IMPRESIÓN DE GRÁFICOS

Hasta ahora hemos estado interesados en la creación de despliegue de gráficos en la pantalla del monitor. Ahora pondremos nuestra atención en obtener impresiones de nuestros gráficos. Los programas descritos en este capítulo fueron diseñados específicamente para usarse con una impresora punto matriz Epson FX-86e. Deberían trabajar igual de bien en otras impresoras Epson de las series MX y FX, y con pequeñas modificaciones en las impresoras de matriz de puntos IBM y otras impresoras de matriz de puntos compatibles con los comandos gráficos de Epson.

Si tienes un monitor EGA de 12 pulgadas el tamaño del despliegue es de 7 x 9- ? pulgadas. Esto corresponde a una resolución horizontal de 68 líneas por pulgadas y una resolución vertical de 50 líneas por pulgada. Sin el recurso de alguna técnica en especial, podremos fácilmente obtener una resolución horizontal de 120 líneas por pulgada y una resolución vertical de 72 líneas por pulgadas desde la impresora. Podremos mejorar esto usando comandos especiales de impresión y un software mas especializado. Es evidente hasta aquí que la impresora es capaz de una resolución mayor que el monitor. Por lo tanto nos enfrentamos con el dilema de que si desarrollamos las funciones de impresión en el monitor; mismas que duplican el despliegue en la impresora, entonces sacrificamos la capacidad de resolución. Mientras que si tratamos de tomar ventaja de las capacidades completas de impresión; necesitamos crear un cuerpo largo de software especializado.

SOFTWARE DE IMPRESIÓN

Estos apuntes no se meten en grandes detalles con el diseño del software para imprimir en alta resolución. Mas adelante vamos a ir directo a lo básico de las técnicas usadas para la impresión de gráficos por las impresoras Epson. Esto debería de darnos un buen entendimiento de cómo mandar alta resolución de datos a la impresora. Para crear tanta información como se necesite, por lo menos, para escribir versiones de todas las funciones en la librería *gdraws* y algunas de la librería *gtools*. Pueden ser esencialmente lo mismo como funciones, excepto que el sistema de coordenadas necesita ser mas largo para acomodar la resolución mas grande de la impresora. En algunos casos esto puede necesitar el uso de enteros largos donde se usaban enteros en las funciones originales. En vez de trazar los puntos directamente a la pantalla, se necesita grabar en un arreglo bidimensional. Si se planea impresión solo en blanco y negro, una arreglo de resolución para 120 x 72 puntos por pulgada podría ser de 120 x 432 bytes o un total de 51,840 bytes. Cada píxel (punto grafico) podría ser asignado como un byte y el software tendría que determinar como posicionar ese byte en particular cuando el píxel debe ser accionado. Luego, cuando el arreglo esta lleno con el despliegue propuesto se puede usar un programa genérico para mandar esta información a la impresora. Si se planea usar algunas formas de salida a impresión que requieran color, el tamaño del arreglo se incrementará por un factor de cuatro.

HERRAMIENTAS BÁSICAS PARA IMPRESIÓN

Antes de que se haga algo con la impresora, se necesitan herramientas básicas para comunicarnos con ella. Algunas versiones de C contienen rutinas I/O estándar para la impresora, Turbo C no las tiene. En cualquier caso, muchas de las rutinas estándard mandan un retorno de carro (CR) al final de cada línea. No siempre se quiere esto cuando se imprimen gráficos, sin embargo, por eso nuestras propias funciones son esenciales. Sólo 2 de estas son necesarias y son simples. La Fig. 1 enlista una función para determinar el status de la impresora. Esto es que no se desea mandar un caracter a la impresora si está desconectada u ocupada. Aquí se usa el servicio de impresión ROM BIOS para verificar el status y devolver un 80 hexadecimal si la impresora está lista o cero si no lo está.

```
char status()
{
    union REGS reg;

    reg.h.ah = 2;
    reg.x.dx = 0;
    int86(0x17,&reg,&reg);
    return (reg.h.ah & 0x80);
}
```

Fig. 1. Función para determinar el status de la impresora.

