calling Function 4 from C

```
test()
{
  int horizontal, vertical;

  SetCursorPos (horizontal, vertical);
}
```

1.2.4.4 Calling Function 4 from Microsoft Pascal

```
program test;

procedure SetCursorPos (horizontal, vertical : Integer);EXTERN;

var
   horizontal, vertical : Integer;

begin
   SetCursorPos (horizontal, vertical);
end.
```

1.2.4.5 Calling Function 4 from Turbo Pascal

```
program test;
{$i mouse.pas}

var
   horizontal, vertical : Integer;

begin
   SetCursorPos (horizontal, vertical);
end.
```

1.2.4.6 Calling Function 4 from Basic

```
200  ' assuming set up as in A.2.1
210  M1% = 4
220  M3% = 0    ' horizontal
230  M4% = 0    ' vertical
240  CALL MOUSE (M1%, M2%, M3%, M4%)
```

1.2.5 FUNCTION 5 - GET BUTTON PRESS INFORMATION

This function returns a count of button presses since the last call to this function. It also returns the horizontal and vertical position of the cursor at the time of the last press of the specified button and the status of the other buttons at that moment.

Input

AX = 5
BX = button of interest (0: Left, 1: Right, 2: Middle)

Output

AX = button status
BX = number of button presses on specified button
CX = horizontal cursor position at last press
DX = vertical cursor position at last press

Notes : The BX register input parameter indicates which button the report is to be made on.
The AX register, on return from the function, contains the current status of all the mouse buttons. Each button is represented by a single bit, 1 => button pressed.

AX : Bit 0 = Left Button
Bit 1 = Right Button
Bit 2 = Middle Button

1.2.5.1 Calling Function 5 from Assembler

```
GET_BUTTON_PRESS:
        MOV     AX, 5
        MOV     BX, BUTTON
        INT     33H
```

1.2.5.2 Calling Function 5 from Modula-2

```
MODULE test;
IMPORT Mouse;
VAR
   button : Mouse.Button;
   buttonStatus : Mouse.ButtonSet;
   buttonPressCount : CARDINAL;
   horizontal, vertical : INTEGER;

BEGIN
   Mouse.GetButPres (button, buttonStatus, buttonPressCount, horizontal, vertical);
END test.
```

1.2.5.3 Calling Function 5 from C

```
test()
{
  int button, buttonStatus, buttonPressCount, horizontal, vertical;

  GetButPres (button, &buttonStatus, &buttonPressCount, &horizontal, &vertical);
}
```

1.2.5.4 Calling Function 5 from Microsoft Pascal

```
program test;
procedure GetButPres (button : Integer;VARS buttonStatus, buttonPressCount, horizontal, vertical :
Integer);EXTERN;

var
  button, buttonStatus, buttonPressCount, horizontal, vertical : Integer;
begin
  GetButPres (button, buttonStatus, buttonPressCount, horizontal, vertical);
end.
```

1.2.5.5 Calling Function 5 from Turbo Pascal

```
program test;
{$i mouse.pas}

var
  button, buttonStatus, buttonPressCount, horizontal, vertical : Integer;
begin
  GetButPres (button, buttonStatus, buttonPressCount, horizontal, vertical);
end.
```

1.2.5.6 Calling Function 5 from Basic

```
200  ' assuming set up as in A.2.1
210  M1% = 5
220  M2% = 1  ' Right Button
230  CALL MOUSE (M1%, M2%, M3%, M4%)
240  IF (M1% AND 2) THEN PRINT "Right button pressed"
```

1.2.6 FUNCTION 6 - GET BUTTON RELEASE INFORMATION

This function returns a count of button releases since the last call to this function. It also returns the horizontal and vertical position of the cursor at the time of the last release of the specified button and the status of the other buttons at that moment.

Input

AX = 6
BX = button of interest (0 :Left, 1 :Right, 2 :Middle)

Output

AX = button status
BX = number of button releases on specified button
CX = horizontal cursor position at last release
DX = vertical cursor position at last press

Input : The BX register input parameter indicates which button the report is to be made on.

Output : The AX register, on return from the function, contains the current status of all the mouse buttons. Each button is represented by a single bit, 1 => button pressed.

AX : Bit 0 = Left Button
Bit 1 = Right Button
Bit 2 = Middle Button

1.2.6.1 Calling Function 6 from Assembler

```
GET_BUT_REL:
        MOV     AX, 6
        MOV     BX, BUTTON
        INT     33H
        MOV     BUTTON_STATUS, AX
        MOV     BUTTON_REL_COUNT, BX
        MOV     HORIZ, CX
        MOV     VERT, DX
```

1.2.6.2 Calling Function 6 from Modula-2

```
MODULE test;
IMPORT Mouse;
VAR
   button : Mouse.Button;
   buttonStatus : Mouse.ButtonSet;
   buttonReleaseCount : CARDINAL;
   horizontal, vertical : INTEGER;

BEGIN
   Mouse.GetButRel (button, buttonStatus, buttonReleaseCount, horizontal, vertical);
END test.
```

1.2.6.3 Calling Function 6 from C

```
test()
{
  int button, buttonStatus, buttonReleaseCount, horizontal, vertical;

  GetButRel (button, &buttonStatus, &buttonReleaseCount, &horizontal, &vertical);
}
```

1.2.6.4 Calling Function 6 from Microsoft Pascal

```
program test;
procedure GetButRel (button : Integer;VARS buttonStatus, buttonReleaseCount,
                                    horizontal, vertical : Integer);EXTERN;

var
  button, buttonStatus, buttonReleaseCount, horizontal, vertical : Integer;
begin
  GetButRel (button, buttonStatus, buttonreleasecount, horizontal, vertical);
end.
```

1.2.6.5 Calling Function 6 from Turbo Pascal

```
program test;
{$i mouse.pas}

var
  button, buttonStatus, buttonReleaseCount, horizontal, vertical : Integer;
begin
  GetButRel (button, buttonStatus, buttonreleasecount, horizontal, vertical);
end.
```

1.2.6.6 Calling Function 6 from Basic

```
200  ' assuming set up as in A.2.1
210  M1% = 6
220  M2% = 2  ' Middle Button
230  CALL MOUSE (M1%, M2%, M3%, M4%)
240  IF (M1% AND 4) THEN PRINT "Middle button released"
```

1.2.7 FUNCTION 7 - SET MINIMUM & MAXIMUM X POSITION

This function sets the minimum and maximum horizontal cursor positions on the screen. All cursor movement is restricted to this area once the area is set.

If the cursor is outside the defined area when the call is made, it moves to just inside the area. If the minimum value is greater than the maximum, the two values are exchanged.

