```
    // Turn off the mouse cursor
    inregs.x.ax = 2;
    int86 (0x33, &inregs, &outregs);

    printf ("Mouse x-coordinate %d\n", x_position);
    printf ("Mouse y-coordinate %d\n", y_position);
}
```

When your programs use Int 33H functions 5H and 6H, the functions return the mouse button status value previously described in Table 8-2 in the AX register and the number of times the button has been clicked in the BX register.

Positioning the Mouse Cursor

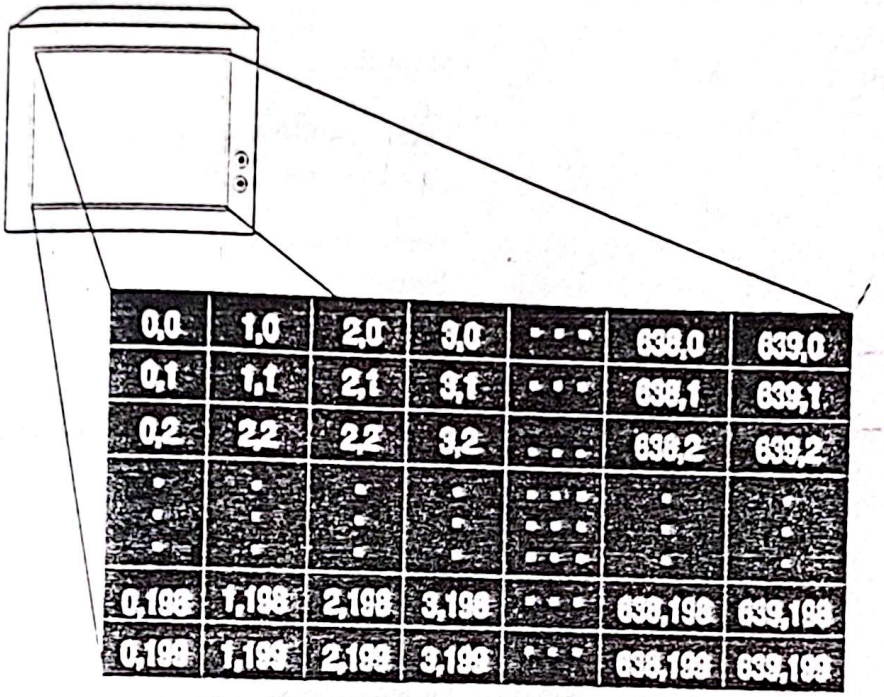
The key to mouse operations is tracking mouse button events that note the mouse cursor's coordinates relative to known positions of objects on your screen. The mouse driver supports four cursor position functions. Depending on a computer's video adapter and monitor, programs can display their output in various video display modes. Generally, we simplify the discussion of video display modes to either text or graphics. Within text and graphics mode, however, the number of screen positions can differ considerably. Therefore, to write programs that successfully use the mouse, your programs need to be aware of the current video mode. Remember, your programs must compare the mouse coordinates to known object positions on the screen.

The mouse services often return the mouse cursor's position using virtual screen coordinates. A *virtual screen* is a grid that maps to your screen as shown in Figure 8-1. In most text and graphics modes, the virtual grid is 640 × 200. As Table 8-1 shows, video modes 0EH, 0FH, 10H, and 12H use more cells along the Y axis.

To determine the mouse cursor's coordinates relative to objects on your screen, you must map the virtual screen coordinates to row and column positions for text mode operations and pixel positions in graphics mode. Table 8-3 lists the operations your programs should perform to map the cursor to screen locations for each video mode.

