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
# TABLE OF CONTENTS

## LOGITECH MOUSE DRIVER FUNCTIONS

### 1.1 PROGRAM INTERFACE
- 1.1.1 Programming Interface for Assembler
- 1.1.2 Programming Interface for Modula-2
- 1.1.3 Programming Interface for C
- 1.1.4 Programming Interface for Microsoft Pascal
- 1.1.5 Programming Interface for Turbo Pascal
- 1.1.6 Programming Interface From the BASIC Interpreter
- 1.1.7 Notes

### 1.2 THE STANDARD MOUSE DRIVER FUNCTIONS

#### 1.2.0 FUNCTION 0 - RESET
- 1.2.0.1 Calling Reset (0) from Assembly Language
- 1.2.0.2 Calling Reset (0) from Modula-2
- 1.2.0.3 Calling Reset (0) from C
- 1.2.0.4 Calling Reset (0) from Microsoft Pascal
- 1.2.0.5 Calling Reset (0) from Turbo Pascal
- 1.2.0.6 Calling Reset (0) from Basic

#### 1.2.1 FUNCTION 1 - SHOW CURSOR
- 1.2.1.1 Calling Show Cursor (1) from Assembler
- 1.2.1.2 Calling Show Cursor (1) from Modula-2
- 1.2.1.3 Calling Show Cursor (1) from C
- 1.2.1.4 Calling Show Cursor (1) from Microsoft Pascal
- 1.2.1.5 Calling Show Cursor (1) from Turbo Pascal
- 1.2.1.6 Calling Show Cursor (1) from Basic

#### 1.2.2 FUNCTION 2 - HIDE CURSOR
- 1.2.2.1 Calling Function 2 from Assembler
- 1.2.2.2 Calling Function 2 from Modula-2
- 1.2.2.3 Calling Function 2 from C
- 1.2.2.4 Calling Function 2 from Microsoft Pascal
- 1.2.2.5 Calling Function 2 from Turbo Pascal
- 1.2.2.6 Calling Function 2 from Basic