Input

AX = 7
CX = new minimum horizontal cursor position
DX = new maximum horizontal cursor position

Output

none

1.2.7.1 Calling Function 7 from Assembler

```
SET_HORIZONTAL_LIMITS:
      MOV      AX, 7
      MOV      CX, MIN_POS
      MOV      DX, MAX_POS
      INT      33H
```

1.2.7.2 Calling Function 7 from Modula-2

```
MODULE test;
IMPORT Mouse;
VAR
   minPos, maxPos : INTEGER;

BEGIN
   Mouse.SetHorizontalLimits (minPos, maxPos);
END test.
```

1.2.7.3 Calling Function 7 from C

```
test()
int minPos, maxPos;
{
  SetHorizontalLimits (minPos, maxPos);
}
```

1.2.7.4 Calling Function 7 from Microsoft Pascal

```
program test;
procedure SetHorizontalLimits (minPos, maxPos : Integer);EXTERN;
begin
  SetVerticalLimits (minPos, maxPos);
end.
```

1.2.7.5 Calling Function 7 from Turbo Pascal

```
program test;
{$i mouse.pas}

var
  minPos, maxPos : Integer;

begin
  SetHorizontalLimits (minPos, maxPos);
end.
```

1.2.7.6 Calling Function 7 from Basic

```
200  ' assuming set up as in A.2.1
210  M1% = 7
220  M3% = 100  ' Min X value 100
230  M4% = 200  ' Max X value 200
240  CALL MOUSE (M1%, M2%, M3%, M4%)
```

1.2.8 FUNCTION 8 - SET MINIMUM & MAXIMUM Y POSITION

This function sets the minimum and maximum vertical cursor positions on the screen. All cursor movement is restricted to this area once the area is set.

If the cursor is outside the defined area when the call is made, it moves to just inside the area. If the minimum value is greater than the maximum, the two values are exchanged.

Input

AX = 8
CX = new minimum vertical cursor position
DX = new maximum vertical cursor position

Output

none

1.2.8.1 Calling Function 8 from Assembler

```
SET_VERTICAL_LIMITS:
        MOV     AX, 8
        MOV     CX, MIN_POS
        MOV     DX, MAX_POS
        INT     33H
```

1.2.8.2 Calling Function 8 from Modula-2

```
MODULE test;
IMPORT Mouse;
VAR
   minPos, maxPos : INTEGER;

BEGIN
   Mouse.SetVerticalLimits (minPos, maxPos);
END test.
```

1.2.8.3 Calling Function 8 from C

```
test()
int minPos, maxPos;
{
  SetVerticalLimits (minPos, maxPos);
}
```

1.2.8.4 Calling Function 8 from Microsoft Pascal

```
program test;
procedure SetVerticalLimits (minPos, maxPos : Integer);EXTERN;
begin
  SetVerticalLimits (minPos, maxPos);
end.
```

1.2.8.5 Calling Function 8 from Turbo Pascal

```
program test;
{$i mouse.pas}
var
   minPos, maxPos : Integer;
begin
  SetVerticalLimits (minPos, maxPos);
end.
```

1.2.8.6 Calling Function 8 from Basic

```
200  ' assuming set up as in A.2.1
210  M1% = 8
220  M3% = 50  ' Min Y value 50
230  M4% = 150  ' Max Y value 150
240  CALL MOUSE (M1%, M2%, M3%, M4%)
```

1.2.9 FUNCTION 9 - DEFINE GRAPHIC CURSOR

Graphics Cursor

The graphics cursor as displayed on the screen is an array of pixels (16x16 or 8x8 depending on screen mode). It is defined by two 16x16 arrays of bits called the screen mask and the cursor mask.

The screen mask is ANDed with the screen contents. The cursor mask is then XORed with the result.

The operational behavior of these bit arrays are summarized by the following table:

screen mask	cursor mask	resulting screen bit
0	0	0
0	1	1
1	0	UNCHANGED
1	1	INVERTED

Hot Spot : When a mouse function refers to the graphics cursor location it is referring to the point on the screen that lies directly under the cursor's target area. The target area is the point in the cursor block that the mouse software uses to determine the cursor coordinates. This point is generally referred to as the hot spot of the cursor.

Define Graphic Cursor determines what the shape and color of the cursor will be when it is in graphics mode, and it identifies the center of the cursor. The cursor block is a 16 X 16 bit pattern. The cursor hot spot values must be within the range of -16 to 16. They define one pixel inside or near the cursor block.

The default graphics cursor (i.e. before function 9 is called) is an arrow, with the hot spot at the upper left-hand corner.

Input

AX = 9
BX = horizontal cursor hot spot
CX = vertical cursor hot spot
ES:DX = address of screen and cursor mask

Output

none

The screen and cursor mask is made of 32 words (16 bits), the first group of 16 words being the screen mask, and the second group the cursor mask.

1.2.9.1 Calling Function 9 from Assembler

```
SET_GRAPHIC_CURSOR:

Cross_CURSOR:
        DW      -4081, -4081,-4081, 7224, 0, 0
        DW      0, 7224, -4081, -4081, -4081, -1
        DW      -1, -1, -1, -1, 0, 2016
        DW      384, 384, 16770, 32766, 16770, 384
        DW      384, 2016, 0, 0, 0, 0
        DW      0, 0
HOT_SPOT_X EQU  7
HOT_SPOT_Y EQU  5

SET_Cross_CURSOR:
        MOV     AX, 9
        MOV     BX, HOT_SPOT_X
        MOV     CX, HOT_SPOT_Y
        PUSH    DS
        POP     ES
        LEA     DX, Cross_CURSOR
        INT     33H
```

1.2.9.2 Calling Function 9 from Modula-2

```
MODULE test;
IMPORT Mouse;
VAR
   cursor : Mouse.GraphicCursor;
BEGIN
  WITH cursor DO
    screenMask[0] := BITSET(2016); screenMask[1] := BITSET(960);
    screenMask[2] := BITSET(-32383); screenMask[3] := BITSET(-16381);
    screenMask[4] := BITSET(-8185); screenMask[5] := BITSET(-4081);
    screenMask[6] := BITSET(-8185); screenMask[7] := BITSET(-16381);
    screenMask[8] := BITSET(-32383); screenMask[9] := BITSET(960);
    screenMask[10] := BITSET(2016); screenMask[11] := BITSET(-1);
    screenMask[12] := BITSET(-1); screenMask[13] := BITSET(-1);
    screenMask[14] := BITSET(-1); screenMask[15] := BITSET(-1);
    cursorMask[0] := BITSET(0); cursorMask[1] := BITSET(30750);
    cursorMask[2] := BITSET(15420); cursorMask[3] := BITSET(7800);
    cursorMask[4] := BITSET(4080); cursorMask[5] := BITSET(2016);
    cursorMask[6] := BITSET(4080); cursorMask[7] := BITSET(7800);
    cursorMask[8] := BITSET(15420); cursorMask[9] := BITSET(30750);
    cursorMask[10] := BITSET(0); cursorMask[11] := BITSET(0);
    cursorMask[12] := BITSET(0); cursorMask[13] := BITSET(0);
    cursorMask[14] := BITSET(0); cursorMask[15] := BITSET(0);
    hotX := 7; hotY := 5;
  END;
  Mouse.SetGraphicCursor (cursor);
END test.
```