The mouse driver provides several services that let your programs control the mouse cursor's position. The first, Int 33H function 4, sets the cursor's screen position. To use this service your programs must assign the desired X coordinate to the CX register and the desired Y coordinate to DX. The following C program, SETMOUSE.C, initially positions the mouse cursor to the location (0,0). When the program begins, click the left mouse button, and the program will move the mouse to location (0,0). The second time you click the mouse, the program will move the mouse to location (10,10), followed by (20,20), and so on. The program will repeat this process until the mouse is at location 100, 100:

```
#include <dos.h>
void main (void)
{
  union REGS inregs, outregs;

  int location = 0;

  // Check for mouse driver
  inregs.x.ax = 0;
  int86 (0x33, &inregs, &outregs);

  if (outregs.x.ax == 0)
    printf ("Mouse not present\n");
  else
  {
      // Turn the mouse cursor display on
    inregs.x.ax = 0x01;
    int86 (0x33, &inregs, &outregs);

    do {

      // position the mouse cursor to the current location
      inregs.x.ax = 0x04;
      inregs.x.cx = location;
      inregs.x.dx = location;
      int86 (0x33, &inregs, &outregs);

      // wait for the user to press the left mouse key
      inregs.x.ax = 0x05;
      inregs.x.bx = 0;      // left button

      do {
        int86 (0x33, &inregs, &outregs);
      } while (outregs.x.bx == 0);

      location += 10;
    } while (location != 110);

    // Turn the mouse display off
    inregs.x.ax = 0x02;
    int86 (0x33, &inregs, &outregs);
  }
}
```



*Depending on the current video mode, the range of Y values may be 199, 349, or 479.

Figure 8-1. Mouse services return mouse cursor positions relative to a virtual screen

Video Mode	Mapping
00H	Text column = x / 16 Text row = y / 8
01H	Text column = x / 16 Text row = y / 8
02H	Text column = x / 8 Text row = y / 8
03H	Text column = x / 8 Text row = y / 8
04H	Pixel column = x / 2 Pixel row = y

Table 8-3. Mapping Mouse Virtual Screen Coordinates to Screen Locations for Various Video Display Modes

Video Mode	Mapping
05H	Pixel column = x / 2 Pixel row = y
06H	Pixel column = x Pixel row = y
07H	Text column = x / 8 Text row = y / 8
0DH	Pixel column = x / 2 Pixel row = y
0EH	Pixel column = x Pixel row = y
0FH	Pixel column = x Pixel row = y
10H	Pixel column = x Pixel row = y
11H	Pixel column = x Pixel row = y
12H	Pixel column = x Pixel row = y
13H	Pixel column = x /2 Pixel row = y

Table 8-3 Mapping Mouse Virtual Screen Coordinates to Screen Locations for Various Video Display Modes (continued)

By default, the user can move the mouse cursor over the entire screen. There may be times when you want to restrict the cursor's movement to a specific area on the screen, such as a menu region. Int 33H functions 7 and 8 let you restrict the area within which the mouse cursor can move. Function 7 restricts the cursor's movement along the X axis (horizontal movement). To use function 7, assign the minimal X axis limit to the CX register and the maximum limit to DX. The coordinates must be specified in virtual screen coordinates from 0 to 639. Int 33H function 8 restricts the mouse cursor's Y axis (vertical) movement. To use function 8, assign the minimal Y axis limit to the CX register and the maximum value to DX. You must specify the screen coordinates in virtual screen coordinates. The following C program, LIMITMOU.C, uses functions 7 and 8 to restrict the mouse cursor's movement to

the rectangular region at the center of the screen whose coordinates are (280,80), (280, 120), (360, 80) and (360,120):

```
#include <dos.h>
#include <conio.h>

void main (void)
  {
    union REGS inregs, outregs;

    // Check for mouse driver
    inregs.x.ax = 0;
    int86 (0x33, &inregs, &outregs);
    if (outregs.x.ax == 0)
      printf ("Mouse not present\n");
    else
     {
        printf ("Press mouse button to continue\n");

        // Turn on the mouse cursor
        inregs.x.ax = 1;
        int86 (0x33, &inregs, &outregs);

        // Restrict mouse movement to the rectangle defined by
        // (280, 80), (280, 120), (360, 80), (360, 120)
        inregs.x.ax = 7;
        inregs.x.cx = 280;
        inregs.x.dx = 360;
        int86 (0x33, &inregs, &outregs);

        inregs.x.ax = 8;
        inregs.x.cx = 80;
        inregs.x.dx = 120;
        int86 (0x33, &inregs, &outregs);

        // Loop until user presses a mouse button
        inregs.x.ax = 3;

        do {
          int86 (0x33, &inregs, &outregs);
        } while (outregs.x.bx == 0);

        // Turn off the mouse cursor
        inregs.x.ax = 2;
```

```
        int86 (0x33, &inregs, &outregs);
      }
  }
```

Run this program and verify the mouse cursor restrictions by moving your mouse. To end this program, click a mouse button.

