0、预备知识

0.1 开发平台: DosBox+Watcom C

① 文档下载

http://10.71.45.100/bhh/watcom.doc

② 软件下载

http://10.71.45.100/bhh/dosboxwc.rar

假定把上述压缩包解压到 D:\

双击 D:\DosBoxWc\wc.exe 进入 dosbox

可以看到 DosBox 中显示的当前文件夹为 c:\watcom\project

请注意此文件夹是虚拟的，实际对应的物理路径为:

D:\DosBoxWc\watcom\project

0.2 如何编辑源程序?

推荐使用第三方编辑器如 editplus(下载链接:http://10.71.45.100/bhh/editplus.rar),也可以用 watcom C 自带的 vi 编辑器。写好的程序应放在 D:\DosBoxWc\watcom\project 内。

0.3 如何编译?

WCL386 hello.c

0.4 如何运行?

hello

若错误信息太多,可以打开 hello.err 查看。

0.5 如何调试?

调试前要对源程序重新编译,编译命令如下:

WCL386 /d2 hello.c

再输入调试命令:

WD /tr=rsi hello

调试按键:

F10 step over 执行一步,不跟踪到函数内部

F8 trace into 执行一步,跟踪到函数内部

F4 user screen 观察程序的输出结果

某行左侧的[]处点击鼠标可以设一个断点

0.6 多个.c 文件如何整合编译成一个.exe ？

先建立 1.c, 2.c

再建立 makefile,其内容如下:

```
12.exe : 1.c  2.c
    wcl386 /fe=12.exe 1.c 2.c
```

最后输入以下命令进行 build:

```
wmake
```

```
#include <stdio.h>
int f(int x);
main()
{
   int x, y;
   x = 3;
   y = f(x);
   printf("y=%d\n", y);
}
```

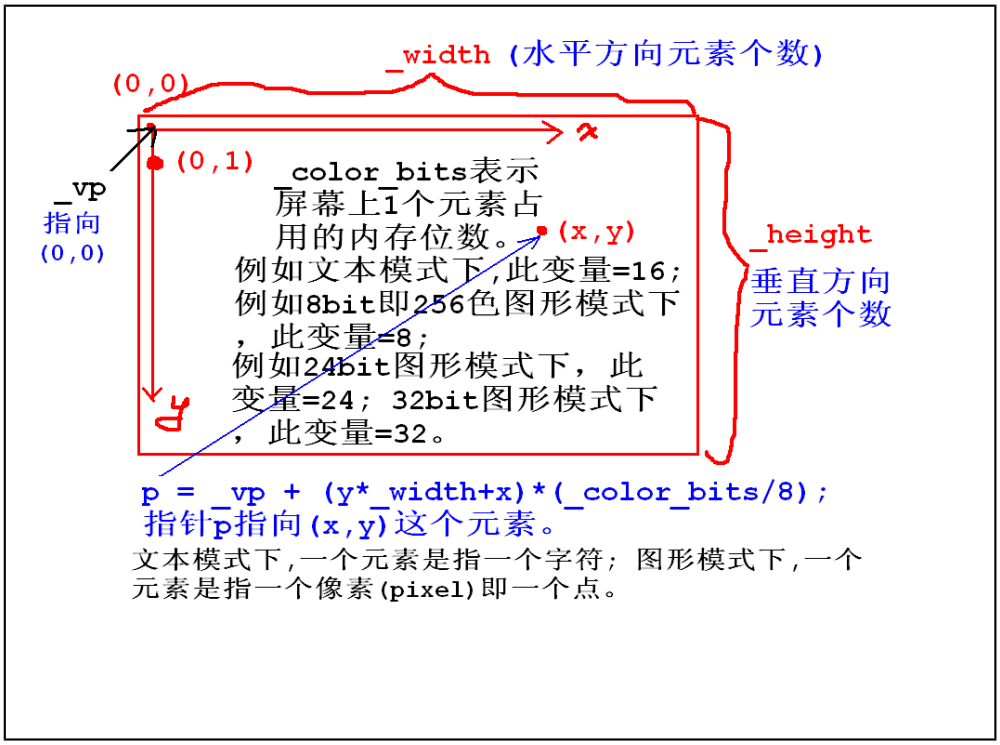
程序 1.c

```
int f(int x)
{
   return x*x;
}
```