1.2.9.3 Calling Function 9 from C

```
main()
{
 graphicCursor_type XCursor = {
    {4080, 2016, -31807, -15997, -8185, -4081, -2017, -4081,
     -8185, -15997, -31807, 2016, 4080, -1, -1, -1,
     0, 28686, 14364, 7224, 3696, 2016, 960, 2016,
     3696, 7224, 14364, 28686, 0, 0, 0, 0,
     7, 6
                              };

 SetGraphicCursor (&XCursor);
}
```

1.2.9.4 Calling Function 9 from Microsoft Pascal

```
program test;

type
  cursortype = record
                 screenMask,
                 cursorMask : array[0..15] of integer;
                 hotX, hotY : integer;
                end;

procedure SetGraphicCursor (VARS cursor : cursortype);EXTERN;

procedure SetCursorToX;
VAR
  cursor : cursortype;
begin
 with cursor do
 begin
   screenMask[0] := 4080; screenMask[1] := 2016;
   screenMask[2] := -31807; screenMask[3] := -15997;
   screenMask[4] := -8185; screenMask[5] := -4081;
   screenMask[6] := -2017; screenMask[7] := -4081;
   screenMask[8] := -8185; screenMask[9] := -15997;
   screenMask[10] := -31807; screenMask[11] := 2016;
   screenMask[12] := 4080; screenMask[13] := -1;
   screenMask[14] := -1; screenMask[15] := -1;

   cursorMask[0] := 0; cursorMask[1] := 28686;
   cursorMask[2] := 14364; cursorMask[3] := 7224;
   cursorMask[4] := 3696; cursorMask[5] := 2016;
   cursorMask[6] := 960; cursorMask[7] := 2016;
   cursorMask[8] := 3696; cursorMask[9] := 7224;
   cursorMask[10] := 14364; cursorMask[11] := 28686;
   cursorMask[12] := 0; cursorMask[13] := 0;
   cursorMask[14] := 0; cursorMask[15] := 0;

   hotX := 0;
   hotY := 0;
 end; (* with *)
 SetGraphicCursor (cursor);
end;
begin
  SetCursorToX;
end.
```

1.2.9.5 Calling Function 9 from Turbo Pascal

```
program test;

{$i mouse.pas} (* LOGITECH MOUSE Turbo Pascal Interface *)

procedure SetCursorToCross;
const
  CrossCursor : GraphicCursor =
           (screenMask: (-4081, -4081, -4081, 7224, 0, 0, 0, 7224,
             -4081, -4081, -4081, -1, -1, -1, -1, -1);
            cursorMask: (0, 2016, 384, 384, 16770, 32766, 16770, 384,
             384, 2016, 0, 0, 0, 0, 0, 0);
            hotX: 7;
            hotY: 5);
begin
  SetGraphicCursor (Crosscursor);
end;

begin
 HiRes;
 ShowCursor;
 SetCursorToCross;
end.
```

1.2.9.6 Calling Function 9 from Basic

```
190  ' assuming setup as in A.2.1
200  ' Set the cursor to Cross
210  ' (this code generated automatically by the
215  '  LOGITECH MOUSE Cursor Design program (LCD))
220
230  CURSOR%(0,0) = -1
240  CURSOR%(1,0) = -4081
250  CURSOR%(2,0) = -4081
260  CURSOR%(3,0) = -4081
270  CURSOR%(4,0) = 7224
280  CURSOR%(5,0) = 0
290  CURSOR%(6,0) = 0
300  CURSOR%(7,0) = 0
310  CURSOR%(8,0) = 7224
320  CURSOR%(9,0) = -4081
330  CURSOR%(10,0) = -4081
340  CURSOR%(11,0) = -4081
350  CURSOR%(12,0) = -1
360  CURSOR%(13,0) = -1
370  CURSOR%(14,0) = -1
380  CURSOR%(15,0) = -1
390
400  CURSOR%(0,1) = 0
410  CURSOR%(1,1) = 0
420  CURSOR%(2,1) = 2016
430  CURSOR%(3,1) = 384
440  CURSOR%(4,1) = 384
450  CURSOR%(5,1) = 16770
460  CURSOR%(6,1) = 32766
470  CURSOR%(7,1) = 16770
480  CURSOR%(8,1) = 384
490  CURSOR%(9,1) = 384
500  CURSOR%(10,1) = 2016
510  CURSOR%(11,1) = 0
520  CURSOR%(12,1) = 0
530  CURSOR%(13,1) = 0
540  CURSOR%(14,1) = 0
550  CURSOR%(15,1) = 0
560
570  M1% = 9
580  M2% = 7
590  M3% = 6
600  CALL MOUSE (M1%, M2%, M3%, CURSOR%(0,0))
```

1.2.10 FUNCTION 10 - DEFINE TEXT CURSOR

The text cursor is a cursor that can be used when the video is in one of the text modes. Two kinds of text cursors are supported: a hardware text cursor and a software text cursor (for more detailed information on the text cursor see Chapter 3 section 6)

The hardware text cursor is the cursor actually placed on the screen by the video controller itself. It is defined in terms of the scan lines of the character cell numbered from 0 starting from the top scan line. The numbers of scan lines on a character cell depends on the actual video controller and monitor (see the controller documentation for details).

The software text cursor is a character or a character attribute that replaces and/or modifies the character cell on the screen where it is positioned.

The behavior of this cursor is defined by two 16-bit values.

The format of the two values is the following:

BIT	DESCRIPTION
15	blinking (1) or non blinking (0) character
14-12	background color
11	high intensity (1) or medium intensity (0)
10-8	foreground color
7-0	character code

The two values are called the screen mask and the cursor mask. The screen mask is used to determine which of the character attributes are preserved (it is ANDed with the screen character and attribute). The cursor mask is used to determine which of the characteristics are changed by the cursor (it is XORed with the result of the previous operation).

The BX input register selects whether the software or hardware cursor is used by the mouse driver.

If the hardware cursor is selected (BX = 1), the CX and DX input registers should contain the first and last scan lines of the cursor which will be shown on the screen. The range of these scan line values depends on the display adapter of the computer (0..7 for IBM monochrome adapter, 0..14 for IBM CGA - see Chapter 3, section 5 for further details).

If the software text cursor is selected (BX = 0), CX and DX specify the screen and cursor masks.