In a similar way, Int 33H function 10H defines the screen region within which the mouse cursor disappears. If you move the cursor into the region, the mouse driver will turn the cursor's display off. Your program must then specifically turn the cursor's display back on if you want the cursor to reappear. To use function 10H, assign the minimum and maximum X axis limits to the CX and SI registers and the minimum and maximum Y axis limits to DX and DI. The following C program, HIDEMOUS.C, hides the mouse cursor's display if you move the mouse into the rectangular region whose coordinates are (0,40), (40,0), (0,40) and (40,40):

```
#include <dos.h>
#include <conio.h>

void main (void)
  {
    union REGS inregs, outregs;

    // Check for mouse driver
    inregs.x.ax = 0;
    int86 (0x33, &inregs, &outregs);

    if (outregs.x.ax == 0)
      printf ("Mouse not present\n");
    else
     {
        printf ("Move mouse to the upper-left screen corner\n");
        printf ("The mouse will disappear\n");
        printf ("Press a mouse button to continue\n");

        // Turn on the mouse cursor
        inregs.x.ax = 1;
        int86 (0x33, &inregs, &outregs);

        // Hide the mouse within the rectangle defined by
        // (0, 0), (40, 0), (0, 40), (40, 40)
        inregs.x.ax = 0x10;
        inregs.x.cx = 0;
        inregs.x.si = 40;
        inregs.x.dx = 0;
```

```
    inregs.x.di = 40;
    int86 (0x33, &inregs, &outregs);

    // Loop until user presses a mouse button
    inregs.x.ax = 3;

    do {
      int86 (0x33, &inregs, &outregs);
    } while (outregs.x.bx == 0);

    // Turn off the mouse cursor
    inregs.x.ax = 2;
    int86 (0x33, &inregs, &outregs);
  }
}
```

When you run this program, the mouse cursor will appear at the center of your screen. Move the mouse to the upper-left corner of your screen. The mouse cursor will disappear. To end the program, click the mouse button. Although the mouse cursor is not visible,the program can still respond to mouse clicks.

Customizing the Cursor

By default, the mouse driver displays the text mode mouse cursor as a rectangle and the graphics mode mouse cursor as an arrow. Int 33H functions 9 and 0AH let your programs customize the graphics mode and text mode cursors.

Customizing the Text Mode Mouse Cursor

Int 33H function 0AH gives you two ways to customize the text mode mouse cursor. First, your programs can change the scan lines used to create the cursor block, much like programs use the ROM-BIOS video services to change the cursor's scan lines. To create a custom text mode cursor in this way, invoke function 0AH, assigning the value 1 to the BX register, the starting scan line to the CX registers, and the ending scan line to DX. Figure 8-2 illustrates several different mouse cursors created using various scan-line combinations.

The second way to change the text mode mouse cursor is to use a bit mask, much like the technique you will use to create a custom graphics mode mouse cursor. To create a custom text-mode cursor using a bit mask, invoke function 0AH, assigning the value 0 to the BX register, the screen bit mask to the CX register, and a cursor bit mask to DX. The screen bit mask determines how the mouse driver treats screen characters at the current mouse cursor position. The driver ANDs the value in the

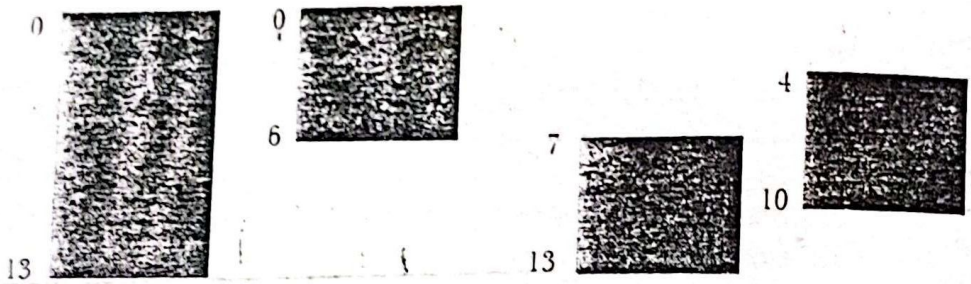


Figure 8-2. Text mode mouse cursors created by changing the cursor's scan lines (the number of cursor scan lines depends on the current video mode) in mode 2, 80 × 25 16-color text mode

