Universidad Nacional Autónoma de México

FACULTAD DE CIENCIAS

ANALISIS Y SINTESIS DE METODOS DE GRAFICACION EN LA COMMODORE

T E S I S

Que para obtener el título de

A C T U A R I O

p r e s e n t a n

LAURA FIERRO VAZQUEZ TEJEDA
SERGIO EDUARDO OCHOA PINZON

México, D. F. 1988

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

I N D I C E

INTRODUCCION

La graficación por computadora no consiste únicamente en representar resultados de ciertos valores como es el caso de las gráficas estadísticas. También existe otro tipo de gráficas que son imágenes de objetos que nos sirven, ya sea para dibujar paisajes o escenas en la animación de juegos o bien, para ilustrar cualquier idea o conocimiento que se quiera expresar.

Las gráficas o imágenes por computadora son una buena ayuda para poder desarrollar juegos de enseñanza, ya que uno de los objetivos en este tipo de juegos es que el usuario aprenda a través del juego y para que éste sea atractivo para el usuario, debe hacerse por medio de imágenes. Si por el contrario, el juego no motiva a las personas para utilizarlo, en poco tiempo lo abandonarán por completo y dicho juego no cumplirá con el objetivo de que el usuario aprenda o refuerce sus conocimientos.

El tratar de mejorar la más alta puntuación en un juego de aprendizaje, además del estímulo visual que pueden producir las imágenes presentadas, puede motivar a una persona para utilizar frecuentemente dicho juego. Para algunos niños dicha motivación puede ser la mejor solución para reforzar sus conocimientos jugando.

Aunque el objetivo en un juego no conlleva forzosamente a hacer imágenes muy sofisticadas, esto puede resultar ser un primer y gran obstáculo para que un juego de aprendizaje no logre motivar y por tal motivo sólo se quede en una idea.

La computadora puede ser un buen maestro, ya que no sólo se asegura que el usuario conteste bien, sino que además es capaz de llevar estadísticas de las respuestas emitidas por él y de esta manera el mismo usuario puede ir viendo el desarrollo que va teniendo.

Por otro lado una ventaja que tenemos al generar objetos o cuerpos geométricos por medio de una computadora, es que se le pueden hacer alteraciones, es decir, es posible cambiarlos de posición, expanderlos, contraerlos o rotarlos, sin necesidad de definirlos nuevamente. Resultaría muy interesante poderle aplicar a estos cuerpos geométricos las transformaciones u operaciones matemáticas, ya sea para obtener la animación en base a éstas o simplemente para ver que resulta de aplicar dichas transformaciones.

Debido al incremento en el uso de las computadoras en casa, la Commodore 64 ha sido una buena alternativa debido a su bajo costo y a su gran facilidad para poder crear imágenes a color y aunado a la introducción de esta computadora al mercado mexicano y a la aceptación que ésta ha tenido, su popularidad ha crecido considerablemente.

Esta tesis tiene como objetivo dar las bases necesarias para que una persona sin grandes conocimientos en computación, pueda hacer programas para generar objetos o imágenes por computadora. Con esto se pretende dar una referencia completa, útil y práctica para que los usuarios de la Commodore cuenten con información detallada y algo más profunda que lo que se ofrece normalmente, la cual es poca, superficial y generalmente dirigida a gente que no está

interesada en profundizar. Además trataremos de explica
algunas de la herramientas más usadas en la graficación.

CAPITULO 1

COMO SE FORMAN LAS GRAFICAS

I.1.- QUE ES LA GRAFICACION.

Las gráficas por computadora se forman por una serie de puntos y líneas que representan imágenes. Estas pueden ser dibujadas en blanco y negro o, si se desean hacer más atractivas, en color.

Hacer estas gráficas es como dibujar por números, es decir, cada punto coloreado en la pantalla corresponde a un número en la memoria.

Para cambiar el color de un punto sólo tendremos que cambiar el número almacenado en la localidad de memoria correspondiente. Para dibujar una imagen usando una computadora se deben colocar los números correspondientes en las localidades de memoria apropiadas.

En la Commodore por ejemplo, la pantalla está formada por 64,000 puntos y cada uno de éstos es afectado por una localidad de memoria específica. Al cambiar el color de los puntos por medio de dichas localidades, se obtienen figuras que previamente se diseñaron y codificaron de una manera especial.

La capacidad de graficación de la Commodore 64 se debe al microprograma o chip de interface de video 6567, también conocido como VIC-II. Este microprograma permite tener varios modos de graficación. Podemos trabajar

sobre la pantalla de texto que consta de 40 columnas por 25 líneas, o bién en una pantalla de alta resolución que consta de 320 puntos horizontales por 200 verticales. Otro recurso son los "sprites", que son figuras pequeñas en movimiento usadas frecuentemente en juegos.

Estos modos de gráficas pueden ser mezclados. Se puede combinar el modo gráfico de texto con el modo gráfico de alta resolución creando una figura detallada en la parte superior de la pantalla y letreros en la parte inferior. Los sprites pueden ser mezclados con cualquiera de los dos modos gráficos anteriores, facilitando esto la escritura en los juegos.

El modo gráfico más sencillo es sin duda el modo de texto. La sencillez en los conceptos no implica la sencillez en los resultados pues se pueden crear gráficas complejas en la pantalla con el modo de texto.

Crear gráficas en los modos de alta resolución y sprites es más difícil y toma más tiempo, pero la calidad y el detalle de éstas es mejor que en las gráficas creadas en el modo de texto.

I.2.- GRAFICACION GRUESA.

Llamamos graficación gruesa a todas aquellas gráficas o figuras que podemos crear con los 62 caracteres gráficos combinados con los 16 colores que tenemos disponibles en la Commodore 64. Cada caracter está formado por ocho puntos.

El apelativo de "graficación gruesa" se debe a que las imágenes que se elaboren no podrán mostrar mucho detalle.

Para crear gráficas o figuras en el modo de graficación gruesa, contamos con caracteres formados por cuadros de ocho por ocho puntos como se muestra en la siguiente figura:

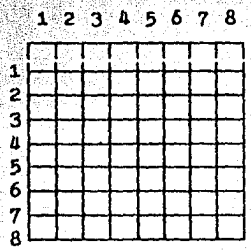


Figura 1.1. Area para dibujar un caracter

Dado lo reducido en el número de puntos muchas de las figuras que se hagan, serán poco detalladas y hasta burdas en ocasiones.

Como cualquier caracter está definido en un cuadro, nos encontramos con la limitación de no poder hacer figuras pequeñas, ya que cuando menos necesitaríamos de dos a cuatro cuadros para poder dibujar una imagen en la pantalla.

Otra de las limitaciones con la que nos podemos encontrar al trabajar con este tipo de gráficas es que en un momento dado no exista algún caracter que necesitemos cuando estemos creando una imagen.

I.3.- GRAFICACION FINA.

Otro camino para crear imágenes, es la graficación fina qu nos permite realizar figuras más detalladas y/o má pequeñas.

Mediante este tipo de graficación podemos realizar imágenes en dos cuadros de la pantalla y algunas veces hasta en ur solo cuadro, lo que nos permite crear una gran variedad de imágenes. Por ello podríamos asegurar que las limitaciones con las que nos encontremos para crear una figura, son casi inexistentes.

Para utilizar este tipo de graficación existen tres caminos que son:

a) Modificación de caracteres.

   Es posible modificar los caracteres convencionales que estemos utilizando. Si necesitamos una figura que sea posible dibujarla en uno o dos cuadros, podemos modificar el caracter que nosotros elijamos y cada vez que deseemos que aparezca dicha imagen sólo tendremos que oprimir la tecla de dicho caracter.

b) Sprites.

   Los sprites son un tipo especial de gráficas que pueden ser creadas en un cuadro de 21 por 24 puntos, es decir, 504 puntos. Dichos puntos se pueden ir prendiendo o apagando para ir conformando una imagen, la cual va a tener propiedades muy importantes para la animación, como la de poderla posicionar en cualquier parte de la pantalla o expanderla horizontal o verticalmente, entre otras.

c) Alta Resolución.

En la alta resolución podemos hacer gráficas muy detalladas ya que disponemos de los 64,000 puntos que forman la pantalla. El tamaño de la figuras que aquí se dibujen puede ser muy pequeño.

Este método de graficación es recomendable. Es posible obtener figuras muy detalladas, debido a que se van dibujando punto por punto y tiene la propiedad de que la figura que se obtenga puede ser posicionada en cualquier parte de la pantalla. Las figuras se obtienen afectando localidades de memoria especificas (apagando o prendiendo puntos en la pantalla, así como asignándoles color a cada uno de éllos).

I.4.- GRAFICACION CON MOVIMIENTO.

El mover una gráfica o imagen es más importante de lo que parece, ya que este desplazamiento de la imagen es lo que nos va a dar la animación o sensación de movimiento, haciendo que las imágenes parezcan más reales.

Una imagen puede ser desplazada por toda la pantalla pero el tipo de movimiento que se le pueda dar a ésta depende mucho del método que se haya utilizado para crear la gráfica. Los posibles desplazamientos que se le van a dar a la figura pueden ser tan largos o cortos, como se desee.