Input

AX = 10
BX = select cursor (0 -> software cursor, 1 -> hardware cursor)
CX = screen mask value/scan line start
DX = cursor mask value/scan line stop

Output

none

1.2.10.1 Calling Function 10 from Assembler

```
SET_TEXT_CURSOR:
        MOV     AX, 10
        MOV     BX, 0
        MOV     CX, 077FFH
        MOV     DX, 07718H
        INT     33H
```

1.2.10.2 Calling Function 10 from Modula-2

```
PROCEDURE SetTextCursorToArrow;
CONST
     UPARROW = 018H;
     BLANK = 0H;
     SCREENMASKATTR = 077H;
     CURSORMASKATTR = 077H;
     SOFTWARECURSOR = 0H;
BEGIN
     Mouse.SetTextCursor (SOFTWARECURSOR,SCREENMASKATTR*256+BLANK, CURSORMASKATTR*256+UPARROW);
END SetTextCursorToArrow;
```

1.2.10.3 Calling Function 10 from C

```
test()
{
 SetTextCursorToArrow()
 {
  int UPARROW = 0x018;
  int BLANK = 0x0;
  int SCREENMASKATTR = 0x077;
  int CURSORMASKATTR = 0x077;
  int SOFTWARECURSOR = 0x0;

  SetTextCursor (SOFTWARECURSOR, SCREENMASKATTR*256+BLANK,
                          CURSORMASKATTR*256+UPARROW);
 }
}
```

1.2.10.4 Calling Function 10 from Microsoft Pascal

```
program test;
procedure SetTextCursor (selectedCursor,
                         screenMaskORscanStart,
                         cursorMaskORscanStop : Integer
                        );EXTERN;

procedure SetTextCursorToArrow;
const
     UPARROW = 018H;
     BLANK = 0H;
     SCREENMASKATTR = 077H;
     CURSORMASKATTR = 077H;
     SOFTWARECURSOR = 0H;
begin
     SetTextCursor (SOFTWARECURSOR,       SCREENMASKATTR*256+BLANK,
                          CURSORMASKATTR*256+UPARROW);
end;

begin
  SetTextCursorToArrow;
end.
```

1.2.10.5 Calling Function 10 from Turbo Pascal

```
program test;
{$i mouse.pas} (* LOGITECH MOUSE Turbo Pascal Interface *)

procedure SetTextCursorToArrow;
const
     UPARROW = 018H;
     BLANK = 0H;
     SCREENMASKATTR = 077H;
     CURSORMASKATTR = 077H;
     SOFTWARECURSOR = 0H;
begin
     SetTextCursor (SOFTWARECURSOR, SCREENMASKATTR*256+BLANK,
                    CURSORMASKATTR*256+UPARROW);
end;

        begin
          SetTextCursorToArrow;
        end.
```

1.2.10.6 Calling Function 10 from Basic

```
200  ' assuming setup as in A.2.1
210  M1% = 10
220  M2% = 0  ' select software cursor
230  M3% = &H77FF  ' screen mask
240  M4% = &H7718  ' cursor mask
250  CALL MOUSE(M1%, M2%, M3%, M4%)
```

1.2.11 FUNCTION 11 - READ MOUSE MOTION COUNTERS

This function returns the horizontal and vertical step count (the distance the mouse has moved in 1/200 inch increments) since the last call to this function. The step count is always within the range -32768 to 32767. A positive horizontal count specifies a motion from left to right while a positive vertical count specifies a motion towards the user (assuming the mouse is in the conventional orientation, cable pointing away from user). The step count is reset to 0 after the call is completed.

Input

AX = 11

Output

CX = horizontal count
DX = vertical count

Note: In the LOGITECH MOUSE driver there is an accelerator which causes cursor motion to be scaled non-linearly against mouse motion. Acceleration only occurs if the mouse is moved at a speed greater than a threshold value. This accelerator is equivalent to the speed-doubling which some other mouse drivers provide. Without acceleration, the mouse resolution is 200 Steps per Inch (SPI). With fast mouse movement it may rise to 400 SPI. Acceleration is enabled by default. To disable acceleration, call function 19 with a very large speed threshold, say 7FFF Hex.

1.2.11.1 Calling Function 11 from Assembler

```
READ_MOTION_COUNTERS:
        MOV     AX, 11
        INT     33H
        MOV     HORIZ, CX
        MOV     VERT, DX
```

1.2.11.2 Calling Function 11 from Modula-2

```
MODULE test;
IMPORT Mouse;
VAR
   horizontal, vertical : INTEGER;
BEGIN
   Mouse.ReadMotionCounters (horizontal, vertical);
END test.
```

1.2.11.3 Calling Function 11 from C

```
test()
{
 int horizontal, vertical;
 ReadMotionCounters (&horizontal, &vertical);
}
```

1.2.11.4 Calling Function 11 from Microsoft Pascal

```
program test;

procedure ReadMotionCounters (VARS horizontal, vertical : Integer);EXTERN;

var
   horizontal, vertical : Integer;
begin
   ReadMotionCounters (horizontal, vertical);
end.
```

1.2.11.5 Calling Function 11 from Turbo Pascal

```
program test;
{$i mouse.pas} (* LOGITECH MOUSE Turbo Pascal Interface *)
var
   horizontal, vertical : Integer;
begin
   ReadMotionCounters (horizontal, vertical);
end.
```

1.2.11.6 Calling Function 11 from Basic

```
190  ' assuming setup as in A.2.1
200  M1% = 11
210  CALL MOUSE(M1%, M2%, M3%, M4%)
```

1.2.12 FUNCTION 12 - DEFINE EVENT HANDLER

This routine sets the address of a user subroutine to be called by the mouse driver on occurrence of a specified event. The driver temporarily stops execution of your main program and calls the specified subroutine whenever one or more of the conditions defined by the call mask occur. The call mask is a single integer value which defines the conditions which will cause an interrupt. Each bit in the call mask corresponds to a specific condition:

Mask Bit	Condition
0	change cursor position
1	press left button
2	release left button
3	press right button
4	release right button
5	press middle button
6	release middle button
7-15	not used

Input

AX = 12
CX = call mask
ES:DX = address of event-handler routine

Output

none

The event-handler itself must be a piece of code which starts at the address passed to Function 12. When the event handler is called, the following information is available in the registers:

Register	Information
AX	Event which occurred (1 bit set in format of event mask)
BX	Button State
CX	Horizontal Cursor Position
DX	Vertical Cursor Position

Note: That the event handler is called from the mouse driver during an interrupt (typically a serial port interrupt), which means that the event handler must behave as such. For example, it is usually not possible to call DOS from the event handler. It is also strongly recommended to spend as little time as possible in the event handler.

The recommended procedure for an event handler is to record the event in some kind of queue that is read by the main program. The mouse events then can be treated at the same level as keyboard input.