La segunda función enlistada en la Fig. 2 manda un caracter a la impresora, la declaración *while* causa que la función se enlace hasta que la impresora esté lista. Hasta este punto, el regreso de la función status es diferente de cero y la función continúa. El caracter que pasó a la función como un parámetro está cargado dentro de un registro apropiado y el servicio ROM BIOS de la impresora es usado para mandarlo.

```
char put_out (char caracter)
{
   union REGS reg;

   while(!status());
   reg.h.ah = 0;
   reg.h.al = caracter;
   reg.x.dx = 0;
   return (reg.h.ah);
}
```

Fig. 2. Función para mandar un caracter a la impresora.

CONSIDERACIONES AL DESCARGAR EL DESPLIEGUE DE PANTALLA A IMPRESORA

Se desea que la impresora reproduzca la pantalla de gráficos con imágenes de la misma figura que aparece en el monitor. Puesto que el despliegue es más ancho que alto, el modo apropiado para crear un despliegue impreso es tener una coordenada "X" para lo ancho de la hoja y una coordenada "Y "para lo estrecho. Esto significa que los pixels "Y" serán impresos por pines en la columna de punto matriz de pines de la impresora. La impresora tiene 9 pines en una columna, pero normalmente solo 8 están usados para gráficos y esos 8 pueden fácilmente ser direccionados por un byte de información gráfica. Los 9 pines pueden ser usados, pero un segundo byte es requerido para suministrar información del noveno únicamente y esto es mas bien ineficiente, cada pin se imprime en un espacio de 1/72 de pulgada. Es por eso que la resolución es esencialmente de 72 líneas por pulgada. Para lograr un espaciamiento correcto de línea para que el siguiente paso de 8 pines sea directamente adyacente al paso actual, se tiene que posicionar la impresora para una línea de espacio de 8/72 de pulgada. Se puede ligeramente reducir el tamaño de la salida a impresión en la dirección "Y", si es necesario, reduciendo la línea de debajo de 8/72 de una pulgada. Entonces coincidirá cada primer punto en el octavo del paso anterior, pero esto puede ser no visible. No se puede alargar el tamaño de impresión de esta forma, porque incrementar la línea de espacio deja estrechas líneas blancas, que son visibles.

Opción	Código primario	Código alterno	Densidad Horizontal
Densidad simple.	ESC*0	ESC K	60
Densidad doble.	ESC*1	ESC L	120
Doble densidad de Alta velocidad.	ESC*2	ESC Y	120
Densidad Cuádruple.	ESC*3	ESC Z	240
CRT I	ESC*4	0	80
Plotter(1:1)	ESC*5	0	72
CRT II	ESC*6	0	90
Plotter de doble Densidad.	ESC*7	0	144

Fig. 3. Modos gráficos en las impresoras Epson y los comandos para activarlos.

Consideremos lo que podría pasar si empezamos seleccionando la densidad simple (modo 1). Tenemos 350 pixels en una línea "Y" del despliegue EGA y esto imprimirá 5.8333 pulgadas de ancho a 60 puntos por pulgada. Para mantener las proporciones apropiadas del despliegue EGA, se necesita tener la impresión 7.7777 pulgadas de ancho, que para 640 pixels "X" requieren que 8 pixels abarquen 21/216 de pulgada. Esto requiere un poco de coincidencia pero no la suficiente para degradar el despliegue del

monitor. El comando Epson para posicionar la línea de espacio en 216 de pulgada es: ESC 3 n.

Donde el 3 es un 3 en código ASCII o 33 Hexadecimal. La n, representa el numero 216 de pulgada para moverse de cada línea, no es ASCII, por eso el comando de línea de espacio 21/216 podría ser:

0 x 1B 0 x 33 0 x 15

IMPRIMIENDO LA PANTALLA.