程序 2.c

```
12.exe : 1.c  2.c
  wcl386 /fe=12.exe 1.c 2.c
```

makefile



一、坐标系统及 graphics 图形库中定义的全局变量

二、 文本模式(text mode)编程指南

DosBox 刚运行时显卡的工作模式就是文本模式;

要让显卡工作在图形模式的话,必须调用函数 initgraph(); 若要从图形模式返回文本模式,则应该调用函数 closegraph() 或 text_mode() 或

```
#include <graphics.h>
#include <stdio.h>
main()
{
   char *p = _vp;
   *p = 'A'; //  下面的语句决定输出的A为蓝底红字
   *(p+1) = (BLUE<<4) + RED; //或
*(p+1)=0x14;
   *(p+2) = 'B'; //下面的语句决定输出的B为绿底白
字
   *(p+3) = (
*(p+3)=0x2F;
```

程序 a.c; 在(0,0)及(1,0)输出字母A、B

```
    }
```

请注意，文本模式下，8 位颜色值中的高 4 位为背景色，低 4 位为前景色，所以当设定蓝底红字时，颜色值应该等于 (BLUE<<4)+RED = (1<<4)+4 = 0x14。

```
#include <graphics.h>
#include <stdio.h>
main()
{
   char *p = _vp;
   int i;
   for(i=0; i<80*25; i++)
   {
      *p = 'A';
      *(p+1) = RED; //  相当于 *(p+1) = 0x04;
      p+=2;
   }
   getchar();
}
```

程序 b.c;　输出 2000 个 A

三、 图形模式(graphics mode)编程指南

3.1 8bit 图形模式

1 个点对应显存的 1 个字节，每个点最多有 256 种颜色变化。

例 3:程序 c.c 画一条红色水平线。

```
#include <graphics.h>
#include <stdio.h>
main()
{
   int driver=0, mode=VESA_1024x768x8bit;
   char *p;
   int x=100, y=100, i;
   initgraph(&driver, &mode, "");
   p = _vp + y*_width + x;
   for(i=x; i<600-x+1; i++)
   {
      *p++ = RED; // 或写成 *p++ = 4;
   }
   getchar();
}
```

程序 c.c；(100,100)-(600,100)画一条红色水平线

8bit 图形模式下的颜色编号与颜色的对应关系（如 4 对应红色）与文本模式类似，具体请查阅头文件 graphics.h 中 enum COLORS 的定义或主页h[illegible]的相关[illegible]例[illegible]：