#### 1.2.3 FUNCTION 3 - GET MOUSE POSITION & BUTTON STATUS
- 1.2.3.1 Calling Function 3 from Assembler

#### 1.2.3.2 CALLING FUNCTION 3 FROM MODULA-2
- 1.2.3.3 Calling Function 3 from C
- 1.2.3.4 Calling Function 3 from Microsoft Pascal
- 1.2.3.5 Calling Function 3 from Turbo Pascal
- 1.2.3.6 Calling Function 3 from Basic
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.2.4</td>
<td>FUNCTION 4 - SET MOUSE CURSOR POSITION</td>
<td>16</td>
</tr>
<tr>
<td>1.2.4.1</td>
<td>Calling Function 4 from Assembler</td>
<td>16</td>
</tr>
<tr>
<td>1.2.4.2</td>
<td>Calling Function 4 from Modula-2</td>
<td>16</td>
</tr>
<tr>
<td>1.2.4.3</td>
<td>Calling Function 4 from C</td>
<td>16</td>
</tr>
<tr>
<td>1.2.4.4</td>
<td>Calling Function 4 from Microsoft Pascal</td>
<td>17</td>
</tr>
<tr>
<td>1.2.4.5</td>
<td>Calling Function 4 from Turbo Pascal</td>
<td>17</td>
</tr>
<tr>
<td>1.2.4.6</td>
<td>Calling Function 4 from Basic</td>
<td>17</td>
</tr>
<tr>
<td>1.2.5</td>
<td>FUNCTION 5 - GET BUTTON PRESS INFORMATION</td>
<td>18</td>
</tr>
<tr>
<td>1.2.5.1</td>
<td>Calling Function 5 from Assembler</td>
<td>18</td>
</tr>
<tr>
<td>1.2.5.2</td>
<td>Calling Function 5 from Modula-2</td>
<td>18</td>
</tr>
<tr>
<td>1.2.5.3</td>
<td>Calling Function 5 from C</td>
<td>19</td>
</tr>
<tr>
<td>1.2.5.4</td>
<td>Calling Function 5 from Microsoft Pascal</td>
<td>19</td>
</tr>
<tr>
<td>1.2.5.5</td>
<td>Calling Function 5 from Turbo Pascal</td>
<td>19</td>
</tr>
<tr>
<td>1.2.5.6</td>
<td>Calling Function 5 from Basic</td>
<td>19</td>
</tr>
<tr>
<td>1.2.6</td>
<td>FUNCTION 6 - GET BUTTON RELEASE INFORMATION</td>
<td>20</td>
</tr>
<tr>
<td>1.2.6.1</td>
<td>Calling Function 6 from Assembler</td>
<td>20</td>
</tr>
<tr>
<td>1.2.6.2</td>
<td>Calling Function 6 from Modula-2</td>
<td>20</td>
</tr>
<tr>
<td>1.2.6.3</td>
<td>Calling Function 6 from C</td>
<td>21</td>
</tr>
<tr>
<td>1.2.6.4</td>
<td>Calling Function 6 from Microsoft Pascal</td>
<td>21</td>
</tr>
<tr>
<td>1.2.6.5</td>
<td>Calling Function 6 from Turbo Pascal</td>
<td>21</td>
</tr>
<tr>
<td>1.2.6.6</td>
<td>Calling Function 6 from Basic</td>
<td>21</td>
</tr>
<tr>
<td>1.2.7</td>
<td>FUNCTION 7 - SET MINIMUM &amp; MAXIMUM X POSITION</td>
<td>22</td>
</tr>
<tr>
<td>1.2.7.1</td>
<td>Calling Function 7 from Assembler</td>