Primero la mayoría de los programas no hace el intento de centrar el despliegue en el papel, entonces normalmente aparece en una esquina. Segundo, usualmente no seleccionan los rangos correctos para que la impresora esté en una porción propia. En el despliegue EGA los círculos también aparecen como óvalos en la impresora. Tercero si el fondo del despliegue en EGA esta de color, todo el despliegue usualmente se imprime como una burbuja negra. La función de la Fig. 4 corrige estas deficiencias. La función primero prepara la impresora para doble espacio de 21/216 por pulgada. Siguiente, 12 líneas son alimentadas para la salida a la impresora para alistar la entrada de la impresora para que esté centrado en la página en la dirección “X”. Siguiente, la función lee un píxel del despliegue en la locación (0,0) y asume que los colores que regresan es el color anterior. La impresora solamente imprime pixels cuyos colores son diferentes al color de fondo. Si se asegura que nunca se tiene el píxel (0,0) en cualquier otro color que se usa de fondo, se evita la burbuja negra. Entonces la función empieza una vuelta que imprime columnas empezadas con el máximo valor de “X” y trabaja debajo hasta el más bajo término de la “X” . La función *printrow* imprime 8 pixels “X” debajo e incluye el valor que pasa como parámetro “X”. Además, la vuelta está puesta para que “X” nunca reduzca a XMIN, esto ha sido encargado en el paso previo del *printrow*. Con cada paso hacia la vuelta, la función revisa si determina si el *keystroke* ha sido activado, y si ha entrado entonces restablece la línea normal de impresión y la salida de la rutina. Al final de la vuelta de impresión, una forma de alimentación es enviada a la impresora para avanzar a la página siguiente.

```
int   XMIN = 0, XMAX = 639, YMIN = 0, YMAX = 349;
unsigned char background;

void printscr(void)
{
       char put_out(char character);
       int i,x,y;

       put_out(0x1B);
       put_out(0x33);
       put_out(0x15);
       for (i=0; i<12; i++)
```

```
       put_out(0x0A);
       background = readPixel(0,0);
       for (x=Xmax; x<XMIN+7; x - =8)
       {
            if (kbhit())
            {
                put_out(0x1B);
                put_out(0x32);
                getch();
                break;
            }
            printrow(x);

       }
       put_out('\x0C');
}


printrow(int x)
{
      unsigned char savechar, temp;
      static unsigned char out_buff[434]={"\x1bk\xAE\x01"};
      unsigned int i, j,newy, y;

      char status();
      char put_out();
      for (y=YMIN, j=84; y<=YMAX; y++,,j++)
      {
          savechar = 0;
          for (i=0; i<8; i++)
          {
               temp = readPixel(x-i,y);
               if (temp !=background)
                    savechar l = 1;
               if (i!=7)
                     savechar <<=1;
          }
            out_buff[j] = savechar;

     }
     for (i=0; i<434; i++)
            put_out(out_buff[i] );
     put_out('\r');
     put_out('\n');

}
```

Fig. 4. Función para imprimir el despliegue EGA.

IMPRIMIENDO UNA COLUMNA DE GRÁFICOS.

Para poder imprimir gráficos en la impresora Epson FX-86e con la densidad que queremos por el despliegue EGA, el siguiente comando debe ser mandado:

ESC K mm nn

Donde la K esta en ASCII K(4b hex) y nn es el bit menos significativo y mm el más significativo del número exacto de bytes de gráficos que tienen que ser enviadas. Este número debe de ser correspondido exactamente con el número de bytes que son transmitidos; y si es menos, la impresora va a hacer el intento de trasladar y operar sobre caracteres gráficos como si ellos fueran ASCII de caracteres o comandos. Si algunos bytes son transmitidos en lugar de ser especificados, cualquiera de los caracteres que se manden después de la cadena que la cual se espera ser impresa como un caracter normal, será impresa como gráfico hasta que el número específico de bytes sea encontrado. La K en el comando superior llama a la densidad única gráfica; se puede sustituir acordando con los requisitos, de la Fig. 3. La función *printrow* acomoda al buffer, el cual después será transferido a la impresora. Los primeros bytes en este buffer son: 1B K AE 01. Cuando son transferidos a la impresora, esto causa que la impresora sea acomodada por una densidad sencilla que designa que 1 AE(hex)bytes gráficos están seguidos. Esto es 430 bytes decimales.