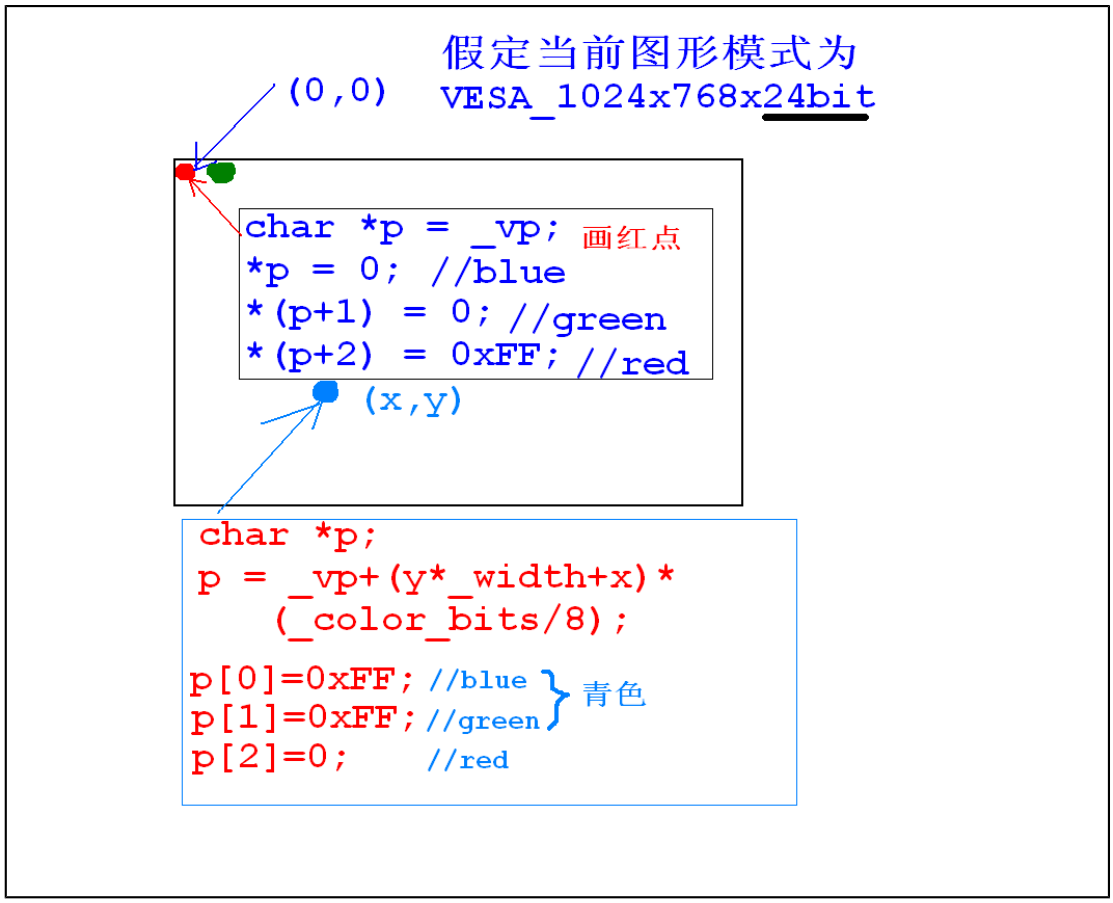
```
#include <graphics.h>
#include <stdio.h>
main()
{
   int driver=0, mode=VESA_1024x768x8bit;
   int i, j, x, y;
   char t[10];
   initgraph(&driver, &mode, "");
   x = 0;
   y = -16;
   for(i=0; i<256; i++)
   {
      setcolor(i); //  设置画线的颜色
      y += 16;       //  移到下一行，行高 =16
      if(y >= _height) //  若已画到屏幕底部
      {
         y = 0;           //  则重回屏幕顶
         x += 100;       //  移到右边一列，列宽 =100
      }
      for(j=0; j<16; j++) //  画 16 条水平线作为 1 行
      {
         line(x, y+j, x+60-1, y+j);
      }
      setcolor(WHITE); // 设置文字的颜色
      sprintf(t, "%02X", i); // 颜色编号转 16 进制
      outtextxy(x+60, y, t); // 输出 16 进制颜色编号
   }
   getchar();
   closegraph();
   return 1;
}
```

程序 256color.c；画 256 种颜色

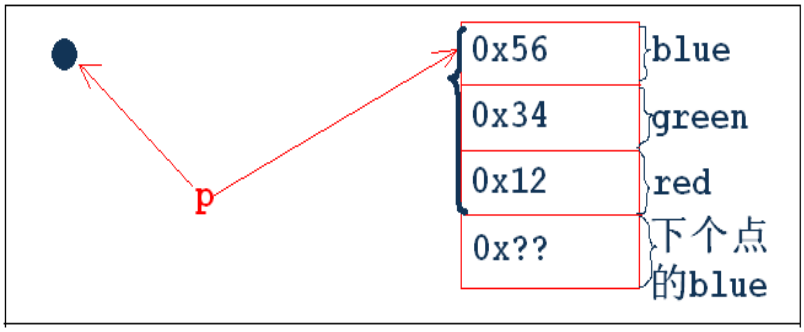
3.2 24bit 图形模式

1 个点对应显存的 3 个字节，每个点的颜色有 2^{24} 种变化。

以下示意图展示了 VESA_1024x768x24bit 模式下，如何在(0,0)处画了一个红点，在(x,y)处画一个青点：



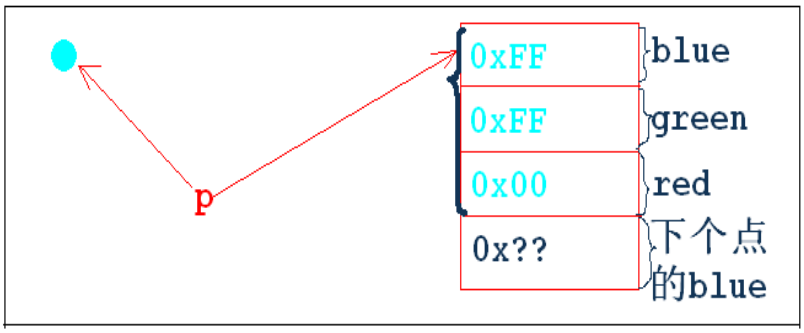
程序 24bit0.c、24bit1.c、24bit2.c、24bit3.c 功能完全一样，都以(200,100)为左上角，以(400,300)为右下角画一个实心的青色矩形。但它们实现的过程有所不同。其中 24bit3.c 是最简单的，它直接调用 bar()函数来画