<td>22</td>
</tr>
<tr>
<td>1.2.7.2</td>
<td>Calling Function 7 from Modula-2</td>
<td>22</td>
</tr>
<tr>
<td>1.2.7.3</td>
<td>Calling Function 7 from C</td>
<td>22</td>
</tr>
<tr>
<td>1.2.7.4</td>
<td>Calling Function 7 from Microsoft Pascal</td>
<td>22</td>
</tr>
<tr>
<td>1.2.7.5</td>
<td>Calling Function 7 from Turbo Pascal</td>
<td>23</td>
</tr>
<tr>
<td>1.2.7.6</td>
<td>Calling Function 7 from Basic</td>
<td>23</td>
</tr>
<tr>
<td>1.2.8</td>
<td>FUNCTION 8 - SET MINIMUM &amp; MAXIMUM Y POSITION</td>
<td>24</td>
</tr>
<tr>
<td>1.2.8.1</td>
<td>Calling Function 8 from Assembler</td>
<td>24</td>
</tr>
<tr>
<td>1.2.8.2</td>
<td>Calling Function 8 from Modula-2</td>
<td>24</td>
</tr>
<tr>
<td>1.2.8.3</td>
<td>Calling Function 8 from C</td>
<td>24</td>
</tr>
<tr>
<td>1.2.8.4</td>
<td>Calling Function 8 from Microsoft Pascal</td>
<td>24</td>
</tr>
<tr>
<td>1.2.8.5</td>
<td>Calling Function 8 from Turbo Pascal</td>
<td>25</td>
</tr>
<tr>
<td>1.2.8.6</td>
<td>Calling Function 8 from Basic</td>
<td>25</td>
</tr>
<tr>
<td>1.2.9</td>
<td>FUNCTION 9 - DEFINE GRAPHIC CURSOR</td>
<td>26</td>
</tr>
<tr>
<td>1.2.9.1</td>
<td>Calling Function 9 from Assembler</td>
<td>27</td>
</tr>
<tr>
<td>1.2.9.2</td>
<td>Calling Function 9 from Modula-2</td>
<td>27</td>
</tr>
<tr>
<td>1.2.9.3</td>
<td>Calling Function 9 from C</td>
<td>28</td>
</tr>
<tr>
<td>1.2.9.4</td>
<td>Calling Function 9 from Microsoft Pascal</td>
<td>29</td>
</tr>
<tr>
<td>1.2.9.5</td>
<td>Calling Function 9 from Turbo Pascal</td>
<td>30</td>
</tr>
<tr>
<td>1.2.9.6</td>
<td>Calling Function 9 from Basic</td>
<td>31</td>
</tr>
<tr>
<td>1.2.10</td>
<td>FUNCTION 10 - DEFINE TEXT CURSOR</td>
<td>32</td>
</tr>
<tr>
<td>1.2.10.1</td>
<td>Calling Function 10 from Assembler</td>
<td>33</td>
</tr>
<tr>
<td>1.2.10.2</td>
<td>Calling Function 10 from Modula-2</td>
<td>33</td>
</tr>
<tr>
<td>1.2.10.3</td>
<td>Calling Function 10 from C</td>
<td>33</td>
</tr>
<tr>
<td>1.2.10.4</td>
<td>Calling Function 10 from Microsoft Pascal</td>
<td>34</td>
</tr>
<tr>
<td>1.2.10.5</td>
<td>Calling Function 10 from Turbo Pascal</td>
<td>34</td>
</tr>
<tr>
<td>1.2.10.6</td>
<td>Calling Function 10 from Basic</td>
<td>35</td>
</tr>
</tbody>
</table>
1.2.11 FUNCTION 11 - READ MOUSE MOTION COUNTERS