La función entonces empieza la vuelta para explorar los pixels “Y” y generar bytes para estar en el buffer. Note que la vuelta esta acomodada para que la primera entrada al buffer sea la locación 84. Esto significa que después que la impresora sea acomodada por cada fila de gráficos, 80 ceros son transmitidos, lo cual causa que la cabeza se mueva sin imprimir, esto centra el despliegue por la coordenada “Y”.

La siguiente función *zeroes savechar* empieza una vuelta la cual lee 8 pixels de la locación (x,y) de regreso a la locación(x-7,y). Por cada píxel que es leído, el color es comparado con el color de fondo, y si es diferente, el bit menos significante de savechar se deja en cero, savechar es ahora enviado un bit a la izquierda (excepto que al último pasa a través de la vuelta) y la vuelta es repetida.

Sobre el cumplimiento de la vuelta, *savchar* contiene bits que determinan cuales pins en la impresora se activan 8 posiciones leídas. Cuando *savchar* ha sido juntado es transferido al buffer y el programa continúa hasta ser llenado con una completa línea de dato. Una vez que la línea datos esta acumulada en el buffer, se manda a impresión seguida por una línea de alimentación y un retorno de carro para preparar la impresora para la siguiente línea de datos.

IMPRIMIR LA REPRESENTACIÓN DE UN DESPLIEGUE A COLOR.

El color puede ser representado en la impresión de un despliegue usando diferentes sombras grises para representar diferentes colores. Si la impresora de punto matriz tuvo la capacidad de variar tamaños de punto una técnica de reproducción fotográfica altamente sofisticada pudo ser desempeñada. Desafortunadamente, todos los puntos son de el mismo tamaño. Todo lo que se puede es asignar un cluster de puntos de cada píxel y variar el número que está impreso de acuerdo al color. Esto hace un tipo muy crudo de impresión y a veces los resultados para los colores son diferentes en la pantalla cuando termina teniendo la misma sombra gris en la impresión. De un *standpoint* práctico, se van a permitir 4 puntos horizontales (en la terminación impresora) para representar un píxel "X" . Cuando se lee el píxel desde pantalla, su color está representado por 4 bits menos significativos. Nosotros asignaremos cada uno de estos bits a un punto, y además tener una combinación de puntos para cada color desafortunadamente este punto no puede ser siempre ser reconocido como sombras grises únicas.