设p当前指向的点的颜色值=0x123456，则此值在内存中的存放顺序为：0x56，0x34，0x12。由于p的类型为long *，
所以*p=0x??123456；现假定新的颜色color=0x00FFFF(即red=0,green=0xFF,blue=0xFF)，则

```
*p = (*p & 0xFF000000) | color
   = (0x??123456 & 0xFF000000) | 0x00FFFF
   = 0x??000000 | 0x00FFFF = 0x??00FFFF
```

结果如下图所示：



请注意下个点的blue并不会因为上述操作而发生变化。

```
#include <graphics.h>
main()
{
   int driver=0, mode=VESA_1024x768x24bit;
   int x, y;
   long color = 0x0000FFFF;
   long *p;
   initgraph(&driver, &mode, "");
   for(y=100; y<=300; y++)
   {
      for(x=200; x<=400; x++)
      {
         p = (long *)(
          _vp + (y*_width+x) *
          (_color_bits/8)
          );
         *p = (*p & 0xFF000000) | color;
      }
   }
   getchar();
   closegraph();
}
```

程序

```
#include <graphics.h>
typedef struct
{
   char blue;
   char green;
   char red;
} RGB;

main()
{
   int driver=0, mode=VESA_1024x768x24bit;
   int x, y;
   RGB c = {0xFF, 0xFF, 0x00};
   RGB (*p)[1024];
   initgraph(&driver, &mode, "");
   p = (RGB (*)[1024]) _vp;
   for(y=100; y<=300; y++)
   {
      for(x=200; x<=400; x++)
      {
         p[y][x] = c; // 此语句相当于以下三句
         //p[y][x].blue = c.blue;
         //p[y][x].green = c.green;
         //p[y][x].red = c.red;
      }
   }
   getchar();
   closegraph();
}
```

程序 24bit1.c

```
#include <graphics.h>
main()
{
   int driver=0, mode=VESA_1024x768x24bit;
   int x, y;
   long color = 0x0000FFFF;
   initgraph(&driver, &mode, "");
   for(y=100; y<=300; y++)
   {
      for(x=200; x<=400; x++)
      {
         putpixel(x, y, color); // 调用画点
函数
      }
   }
   getchar();
   closegraph();
}
```

以

心

例8：程序24bit3.c调用函数bar()，以(200,100)为左上角，以(400,300)为右下角画一个实心的青色矩形。

```
#include <graphics.h>
main()
{
   int driver=0, mode=VESA_1024x768x24bit;
   long color = 0x0000FFFF;
   initgraph(&driver, &mode, "");
   //设定填充的模式及颜色
  setfillstyle(SOLID_FILL, color);
   bar(200, 100, 400, 300);//调用画实心矩形函
数
   getchar();
   closegraph();
}
```

程序

3.3 32bit 图形模式

1 个点对应显存的 4 个字节，每个点的颜色有 2^{24} 种变化。

程序 32bit.c 也是以(200,100)为左上角，以(400,300)为右下角画一个实心的青色矩形。

例 9：程序 32bit.c 在 32bit 图形模式下使用二维结构数组画点，以(200,100)为左上角，以(400,300)为右下角画一个实心的青色矩形。

```
#include <graphics.h>
typedef struct
{
   char blue;
   char green;
   char red;
   char zero;
} RGB;

main()
{
   int driver=0, mode=VESA_1024x768x32bit;
   int x, y;
   RGB c = {0xFF, 0xFF, 0x00, 0x00};
   RGB (*p)[1024];
   initgraph(&driver, &mode, "");
   p = (RGB (*)[1024]) _vp;
   for(y=100; y<=300; y++)
   {
      for(x=200; x<=400; x++)
      {
         p[y][x] = c;
      }
   }
   getchar();
   closegraph();
}
```