1.2.12.1 Calling Function 12 from Assembler

```
COUNTER label WORD
        DW      0

MY_EVENT_HANDLER:
        PUSH    AX
        MOV     AX, COUNTER
        ADD     AX, 1
        MOV     COUNTER, AX
        POP     AX

SET_EVENT_HANDLER:
        MOV     AX, 12
        MOV     CX, 1
        PUSH    CS
        POP     ES
        LEA     DX, MY_EVENT_HANDLER
        INT     33H
```

1.2.12.2 Calling Function 12 from Modula-2

```
MODULE test;

IMPORT Mouse;

PROCEDURE EventHandler (events : Mouse.EventSet;
                                        buttons : Mouse.ButtonSet;
                                        x, y : INTEGER);
BEGIN
(* code to process the events - if there were many different
   events to process, this procedure could call other procedures
   to handle the different events *)
END EventHandler;

VAR
   eventMask : EventSet;

BEGIN (* module initialization *)
  eventMask := {};       (* initialize event mask to no events *)
  INCL (eventMask, LeftButtonPress);
  (* set event mask so that EventHandler is called only when the left button
                                                                  is pressed *)
  SetEventHandler (eventMask, EventHandler);
  (* main body of program *)
END test.
```

1.2.12.3 Calling Function 12 from C

```
#define FALSE 0
#define TRUE 1

int finished = FALSE;
int quit = 0;

/* the procedure EventHandler is called by the driver asynchronously whenever movement of the
mouse occurs - the button status is then checked. If the right button is pressed, the global
variable 'finished' is set to TRUE, the eventhandler returns and the program terminates.
*/

EventHandler (eventSet, buttonSet, horiz, vert)
int eventSet, buttonSet, horiz, vert;
{
 if (buttonSet == 2) {finished = TRUE; return;}
 /* if right button pressed then quit */
}

test()
{
 SetEventHandler (9, EventHandler); /* event is called when mouse moves */

 finished = FALSE;
 while (finished == FALSE)
{
  /* loop */
 }
}
```

1.2.12.4 Calling Function 12 from Microsoft Pascal

```
program test;
procedure SetEventHandler (mask : Integer; VARS handler : Integer);EXTERN;
(* NB: declaration of the event handler must be as follows - it must be made PUBLIC so that it is
identified by a long (doubleword) address to be called from the mouse driver. Note also that DOS
function calls should not be made from within the event handler - this is because DOS is not re-
entrant. Basically, this means be careful of calls such as Write, WriteLn etc which are usually
translated into DOS calls. *)

procedure MyEventHandler (eventSet, ButtonSet, horiz, vert : Integer)
                                                                    [PUBLIC];
begin
  (* do something *)
end;

var
   eventMask : Integer;
begin
  eventMask := 2;  (* left button presses only *)
  SetEventHandler (eventMask, MyEventHandler);
end.
```

1.2.12.5 Calling Function 12 from Turbo Pascal

```
program test;
{$i mouse.pas} (* LOGITECH MOUSE Turbo Pascal Interface *)

procedure MyEventHandler (eventSet,buttonSet,horiz,vert : Integer);
begin
  (* do something *)
end MyEventHandler;

begin
   SetEventHandler (2 (*left button*), Ofs(MyEventHandler));
end.
```

1.2.12.6 Calling Function 12 from Basic

This function will require different treatment for different implementations of BASIC.

1.2.13 FUNCTION 13 - LIGHT PEN EMULATION MODE ON

This function enables light pen emulation by the mouse. When light pen emulation is enabled, a simultaneous press of the left and right mouse buttons will emulate the pen-down state of the light pen. Both mouse buttons released emulates the pen-up state of the light pen.

Thus, programs which expect a light pen can be run with a mouse instead by first calling function 13. If a light pen is to be used along with a mouse, function 14 will switch off the light pen emulation.

Light pen emulation is OFF by default.

Input

AX = 13

Output

none

1.2.13.1 Calling Function 13 from Assembler

```
LIGHT_PEN_ON:
        MOV     AX, 13
        INT     33H
```

1.2.13.2 Calling Function 13 from Modula-2

```
MODULE test;
IMPORT Mouse;
BEGIN
  Mouse.LightPenOn;
END test.
```

1.2.13.3 Calling Function 13 from C

```
test()
{
  LightPenOn();
}
```

1.2.13.4 Calling Function 13 from Microsoft Pascal

```
program test;

procedure LightPenOn;EXTERN;

begin
  LightPenOn;
end.
```

1.2.13.5 Calling Function 13 from Turbo Pascal

```
program test;
{$i mouse.pas} (* LOGITECH MOUSE Turbo Pascal Interface *)

begin
  LightPenOn;
end.
```

1.2.13.6 Calling Function 13 from Basic

```
190  ' assuming setup as in A.2.1
200  M1% = 13
210  CALL MOUSE(M1%, M2%, M3%, M4)
```

Here is a small test program to illustrate the use of a mouse in place of a light pen. It is written in IBM BASICA. The reference for the light pen functions is the IBM BASIC Reference 3.0.

```
2   REM THIS PROGRAM TESTS THE LIGHT PEN EMULATION OF THE MOUSE DRIVER
3   REM
5   SCREEN 0 : SCREEN 2 ' graphics mode 640x200
10  DEF SEG=0
20  MSEG=256*PEEK(51*4+3)+PEEK(51*4+2)
30  IF MSEG=0 THEN 90
40  ENTRY=256*PEEK(51*4+1)+PEEK(51*4)
50  DEF SEG=MSEG
60  IF PEEK(ENTRY)=207 THEN 90
65  MOUSE=ENTRY+2
70  PRINT "MOUSE DRIVER PRESENT"
80  GOTO 150
90  PRINT "MOUSE DRIVER NOT LOADED"
100 END
150 GOSUB 800  ' mouse driver function 0, Flag Reset
170 IF M1%=-1 THEN 200
180 PRINT "MOUSE NOT RESPONDING" : END
200 PRINT "SETTING LIGHT PEN EMULATION ON"
220 GOSUB 900 ' mouse driver function 13, Light Pen Emulation On
224 GOSUB 600 ' mouse driver function 1, Show Cursor
230 PRINT "TESTING LIGHT PEN EMULATION"
232 PRINT
235 PRINT "RIGHT BUTTON TO EXIT"
236 PRINT "LEFT BUTTON STEPS THROUGH THE PEN FUNCTIONS 0-9"
237 PRINT
240 PEN ON
242 PENFUNCTION=0
245 GOSUB 700 ' mouse driver function 3, Get Position and Buttons
246 IF M2%=0 THEN 245
249 IF M2% AND 2 THEN SCREEN 0 : END  'right button pressed
255 GOSUB 700  ' Get Position and Buttons
260 IF M2% AND 1 THEN 255
270 IF PENFUNCTION < 9 THEN PENFUNCTION = PENFUNCTION+1 : GOTO 245
280 PENFUNCTION = 0 : GOTO 245
490 REM
500 ' SUBROUTINE HIDECURSOR
510 M1%=2
520 CALL MOUSE(M1%, M2%, M3%, M4%)
530 RETURN
540 REM
600 ' SUBROUTINE SHOWCURSOR
610 M1%=1
620 CALL MOUSE(M1%, M2%, M3%, M4%)
630 RETURN
640 REM
```

```
700 ' SUBROUTINE GET POSITION AND BUTTONS
710 M1%=3
720 CALL MOUSE(M1%, M2%, M3%, M4%)
730 RETURN
740 REM
800 ' SUBROUTINE FLAG RESET
810 M1%=0
820 CALL MOUSE(M1%, M2%, M3%, M4%)
830 RETURN
840 REM
900 ' SUBROUTINE LIGHT PEN EMULATION ON
910 M1%=13
920 CALL MOUSE(M1%, M2%, M3%, M4%)
930 RETURN
```

1.2.14 FUNCTION 14 - LIGHT PEN EMULATION MODE OFF

This function disables the light pen emulation mode.