Una buena animación depende de los desplazamientos antes mencionados, ya que si éstos son lo suficientemente pequeños la sensación de movimiento será muy buena y si por el

contrario son muy largos no habrá efecto de animación.

Si una figura fue creada por cuadros, con la graficación gruesa, los movimientos se tendrán que realizar cuadro por cuadro, es decir, los desplazamientos serán de ocho en ocho puntos.

Otro método para dar sensación de movimiento en graficación gruesa es mediante la modificación de los caracteres. Además de poderlos desplazar por la pantalla, también es posible intercambiar dos o tres caracteres modificados en un mismo lugar de la pantalla cada determinado tiempo, lo que puede producir una buena sensación de movimiento.

Si las figuras fueron creadas por medio de los métodos de graficación fina, los desplazamientos de la figura pueden ser mucho más cortos, ya que se pueden hacer punto por punto.

El tiempo que se tarde en desplazar una figura o en cambiarla en el mismo lugar, es muy importante ya que de ello depende en mucho, un buen efecto de animación.

CAPITULO II

DESCRIPCION DE LA MEMORIA EN LA COMMODORE

II.1.- ORGANIZACION DE LA MEMORIA.

Llamaremos "mapa de la memoria" a una lista de las localidades de memoria y sus funciones. Las localidades se denotan con enteros positivos y se habla de localidades "altas" o "bajas" dependiendo de si los enteros que las denotan están cerca del entero máximo o mínimo respectivamente.

Para analizar el mapa de la memoria, empezaremos por la parte más alta que es donde se localiza el núcleo del sistema operativo, denominado Kernel.

El Kernel determina cómo debe comportarse la Commodore en sus funciones más primitivas. Por ejemplo, cuando uno se encuentra editando un programa en BASIC, la mayor parte del trabajo se hace con el teclado y la pantalla, que son controlados por el Kernel.

El Kernel está alojado permanentemente en la ROM (Read Only Memory) y tiene una capacidad de 8k. Esencialmente, es una tabla estandarizada de salto a rutinas de entrada, salida y manejo de memoria en el sistema operativo. Esta tabla está colocada en una porción de 256 bytes, los más altos, dentro de la memoria ROM.

El generador de caracteres ROM se encuentra localizado en la parte inferior inmediata al Kernel y ocupa 4k de memoria. Este generador de caracteres contiene todos los puntos o

información necesaria para poder generar cada caracter que la Commodore puede desplegar.

En esta misma parte de la memoria tenemos una segunda ROM que contiene rutinas de entradas y salida, es decir, pequeños programas que se encargan de manejar la impresora o el disco.

La Commodore puede escoger entre estas dos secciones de 4k de memoria cada una. Por ejemplo, si el Kernel necesita encontrar la información necesaria para desplegar un caracter, aquél acude al generador de caracteres. De no ser así, la Commodore acude a las rutinas de entrada y salida.

Para que esta área pueda realizar dichas funciones contiene:

- La interface para el video (VIC-II)
- La unidad de interface para el sonido (SID)
- Dos complejas interfaces para adaptadores (CIA) para manejar otras entradas y salidas como la palanca para juegos ("joystick"), entre otras.

Para poder hacer uso de estas tres interfaces, lo que tendremos que hacer es, por medio de la función PEEK y el procedimiento POKE afectar las localidades correctas.

La siguiente área, que ocupa 4k de memoria, es una parte de RAM (Random Acces Memory). Este tipo de memoria, a diferencia de la ROM, si puede ser modificada por el programador.

Esta parte de la memoria es muy usada, ya que se encuentra aislada de la mayor parte de la RAM y por lo tanto está libre para usar o guardar información acerca de las gráficas.

En los siguientes 8k podemos encontrar la ROM donde se encuentra el intérprete de BASIC. Este intérprete de BASIC es adecuado excepto por su falta de comandos que gobiernen las sofisticadas gráficas y los sonidos que produce la Commodore.

La parte baja de la memoria se encuentra formada por 40k de memoria que supuestamente se encuentra libre y es la que pertenece a la RAM. Sin embargo, no se encuentra del todo libre como nosotros lo desearíamos ya que parte de esta área de memoria es usada para almacenar los datos de la pantalla y algunos otros datos para las gráficas, como lo son los sprites y la alta resolución.

La parte superior de esta área la asignaremos para guardar datos de sonidos y gráficas, además de almacenar también ahí nuestros programas en BASIC.

La parte más baja de la memoria, está formada por 256 bytes y es llamada la página cero ("zero page"). Una página está constituída por 256 bytes y en la Commodore existen 256 páginas.

Esta es una vista muy general del mapa de la memoria y en éste pueden existir algunas variaciones como por ejemplo, es posible apagar la ROM completamente y hacer de la Commodore una máquina basada únicamente en la RAM.

Con programas que no están escritos en BASIC puede uno apagar la ROM y ganar un área adicional de 8k para espacio de datos o programas.

A continuación se muestra el mapa general de la memoria de la Commodore 64:

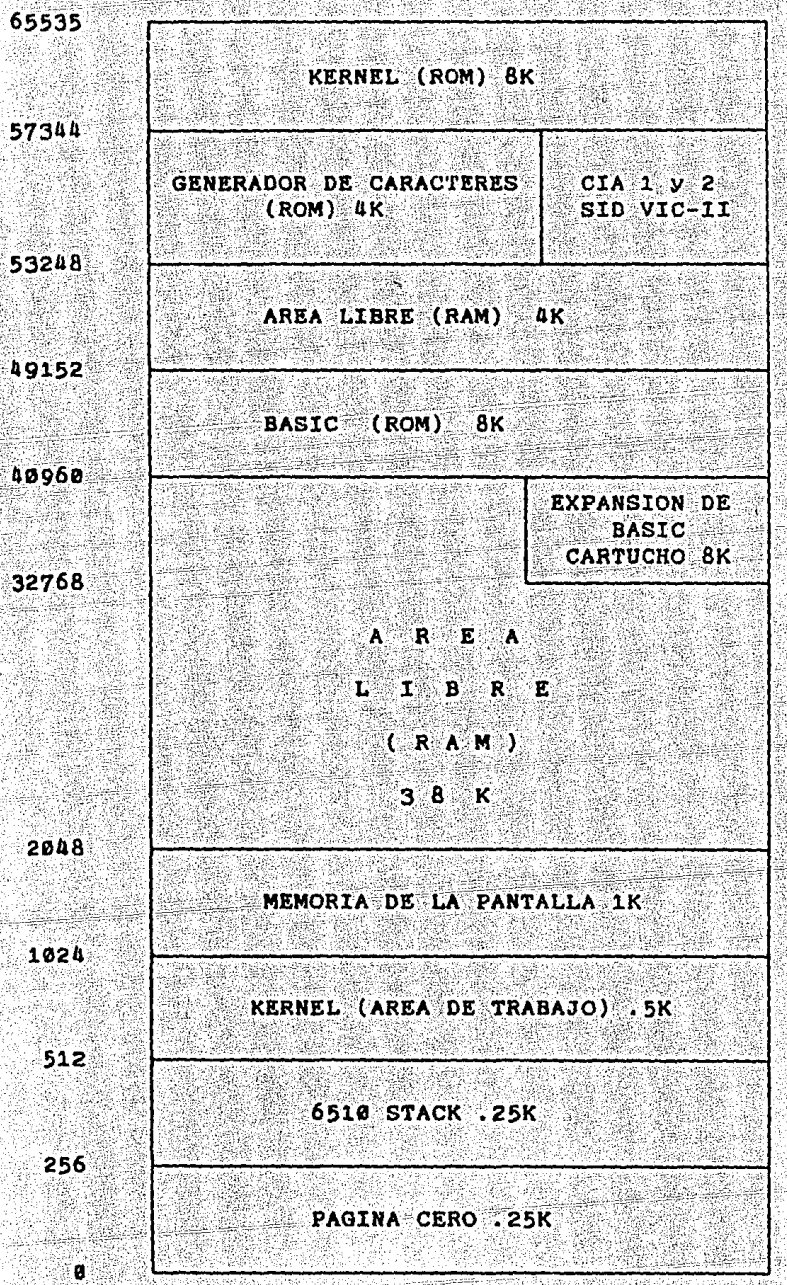


Figura 2.1. Mapa de la memoria.

código binario de un caracter. Después de reconocer el primer código de la pantalla, el procesador lee la segunda localidad de memoria, es decir, el byte 1, el siguiente caracter a la derecha de éste es el byte 2 y continúa este mismo proceso hasta llegar al byte 39. Una vez leído éste baja a la primera posición de la izquierda del segundo renglón, que corresponde al byte 40 y este renglón comprende hasta el byte 79. Después continúa leyendo los renglones siguientes con este procedimiento hasta llegar a los bytes 960 al 999, que corresponden al último renglón, donde termina la memoria de la pantalla. En menos de 1/60 de segundo se puede volver a iniciar este proceso.