CX register with the attribute and character values stored at the current location. In general, you probably don't want the cursor to appear. You can use the value FF00H to keep the existing screen attributes while causing no character to be displayed. The cursor mask lets you control the cursor's appearance and attributes. The mouse driver Exclusive ORs the value of the cursor mask with the result of the previous AND operation. The following C program, HAPPY.C, uses the screen mask FF00H and the cursor mask 1 to display the ASCII happy face as the mouse cursor. If you examine the ASCII chart in Appendix C, you will find the value 1 corresponds to the happy face:

```
#include <dos.h>
#include <conio.h>

void main (void)
  {
    union REGS inregs, outregs;

    // Check for mouse driver
    inregs.x.ax = 0;
    int86 (0x33, &inregs, &outregs);

    if (outregs.x.ax == 0)
      printf ("Mouse not present\n");
    else
     {
        printf ("Press any key to continue\n");
```

```
        // Select the custom cursor
        inregs.x.ax = 0x0A;
        inregs.x.bx = 0;
        inregs.x.cx = 0xFF00;  // screen mask: FFH saves attribute
                               // 00 hides current character
        inregs.x.dx = 1;       // ASCII happy face
        int86 (0x33, &inregs, &outregs);

        // Turn the mouse cursor display on
        inregs.x.ax = 0x01;
        int86 (0x33, &inregs, &outregs);

        // Wait for the user to press a key
        getch();

        // Turn the mouse cursor display off
        inregs.x.ax = 0x02;
        int86 (0x33, &inregs, &outregs);
    }
}
```

Customizing the Graphics Mode Mouse Cursor

The default graphics mode mouse cursor appears as an arrow. The following C program, DEFAULT.C, changes the current video mode to 640 × 200 two-color graphics (which is supported by all video adapters except monochrome) and then displays the default graphics mode mouse cursor:

```
#include <dos.h>
#include <conio.h>

void main (void)
  {
    union REGS inregs, outregs;

    int done = 0;       // Controls the program loop
    int button_count;   // Number of mouse buttons
    int original_mode; // Original video display mode

    // Check for mouse driver
    inregs.x.ax = 0;
    int86 (0x33, &inregs, &outregs);
```

```
if (outregs.x.ax == 0)
  printf ("Mouse not present\n");
else
 {
    button_count = outregs.x.bx;

    // Save the current display mode
    inregs.h.ah = 0x0F;
    int86 (0x10, &inregs, &outregs);
    original_mode = outregs.h.al;

    // Set the display mode to 640x200 graphics
    inregs.h.ah = 0;  // Set video mode
    inregs.h.al = 6;  // 640x200
    int86 (0x10, &inregs, &outregs);

    printf ("Press mouse button to continue\n");

    // Turn on the mouse cursor
    inregs.x.ax = 1;
    int86 (0x33, &inregs, &outregs);

    // Loop until user presses a mouse button
    inregs.x.ax = 5;

    do {
      // Check the left mouse button
      inregs.x.bx = 0;
      int86 (0x33, &inregs, &outregs);
      if (outregs.x.bx != 0)
        done = 1;
      else   // Check the right button
        {
          inregs.x.bx = 1;
          int86 (0x33, &inregs, &outregs);
          if (outregs.x.bx != 0)
            done = 1;
          else if (button_count == 3)  // check center
            {
              inregs.x.bx = 2;
              int86 (0x33, &inregs, &outregs);
              if (outregs.x.bx != 0)
                done = 1;
            }
        }
```

```
    } while (!_done);

    // Turn off the mouse cursor
    inregs.x.ax = 2;
    int86 (0x33, &inregs, &outregs);

    // Restore the original video mode
    inregs.h.ah = 0;  // Set video mode
    inregs.h.al = original_mode;
    int86 (0x10, &inregs, &outregs);
  }
}
```