Input

AX = 14

Output

none

1.2.14.1 Calling Function 14 from Assembler

```
LIGHT_PEN_OFF:
        MOV     AX, 14
        INT     33H
```

1.2.14.2 Calling Function 14 from Modula-2

```
MODULE test;
IMPORT Mouse;
BEGIN
  Mouse.LightPenOff;
END test.
```

1.2.14.3 Calling Function 14 from C

```
test()
{
  LightPenOff();
}
```

1.2.14.4 Calling Function 14 from Microsoft Pascal

```
program test;

procedure LightPenOff;EXTERN;

begin
  LightPenOff;
end.
```

1.2.14.5 Calling Function 14 from Turbo Pascal

```
program test;
{$i mouse.pas} (* LOGITECH MOUSE Turbo Pascal Interface *)
begin
  LightPenOff;
end.
```

1.2.14.6 Calling Function 14 from Basic

```
190  ' assuming setup as in A.2.1
200  M1% = 14
210  CALL MOUSE(M1%, M2%, M3%, M4)
```

1.2.15 FUNCTION 15 - SET MOUSE MOTION/PIXEL RATIO

This function sets the mouse motion to screen pixel ratio.

The horizontal and vertical ratios specify the amount of mouse motion which is required to move the cursor 8 pixels. The allowable range of values is 1 to 32767 mickeys where a mickey is defined to be the unit of physical mouse motion (or if you like, hand movement). The value which should be set is dependent on the number of mickeys the mouse records per inch - some mice are 100 mickeys/inch, others 200 or even 320.

The default values are 8 steps to 8 pixels horizontally and 16 steps to 8 pixels vertically. For the LOGITECH MOUSE 200 mickeys/inch, this is equivalent to 3.2 inches of horizontal mouse movement and 2.0 inches of vertical mouse movement to move the cursor over the extent of a 640x200 pixel screen.

Input

AX = 15
CX = horizontal step to pixel ratio
DX = vertical step to pixel ratio

Output

none

1.2.15.1 Calling Function 15 from Assembler

```
SET_MICKEYS_PER_PIXEL:
        MOV     AX, 15
        MOV     CX, 16   ; horizontal
        MOV     DX, 16   ; vertical
        INT     33H
```

1.2.15.2 Calling Function 15 from Modula-2

```
MODULE test;
IMPORT Mouse;
VAR
   horPix, verPix : CARDINAL;
BEGIN
  horPix := 16;
  verPix := 16;
  Mouse.SetMickeysPerPixel (horPix, verPix);
END test.
```

1.2.15.3 Calling Function 15 from C

```
test()
{
  unsigned horPix, verPix;
  horPix = 16;
  verPix = 16;
  SetMickeysPerPixel (horPix, verPix);
}
```

1.2.15.4 Calling Function 15 from Microsoft Pascal

```
program test;
procedure SetMickeysPerPixel (horPix, verPix : Integer);EXTERN;
var
   horPix, verPix : Integer;
begin
  horPix := 16;
  verPix := 16;
  SetMickeysPerPixel (horPix, verPix);
end.
```

1.2.15.5 Calling Function 15 from Turbo Pascal

```
program test;
{$i mouse.pas} (* LOGITECH MOUSE Turbo Pascal Interface *)
var
  horPix, verPix : Integer;
begin
  horPix := 16;
  verPix := 16;
  SetMickeysPerPixel (horPix, verPix);
end.
```

1.2.15.6 Calling Function 15 from Basic

```
190  ' assuming setup as in A.2.1
200  M1% = 15
210  M3% = 16
220  M4% = 16
230  CALL MOUSE(M1%, M2%, M3%, M4%)
```

1.2.16 FUNCTION 16 - CONDITIONAL HIDE CURSOR

This function allows the user to define an area on the screen within which the mouse cursor will automatically be turned off. This function is used to guard a portion of the screen which your program is about to update.

Calling the Show Cursor Function (function 1) - resets the region (i.e. cursor enabled over the whole screen).

Like HideCursor, this function decrements the cursor counter. See ShowCursor and HideCursor for details.

Input

AX = 16
CX = left margin
DX = top margin
SI = right margin
DI = bottom margin

Output

none

1.2.16.1 Calling Function 16 from Assembler

```
CONDITIONAL_OFF:
        MOV     AX, 16
        MOV     CX, 50  ; left margin
        MOV     DX, 60  ; top margin
        MOV     SI, 60   ; right margin
        MOV     DI, 70   ; bottom margin
        INT     33H
```

1.2.16.2 Calling Function 16 from Modula-2

```
MODULE test;
IMPORT Mouse;
BEGIN
  Mouse.ConditionalOff (50, 60, 60, 70);
END test.
```

1.2.16.3 Calling Function 16 from C

```
test()
{
  ConditionalOff (50, 60, 60, 70);
}
```

1.2.16.4 Calling Function 16 from Microsoft Pascal

```
program test;
procedure ConditionalOff (left, top, right, bottom : Integer);EXTERN;
begin
  ConditionalOff (50, 60, 60, 70);
end.
```

1.2.16.5 Calling Function 16 from Turbo Pascal

```
program test;
{$i mouse.pas} (* LOGITECH MOUSE Turbo Pascal Interface *)
begin
  ConditionalOff (50, 60, 60, 70);
end.
```

1.2.16.6 Calling Function 16 from Basic

This function may not generally be used from BASIC due to the number of parameters to be passed.

1.2.19 FUNCTION 19 - SET SPEED THRESHOLD

This function allows the user to define a threshold speed (in units of mouse velocity, mickeys per second) of the mouse above which the driver will add in an acceleration component. This has the effect of allowing a fast movement of the mouse to move the cursor further than a slow movement. The acceleration component varies according to the implementation of the driver. On some drivers it is a constant multiplier, generally with a value of 2, on others it changes according to the increasing speed of the mouse, with multiplier values increasing over an acceleration curve. The Logimouse driver has the latter type of accelerator.

To cancel the acceleration effect, SetSpeedThreshold may be called with a large value for the speed threshold - 07FFFH or some equivalently large number will certainly remove all acceleration. To restore acceleration again, SetSpeedThreshold should be called with a speed threshold value of 0.