Por medio de la función PEEK se puede desplegar en la pantalla, el código de el caracter que se está leyendo de la memoria de la pantalla y mediante el procedimiento POKE se puede cambiar lo que está siendo desplegado en la pantalla cambiando el valor de la localidad de memoria correspondiente.

II.2.2.- LA MEMORIA DE LOS CARACTERES.

Al igual que la memoria de la pantalla, la memoria de los caracteres ocupa una sección predeterminada de la memoria.

El código de la pantalla no es por sí solo suficiente para poder poner un caracter completo en la pantalla, ya que contiene solamente un índice que selecciona a un caracter del conjunto de caracteres.

La máquina dispone de 512 "dibujos" de símbolos o caracteres diferentes, formados por dos grupos de 128

caracteres y sus inversos correspondientes.

Cada dibujo de un caracter es guardado en una serie de 8 bytes, por lo que la memoria de los caracteres consta de 4,096 bytes (512x8).

Los ocho bytes que forman un caracter son guardados en el mismo orden que el código de la pantalla. El primer código de la pantalla es el 0, el cual corresponde al caracter de la "@", el segundo código de la pantalla es un 1, que a su vez corresponde al caracter de la "A". (Apéndice A)

Para encontrar el patrón de cualquier caracter en la memoria de los caracteres, todo lo que se tiene que hacer es multiplicar el código de la pantalla, correspondiente a ese caracter, por ocho. Por ejemplo: el código de la pantalla para la "Z" es 26, entonces el primer byte que define a este caracter será el byte 208 (8x26) en la memoria de los caracteres.

II.2.3.- LA MEMORIA DEL COLOR.

Además de la memoria de la pantalla existe otra memoria con su mapa respectivo, paralela a ésta, en la cual se tiene un control acerca de los colores que se despliegan en la pantalla.

Se puede seleccionar individualmente un color para cada caracter en la memoria de la pantalla, cambiando la localidad correspondiente en la memoria del color.

El mapa del código de la pantalla de los caracteres y el mapa de los códigos de los colores tienen una relación uno a

uno; siempre que sea llamado el caracter que está en el k-ésimo byte de la memoria de la pantalla, éste será desplegado con el color que existe o que tiene en ese momento el k-ésimo byte en la memoria del color.

Las tres memorias antes descritas quedarían localizadas en la memoria de la siguiente manera:

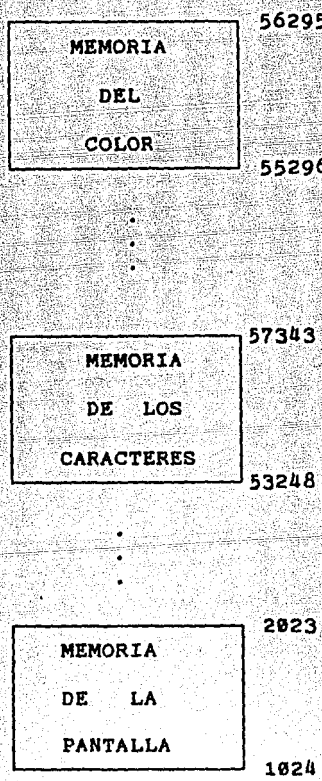


Figura 2.2 . Localidades correspondientes a la la memoria de la pantalla, la memoria de los caracteres y la memoria del color.

II.3.- USO DE LA FUNCION PEEK Y EL PROCEDIMIENTO POKE.

La función PEEK y el procedimiento POKE permiten el acceso directo a posiciones de memoria, pasando por encima de los resguardos que en este sentido tienen los lenguajes de alto nivel como BASIC.

Un uso muy generalizado en las microcomputadoras y en particular en la Commodore es la de leer datos de música, sonidos o gráficas y en general datos que se usan para controlar el funcionamiento de la computadora.

El procedimiento POKE se usa para almacenar un número en una localidad de memoria entre la 0 y la 65535. El valor que se guarde en dicha localidad debe ser un entero entre 0 y 255. La palabra POKE deberá ir seguida de una localidad de memoria que se va a afectar y el valor que se va a guardar en ella separados por una coma. Por ejemplo, para cambiar el color del fondo de la pantalla a blanco debe guardarse en la localidad 53281 un 1, que es el valor correspondiente a este color.

```
POKE 53281,1
```

La función PEEK sirve para leer el valor que se encuentra en una determinada localidad de memoria. Esta función no puede alterar los valores. Su invocación deberá ir seguida de la localidad de memoria que se quiere leer entre paréntesis y en el contexto de una expresión. Por ejemplo, si deseamos saber el valor que se encuentra en la localidad 53281 deberemos escribir:

```
PRINT PEEK(53281)
```

Tanto la función PEEK como el procedimiento POKE tienen como

argumento una localidad de memoria a la que utilizan como un todo. Para muchos propósitos esto no es suficiente ya que en muchos de los casos lo único que se tiene que hacer es cambiar una parte de un byte y en otros sólo un bit, para lo cual se utilizan los operadores lógicos AND y OR.

II.4.- OPERACIONES BINARIAS MEDIANTE EL AND Y EL OR.

Un problema al que nos enfrentamos es el de la necesidad de poder apagar un solo punto (bit) en la pantalla (mediante la función PEEK o el procedimiento POKE no podemos cambiar un solo bit, sino todo un byte). Si nosotros queremos cambiar un punto en la pantalla, tendremos que introducir mediante el procedimiento POKE el byte completo, controlando los ocho puntos de éste y no sólo uno.

La Commodore contempla algunos comandos que nos permiten tomar un byte de la memoria de la pantalla, cambiar individualmente un bit o más y regresar el byte modificado a su lugar de origen. En general los operadores lógicos de las computadoras ofrecen estas facilidades.

AND.- Cuando usamos una expresión lógica como A=5 AND 3, la palabra AND causa una operación binaria en la que dos números se comparan bit a bit.

En el momento en el que se comparan dos números por medio de un AND, se está buscando un 1 en la misma posición de ambos números, es decir, compara un bit del primer número con el bit correspondiente en el segundo número. Cuando está comparando dos números y se encuentra en ambas posiciones un

1 el resultado de la operación es 1. En cualquier otro caso el resultado es 0, como se muestra en el siguiente ejemplo:

```
        bit   7 6 5 4 3 2 1 0

          5   0 0 0 0 0 1 0 1
  AND     3   0 0 0 0 0 0 1 1
            -----------------
              0 0 0 0 0 0 0 1
```

Por lo que tenemos que (5 AND 3) = 1.

OR.- La operación tiene como argumentos dos números, al igual que la operación AND, sólo que ahora busca que alguno de los dos números o los dos sean 1, siendo en este caso el resultado 1 y 0 en otro caso, como se muestra en el siguiente ejemplo:

```
        bit   7 6 5 4 3 2 1 0

          5   0 0 0 0 0 1 0 1
  OR      3   0 0 0 0 0 0 1 1
            -----------------
              0 0 0 0 0 1 1 1
```

Por lo que (5 OR 3) = 7.

II.4.1.- USO DEL AND Y EL OR.

Como ya vimos anteriormente, usamos el AND y el OR para prender o apagar uno o más bits y respetar los restantes bits del byte. Para ello construímos lo que se conoce como una "máscara" de bits en un byte que se usa como uno de los operandos con estos operadores lógicos. El otro operando es el byte que se desea manipular.

El OR es usado para prender uno o más bits, respetando los demás. Si deseamos encender el n-ésimo bit de un byte, el número que tenemos que usar como argumento para el OR será

2^n, es decir, si deseamos encender el bit 1 tenemos que $2^1=2$ por lo que la expresión será OR 2. Aquellos bits que deseemos dejar inalterados en el byte original basta operarlos, bit a bit, con ceros. Al efectuarse la operación binaria tendríamos:

```
     bit   7 6 5 4 3 2 1 0

           x x x x x x x x
  OR  2    0 0 0 0 0 0 1 0
           ---------------
           x x x x x x 1 x
```

donde x puede ser 0 o 1.

Podemos notar que el único bit que estamos encendiendo es el bit 1, respetando el valor de los demás bits. El 0 en la operación OR respeta el estado anterior del primer operando, como ya lo mencionamos.

En el caso de que se desee encender más de un bit sólo tendremos que encontrar el valor decimal de los bits que queremos encender, sumar éstos y el resultado usarlo como argumento para el OR, tomando en cuenta que, numerados los bits de derecha a izquierda a partir del 0, el i-ésimo bit tiene un peso o valor de 2^i.

Para prender todos los bits del byte, el argumento que tenemos que usar es 255 ya que:

$$255 = 1+2+4+8+16+32+64+128$$

$$255 = 2^0+2^1+2^2+2^3+\ldots+2^7$$

El AND, por el contrario del OR, es usado para apagar los bits individualmente. Si deseamos apagar un punto en la pantalla, sólo tendremos que poner un cero en el bit que

nosotros tendremos que modificar sólo los bits entre el 0 y 3 de esta localidad.

Similarmente, para cambiar el modo de 25 renglones que tiene normalmente la pantalla, al modo de 24 renglones, tendremos que cambiar sólo el bit 3 de la localidad 53265 sin cambiar los demás bits.