Int 33H function 9H lets you customize the graphics mode cursor. As was the case with the text mode cursor, customizing the graphics mode mouse cursor requires you to specify a screen and cursor bit mask. For graphics mode, these bit masks are 16 × 16. The mouse driver ANDs the screen bit mask with the existing screen image and then Exclusive ORs the result with cursor mask. To customize the cursor using Int 33H function 9H, your programs must assign the address of the screen and cursor bit masks to the ES:DX register combination. The following C structure illustrates a screen and cursor mask that creates a custom graphics mode mouse cursor, shaped like, what else—a mouse:

```
unsigned masks[] = {
  0,0,0,0,0,0,0,0,  // Screen mask, all zero
  0,0,0,0,0,0,0,0,

  // Cursor mask
  0x0000, // 0000000000000000
  0x0000, // 0000000000000000
  0x0000, // 0000000000000000
  0x0000, // 0000000000000000
  0x0000, // 0000000000000000
  0x0000, // 0000000000000000
  0x1000, // 0001000000000000
  0x13C0, // 0001001111000000
  0x3FF0, // 0011111111110000
  0x7FF8, // 0111111111111000
  0xFFF8, // 1111111111111000
  0xFFF8, // 1111111111111000
  0x0824, // 0000100000100100
  0x0822, // 0000100000100010
  0x1CE2, // 0001110011100010
  0x0000  // 0000000000000000
};
```

In graphics mode, the cursor has a predefined "hot spot" the driver uses to define the actual mouse position. Your programs must specify the hot spot as X and Y offsets from the upper-left corner of the 16 × 16 bit mask. The upper-left corner, for example, would have the offsets 0,0. Likewise, the lower-right corner would have the offsets 15,15. When you customize the mouse cursor using Int 33H function 9H, your programs must assign the hot spot X offset to the BX register and the Y offset to CX. The following C program, MOUSECSR.C, uses the previous screen and cursor mask to display the custom "mouse" cursor:

```
#include <dos.h>
#include <conio.h>

void main (void)
  {
    union REGS inregs, outregs;
    struct SREGS segs;

    unsigned masks[] = {
      0,0,0,0,0,0,0,0,   // Screen mask, all zero
      0,0,0,0,0,0,0,0,

      // Cursor mask
      0x0000, // 0000000000000000
      0x0000, // 0000000000000000
      0x0000, // 0000000000000000
      0x0000, // 0000000000000000
      0x0000, // 0000000000000000
      0x0000, // 0000000000000000
      0x1000, // 0001000000000000
      0x13C0, // 0001001111000000
      0x3FF0, // 0011111111110000
      0x7FF8, // 0111111111111000
      0xFFF8, // 1111111111111000
      0xFFF8, // 1111111111111000
      0x0824, // 0000100000100100
      0x0822, // 0000100000100010
      0x1CE2, // 0001110011100010
      0x0000  // 0000000000000000
    };

  int done = 0;          // Controls the program loop
  int button_count;  // Number of mouse buttons
  int original_mode; // Original video display mode
```

```
// Check for mouse driver
inregs.x.ax = 0;
int86 (0x33, &inregs, &outregs);

if (outregs.x.ax == 0)
  printf ("Mouse not present\n");
else
 {
    button_count = outregs.x.bx;

    // Save the current display mode
    inregs.h.ah = 0x0F;
    int86 (0x10, &inregs, &outregs);
    original_mode = outregs.h.al;

    // Set the display mode to 640x200 graphics
    inregs.h.ah = 0;   // Set video mode
    inregs.h.al = 6;   // 640x200
    int86 (0x10, &inregs, &outregs);

    printf ("Press mouse button to continue\n");

    // Customize the mouse driver
    inregs.x.ax = 9;
    inregs.x.bx = 0;  // X hot spot
    inregs.x.cx = 11; // Y hot spot
    inregs.x.dx = (unsigned) masks;
    segread (&segs);
    int86 (0x33, &inregs, &outregs);

    // Turn on the mouse cursor
    inregs.x.ax = 1;
    int86 (0x33, &inregs, &outregs);

    // Loop until user presses a mouse button
    inregs.x.ax = 5;

    do {
      // Check the left mouse button
      inregs.x.bx = 0;
      int86 (0x33, &inregs, &outregs);
      if (outregs.x.bx != 0)
        done = 1;
```

```
        else    // Check the right button
          {
             inregs.x.bx = 1;
             int86 (0x33, &inregs, &outregs);
             if (outregs.x.bx != 0)
               done = 1;
             else if (button_count == 3)  // check center
               {
                 inregs.x.bx = 2;
                 int86 (0x33, &inregs, &outregs);
                 if (outregs.x.bx != 0)
                   done = 1;
               }
          }
      } while (! done);

      // Turn off the mouse cursor
      inregs.x.ax = 2;
      int86 (0x33, &inregs, &outregs);

      // Restore the original video mode
      inregs.h.ah = 0;  // Set video mode
      inregs.h.al = original_mode;
      int86 (0x10, &inregs, &outregs);
    }
}
```