1.2.11.1 Calling Function 11 from Assembler

1.2.11.2 Calling Function 11 from Modula-2

1.2.11.3 Calling Function 11 from C

1.2.11.4 Calling Function 11 from Microsoft Pascal

1.2.11.5 Calling Function 11 from Turbo Pascal

1.2.11.6 Calling Function 11 from Basic

1.2.12 FUNCTION 12 - DEFINE EVENT HANDLER

1.2.12.1 Calling Function 12 from Assembler

1.2.12.2 Calling Function 12 from Modula-2

1.2.12.3 Calling Function 12 from C

1.2.12.4 Calling Function 12 from Microsoft Pascal

1.2.12.5 Calling Function 12 from Turbo Pascal

1.2.12.6 Calling Function 12 from Basic

1.2.13 FUNCTION 13 - LIGHT PEN EMULATION MODE ON

1.2.13.1 Calling Function 13 from Assembler

1.2.13.2 Calling Function 13 from Modula-2

1.2.13.3 Calling Function 13 from C

1.2.13.4 Calling Function 13 from Microsoft Pascal

1.2.13.5 Calling Function 13 from Turbo Pascal

1.2.13.6 Calling Function 13 from Basic

1.2.14 FUNCTION 14 - LIGHT PEN EMULATION MODE OFF

1.2.14.1 Calling Function 14 from Assembler

1.2.14.2 Calling Function 14 from Modula-2

1.2.14.3 Calling Function 14 from C

1.2.14.4 Calling Function 14 from Microsoft Pascal

1.2.14.5 Calling Function 14 from Turbo Pascal

1.2.14.6 Calling Function 14 from Basic

1.2.15 FUNCTION 15 - SET MOUSE MOTION/PIXEL RATIO

1.2.15.1 Calling Function 15 from Assembler

1.2.15.2 Calling Function 15 from Modula-2

1.2.15.3 Calling Function 15 from C

1.2.15.4 Calling Function 15 from Microsoft Pascal

1.2.15.5 Calling Function 15 from Turbo Pascal

1.2.15.6 Calling Function 15 from Basic

1.2.16 FUNCTION 16 - CONDITIONAL HIDE CURSOR

1.2.16.1 Calling Function 16 from Assembler

1.2.16.2 Calling Function 16 from Modula-2

1.2.16.3 Calling Function 16 from C

1.2.16.4 Calling Function 16 from Microsoft Pascal

1.2.16.5 Calling Function 16 from Turbo Pascal

1.2.16.6 Calling Function 16 from Basic

1.2.19 FUNCTION 19 - SET SPEED THRESHOLD

1.2.19.1 Calling Function 19 from Assembler

1.2.19.2 Calling Function 19 from Modula-2

1.2.19.3 Calling Function 19 from C

1.2.19.4 Calling Function 19 from Microsoft Pascal

1.2.19.5 Calling Function 19 from Turbo Pascal

1.2.19.6 Calling Function 19 from Basic
1.3 EGA FUNCTIONS
  1.3.1 Read EGA Register 54
  1.3.2 Write EGA Register 55
  1.3.3 Read EGA Register Group 55
  1.3.5 Read EGA Register List 56
  1.3.6 Write EGA Register List 56
  1.3.7 Reset to Default Values 56
  1.3.8 Set the Default Values 56
  1.3.9 EGA Functions Installed 57

CHAPTER 2
THE LOGIMOUSE CURSOR DESIGNER 59

  2.1 FILES 59
  2.2 USING LCD 59
  2.3 THE MOUSE BUTTONS 59
  2.4 THE ICONS 60
  2.5 MAKING SKELETONS FOR OTHER LANGUAGES 62
    2.5.1 LCD Skeleton Variables 62
    2.5.2 Example 1
      The Skeleton for LOGITECH MODULA-2 63
    2.5.3 Example 2
      The Skeleton for Microsoft C 64
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(M1X,M2X,M3%,M4X)
```

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.

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

\[
AX = \text{mouse status} \\
BX = \text{number of buttons}
\]

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).

```assembly
; ; 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.

```modula2
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
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

```pascal
program test;
{si mouse.pas)

begin
  ShowCursor;
end.
```

1.2.1.6 Calling Show Cursor (1) from Basic

```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 testi
IMPORT Mousei
BEGIN
    Mouse.HideCursori
END testi.
```

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

```pascal
program test;
{$i mouse.pas}
begin
  HideCursor;
end.
```

1.2.2.6 Calling Function 2 from Basic

```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

```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

```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

```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);
}
Function 4 - Set Cursor Position

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**

\[
\begin{align*}
AX & = 5 \\
BX & = \text{button of interest} \ (0: \text{Left}, 1: \text{Right}, 2: \text{Middle})
\end{align*}
\]

**Output**

\[
\begin{align*}
AX & = \text{button status} \\
BX & = \text{number of button presses on specified button} \\
CX & = \text{horizontal cursor position at last press} \\
DX & = \text{vertical cursor position at last press}
\end{align*}
\]

**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.

\[
\begin{align*}
AX: & \quad \text{Bit 0 = Left Button} \\
& \quad \text{Bit 1 = Right Button} \\
& \quad \text{Bit 2 = Middle Button}
\end{align*}
\]

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

\[
\begin{align*}
AX &= 6 \\
BX &= \text{button of interest} \ (0: \text{Left}, 1: \text{Right}, 2: \text{Middle})
\end{align*}
\]

Output

\[
\begin{align*}
AX &= \text{button status} \\
BX &= \text{number of button releases on specified button} \\
CX &= \text{horizontal cursor position at last release} \\
DX &= \text{vertical cursor position at last press}
\end{align*}
\]

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.

\[
\begin{align*}
\text{AX: } & \text{ Bit 0 = Left Button} \\
& \text{ Bit 1 = Right Button} \\
& \text{ Bit 2 = Middle Button}
\end{align*}
\]

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;
begi
genVerticalLimits (minPos, maxPos);
ed.
1.2.7.5 Calling Function 7 from Turbo Pascal

```pascal
program test;
{#i mouse.pas}
var
    minPos, maxPos : integer;
begin
    SetHorizontalLimits (minPos, maxPos);
end.
```

1.2.7.6 Calling Function 7 from Basic

```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.SetFloatVerticalLimits (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:

<table>
<thead>
<tr>
<th>screen mask</th>
<th>cursor mask</th>
<th>resulting screen bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>UNCHANGED</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>INVERTED</td>
</tr>
</tbody>
</table>