Input

AX = 19
DX = threshold speed in mickeys/second

Output

none

1.2.19.1 Calling Function 19 from Assembler

```
SET_SPEED_THRESHOLD:
        MOV     AX, 19
        MOV     DX, 300  ; speed threshold
        INT     33H
```

1.2.19.2 Calling Function 19 from Modula-2

```
MODULE test;
IMPORT Mouse;
BEGIN
  Mouse.SetSpeedThreshold (300);
END test.
```

1.2.19.3 Calling Function 19 from C

```
test()
{
  SetSpeedThreshold (300);
}
```

1.2.19.4 Calling Function 19 from Microsoft Pascal

```
program test;
procedure Setspeedthreshold (threshold : Integer);EXTERN;
begin
  SetSpeedThreshold (300);
end.
```

1.2.19.5 Calling Function 19 from Turbo Pascal

```
program test;
{$i mouse.pas} (* LOGITECH MOUSE Turbo Pascal Interface *)
begin
  SetSpeedThreshold (300);
end.
```

1.2.19.6 Calling Function 19 from Basic

```
190  ' assuming setup as in A.2.1
200  M1% = 19
210  M4% = 300
220  CALL MOUSE(M1%, M2%, M3%, M4%)
```

1.3 EGA FUNCTIONS

The EGA functions are installed as an extension of interrupt 10H. They are installed only if an Enhanced Graphics Adapter board is present on the system. Using these functions gives the application the capability to read back values in write-only EGA registers. In order to assure correct behavior, the application must use these functions exclusively instead of writing directly to the EGA.

The registers are grouped as follow. A reference to an EGA register is made with a pair (group number, register number), or a whole group of registers may be referred to by a group number.

Group	Register	Port	Description
0	0 to 24	3D4	CRT Controller
8	0 to 4	3C4	Sequencer
16	0 to 8	3CE	Graphics Controller
24	0 to 19	3C0	Attribute Controller
32	0	3C2	Output Register
40	0	3DA	Feature Control Register
48	0	3CC	Graphics 1 Position
56	0	3CA	Graphics 2 Position

Calling an EGA function consists of loading the input registers with the appropriate values as listed below and then initiating software interrupt 10 Hex. In 8086 assembly code, the sequence for function 250 is as follows:

```
CALL_FN_250:
        MOV AX, 250
        MOV BX, 0
        INT 10H
```

1.3.1 Read EGA Register - EGA Function 240

Input
AH = 240
BX = Register number
DX = Group number

Output
BL = Value of register

The value of the register indicated by (DX,BX) is returned in BL.

1.3.2 Write EGA Register - EGA Function 241

Input
AH = 241
DX = Group number
Single Register:
BL = Value to be written

Register from a group:
BL = Register number
BH = Value to be written

The register indicated by (DX, BL) is set to the value of BH.

1.3.3 Read EGA Register Group - Function 242

Input
AH = 242
ES:BX = Buffer address
CH = Starting Register Number
CL = Count
DX = Group Number

Output
ES:BX to ES:BX+Count-1 contains the values of the designated registers

The values of the registers from group DX, numbered from CH to CH+CL-1 are copied to the buffer indicated by ES:BX.

1.3.4 Write EGA Register Group - Function 243

Input
AH = 243
ES:BX = Buffer address
CH = Starting Register Number
CL = Count
DX = Group Number

The registers from group DX, numbered from CH to CH+CL-1 are set to the values contained in the buffer indicated by ES:BX.

1.3.5 Read EGA Register List - Function 244

Input
AH = 244
ES:BX = Address of Register List
CX = Count

Output
ES:BX to ES:BX+Count*4 contains the values read

The values of the registers described in the list indicated by ES:BX are transferred to the list. The list is a sequence of records in the form:

```
WORD    Group Number
BYTE    Register Number
BYTE    Value
```

1.3.6 Write EGA Register List - Function 245

Input
AH = 245
ES:BX = Address of Register List
CX = Count

The values contained in the list indicated by ES:BX are transferred to the indicated registers. The list has the same form as for function 244.

1.3.7 Reset to Default Values - Function 246

Input
AH = 246

This function resets the values of all the registers to the values defined using function 247.

1.3.8 Set the Default Values - Function 247

Input
AH = 247
ES:BX = Pointer to a table of one-byte register
values, one for each register of the group
DX = Group Number

This function defines the table of default values used by function 246 for the given group.

1.3.9 EGA Functions Installed - Function 250

Input
AH = 250
BX = 0

Output
BX = 0 if not installed

This function allows the application to detect if the EGA functions are installed. Register BX should be set to 0 before calling the function. If it is still 0 on return, the EGA functions are not installed.

NOTES:

CHAPTER 2
THE LOGIMOUSE CURSOR DESIGNER FOR IBM PC & COMPATIBLES WITH CGA GRAPHICS.

The Logimouse Cursor Designer (LCD) facilitates design of mouse cursors. It produces actual code to integrate in your application and is language independent. This is accomplished by means of loadable skeletons (templates) which define the structure of the output. Skeletons for several popular languages are provided. Customizing or creating your own skeleton files is simple.

2.1 FILES

The LCD program is the file *LCD.EXE*

The skeleton files are called *SKEL.XXX* where XXX is a filename extension denoting the language. Thus *SKEL.C* is a skeleton for the C language. The output is simple enough that in most cases it is compatible with various implementations of a language.

2.2 USING LCD

Type LCD to run the program.

LCD need the mouse driver to work and so this should be loaded first by typing MOUSE. If you try to run LCD without the mouse driver you will get the following error message:

```
mouse driver not loaded
```

2.3 THE MOUSE BUTTONS

Left Button	paints a mask cell
Right Button	clears a mask cell
Middle Button	sets the hot spot (the place on the cursor which dictates exactly where the cursor is actually pointing)

2.4 THE ICONS

- **Load Skeleton**

  Loads a skeleton file (SKEL.XXX). A directory of skeleton files available in the current directory is displayed. The default skeleton is SKEL.MOD (it is loaded automatically at the start).

- **Save Cursor**

  Saves cursor to a file in the format given by the current skeleton. Prompts for a name for the cursor. The file is created with the given name and an extension is added according to the current skeleton (eg if the cursor is given the name 'CROSS' and the current skeleton is 'SKEL.PAS', the output file will be called 'CROSS.PAS').

- **Arrow**

  Resets the cursor to an arrow.

- **Install Cursor**

  Installs the designed cursor as the current mouse cursor.

- **Load Cursor**

  Loads a cursor file for editing. Prompts for cursor name. Only the name of the cursor should be given and NOT the file extension (this is generated automatically from the current skeleton loaded). LCD uses the format of the loaded skeleton (default is Modula-2) to interpret the cursor file. The cursor file must have been created from the loaded skeleton file, otherwise LCD will not be able to interpret it and the message

  ```
  Cursor does not match skeleton
  ```

  will appear.