程序

```
#include <graphics.h>
main()
{
   int driver=0, mode=VESA_1024x768x32bit;
   int x, y;
   long c = 0x0000FFFF;
   // red=0x00, green=0xFF, blue=0xFF
   long (*p)[1024];
   initgraph(&driver, &mode, "");
   p = (long (*)[1024]) _vp;
   for(y=100; y<=300; y++)
   {
      for(x=200; x<=400; x++)
      {
         p[y][x] = c;
      }
   }
   getchar();
   closegraph();
}
```

程序 32bitx.c

四、图片输出

4.1 显示 8bit bmp 图片

调用函数 load_8bit_bmp()可以在指定坐标位置显示一张 256 色 bmp 图片。

例 11: 程序 8bitbmp.c 调用函数 load_8bit_bmp()在坐标(0,0)显示一张图片 pic256.bmp，再在坐标(200,200)显示另一张图片 hello.bmp。程序 8bitbmp.c 及配套图片文件打包下载链接:

http://10.71.45.100/bhh/8bitbmp.rar

```
#include <graphics.h>
#include <stdio.h>
main()
{
   int x, y;
   int driver=0, mode=VESA_1024x768x8bit;
   initgraph(&driver, &mode, "");
   load_8bit_bmp(0, 0, "pic256.bmp");
   load_8bit_bmp(200, 200, "hello.bmp");
   getchar();
   closegraph();
}
```

程序 8bitbmp.c

```
#include <graphics.h>
#include <stdio.h>
#include <mem.h>
int load_24bit_bmp(int x, int y, char *filename)
{
   FILE *fp = NULL;
   byte *p = NULL; /* pointer to a line of bmp data */
   byte *vp = _vp + (y*_width + x) * (_color_bits/8);
   dword width, height, bmp_data_offset, bytes_per_line, offset;
   int i;
   p = malloc(1024L * 3);  /* memory for holding a line of bmp data */
   if(p == NULL)  /* cannot allocate enough memory for drawing 1 line
*/
      goto display_bmp_error;
   fp = fopen(filename, "rb");
   if(fp == NULL) /* cannot open bmp file */
      goto display_bmp_error;
   fread(p, 1, 0x36, fp);     /* read BMP head */
   if(*(word *)p != 0x4D42)   /* check BMP signature */
      goto display_bmp_error; /* not a BMP file */
   if(*(word *)(p+0x1C) != 24)
      goto display_bmp_error; /* not a 24-bit-color BMP file */
   width = *(dword *)(p+0x12);
   height = *(dword *)(p+0x16);
   bmp_data_offset = *(dword *)(p+0x0A);
   fseek(fp, bmp_data_offset, SEEK_SET); /* skip BMP head */
   bytes_per_line = (width * 3 + 3) / 4 * 4; /* must be multiple of 4
*/
   for(i=height-1; i>=0; i--)          /* draw from bottom to top */
   {
      fread(p, 1, bytes_per_line, fp); /* read a line of bmp data */
      offset = i * 1024 * 3;
      memcpy(vp+offset, p, width*3);
   }
   free(p);
   fclose(fp);
   return 1;
display_bmp_error:
   if(p != NULL)
      free(p);
   if(fp != NULL)
      fclose(fp);
   return 0;
}

main()
{
   int driver=0, mode=VESA_1024x768x24bit;
   initgraph(&driver, &mode, "");
   load_24bit_bmp(0, 0, "pic.bmp");
   getchar();
   closegraph();
}
```

五、文字输出

5.1 文本模式下的字符输出

文本模式下只能输出英文，不能输出中文。如果不考虑输出的坐标位置，则可以调用 printf()、puts()输出字

```
#include <graphics.h>
#include <stdio.h>
main()
{
   clrscr(); // 清屏
   gotoxy(1,1);
   puts("Hello,Tom!");
   gotoxy(2,2);
   puts("Hello,Jerry!");
   getchar();
}
```