La Fig. 5 lista funciones que desempeñan el descargue de una pantalla a color EGA a la impresora. Se parece a las funciones de la impresión de pantalla de la Fig. 4. La primera diferencia es de que estamos usando el modo de impresión de densidad cuádruple. Desafortunadamente en este modo no es posible imprimir puntos adyacentes. Consecutivamente, cuando imprimimos información fuera del buffer, se imprime cualquier otro byte, esparciendo cero bytes entre ellos. Entonces se hace un retorno de carro (CR) sin una línea de alimentación (LF) e imprime todo lo de los bytes que se perdió la primera vez, esparciendo 0 bytes en la misma línea. Se hace un CR y una LF para prepararse para la siguiente línea. Si este es el único cambio que se hace, ahora se tendrá una impresión que es un solo cuarto de altura como el que se tenía para blanco y negro. Se puede regresar a la misma sombra dispersada que se tenía antes mandando cada caracter del buffer a la impresora cuatro veces. Esto es esencialmente lo que se hace, excepto que en lugar de imprimir 4 puntos idénticos, se determina si un punto en particular está prendido o apagado por el color que fue leído desde la pantalla. Todo esto requiere un buffer 4 veces de largo como el que fué usado para blanco y negro y un cambio en el buffer de inicialización, no solo para especificar el nuevo modo, sino también para declarar un número de caracteres a ser transmitidos por la impresora que incluye todo lo de los duplicados.

```
int  XMIN = 0, XMAX = 639, YMIN = 0, YMAX = 349;
unsigned char background;

void color_printscr(void)
{
           char put_out(char character);

           int i, x, y ;

           put_out( 0x1b);
```

```
            put_out( 0x33);
            put_out( 0x14);
            for (i=0; i<12; i++)
            {

                  if (kbhit())
                  {

                          put_out( 0x1b);
                          put_out( 0x32);
                          getch();
                           break;

                   }
                   color_printrow(x);

            }
            put_out('\x0C');

}

color_printrow(int x)
{
        unsigned char savechar1,savechar2, savechar3,
                savechar4, temp;
        static unsigned char out_buff[1724] = {"\x1bz\xB8\x06};

        char status();
        char put_out();
        unsigned i, j, newy, y;
        for (y=YMIN, j=324; y<=YMAX; y++, j+=4)
        {

               savechar1 = 0;
               savechar2 = 0;
               savechar3 = 0;
               savechar4 = 0;
               for (i=8; i<8; i++)
               {
                     temp = readPixel(x-I,y);
                     if (temp != background)
                     {
                          if ((temp & 0x01) != 0)
                                savechar 1 l=1;
                          if ((temp & 0x02) != 0)
                                savechar 2 l=1;
                          if ((temp & 0x04) != 0)
                                savechar 3 l=1;
                           if ((temp & 0x08) != 0)
                                savechar 4 l=1;
                      }
                      if (i!=7)
                      {
                            savechar1<<=1;
                            savechar2<<=1;
                            savechar3<<=1;
```

```
                        savechar4<<=1;
                    }

                }
                out_buff[ j ]= savechar 1;
                out_buff[ j +1]= savechar 2;
                out_buff[ j +2]= savechar 3;
                out_buff[ j +3]= savechar 4;
        }
        for (i=0; i<324; i++)
        put_out(out_buff [ i ]);


        for (i=324; i<1724; i+=2)
        {
            put_out(out_buff[ i ];
            put_out(0 x00);
        }
        put_out('\r');
        for (i=0; i<324; i++)
              put_out(out_buff [ i ];
        for (i=325; i<1724; i+=2)
        {
            put_out(0 x00);
            put_out(out_buff[ i ];
         }
         put_out('\r');
         put_out('\n');

}
```

Fig. 5. Función para imprimir un despliegue EGA con sombra para ejecución a color.