Understanding Mouse Movement

As you move the mouse, the mouse driver automatically moves the mouse cursor on your screen. There may be times when your programs are interested in the amount of mouse movement. Int 33H function 0BH determines the amount of mouse movement since the last call to the function. The function returns the amount of horizontal movement in the CX register and the amount of vertical movement in DX. The movement is measured in *mickeys*. One mickey is approximately equal to 0.127 millimeters or 1/200 inch. In the case of horizontal movement, a negative value indicates the mouse was moved left. For vertical movement, a negative value means the mouse was moved up (away from the user). The following C program, MOVEMENT.C, monitors the mouse movement. Each time the mouse moves 100 mickeys in any direction, the program displays a message describing the movement. To end the program, click the mouse button.

```
#include <dos.h>

void main (void)
  {
    union REGS inregs, outregs;

    int num_buttons;
    int horizontal = 0;
    int vertical = 0;
    int button_pressed = 0;  // Controls loop

    // Check for mouse driver
    inregs.x.ax = 0;
    int86 (0x33, &inregs, &outregs);

    if (outregs.x.ax == 0)
      printf ("Mouse not present\n");
    else
     {
        num_buttons = outregs.x.bx;

        // Turn the mouse display on
        inregs.x.ax = 0x01;
        int86 (0x33, &inregs, &outregs);

        // Call function 0BH once to initialize counters to zero
        inregs.x.ax = 0x0B;
        int86 (0x33, &inregs, &outregs);

        // Loop until a button is pressed

        do {
          }//:Display the mouse
          inregs.x.ax = 0x01;
          int86 (0x33, &inregs, &outregs);

          // Check counters
          inregs.x.ax = 0x0B;
          int86 (0x33, &inregs, &outregs);
          horizontal += outregs.x.cx;
          vertical += outregs.x.dx;

          if ((horizontal >= 100) || (horizontal <= -100))
            {
              // Turn mouse display off
```

```
      inregs.x.ax = 0x02;
      int86 (0x33, &inregs, &outregs);
      printf ("Horizontal movement %d\n", horizontal);
      // Turn mouse display on
      inregs.x.ax = 0x01;
      int86 (0x33, &inregs, &outregs);
      horizontal = 0;
    }

  if ((vertical >= 100) || (vertical <= -100))
    {
      // Turn mouse display off
      inregs.x.ax = 0x02;
      int86 (0x33, &inregs, &outregs);
      printf ("Vertical movement %d\n", vertical);
      // Turn mouse display on
      inregs.x.ax = 0x01;
      int86 (0x33, &inregs, &outregs);
      vertical = 0;
    }

  // Check for left mouse button press
  inregs.x.ax = 0x05;
  inregs.x.bx = 0;     // left button
  int86 (0x33, &inregs, &outregs);
  if (outregs.x.bx)
    button_pressed = 1;
  else
    {
      // Check for left mouse button press
      inregs.x.ax = 0x05;
      inregs.x.bx = 1;     // right button
      int86 (0x33, &inregs, &outregs);
      if (outregs.x.bx)
        button_pressed = 1;
      else if (num_buttons == 3)
        {
          // Check for left mouse button press
          inregs.x.ax = 0x05;
          inregs.x.bx = 2;    // center button
          int86 (0x33, &inregs, &outregs);
          if (outregs.x.bx)
            button_pressed = 1;
        }
```

```
        }
    } while (! button_pressed);

    // Turn the mouse display off
    inregs.x.ax = 0x02;
    int86 (0x33, &inregs, &outregs);
  }
}
```