Si vemos cual es el número que contiene la localidad 53265 tendremos que:

```
PRINT PEEK(53265)
27
```

Lo que nos indica que tenemos el número 27 en dicha localidad y que en binario sería el siguiente:

```
bit   7 6 5 4 3 2 1 0
27    0 0 0 1 1 0 1 1
```

Eso nos dice que el bit 3 de la localidad está prendido, lo que significa que está el modo de 25 renglones y para pasar al de 24 debemos apagar este bit únicamenete, independientemente de los valores que tengan los demás bits. Para apagar dicho bit tendremos que restar $2^3=8$ del valor que existe en esa localidad, por lo que tendremos 27-8=19. Esto lo tendremos que hacer de la siguiente manera:

```
POKE 53265,PEEK(53265)-8
```

Así podemos pasar al modo de 24 renglones en la pantalla. Para volver al modo de 25 renglones tendremos que ejecutar la operación inversa que sería:

```
POKE 53265,PEEK(53265)+8
```

La otra manera es afectando únicamente los bits involucrados utilizando los operadores lógicos AND y OR. Las operaciones

equivalentes para lo que hicimos anteriormente, pero usando los operadores lógicos serían:

La operación binaria equivalente, para pasar al modo de 24 renglones sería:

```
POKE 53265,PEEK(53265) AND 247
```

Ya que es el tercer bit el que deseamos apagar ($2^3=8$) tenemos que 255-8=247 y la operación que estaríamos efectuando con el AND sería la siguiente:

```
    BIT          7 6 5 4 3 2 1 0
PEEK(53265)      0 0 0 1 1 0 1 1   = 27
  AND 247        1 1 1 1 0 1 1 1
                 ---------------
                 0 0 0 1 0 0 1 1   = 19
```

La operación inversa que nos regresaría al modo de 25 renglones sería:

```
POKE 53265,PEEK(53265) OR 8
```

Se utiliza el 8 ya que este valor en binario corresponde a un byte que tiene únicamente el bit 3 encendido, que es el que nos interesa prender. En caso de que ya se encuentre encendido, con el OR conservará su valor original.

La operación binaria que estaríamos efectuando con el operador OR sería la siguiente:

```
    BIT          7 6 5 4 3 2 1 0
PEEK(53265)      0 0 0 1 0 0 1 1   = 19
  OR 8           0 0 0 0 1 0 0 0
                 ---------------
                 0 0 0 1 1 0 1 1   = 27
```

Hay que hacer notar que por medio de estas operaciones el único bit que se estuvo cambiando fue el tercero. En este caso, que sólo modificamos un solo bit resulta simple

hacerlo con sumas y restas. Sin embargo, al sumar y restar determinadas cantidades tenemos que garantizar el estado anterior; el o los bits correspondientes deben estar prendidos para restar o apagados para sumar. Si no es así, el resultado es impredecible. Cuando nos encontramos en el caso de cambiar no sólo un bit sino varios, a veces no consecutivos y no podemos garantizar cuál es el estado ó configuración de los mismos antes del cambio, resulta más fácil y seguro hacerlo por medio de los operadores lógicos AND y OR.

II.5.- LOCALIDADES BASE PARA LAS GRAFICAS.

El microprograma VIC-II no tiene capacidad para manejar 64k. Debido a esto el CPU 6510 y el VIC-II usan la memoria como si estuviera dividida en 4 bancos de 16k cada uno como sigue:

BANCO	LOCALIDAD
0	0 - 16383
1	16384 - 32767
2	32768 - 49151
3	49152 - 65535

Figura 2.3. Localidades correspondientes a cada uno de los cuatro bancos de memoria en que se divide la Commodore 64.

El VIC-II puede leer de cualquiera de los cuatro bancos solamente que uno a la vez. Esto significa que si colocamos el mapa de bits en el banco 3, entonces la memoria de la pantalla deberá estar en este mismo banco.

Para indicarle al VIC-II cuál banco usar, tenemos los bits 0 y 1 de la localidad 56576.

BANCO	VALOR DE LOS BITS 0 Y 1 DE LA LOCALIDAD 56576	
	BINARIO	DECIMAL
0	11	3
1	10	2
2	01	1
3	00	0

Figura 2.4. Valor de los bits 0 y 1 de la localidad 56576 para indicar al VIC-II cuál banco se va utilizar.

La instrucción POKE 56576,1 indicará al VIC-II que use el banco 2, que comienza en la localidad 32768.

A continuación se da una breve descripción de cada uno de los bancos.

BANCO 0:

Esta área se usa normalmente para las variables del sistema y los textos de programas en BASIC.

De la localidad 1024 a la 2048 están reservadas por "default" para la memoria de la pantalla.

Una restricción de este bloque es que todos los datos a los que el VIC-II pueda tener acceso deben estar dentro del mismo bloque de 16k, incluyendo la información del generador de caracteres ROM, el cual le dice al VIC-II como dibujar la forma de cada caracter en la pantalla.

Como la memoria ROM no puede colocarse en la mitad del área

de BASIC se requiere de un truco. Este truco consiste en que el VIC-II ubica al generador de caracteres ROM en las localidades 4096-8191, a pesar de que el microprocesador 6510 localiza esta parte de ROM a partir de la localidad 53248. Por lo tanto mientras el 6510 usa la memoria RAM en estas localidades para textos de programas, el VIC-II ve sólo la memoria de ROM y no pone atención a lo que se encuentra en esas localidades de memoria RAM. Debido a lo anterior, esta parte de la memoria no está disponible para sprites, caracteres definidos por el usuario o para la memoria de la pantalla.

El área de la localidad 679 a la 767 no tiene ningún uso, por lo que puede emplearse para definir un sprite o para colocar los datos de 11 caracteres.

De la localidad 828 a la 1019 se encuentra el buffer del cassette. Como de la localidad 820 a la 827 y de la 1020 a la 1023 no tiene ningún uso, entonces el área formada por estas localidades, aunada al área designada para el buffer del cassette estaría disponible (si no se usa la grabadora de cassette) para almacenar información de gráficas. Pudiendo colocar 3 sprites o la información para 25 caracteres.

Para la alta resolución, una solución es usar una parte del área destinada para los programas en BASIC. Esto puede hacerse disminuyendo el tope de la memoria usada para los programas, con lo cual se protege la parte más alta de la memoria de una colisión con BASIC.

Las localidades 55 y 56 apuntan a la localidad más alta

usada para programas de BASIC. Por ejemplo, la expresión POKE 56,32 cambia el tope de BASIC a la localidad 8192 (32X256) y de esta manera pueden usarse de la localidad 8192 a la 16384 para gráficas de alta resolución, para nuevos conjuntos de caracteres, para sprites o para alternar pantallas de texto.

Otra alternativa es correr la localidad donde principia BASIC y marcar la memoria inferior hasta este punto como protegida. Por ejemplo pueden colocarse los 8K de la pantalla de alta resolución a partir de la localidad 8192 a la 16384 y proteger esa memoria de BASIC, dejando libres 24k para programas.

BANCO 1:

Esta sección se usa normalmente para almacenar programas de BASIC. Al usar este bloque el VIC-II no tiene acceso al generador de caracteres ROM.

Si se reduce el límite superior de BASIC, esta área puede usarse para datos de sprites, caracteres y gráficas de alta resolución. Las desventajas de usar este bloque es que se pierde el acceso al generador de caracteres ROM además de que se reduce a 14k la memoria para programas de BASIC. La ausencia de los caracteres de ROM puede no afectar mucho, ya que se puede conectar la memoria ROM y copiar todo o una parte del conjunto de caracteres a RAM.

Debido a que este bloque de 16k es el único formado totalmente de memoria RAM, es una buena elección usarlo para

gráficas de alta resolución. Usando los últimos 9k para la pantalla de alta resolución y el mapa del color, quedarán 21k libres para el programa.

La falta de la memoria de caracteres ROM no es importante para la alta resolución y en cambio si es una ventaja porque permite escoger cualquiera de las dos secciones de 8K.

BANCO 2:

Este bloque consiste de 8k de memoria RAM, mitad de la cual es "vista" por el VIC-II como la memoria de caracteres ROM, y los 8k restantes como pertenecientes al intérprete de BASIC (ROM).

Debido a esto, no es posible usar la pantalla de alta resolución si se coloca en esta área, sin hacer uso de trucos en lenguaje de máquina.

Sin las localidades 36864 a la 40959 usadas por el generador de caracteres ROM y de la 40960 a la 49151 usadas por el intérprete de BASIC, únicamente hay 4K verdaderamente libres para usarse para la memoria de la pantalla, los cuales no son suficientes para una pantalla de alta resolución. Por lo tanto no es recomendable usar este bloque para gráficas de alta resolución.