程序 text1.c

请注意 gotoxy()函数中， 横坐标及纵坐标都是以 1 为基的，即(1,1)相当于实际坐标(0,0)。

在文本模式下，除了调用 gotoxy()，还可以使用视频指针_vp 来控制输出的坐标及内容。

例 14： 程序 text2.c 使用指针 vp 控制输出坐标及颜色和

内容，先在坐标(0,0)处输出黑色背景白色前景的字符串"Hello,Tom!"，再在坐标(1,1)处输出蓝色背景红色前景的字符串"Hello,Jerry"。

```
#include <graphics.h>
#include <stdio.h>
void putsxy(int x, int y, char *s, char color)
{
   char *vp = _vp + (y*_width+x)*2;
   int i = 0;
   while(s[i] != '\0')
   {
      *vp = s[i];
      *(vp+1) = color;
      vp += 2;
      i++;
   }
}

main()
{
   clrscr();
   putsxy(0, 0, "Hello,Tom!", BLACK<<4 | WHITE);
   putsxy(1, 1, "Hello,Jerry!", BLUE<<4 | RED);
   getchar();
}
```

程序 text2.c

5.2 图形模式下的字符输出

5.2.1 图形模式下的英文输出

图形模式下可调用 outtextxy()在指定坐标位置输出一个英文字符串。

```
#include <graphics.h>
#include <stdio.h>
main()
{
   int driver=0, mode=VESA_1024x768x24bit;
   initgraph(&driver, &mode, "");
   // 默认颜色为白色
  outtextxy(100,100, "Hello,Tom!");
   setcolor(0xFF0000); // 设置颜色为红色
  // 其中 red=0xFF, green=0x00, blue=0x00
  // 注意图形模式下的颜色不能使用 RED、GREEN、BLUE
  // 等常数, 而应该用 RGB 成份进行定义
   outtextxy(200,200, "Hello,Jerry!");
   getchar();
   closegraph();
}
```

程序 outtext.c

请注意，outtextxy()输出字符串的颜色是指前景色，该颜色由 setcolor()函数指定。outtextxy()并不描绘字符串的背景，也就是说 setbkcolor()函数指定的背景色对 outtextxy()无任何影响。

另外，outtextxy()是按 8*16 点阵输出字符，即字体大小是固定的。

5.2.2 图形模式下的中文输出

图形模式下可以自己编程读取汉字的点阵字库描出16*16个点。

例16：程序hz.c读取点阵字库hzk中"我"字的16*16点阵共32字节信息并在屏幕中间位置画出该字。程序hz.c及相关的字库文件hzk打包下载链接：

http://10.71.45.100/bhh/hz.rar

程序hz.c通过画一个4*4的方块来描绘一个点，因此实际上把汉字放大了4倍。但是，放大以后的汉字呈明显的锯齿形。

为了克服点阵汉字不能放大的缺陷，我们可以使用TTF曲线字库来输出汉字。其中快速输出可以通过调用函数get_ttf_text_pic()及draw_picture()实现，慢速输出可以通过调用函数out_ttf_text_xy()实现。

例17：程序ttftest.c演示了TTF字体的快速输出及慢速输出两种方式。程序ttftest.c及配套字库打包下载链接：

http://10.71.45.100/bhh/ttftest.rar

六、声音输出

6.1 扬声器输出

例18：程序music.c演示了如何控制杨声器发声，演奏一

首歌。程序 music.c 下载链接:

http://10.71.45.100/bhh/music.c

6.2 声卡输出

例 19: 程序 sound.c 演示了如何调用函数 play_wave()及 play_midi()输出 wav 波形文件及 midi 音乐文件。程序 sound.c 及配套声音文件打包下载链接:

http://10.71.45.100/bhh/sound.rar

七、输入

7.1 键盘输入

```
char x;
scanf("%c", &x);
x = getchar();
```

上述两个函数在输入一个字符时需要敲回车表示输入的结束；输入的符号会显示在屏幕上；一些特殊的键无法输入如上下左右方向键。

函数 bioskey()可以代替上述函数，并且克服以上 3 个不足之处。

```
int key;
key = bioskey(0);
```

其中 key 是一个 16 位整数。若当前键盘缓冲区是空的，

则执行 bioskey(0)会等待用户敲键；若当前键盘缓冲区非空，则执行 bioskey(0)会把键盘缓冲区中的键读走，不会等待用户输入。bioskey(1)用来检测键盘缓冲区是否为空，若缓冲区为空，则 bioskey(1)返回 0，否则返回非零。

什么是键盘缓冲区?