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[8] := BITSET(-32383); screenMask[9] := BITSET(960);
    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[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,
        7, 6
    };

    SetGraphicCursor (&XCursor);
}
1.2.9.4 Calling Function 9 from Microsoft Pascal

```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
    cursorMask[0] := 0; cursorMask[1] := 28686;

    hotX := 0;
    hotY := 0;
  end; (* with *)
  SetGraphicCursor (cursor);
end;
begin
  SetCursorToX;
end.
```
30

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:

<table>
<thead>
<tr>
<th>BIT</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>blinking (1) or non blinking (0) character</td>
</tr>
<tr>
<td>14-12</td>
<td>background color</td>
</tr>
<tr>
<td>11</td>
<td>high intensity (1) or medium intensity (0)</td>
</tr>
<tr>
<td>10-8</td>
<td>foreground color</td>
</tr>
<tr>
<td>7-0</td>
<td>character code</td>
</tr>
</tbody>
</table>

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
Function 10 - Define Text Cursor

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 = 018;
    BLANK = 0;
    SCREENMASKATTR = 077;
    CURSORMASKATTR = 077;
    SOFTWARECURSOR = 0;
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

```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

```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**

\[ CX = \text{horizontal count} \]
\[ DX = \text{vertical count} \]

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

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

1.2.11.2 Calling Function 11 from Modula-2

```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

```c
void test()
{
    int horizontal, vertical;
    ReadMotionCounters (&horizontal, &vertical);
}
```
1.2.11.4 Calling Function 11 from Microsoft Pascal

```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

```pascal
program test;
{Si 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

```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:

<table>
<thead>
<tr>
<th>Mask Bit</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>change cursor position</td>
</tr>
<tr>
<td>1</td>
<td>press left button</td>
</tr>
<tr>
<td>2</td>
<td>release left button</td>
</tr>
<tr>
<td>3</td>
<td>press right button</td>
</tr>
<tr>
<td>4</td>
<td>release right button</td>
</tr>
<tr>
<td>5</td>
<td>press middle button</td>
</tr>
<tr>
<td>6</td>
<td>release middle button</td>
</tr>
<tr>
<td>7-15</td>
<td>not used</td>
</tr>
</tbody>
</table>

**Input**

\[ AX = 12 \]
\[ CX = \text{call mask} \]
\[ ES:DX = \text{address of event-handler routine} \]

**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:

<table>
<thead>
<tr>
<th>Register</th>
<th>Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>AX</td>
<td>Event which occurred (1 bit set in format of event mask)</td>
</tr>
<tr>
<td>BX</td>
<td>Button State</td>
</tr>
<tr>
<td>CX</td>
<td>Horizontal Cursor Position</td>
</tr>
<tr>
<td>DX</td>
<td>Vertical Cursor Position</td>
</tr>
</tbody>
</table>

**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.
Function 12 - Define Event Handler

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

```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

```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

```pascal
program test;
{Si 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

```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.

```basica
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
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 HIDE_CURSOR
510 M1%=2
520 CALL MOUSE(M1%, M2%, M3%, M4%)
530 RETURN
540 REM
600 ' SUBROUTINE SHOW_CURSOR
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)
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

Calling Function 15 from Assembler

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

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.
```

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

```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