BANCO 3:

Este bloque contiene normalmente 4K de memoria RAM que no es usada por el sistema, 4K de registros de entrada y salida y 8K de memoria ROM ocupados por el Kernel del sistema

operativo. Es muy conveniente usarlo cuando se necesita mucha memoria para gráficas. En primer lugar está lo suficientemente arriba del área usada por BASIC, por lo cual no es necesario cambiar los apuntadores para proteger las gráficas del BASIC y no tiene que limitarse la memoria para los programas. Además, si se usa este bloque, puede usarse para los programas de BASIC de la localidad 1024 a la 2048 debido a que no se usarán para la memoria de la pantalla y de esta manera se gana 1K. En segundo lugar tiene suficiente memoria RAM libre para cuatro pantallas de texto, mientras que el área de ROM puede usarse para guardar dos conjuntos de caracteres y los datos de 64 sprites simultáneamente.

Aunque la memoria de caracteres ROM no está disponible, puede copiarse a los últimos 4K debajo del Kernel (mediante un programa en lenguaje de máquina).

De lo anterior puede deducirse que el mejor banco para alojar las gráficas de alta resolución es el banco 1.

La primera localidad de cada banco la usaremos como la dirección o localidad base para las gráficas, es decir, como un apuntador al inicio del banco. Esto nos ayuda para ir obteniendo las demás localidades del banco por medio de desplazamientos, además de tener la facilidad de pasar eventualmente de un banco a otro, pues las direcciones están dadas relativas al inicio del banco, y no en forma absoluta a las localidades de la memoria.

CAPITULO III

GRAFICACION GRUESA EN LA COMMODORE

III.1.- LA PANTALLA.

La pantalla de la Commodore 64 puede pensarse y ser utilizada como una hoja cuadriculada de 40 columnas por 25 renglones en la cual cada cuadro puede ser ocupado por cualquier caracter, sea gráfico o no.

Uno puede diseñar su propia figura en la pantalla, seleccionando el color y caracter deseado utilizando los espacios y las teclas que controlan el cursor, posicionando de esta manera los caracteres en el lugar deseado. Así se podrá tener una idea exacta de la figura que se obtendrá finalmente.

Una vez terminado este boceto hay que traducirlo a un programa que solamente incluye la proposición PRINT para posicionar por renglones los caracteres, en el color y lugar deseado.

A continuación se muestra un programa utilizando los caracteres gráficos para crear una figura

```
2    REM ************************************************
4    REM *          PROGRAMA PARA DESPLEGAR UN          *
6    REM *          LETRERO DENTRO DE UN CUADRO         *
7    REM *       UTILIZANDO LOS CARACTERES GRAFICOS     *
8    REM ************************************************
10   PRINT "<SHFT CLR/HOME>"
20   FOR I = 1 TO 7
30     PRINT
40   NEXT I
50   PRINT SPC(9);"<COMM A><18 SHFT C><COMM S>"
60   PRINT SPC(9);"<SHFT B><18 SPC><SHFT B>
70   PRINT SPC(9);"<SHFT B><3 SPC>ESTE LETRERO<3 SPC><SHFT
     B>"
80   PRINT SPC(9);"<SHFT B><5 SPC>APARECERA<4 SPC><SHFT B>"
90   PRINT SPC(9);"<SHFT B><3 SPC>DENTRO DE UN<3 SPC><SHFT
     B>"
100  PRINT SPC(9);"<SHFT B><6 SPC>CUADRO<6 SPC><SHFT B>"
110  PRINT SPC(9);"<SHFT B><18 SPC><SHFT B>"
120  PRINT SPC(9);"<COMM Z><18 SHFT C><COMM X>"
130  END
```

Programa 3.1

III.1.1.- GRAFICAS POR MEDIO DE LA FUNCION CHR$.

Otra manera para desplegar los caracteres gráficos y letras en la pantalla es usando la función CHR$. Cuando uno manda imprimir el caracter (CHR$) correspondiente a un número que se encuentre entre 0 y 255, el caracter correspondiente en el código ASCII será desplegado en la pantalla.

También existe una función inversa para CHR$, que es muy útil cuando se desconoce el código de un caracter. Esta es la función ASC, que al mandarla a imprimir nos entrega el código ASCII correspondiente a ese caracter.

A continuación mostramos el programa 3.2, que produce el mismo efecto que el programa 3.1, con la diferencia que este programa se ejecuta usando la función CHR$.

```
2     REM ***********************************************
4     REM *        PROGRAMA  PARA DESPLEGAR UN          *
6     REM *        LETRERO DENTRO DE UN CUADRO          *
7     REM *        UTILIZANDO LA FUNCION  CHR$          *
8     REM ***********************************************
10     PRINT CHR$(147)
20     FOR I = 1 TO 7
30       PRINT
40     NEXT I
50     PRINT SPC (9);CHR$(176);
60     FOR I = 1 TO 18
70       PRINT CHR$(195);
80     NEXT I
90     PRINT CHR$(174)
100    PRINT SPC(9);CHR$(194);SPC(18);CHR$(194)
110    PRINT SPC(9);CHR$(194);"   ESTE LETRERO   ";CHR$(194)
120    PRINT SPC(9);CHR$(194);"     APARECERA    ";CHR$(194)
130    PRINT SPC(9);CHR$(194);"   DENTRO DE UN   ";CHR$(194)
140    PRINT SPC(9);CHR$(194);"      CUADRO      ";CHR$(194)
150    PRINT SPC(9);CHR$(194);SPC(18);CHR$(194)
160    PRINT SPC(9);CHR$(173);
170    FOR I = 1 TO 18
180      PRINT CHR$(195);
190    NEXT I
200    PRINT CHR$(189)
210    END
```

Programa 3.2

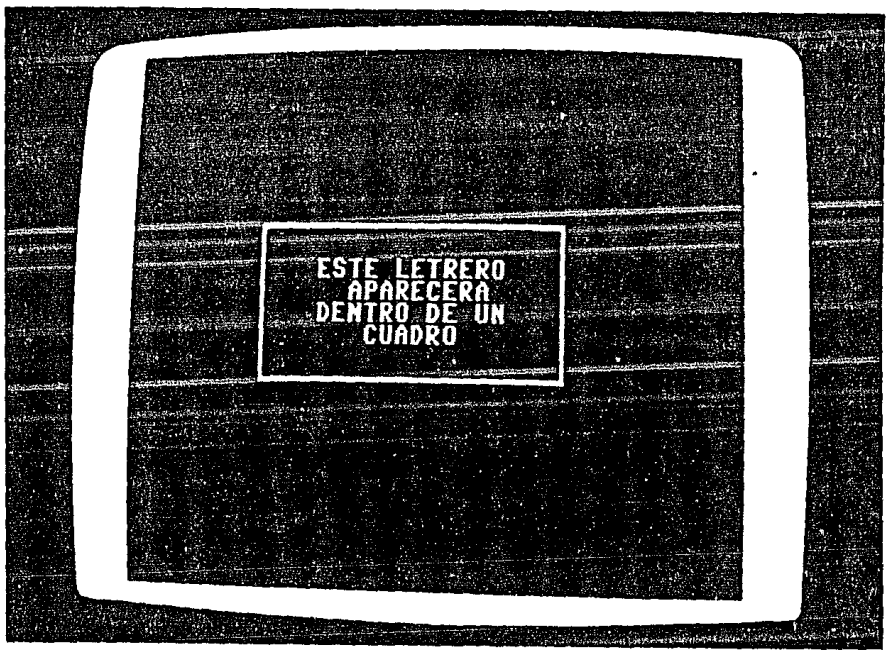


Figura 3.1. Imagen generada por los programas 3.1 y 3.2

III.1.2.- GRAFICAS POR MEDIO DEL PROCEDIMIENTO POKE.

Cuando uno crea gráficas mediante la proposición PRINT se manejan una serie de datos de una manera secuencial y los caracteres son desplegados de la misma manera, comenzando desde un lugar determinado en la pantalla. Este método requiere de más tiempo para elaborar el programa y casi siempre resulta en un programa más grande.

Aún cuando la proposición PRINT puede ser usada para crear una variedad de gráficas a color, existe otro método que es mucho más versátil y en muchas ocasiones más fácil de usar. Este método consiste en alterar las localidades de memoria. Para poder entrar directamente a estas localidades, cuando uno crea gráficas en la pantalla, se utiliza como ya dijimos, el procedimiento POKE. Este método es el más usado para crear gráficas en la Commodore 64.

El propósito de este procedimiento, como ya se vió anteriormente, es el de afectar las localidades de memoria. De esta manera se controla directamente cada localidad de la pantalla.

Aunque se puede afectar cualquier localidad entre 0 y 65535 y guardar un valor entre 0 y 255, sólo algunas localidades y valores son usados frecuentemente para crear gráficas.

La memoria es una larga sucesión de localidades y una sección de ésta es usada por la pantalla. Estas localidades son asociadas a la pantalla desde el sistema operativo, pero son iguales a cualesquiera otras posiciones de memoria excepto que el sistema operativo no las asigna para variables o programas.

Por medio del procedimiento POKE se pueden desplegar en la pantalla 1,000 caracteres en un campo de 40 columnas por 25 renglones, que corresponden a 1000 localidades de memoria reservadas para manejar lo que aparece en la pantalla.