The program turns the mouse display off before displaying output with printf and then back on after the output completes. By default, the text mouse does not respond well to vertical screen scrolling, which occurs when printf performs a carriage return/line feed when the cursor is at the bottom of the screen. Actually, your screen may end up with two or more mouse cursor characters, one of which is the actual mouse cursor, and the other is a result of the scroll.

Controlling Mouse Speed and Responsiveness

By default, each time you move the mouse 8 mickeys in the horizontal direction or 16 mickeys in the vertical, the mouse driver moves the cursor 8 pixel locations. Int 33H function 0FH lets your programs control the mickey-to-pixel ratio. By decreasing the number of mickeys to pixels, you increase the speed at which the mouse moves across your screen. By increasing the number of mickeys to pixels, you slow down the mouse speed. The following C program, SLOWFAST.C, uses Int 33H function 0FH to change the mouse speed. If you click the right mouse button, the program doubles the mickey to pixel ratio, slowing down the mouse. If you press the left mouse button, the program cuts the current ratio in half, increasing the mouse speed. To end the program, press any key:

```
#include <dos.h>
#include <conio.h>

void main (void)
  {
    union REGS inregs, outregs;

    int horizontal_ratio = 8;  // Mouse driver defaults
    int vertical_ratio = 16;

    // Check for mouse driver
    inregs.x.ax = 0;
    int86 (0x33, &inregs, &outregs);
```

```
if (outregs.x.ax == 0)
  printf ("Mouse not present\n");
else
 {
    // Turn the mouse display on
    inregs.x.ax = 0x01;
    int86 (0x33, &inregs, &outregs);

    do {
      // Check for left mouse button press
      inregs.x.ax = 0x05;
      inregs.x.bx = 0;     // left button
      int86 (0x33, &inregs, &outregs);
      if ((outregs.x.bx) && (horizontal_ratio > 8))
        {
          horizontal_ratio = horizontal_ratio / 2;
          vertical_ratio = vertical_ratio / 2;
          inregs.x.ax = 0x0F;
          inregs.x.cx = horizontal_ratio;
          inregs.x.dx = vertical_ratio;
          int86 (0x33, &inregs, &outregs);
        }
      else
        {
          // Check for right mouse button press
          inregs.x.ax = 0x05;
          inregs.x.bx = 1;     // right button
          int86 (0x33, &inregs, &outregs);

          if ((outregs.x.bx) && ((vertical_ratio * 2) > 0))
            {
              horizontal_ratio = horizontal_ratio * 2;
              vertical_ratio = vertical_ratio * 2;
              inregs.x.ax = 0x0F;
              inregs.x.cx = horizontal_ratio;
              inregs.x.dx = vertical_ratio;
              int86 (0x33, &inregs, &outregs);
            }
        }
      } while (! kbhit());
 }

// Turn the mouse display off
inregs.x.ax = 0x02;
```

```
    int86 (0x33, &inregs, &outregs);
}
```

The minimum mickey-to-pixel ratio is 1:1, and the maximum is 32,767:1.