```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

```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

\[
\begin{align*}
AX &= 16 \\
CX &= \text{left margin} \\
DX &= \text{top margin} \\
SI &= \text{right margin} \\
DI &= \text{bottom margin}
\end{align*}
\]

Output

none

1.2.16.1 Calling Function 16 from Assembler

\[
\text{CONDITONAL\_OFF:} \\
\begin{align*}
\text{MOV} & \quad AX, 16 \\
\text{MOV} & \quad CX, 50 \; ; \text{left margin} \\
\text{MOV} & \quad DX, 60 \; ; \text{top margin} \\
\text{MOV} & \quad SI, 60 \; ; \text{right margin} \\
\text{MOV} & \quad DI, 70 \; ; \text{bottom margin} \\
\text{INT} & \quad 33H
\end{align*}
\]

1.2.16.2 Calling Function 16 from Modula-2

\[
\text{MODULE test;} \\
\text{IMPORT Mouse;} \\
\text{BEGIN} \\
\quad \text{Mouse.ConditionOff} \; (50, 60, 60, 70); \\
\text{END test.}
\]

1.2.16.3 Calling Function 16 from C

\[
\text{test()} \\
\quad \{ \\
\quad \quad \text{ConditionalOff} \; (50, 60, 60, 70); \\
\quad \}
\]
1.2.16.4 Calling Function 16 from Microsoft Pascal

```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

```pascal
program test;
{si 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

```pascal
program test;

(* 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.

<table>
<thead>
<tr>
<th>Group</th>
<th>Register</th>
<th>Port</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0 to 24</td>
<td>3D4</td>
<td>CRT Controller</td>
</tr>
<tr>
<td>8</td>
<td>0 to 4</td>
<td>3C4</td>
<td>Sequencer</td>
</tr>
<tr>
<td>16</td>
<td>0 to 8</td>
<td>3CE</td>
<td>Graphics Controller</td>
</tr>
<tr>
<td>24</td>
<td>0 to 19</td>
<td>3C0</td>
<td>Attribute Controller</td>
</tr>
<tr>
<td>32</td>
<td>0</td>
<td>3C2</td>
<td>Output Register</td>
</tr>
<tr>
<td>40</td>
<td>0</td>
<td>3DA</td>
<td>Feature Control Register</td>
</tr>
<tr>
<td>48</td>
<td>0</td>
<td>3CC</td>
<td>Graphics 1 Position</td>
</tr>
<tr>
<td>56</td>
<td>0</td>
<td>3CA</td>
<td>Graphics 2 Position</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>WORD</th>
<th>Group Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>BYTE</td>
<td>Register Number</td>
</tr>
<tr>
<td>BYTE</td>
<td>Value</td>
</tr>
</tbody>
</table>

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:
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 \textit{LCD.EXE}

The skeleton files are called \textit{SKEL.xxx} where \textit{XXX} is a filename extension denoting the language. Thus \textit{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 \texttt{LCD} to run the program.

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

\begin{verbatim}
mouse driver not loaded
\end{verbatim}

2.3 THE MOUSE BUTTONS

<table>
<thead>
<tr>
<th>Button</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left Button</td>
<td>paints a mask cell</td>
</tr>
<tr>
<td>Right Button</td>
<td>clears a mask cell</td>
</tr>
<tr>
<td>Middle Button</td>
<td>sets the hot spot (the place on the cursor which dictates exactly where the cursor is actually pointing)</td>
</tr>
</tbody>
</table>
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 (e.g., 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 'y_yyy' 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 SetCursorToXP;

(* 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(50%); screenMask[1] := BITSET(51%);
    cursorMask[0] := BITSET(0%); cursorMask[1] := BITSET(1%);
    hotX := %5X%; hotY := %5Y%;
  END;
  Mouse.SetGraphicCursor (cursor);
END SetCursorToXP;
When this skeleton is loaded, a cursor designed and called "Cross", the resultant output file "CROSS.MOD" contains the following:

```pascal
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[6] := BITSET(0); screenMask[7] := BITSET(0);
    screenMask[8] := BITSET(0); screenMask[9] := BITSET(0);
    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[8] := BITSET(32766); cursorMask[9] := 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

```pascal
graphicCursor_type XPCursor =
  %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:

```pascal
graphicCursor_type CrossCursor =
  -1, -1, -1, -2017, -2017, -2017, 0, 0,
  0, 0, -2017, -2017, -2017, -1, -1,
  0, 0, 0, 0, 960, 960, 960, 32766,
  32766, 960, 960, 960, 0, 0, 0, 0,
  7, 8);
```