Como ya se mencionó anteriormente, cada localidad de memoria puede aceptar un valor entre 0 y 255 y cambiando este valor uno puede seleccionar dónde y qué se va a desplegar en la pantalla.

La memoria de la pantalla en la Commodore 64 normalmente comienza en la localidad 1024 y termina en la 2023, siendo la primera localidad la que corresponde a la esquina superior izquierda de la pantalla, mientras que la última corresponde a la esquina inferior derecha. Si uno quiere colocar un caracter en el centro de la pantalla, es decir, en la columna 20 y el renglón 12, hay que encontrar la localidad de memoria que corresponde a esa posición. Para ésto utilizamos un polinomio de direccionamiento que transforma la localización de un elemento de una matriz, en este caso la pantalla, al elemento correspondiente de un vector, en este caso la memoria.

Para encontrar dicha localidad se multiplica el número de renglón por 40 y sumándole el número de columna nos posiciona en el cuadro deseado, es decir, 12*40+20=500. Si a dicha cantidad le sumamos la primera localidad de memoria de la pantalla (1024) nos da exactamente la localidad de memoria que deseamos afectar.

El polinomio de direccionamiento para encontrar cualquiera

de las 1000 localidades de la memoria de la pantalla que se desea afectar es:

LOCALIDAD = 1024 + 40 * RENGLON + COLUMNA

III.2.- CONTROL DEL COLOR.

Cuando se enciende la computadora se despliega por "default" una combinación de un solo color: los textos y el borde de la pantalla en azul claro y en azul oscuro el fondo de la misma.

Los colores de los caracteres pueden ser cambiados individualmente, sean gráficos o no. También es posible cambiar el color del fondo de la pantalla o del borde de ésta.

Existen algunas maneras de cambiar el color de los caracteres, pero para cambiar el color de la pantalla o del borde sólo es posible hacerlo a través de localidades de memoria específicas, sobre las cuales hablaremos posteriormente.

Las teclas del color contenidas en el teclado funcionan oprimiendo simultáneamente las teclas de COMMODORE o CONTROL, según sea el caso, junto con el número del color correspondiente.

En la Commodore 64 contamos con 16 colores distintos y a continuación mostramos una tabla de éstos, así como las teclas que los producen.

<CTRL 1> - NEGRO	<COMM 1> - NARANJA
<CTRL 2> - BLANCO	<COMM 2> - CAFE
<CTRL 3> - ROJO	<COMM 3> - ROJO CLARO
<CTRL 4> - AZUL CIELO	<COMM 4> - GRIS OSCURO
<CTRL 5> - PURPURA	<COMM 5> - GRIS MEDIO
<CTRL 6> - VERDE	<COMM 6> - VERDE CLARO
<CTRL 7> - AZUL	<COMM 7> - AZUL CLARO
<CTRL 8> - AMARILLO	<COMM 8> - GRIS CLARO

Figura 3.2. Teclas que producen los 16 colores que existen en la Commodore 64.

III.2.1.- COMO COLOREAR TEXTOS.

El cambiar el color a los caracteres resulta fácil y existen varias maneras de hacerlo, pero la más sencilla de todas es por medio de las teclas que controlan el color (Figura 3.2). Al presionar estas teclas el cursor cambiará al color respectivo y todo lo que se escriba será del color que tiene el cursor en el momento de escribir.

La manera de mandar escribir un texto de colores es mediante la proposición PRINT, poniendo como parte del texto el caracter de control que corresponde a un color. El texto será desplegado en el color que se haya seleccionado. De igual manera sucede con los caracteres gráficos. Todo ello se puede usar como lo muestra el siguiente programa, que cambia directamente los colores en el texto.

```
2   REM ***********************************************
4   REM *          PROGRAMA PARA CAMBIAR UN           *
6   REM *          TEXTO DE COLOR UTILIZANDO          *
7   REM *   LAS TECLAS QUE  CONTROLAN  EL COLOR       *
8   REM ***********************************************
10  PRINT "<SHFT CLR/HOME>"
20  PRINT : PRINT : PRINT
30  PRINT "<CTRL 1>     ESTE RENGLON APARECERA EN NEGRO"
40  PRINT
50  PRINT "<CTRL 2>     ESTE RENGLON APARECERA EN BLANCO"
60  PRINT
70  PRINT "<CTRL 8>     DE ESTA MANERA PUEDE SER CAMBIADO"
80  PRINT
90  PRINT "<COMM 5>     EL COLOR DEL TEXTO, NOTE QUE EL"
100 PRINT
110 PRINT "    CAMBIO DEL COLOR NO ES SOLO POR UN"
120 PRINT
130 PRINT "<COMM 7>     RENGLON, SINO HASTA SER ENCONTRADO"
140 PRINT
150 PRINT "<COMM 6>     OTRO COMANDO.<COMM 7>"
160 END
```

Programa 3.3

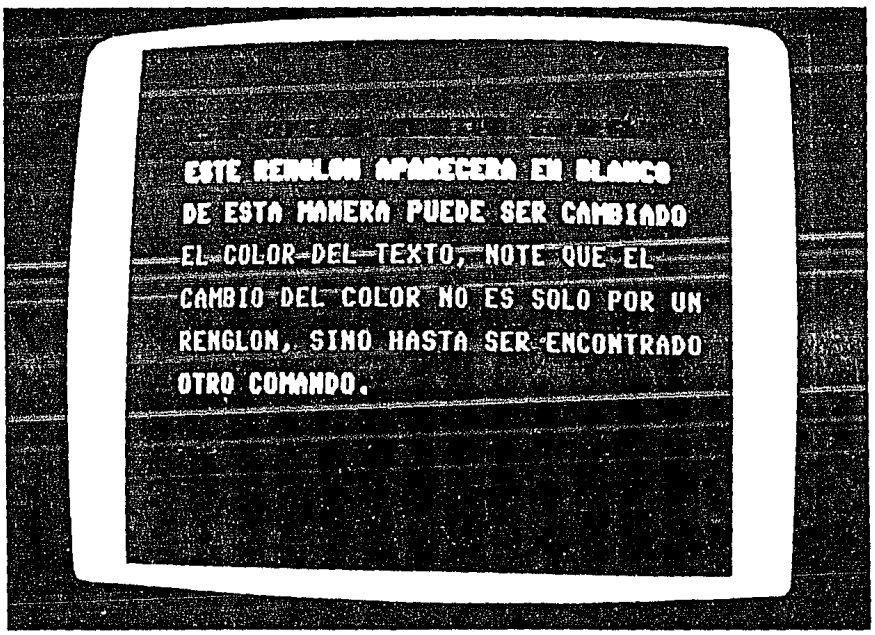


Figura 3.3. Imagen generada por los programas 3.3 y 3.4

Otra manera para desplegar los caracteres con distintos colores en la pantalla es usando la función CHR$, de la cual ya se explicó su funcionamiento.
A continuación mostramos una tabla del código ASCII que corresponde a cada uno de los 16 colores.

ASCII	COLOR	ASCII	COLOR
144	NEGRO	129	NARANJA
5	BLANCO	149	CAFE
28	ROJO	150	ROJO CLARO
159	AZUL CIELO	155	GRIS OSCURO
156	PURPURA	152	GRIS MEDIO
30	VERDE	153	VERDE CLARO
31	AZUL	154	AZUL CLARO
158	AMARILLO	151	GRIS CLARO

Figura 3.4. Código ASCII de los 16 colores.

El siguiente programa es equivalente al programa 3.3, con la diferencia que este programa se ejecuta usando la función CHR$.

```
2   REM ************************************************
4   REM *          PROGRAMA PARA CAMBIAR UN              *
6   REM *               TEXTO DE COLOR                   *
7   REM *          UTILIZANDO LA FUNCION CHR$            *
8   REM ************************************************
10  PRINT CHR$(147)
20  PRINT : PRINT : PRINT
30  PRINT CHR$(144)"    ESTE RENGLON APARECERA EN NEGRO"
40  PRINT
50  PRINT CHR$(5)"    ESTE RENGLON APARECERA EN BLANCO"
60  PRINT
70  PRINT CHR$(158)"    DE ESTA MANERA PUEDE SER CAMBIADO"
80  PRINT
90  PRINT CHR$(152)"    EL COLOR DEL TEXTO, NOTE QUE EL"
100 PRINT
110 PRINT "   CAMBIO DEL COLOR NO ES SOLO POR UN"
120 PRINT
130 PRINT CHR$(154)"    RENGLON, SINO HASTA SER ENCONTRADO"
140 PRINT
150 PRINT CHR$(155)"    OTRO COMANDO."CHR$(154)
160 END
```

Programa 3.4

III.2.2.- COMO CAMBIAR EL COLOR A LOS CARACTERES.

Es posible cambar el color de un caracter, si por medio del procedimiento POKE se almacena el código del color en la localidad de memoria correspondiente.

La memoria del color se distribuye de la misma manera que la memoria de la pantalla y por consiguiente ambas poseen la misma longitud, solamente que la memoria del color comienza en la localidad 55296 y termina en la localidad 56295.