The previous function increased or decreased the speed at which the mouse cursor moves for all operations. Int 33H function 13H lets you specify a speed threshold that, once the mouse exceeds, the cursor moves across the screen at twice its normal speed. The default speed threshold is 64 mickeys per second. If the user moves the mouse at a rate greater than 64 mickeys per second, the movement of the mouse cursor's speed across the screen doubles. To change the threshold value using function 13H, you must assign the desired value to the DX register. The following C program, THRESHOL.C, lets you increase or decrease the double-speed threshold by clicking the right or left mouse button. Pressing the right mouse button doubles the current threshold. Pressing the left mouse button divides the current threshold in half. To end the program, press any key.

```
#include <dos.h>
#include <conio.h>

void main (void)
  {
    union REGS inregs, outregs;

    int threshold = 64;  // Mouse driver default

    // Check for mouse driver
    inregs.x.ax = 0;
    int86 (0x33, &inregs, &outregs);

    if (outregs.x.ax == 0)
      printf ("Mouse not present\n");
    else
      {
        // Turn the mouse display on
        inregs.x.ax = 0x01;
        int86 (0x33, &inregs, &outregs);

        do {
          // Check for left mouse button press
          inregs.x.ax = 0x05;
          inregs.x.bx = 0;     // left button
          int86 (0x33, &inregs, &outregs);
          if ((outregs.x.bx) && (threshold > 1))
            {
```

```
                threshold = threshold / 2;
                inregs.x.ax = 0x13;
                inregs.x.dx = threshold;
                int86 (0x33, &inregs, &outregs);
              }
            else
              {
                // Check for right mouse button press
                inregs.x.ax = 0x05;
                inregs.x.bx = 1;      // right button
                int86 (0x33, &inregs, &outregs);

                if ((outregs.x.bx) && ((threshold * 2) > 0))
                  {
                    threshold = threshold * 2;
                    inregs.x.ax = 0x13;
                    inregs.x.dx = threshold;
                    int86 (0x33, &inregs, &outregs);
                  }
              }
          } while (! kbhit());
      }

    // Turn the mouse display off
    inregs.x.ax = 0x02;
    int86 (0x33, &inregs, &outregs);
}
```

As you can see, Int 33H function 0FH lets you set the mouse speed, and function 13H lets you set the threshold at which the cursor doubles its speed across the screen. Int 33H function 1AH lets you combine functions 0FH and 13H into one.

To use function 1AH your program must assign the horizontal ratio of mickeys to pixels to the BX register, the vertical ratio to CX, and the double-speed threshold to DX. Int 33H function 1BH lets your programs get the current mouse sensitivity set by function 1AH. The function returns the horizontal mickey to pixel ratio in BX, the vertical ratio in CX, and the double-speed threshold in DX. The following C program, MOUSPEED.C, uses Int 33H function 1AH to set the mouse sensitivity and function 1BH to retrieve the values for display:

```
#include <dos.h>

void main (void)
  {
    union REGS inregs, outregs;
```

```
// Check for mouse driver
inregs.x.ax = 0;
int86 (0x33, &inregs, &outregs);

if (outregs.x.ax == 0)
  printf ("Mouse not present\n");
else
 {
    // Set the mouse sensitivity to the driver defaults
    inregs.x.ax = 0x1A;
    inregs.x.bx = 8;   // horizontal mickey/pixel ratio
    inregs.x.cx = 16;  // vertical mickey/pixel ratio
    inregs.x.dx = 64;  // double-speed threshold
    int86 (0x33, &inregs, &outregs);

    inregs.x.ax = 0x1B;
    int86 (0x33, &inregs, &outregs);

    printf ("Mouse double-speed threshold %d\n",
      outregs.x.dx);
    printf ("Horizontal mickey-to-pixel ratio %d\n",
      outregs.x.bx);
    printf ("Vertical mickey-to-pixel ratio %d\n",
      outregs.x.cx);
 }
}
```

The previous mouse functions control the speed at which the mouse driver moves the mouse cursor across the screen. Int 33H function 1CH, on the other hand, sets the speed at which the mouse driver polls the mouse and, thus, the speed at which the driver can recognize mouse clicks. Int 33H function 1CH only supports the Microsoft InPort mouse. To use function 1CH, your programs must assign one of the speed values listed in Table 8-4 to the BX register.

Saving the Mouse Driver State

In Chapter 9, "Creating Child Processes," you will learn how to temporarily suspend one program to run a second program called a child. If both programs use the mouse, it is possible that the child program may change the mouse driver settings in some way. Int 33H functions 15H, 16H, and 17H help your program allocate space for a buffer into which your program can save and later restore the mouse driver's state. Function 15H returns the number of bytes your program must allocate to store the