CPU 能在运行程序时处理外部中断(interrupt)。当用户按下某个键时，键盘会向 CPU 发出一个中断请求，此时 CPU 会暂停正在运行的程序而转去处理键盘中断，处理的结果是把当前按住的键读走并存放到键盘缓冲区中。键盘缓冲区是一个队列(queue)，相当于一个数组。

例 20: 程序 blkctrl1.c 演示了如何用 bioskey(0)控制键移动屏幕上的一个方块。程序 blkctrl1.c 下载链接:

http://10.71.45.100/bhh/blkctrl1.c

例 21: 程序 blkctr12.c 演示了如何用 bioskey(1)检测键盘缓冲区从而控制方块做随机运动。程序 blkctrl2.c 下载链接:

http://10.71.45.100/bhh/blkctrl1.c

除了使用 bioskey()控制键盘外，还可以用键盘中断来控制键盘。使用键盘中断不仅可以读取 bioskey()无法读取的键如 Ctrl、Shift、Alt、CapsLock 等，而且可以在用户按下某个键的瞬间作出实时响应。

例 22：程序 blkctr13.c 演示了如何用键盘中断控制方块的移动。程序 blkctrl3.c 下载链接：

http://10.71.45.100/bhh/blkctrl3.c

7.2 鼠标输入

鼠标要用到 int 33h 中断调用。例如 0003 号功能用来获取当前的鼠标状态，包括鼠标的坐标、鼠标的按键状态。汇编代码如下：

```
mov ax, 0003h
int 33h; 返回 cx=x 坐标,dx=y 坐标,bx=按键状态
```

其中 bx 的第 0 位=1 时表示左键被按下；

其中 bx 的第 1 位=1 时表示右键被按下；

其中 bx 的第 2 位=1 时表示中间键被按下。

获取鼠标当前状态的 C 语言代码如下：

```
int x, y, b;
union REGS r;
r.w.ax = 0x0003;
int86(0x33, &r, &r);
x = r.w.cx; /* 当前鼠标的 x 坐标 */
y = r.w.dx; /* 当前鼠标的 y 坐标 */
b = r.w.bx; /* 当前鼠标的按键状态 */
```

鼠标的其它功能调用包括：

功能号	用途

0000　　检测鼠标驱动是否安装

0004　　把鼠标定位到(x,y)坐标

0007　　设置水平方向的鼠标移动范围

0008　　设置垂直方向的鼠标移动范围

以上功能的具体参数及返回值请查阅中断大全：

http://www.ctyme.com/intr/int.htm

例23：程序hzmouse.c演示了用查询方式获取鼠标当前状态。按鼠标左键画绿色圆点，按Esc键结束程序。hzmouse.c及相关资源打包下载链接：

http://10.71.45.100/bhh/hzmouse.rar

例24：程序intmouse.c演示了用中断方式获取鼠标当前状态。按鼠标左键画点，按鼠标右键结束程序intmouse.c及相关资源打包下载链接：

http://10.71.45.100/bhh/intmouse.rar

八、动画

例25：程序bugwc.c演示了文本模式下一条沿水平方向蠕动的虫子。程序bugwc.c下载链接：

http://10.71.45.100/bhh/bugwc.c

例26：程序bugeatwc.c演示了文本模式下一条会走迷宫的虫子。按上下左右键可以控制青色方块不被虫子捉住。程

序 bugeatwc.c 下载链接：

http://10.71.45.100/bhh/bugeatwc.c

例 27：程序 dm8bit.c 演示了 256 色图形模式下一个会运动的方块，按上下左右键可以控制方块的运动方向。程序 dm8bit.c 及相关图片文件打包下载链接：

http://10.71.45.100/bhh/dm8bit.rar

例 28：程序 dm24bit.c 演示了 24bit 色图形模式下一个会运动且会变化的图片，按上下左右键可以控制图片的运行方向。程序 dm24bit.c 及相关图片文件打包下载链接：

http://10.71.45.100/bhh/dm24bit.rar

例 29：程序 asteroid.c 演示了一个打坝石的游戏，按左右键移动飞机，按朝上键加速，按空格键开火。程序 asteroid.c 及配套资源文件打包下载链接：

http://10.71.45.100/bhh/asteroid.rar