Hay que hacer notar que la diferencia entre la primera localidad de la memoria de la pantalla y la primera localidad de la memoria del color es de 54272 localidades, por lo que cuando ya se posicionó una figura en una localidad de la memoria de la pantalla, sólo bastará sumarle a esta localidad 54272, para encontrar la localidad correspondiente en la memoria del color.

Otra manera de encontrar la localidad de la memoria del color que se quiere afectar es mediante el polinomio de direccionamiento que utilizamos en la memoria de la pantalla, sólo que ahora se sustituye 1024 por 55296, quedando el polinomio de la siguiente manera:

LOCALIDAD = 55296 + 40 * RENGLON + COLUMNA

Existen dos localidades de memoria que nos sirven para el control del color de la pantalla. Esta son la 53280 y la 53281. La primera de ellas controla el color del borde de la pantalla y la segunda controla el color del fondo de la misma. Es decir, si almacenamos un 2 en la localidad 53281 por medio del procedimiento POKE 53281,2 se cambiará a rojo

el color del fondo de la pantalla y si se quisiera cambiar el color del borde de la pantalla al mismo color que el del fondo de ésta, se tendría POKE 53280,2.

Utilizando el cambio de colores es posible hacer que los letreros y las figuras puedan tener un efecto de parpadeo y por lo tanto un efecto de animación. Si dibujamos y cambiamos el color de una figura al color del fondo de la pantalla, logramos un efecto de borrar dicha figura y si dibujamos la misma figura en otra posición o en un lugar inmediatamente contiguo se logra un efecto de movimiento de la figura.

El siguiente programa muestra cómo se cambia el color del borde y del fondo de la pantalla, además de cambiar el color de algunos de los caracteres de un letrero en movimiento.

```
2   REM ************************************************
4   REM *   PROGRAMA  PARA  DESPLEGAR  UN LETRERO FIJO  *
6   REM *   Y OTRO EN MOVIMIENTO CAMBIANDO EL COLOR EN  *
8   REM *                 UNA PARTE DE ELLOS            *
9   REM ************************************************
20  INPUT "<SHFT CLR/HOME><2 CR/AB>CUANTOS MENSAJE QUIERE
    S MANDAR : " ; N
30  B$ = "                                              "
40  FOR I = 1 TO N
50    PRINT "<CR/AB>DAME EL MENSAJE No. ";I
60    INPUT C$(I)
70    A$(I) = B$+C$(I)+B$
80  NEXT I
90  FOR M = 1 TO N
100   PRINT "<SHFT CLR/HOME>"
110   FOR L = 1 TO 5 : PRINT : NEXT L
120   ES = 40 - LEN (C$(M))
130   PRINT SPC(INT(ES/2));c$(M)
140   FOR K = 13 TO 23
150     POKE 55537+K,1
160   NEXT K
170   FOR L = 1 TO 7 : PRINT : NEXT L
180   FOR J = 1 TO LEN(A$(M)) - 40
200     FOR K = 13 TO 23
210       POKE 55857+K,1
```

```
220      NEXT K
230      PRINT MID$(A$(M),J,40);"<CR/AB>";
240      FOR K = 13 TO 23
250        POKE 55857+K,1
260      NEXT K
270    NEXT J
280 NEXT M
290 PRINT <SHFT CLR/HOME>
300 END
```

Programa 3.5

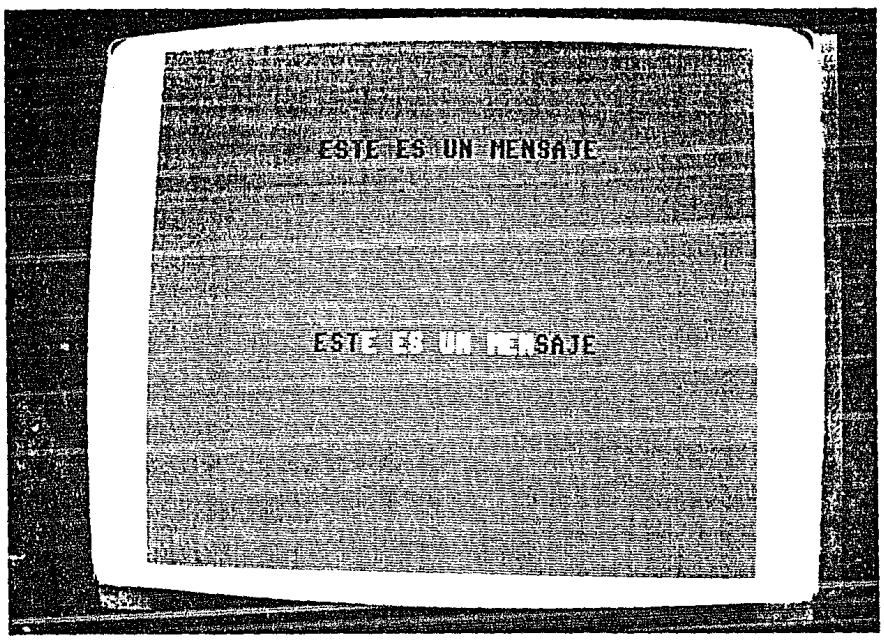


Figura 3.5. Imagen generada por el programa 3.5

CAPITULO IV

DIFERENTES FORMAS DE AFECTAR LA IMAGEN

IV.1.- COMO SE FORMA UN CARACTER.

Como ya hemos visto la pantalla está dividida en 40 cuadros horizontales por 25 verticales, y cada uno de estos cuadros consta de ocho puntos horizontales por ocho verticales.

Cada patrón de un caracter se representa por una matriz de ocho por ocho puntos. Cada punto es representado por un bit en memoria, por lo que cada caracter requiere ocho bytes, es decir, un byte para cada una de las líneas horizontales del caracter. Cada bit puede tener un valor de 0 ó 1. Si el bit vale 1 el punto correspondiente a ese bit estará prendido y por lo tanto será desplegado en la pantalla. Si por el contrario el bit vale 0 el punto correspondiente a él estará apagado y no aparecerá en la pantalla.

La matriz en la cual se define a un caracter puede ser representada de la siguiente manera:

bits	7	6	5	4	3	2	1	0
bytes								
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0

Figura 4.1. Matriz de un caracter.

Este rectángulo tiene una relación uno a uno con los bits de los ocho bytes del patrón de caracteres.

Por ejemplo el patrón para la letra A será:

bits bytes	76543210	IMAGEN DEL CARACTER	VALOR DECIMAL
0	00000000		0
1	00011000	11	24=2 +2 (3 4)
2	00111100	1111	60=2 +2 +2 +2 (2 3 4 5)
3	01100110	11 11	102=2 +2 +2 +2 (1 2 5 6)
4	01111110	111111	126=2 +2 +2 +2 +2 +2 (1 2 3 4 5 5)
5	01100110	11 11	102=2 +2 +2 +2 (1 2 6 6)
6	01100110	11 11	102=2 +2 +2 +2 (1 2 5 6)
7	00000000		0

Figura 4.2. Diseño del caracter correspondiente a la letra A.

A la derecha de cada byte se encuentra su valor decimal, que es el que tendremos que introducir en la localidad de memoria correspondiente al patrón de caracteres mediante el procedimiento POKE, conservando el orden de los bytes.
Un método con el que cualquiera puede crear sus propios caracteres es el que se mostró en el ejemplo anterior.

IV.2.- COMO MOVER LA MEMORIA DE LOS CARACTERES.

Hay dos conjuntos de caracteres en la Commodore 64 que constan cada uno de 256 caracteres. El primer conjunto consta de las letras mayúsculas y caracteres gráficos y el segundo de las letras mayúsculas y minúsculas.
Como ya se explicó anteriormente, al encender la máquina la memoria de la pantalla comienza en la localidad 1024, la

memoria del color en la 55296 y la memoria de los caracteres en la 53248 (Esto puede no ser permanente, ya que uno puede indicarle al VIC-II que busque cualquiera de estas tres memorias en algún otro lado).

Como se podrá observar, las localidades ocupadas por el generador de caracteres ROM son las mismas localidades de control que ocupa el VIC-II. Esto es posible debido a que las localidades no son ocupadas al mismo tiempo para ambos objetivos. El conjunto de caracteres ROM está disponible únicamente cuando el VIC-II lo requiere. En cualquier otro momento esta área consiste de localidades de memoria RAM que sirven para el control del VIC-II.

Para crear o cambiar un caracter se tendrá que transferir la memoria de los caracteres a la memoria de la pantalla y en ésta última poder hacer las alteraciones necesarias a las localidades correspondientes.

La razón por la cual se tiene que hacer este cambio es porque el conjunto de caracteres se encuentra en las localidades de memoria que pertenecen a ROM, las cuales no son borradas cada vez que se apaga la máquina. Por ello dichas localidades no pueden ser afectadas directamente mediante el procedimiento POKE, pues existen protecciones para que no sean alteradas. En principio sólo pueden ser leídas o consultadas (mediante la función PEEK).