  When a cursor has been successfully loaded, a different skeleton may be loaded and the cursor saved in the new language format. Thus, if a cursor has been saved in C and it is required in Assembler, the C skeleton should be loaded, then the required cursor loaded followed by the Assembler skeleton and finally the cursor saved.

- **Make S From C**

  Attempts to automatically generate a reasonable screen mask from the cursor mask. It does this by copying the cursor mask to the screen mask, inverting the screen mask and then 'padding-out' the outline. The screen mask may require some final editing to achieve the desired effect.

- **Invert C**

  Inverts the cursor mask

- **Invert S**

  Inverts the screen mask

- **Clear C**

  Clears the cursor mask

- **Clear S**

  Clears the screen mask

2.5 MAKING SKELETONS FOR OTHER LANGUAGES

Skeleton files can be defined for other languages and you may wish to modify the existing ones. The structure of the skeleton file is very informal. The skeleton represents the code of the particular language being used. In a skeleton, the actual values of the cursor fields are replaced by variables, denoted by surrounding percentage signs, for example %P% denotes the cursor name, %HX% denotes the x coordinate of the hot spot. The LCD program reads the skeleton file and when it writes a cursor file, it copies the skeleton back out to the output file but substitutes the values of the designed cursor for the variables, for example, %P% might become MY_CURSOR, %HX% might become 7.

Note: Be careful when using the name variable *%P%*. It must be followed by a non-alphabetic character such as *'yy_yy'* or 'yy;yy' etc. Names like *%P%_Cursor* or *SetCursorTo%P%* are legal but not *%P%Cursor* or *Set%P%Cursor.*

2.5.1 LCD Skeleton Variables

%P% The name of the cursor.

%S0% The fields of the screen mask.

.

.

%S15%

%C0% The fields of the cursor mask.

.

.

%C15%

%HX% The x coordinate of the cursor hot spot.

%HY% The y coordinate of the cursor hot spot.

2.5.2 Example 1 : The Skeleton for LOGITECH MODULA-2

```
PROCEDURE SetCursorTo%P%;

(* the use of the LOGITECH MODULA-2 interface to the mouse driver is assumed *)

VAR
   cursor : Mouse.GraphicCursor;

BEGIN
  WITH cursor DO
    screenMask[0] := BITSET(%S0%); screenMask[1] := BITSET(%S1%);
    screenMask[2] := BITSET(%S2%); screenMask[3] := BITSET(%S3%);
    screenMask[4] := BITSET(%S4%); screenMask[5] := BITSET(%S5%);
    screenMask[6] := BITSET(%S6%); screenMask[7] := BITSET(%S7%);
    screenMask[8] := BITSET(%S8%); screenMask[9] := BITSET(%S9%);
    screenMask[10]:= BITSET(%S10%);screenMask[11]:= BITSET(%S11%);
    screenMask[12]:= BITSET(%S12%);screenMask[13]:= BITSET(%S13%);
    screenMask[14]:= BITSET(%S14%);screenMask[15]:= BITSET(%S15%);
    cursorMask[0] := BITSET(%C0%); cursorMask[1] := BITSET(%C1%);
    cursorMask[2] := BITSET(%C2%); cursorMask[3] := BITSET(%C3%);
    cursorMask[4] := BITSET(%C4%); cursorMask[5] := BITSET(%C5%);
    cursorMask[6] := BITSET(%C6%); cursorMask[7] := BITSET(%C7%);
    cursorMask[8] := BITSET(%C8%); cursorMask[9] := BITSET(%C9%);
    cursorMask[10]:= BITSET(%C10%);cursorMask[11]:= BITSET(%C11%);
    cursorMask[12]:= BITSET(%C12%);cursorMask[13]:= BITSET(%C13%);
    cursorMask[14]:= BITSET(%C14%);cursorMask[15]:= BITSET(%C15%);
    hotX := %HX%; hotY := %HY%;
  END;
  Mouse.SetGraphicCursor (cursor);
END SetCursorTo%P%;
```

When this skeleton is loaded, a cursor designed and called "Cross", the resultant output file "CROSS.MOD" contains the following:

```
PROCEDURE SetCursorToCross;
(* the use of the LOGITECH MODULA-2 interface to the mouse driver is assumed *)

VAR
   cursor : Mouse.GraphicCursor;
BEGIN
  WITH cursor DO
    screenMask[0] := BITSET(-1); screenMask[1] := BITSET(-1);
    screenMask[2] := BITSET(-1); screenMask[3] := BITSET(-2017);
    screenMask[4] := BITSET(-2017);screenMask[5] :=BITSET(-2017);
    screenMask[6] := BITSET(0);  screenMask[7] :=BITSET(0);
    screenMask[8] := BITSET(0);  screenMask[9] :=BITSET(0);
    screenMask[10]:= BITSET(-2017);screenMask[11]:=BITSET(-2017);
    screenMask[12]:= BITSET(-2017);screenMask[13]:=BITSET(-1);
    screenMask[14]:= BITSET(-1); screenMask[15]:=BITSET(-1);
    cursorMask[0] := BITSET(0);  cursorMask[1] :=BITSET(0);
    cursorMask[2] := BITSET(0);  cursorMask[3] :=BITSET(0);
    cursorMask[4] := BITSET(960);cursorMask[5] :=BITSET(960);
    cursorMask[6] := BITSET(960);cursorMask[7] :=BITSET(32766);
    cursorMask[8] := BITSET(32766);cursorMask[9] :=BITSET(960);
    cursorMask[10]:= BITSET(960);cursorMask[11]:=BITSET(960);
    cursorMask[12]:= BITSET(0);  cursorMask[13]:=BITSET(0);
    cursorMask[14]:= BITSET(0);  cursorMask[15]:=BITSET(0);
    hotX := 7; hotY := 8;
  END;
  Mouse.SetGraphicCursor (cursor);
END SetCursorToCross;
```

2.5.3 Example 2 : The Skeleton for Microsoft C

```
graphicCursor_type %P%Cursor =
    {%S0%, %S1%, %S2%, %S3%, %S4%, %S5%, %S6%, %S7%,
     %S8%, %S9%, %S10%, %S11%, %S12%, %S13%, %S14%, %S15%,
     %C0%, %C1%, %C2%, %C3%, %C4%, %C5%, %C6%, %C7%,
     %C8%, %C9%, %C10%, %C11%, %C12%, %C13%, %C14%, %C15%,
     %HX%, %HY% };
```

When this skeleton is loaded, a cursor designed and called "Cross", the resultant output file "CROSS.C" contains the following:

```
graphicCursor_type CrossCursor =
    {-1, -1, -1, -2017, -2017, -2017, 0, 0,
     0, 0, -2017, -2017, -2017, -1, -1, -1,
     0, 0, 0, 0, 960, 960, 960, 32766,
     32766, 960, 960, 960, 0, 0, 0, 0,
     7, 8 };
```