CREAR IMPRESIÓN EN PANTALLA RESIDENTE

La utilidad de imprimir pantalla que se provee en el software MS-DOS no es usada cuando se está en el modo de gráficos. Responde solo al modo de texto y será confundida por la pantalla gráfica. Las funciones *printscr* y *color_printscr* que se muestran, manejan gráficos muy bien y si se insertan en el software en el punto apropiado, reproducirá la imagen efectivamente. Sin embargo, no soluciona el problema cuando viene a capturar una imagen en el vuelo. Afortunadamente es un hecho el convertirlos en programas residente para reposicionar la función *print screen* actual. La Fig. 6 lista un programa que se hace sólo para el modo EGA.

```
#include <dos.h>

unsigned char out_buff[434] = {“\x1bK\xAE\x01};

void interrupt print_handler()
{
      unsigned char background;
      int i, x ;
      void put_out(char character);

      put_out(0*1b);
      put_out(0*33);
      put_out(0*15);
      for (i=0; i<12; i++)
            put_out(0*0A);
      background = readPixel (0,0) ;
      for (x=639; x>=7; x - =8)
      {
            printrow(x,background) ;
      }
      put_out(‘\x0C’);
}

printrow(int x,int background)
{
      unsigned char savechar,temp ;
      unsigned i, j, newy, y ;

      char status();
            char put_out();
            for (y=0; y<=349; y++, j++)
      {
                  savechar = 0;
                  for (i=0; i<8; i++)
                  {
                        temp = readPixel(x-i, y) ;
             if (temp != background)
                 savechar l =1;

                        if (i!=7)
                 savechar <<= 1 ;
                  }
                  out_buff [ j ] = savechar;
             }
             for (i=0; i<434; i++)
             {
                   put_out(out_buff[ i ] ;
             }
             put_out(‘\r’);
             put_out(‘\n’);
}
char status()
{
             char temp;
             temp = inportb( 0 x 379);
             return (temp & 0x80);
```

```
}

void put_out(char character)
{
            int temp;

            while(!status());
      outportb(0x378,character);
      outportb(0x37A,0*0D);
      outportb(0x37A,0*0C);
}
int readPixel (int x, int y)
{
      #define DISPLAY_OUT(index, val)     {outp(0x3CE,index);\

outp(0x3CF, val    }

      int i, j, color=0;
      unsigned char mask, exist_color;
      char far *base;
      base = (char far*)  (0*A0000000L + ((long)y * 80L+
                  ((long)x\8L)));
      mask = 0x80 >> (x % 8);

            for (i=0; i<4; i++)
      {
                  DISPLAY_OUT(4, i );
             DISPLAY_OUT(5, 0 );
             exist_color = *base & mask ;
                  if (exist_color != 0)
                      color l= 0x01<< i ;
             }
      return color;
}
main()
{
      setvect(5, print_handler);
      printf("\nEGA Graphic Screen Printing Routine"
                        "Installed\n") ;
      keep(0, 0x9FF);
}
```

Fig. 6. Programa para reposicionar la función *print screen* IBM con la función de gráficos EGA *print screen.*

No hay mucha diferencia en los programas residentes necesitados para imprimir CGA, VGA o despliegue Hercules. Sin embargo el tamaño del buffer y el modo de densidad de impresión necesitan ser diferentes para obtener el tamaño y figura correcto de despliegue y para desplegar toda la información que esta disponible. La Fig. 8 enlista el programa residente de impresión para el VGA.

Los programas son como la función *printscr* descrita, pero hay algunas diferencias significantes. Primero note que se había que reposicionar las 3 subrutinas que hacen uso de llamadas ROM BIOS. El ROM BIOS es llamado para la interrupción de manejo de rutinas y no es entrante, por eso no podemos llamarlo de nuevo desde dentro de nuestra función. También, aunque no es demasiado evidente la función *Kbhit* de Turbo C, que fue usada para permitir la interrupción de la rutina *printscr ;* usa una llamada ROM BIOS , por eso esa porción del programa tuvo que ser borrada para que la función de memoria residente trabajara correctamente. La función *status* simplemente lee el status registrado de la impresora y revisa si está o no ocupada.
La función para escribir a la impresora es un poco mas complicada. Primero se da la vuelta hasta que no esté ocupada la impresora. Después producimos el caracter deseado al registro de información del puerto paralelo de la impresora. Después producimos una salida de luz (strobe) para mandar el caracter a la impresora. Esto se hace mandando 0x0D al puerto de control de la impresora para empezar la impresión y 0x0C para controlar el puerto de salida.

```
#include <dos.h>

unsigned char out_buff[ 565 ] = { "\x1b*5\x30\x02" } ;

void interrupt print_handler ()
{
      unsigned char background ;
      int i, x ;

      void put_out( char character ) ;
      put_out ( 0x1b) ;
      put_out ( 0x33) ;
      put_out ( 0x15) ;
      for (i=0; i<12; i++)
      {
            put_out(0x0A) ;
      background = readPixel (0,0);
      for (x=639; x>=7; x -=8)
      {
            printrow(x, background) ;
      }
      put_out ('\x0C') ;
}
printrow (int x, int background);
{
      unsigned char savechar, temp ;
      unsigned i, j, newy, y ;

      char status () ;
      void put_out();
      for (y=0; j=84; y<=479; y++, j++)
      {
            savechar = 0;
            for (i=0; i<8; i++)
            {
                  temp = readPixel (x – i, y) ;
                  if (temp != background)
```

```
                        savechar l = 1;
                  if (i != 7)
                        savechar <<=1;
            }
            out_buff [ j ] = savechar;
}

      for (i=0; i<485; i++)
      {
            put_out(out_buff [ i ]) ;
      }
      put_out('\r') ;
      put_out('\n') ;
}

char status ()
{
      char temp;
      temp = inportb (0x379) ;
      return (temp & 0x80) ;
}

void put_out ( char character)
{
      int temp;

      while (! Status () ) ;
      outportb ( 0x378, character);
      outportb ( 0x37A, 0x0D);
      outportb ( 0x37A, 0x0C);
}

int readPixel (int x, int y)
{
      #define DISPLAY_OUT (index, val ) {outp(0x3CE, index) ;\
                                         outp(0x3CF, val) ;}

      int i, j, color =0;
      unsigned char mask, exist_color;
      char far*base ;
      base = (char far *)  (0xA0000000L + ((long)y * 80L +
               ((long)y /8L))) ;
      mask = 0x80 >> (x % 8);

            for (i=0; i<4; i++)
      {
                  DISPLAY_OUT(4, i );
             DISPLAY_OUT(5, 0 );
             exist_color = *base & mask ;
                  if (exist_color != 0)
                      color l= 0x01<< i ;
             }
      return color;
}
```

```
main()
{
      setvect(5, print_handler);
      printf("\nVGA Graphic Screen Printing Routine"
                        "Installed\n") ;
      keep(0, 0x9FF); / / / /
}
```

Fig. 8. Programa para reemplazar la función IBM print screen con una función de gráficos print screen VGA.

Siguiente, nótese que todas las funciones que son parte de la interrupción son posicionadas al principio de el programa. El principal (main) está definido como tipo *interrupt.* Esto causara que la función sea compilada como un manejador de interrupción, incluyendo el grabado de todos los registros al principio y restaurándolos al final. El programa *main* es mínimo. La función *setVect* automáticamente repone la dirección en el vector de PCs con la dirección del programa tipo *interrupt* , por eso cuando la interrupción está activada, el programa es llamado. La función *setVect* requiere 2 parámetros: primero el número de la interrupción ha ser reemplazada y segundo el nombre de la función C que debe ser tipo *interrupt*. Siguiente se imprime una declaración que la interrupción ha estado reemplazando. La función *printf* toma mucho espacio de memoria pero esto no importa porque no se va a grabar esta parte del programa. Finalmente, la función *keep* termina pero mantiene el programa residente en memoria. Esta función tiene 2 parámetros. El primero no se usa, el segundo es el tamaño del programa a ser usado. Para encontrar lo que esto significa, se elimina temporalmente el programa principal (main) y compila lo que esta a la izquierda con la opción Turbo C seleccionada, El *linker* y el segmento de mapa.
Uno de los resultados de la compilación será activado como *prntinst.map* Si se ve el programa, dará el número hexadecimal del segmento más largo usado en el programa residente; usa eso en la declaración *define prog_size* al principio del programa. Ahora se compila todo el programa, incluyendo el *main* y pone los resultados en un archivo *prntinst.exe* en un directorio en una de las vías principales. Al principio de la sesión se podrá instalar la función nueva *printscreen* escribiendo *prntinst* . Después de eso cuando se oprima *print scr* este programa correrá e imprimirá los gráficos de pantalla.

Cada diferente modo grafico requiere su propio *print screen* programa residente. No se trate de usar uno con un modo diferente de gráficos que el que fué diseñado o se tendrán impresiones extrañas.

Desafortunadamente, las funciones *setVect* y *keep* no son soportadas por Microsoft C, por eso las funciones *prntinst* listadas en este capitulo no trabajan si se usa Microsoft C. Se necesita escribir las propias funciones usando los servicios ROM BIOS para reponer *setVect* y *keep* si se requiere imprimir rutinas que son compatibles con Microsoft C.