La máquina puede utilizar otro conjunto nuevo de caracteres, sólo que tendremos que indicarle en qué localidad va a empezar la nueva memoria de los caracteres, para lo cual

tendremos que alterar la localidad 53272.

La localidad 53272 controla dónde encuentra los caracteres el VIC-II y también se usa para indicarle dónde colocar la memoria de la pantalla. De los bits 7 al 4 se controla dónde estarán las localidades correspondientes a la memoria de la pantalla. Los bits del 3 al 1 controlan la localización de los conjuntos de caracteres. Por último, el bit 0 no tiene alguna función definida por Commodore aunque parece estar siempre encendido aún cuando se apague mediante el procedimiento POKE.

Para saber dónde colocar la memoria de los caracteres debe recordarse que el VIC-II sólo puede buscar en uno de los bancos de 16K de memoria RAM a la vez, por lo que la memoria de la pantalla y la de los caracteres deben estar siempre dentro de un mismo banco, a menos de que se use el conjunto de caracteres de ROM.

Al encender la máquina la localidad 53272 tienen el valor de 21.

```
BIT   7 6 5 4 3 2 1 0

      0 0 0 1 0 1 0 1
      └─────┬─┬─┬─┘
            16  4  1
```

Figura 4.3. Configuración inicial de la palabra que elige el conjunto de caracteres.

Lo que significa que la memoria de la pantalla comenzará en la localidad 4096 y la memoria de los caracteres en la localidad 2048, como lo veremos más adelante.

Para saber dónde colocar la pantalla es necesario dividir entre 64 la localidad en donde empieza la memoria de la

pantalla. Por ejemplo normalmente comienza en la localidad 1024, por lo que, el valor que deberá ir en la localidad 53272 será 16, es decir, 1024/64=16.

Basándonos en lo anterior se obtiene la siguiente tabla de posibles posiciones de inicio de la memoria de la pantalla, así como el valor que debe colocarse en la localidad 53272.

VALOR	LOCALIDAD	VALOR	LOCALIDAD
0	0	128	8192
16	1024	144	9216
32	2048	160	10240
48	3072	176	11264
64	4096	192	12288
80	5120	208	13312
96	6144	224	14336
112	7168	240	15360

Figura 4.4. Tabla de valores para el control del inicio de la memoria de la pantalla.

Como se podrá observar en esta tabla únicamente aparecen las localidades del primer banco de 16K de memoria. Para indicar al VIC-II en cuál banco debe buscar, es necesario modificar los bits 0 y 1 de la localidad 56576.

Para colocar la memoria de los caracteres debe tomarse en cuenta que cada conjunto de caracteres usa 2K de memoria y debido a ésto, sólo hay ocho posibles localidades en donde puede colocarse el inicio de ellos dentro del primer banco de 16K de memoria y estas son las siguientes:

LOCALIDAD
0
2048
4096
6144
8192
10240
12288
14336

Figura 4.5. Localidades donde pueden empezar los conjuntos de caracteres.

De la localidad 0 a la 2048 no pueden ser usadas con BASIC, pues de la 0 a la 1023 se guarda información importante del sistema y de la 1024 a la 2047 están reservadas para la memoria de la pantalla. De la 2048 a la 40959 se usa para los programas de BASIC. Si se van a usar los dos conjuntos de caracteres, la localidad más apropiada para colocar el inicio de ellos es la 12288, pero deben protegerse estas localidades de los programas de BASIC, es decir, se debe afectar la localidad 56 que es la que controla el límite superior del área destinada para los programas de BASIC. Para ello se tiene la siguiente expresión:

POKE 56,12288/256 o POKE 56,48

Con ello sólo se tendrán 10K libres para programas de BASIC, es decir, de la localidad 2048 a la 12287.

Si únicamente se va a copiar un conjunto de caracteres sólo son necesarios 2K de memoria. Debido a ello las mejores localidades para colocar dicho conjunto son de la 14336 a la 16383. Para protegerlas de BASIC se haría lo siguiente:

POKE 56,14336/256 o POKE 56,56

Para saber qué valor debe colocarse en la localidad 53272 e indicar dónde se encontrará la memoria de los caracteres, debe dividirse entre 1024 cualquiera de las localidades mencionadas anteriormente y obtenerse de esta forma el valor con el cual se afectará dicha localidad, como se muestra en la siguiente tabla:

VALOR	LOCALIDAD
0	0
2	2048
4	4096
6	6144
8	8192
10	10240
12	12288
14	14336

Figura 4.6. Tabla de valores para el control de inicio de la memoria de los caracteres.

De todo lo anterior, para indicarle al VIC-II dónde encontrar los caracteres (sin afectar las localidades que indican dónde se encuentra la pantalla) se tiene la siguiente expresión:

```
POKE 53272,PEEK(53272) AND 240 OR MC
```

donde MC es el valor correspondiente a la localidad donde comenzará el conjunto de caracteres (Figura 4.6).

Ejemplo :

```
POKE 53272,PEEK(53272) AND 240 OR 12

  PEEK(53272)        0 0 0 1 0 1 0 1
  AND 240            1 1 1 1 0 0 0 0
                     ---------------
                     0 0 0 1 0 0 0 0  = 16
  OR 12              0 0 0 0 1 1 0 0
                     ---------------
                     0 0 0 1 1 1 0 0  = 28
```

En este ejemplo, la pantalla quedará en la localidad 1024 y los conjuntos de caracteres a partir de la localidad 12288.
Una vez sabiendo como decirle a la máquina cómo obtener sus caracteres de algún otro lado de la memoria, necesitamos poner algunos caracteres ahí. Lo más conveniente es copiar el conjunto de caracteres completo a esta parte de la memoria y después cambiar únicamente los caracteres que se deseen distintos al conjunto original.
Para obtener el conjunto de caracteres de ROM es necesario utilizar una localidad que indica al Kernel como controlar la información de entrada y salida, sin la cual el VIC-II no podrá funcionar.
Se presenta un problema pues una vez que esta localidad haya sido modificada la Commodore no tendrá capacidad de entrada y salida, quedando la máquina aislada. Hablaremos sobre cómo resolver este problema más adelante.

IV.3.- CREACION DE NUEVOS CARACTERES.

Ya vimos que para poder modificar los patrones de los caracteres que normalmente usamos, lo que hacemos es copiar el conjunto de caracteres de ROM y cambiarlos a un área en RAM, que será la nueva memoria de los caracteres, para que ahí puedan ser modificados los patrones que sea necesario cambiar. Los demás permanecerán sin alteración alguna.
Para copiar el conjunto de caracteres de ROM tendremos que alterar algunas localidades de tal forma de quitar las protecciones con que cuenta el sistema para que no sean

manipuladas (por error) localidades clave para su propio manejo.

Como el conjunto de patrones de caracteres en ROM es un banco de memoria, normalmente tiene una protección para que esta área no pueda ser leída mediante la función PEEK. Para quitar dicha protección es necesario usar una localidad que controla la entrada y salida de información. Al "desconectar" esta localidad se desconectan asimismo todas las protecciones del sistema además de otras funciones, sin las cuales el VIC-II y otros procesadores no pueden trabajar. De lo anterior, al desconectar las protecciones queda la máquina bloqueada.

Las acciones a realizarse son:

- Desconectar el registro de control de entrada y salida
- Conectar la memoria de caracteres de ROM
- Copiar los patrones de caracteres que se deseen
- Desconectar el área de ROM
- Conectar nuevamente el registro de entrada y salida.

Para hacer las tareas antes mencionadas el procedimiento será el siguiente:

- Desconectar los interruptores :

```
POKE 56334,PEEK(56334) AND 254
```

- Desproteger el área de ROM :

```
POKE 1,PEEK(1) AND 251
```

- Copiar el conjunto de caracteres :

```
FOR I=0 TO 1023
POKE 12288+I,PEEK(53248+I)
NEXT I
```

- Proteger el área de ROM :

```
POKE 1,PEEK(1) OR 4
```

- Conectar los interruptores :

```
POKE 56334,PEEK(56334) OR 1
```

El siguiente programa copia los patrones de los caracteres de ROM y modifica los primeros ocho caracteres de este conjunto al prender el primer byte en el primer caracter, el segundo byte en el segundo caracter y así sucesivamente hasta llegar al octavo caracter.

```
10    REM ********************************
20    REM *     PROGRAMA QUE COPIA EL      *
30    REM *   CONJUNTO  DE CARACTERES      *
40    REM *         Y LOS MODIFICA         *
50    REM ********************************
60    REM LOCALIDAD DE INICIO DE :
70    REM CM - NUEVA MEMORIA DE LOS CARACTERES
80    REM CX - MEMORIA DE CARACTERES EN ROM
90    CM = 12288
100   CX = 53248
110   GOSUB 250
120   PRINT "<8 CR/IZ>"
130   PRINT "ESTOY COPIANDO EL CONJUNTO"
140   PRINT SPC(12);"DE CARACTERES"
150   GOSUB 320
160   POKE 53272, (PEEK(53272) AND 240) OR 12
170   FOR I = 0 TO 63 STEP 9
180     POKE CM+I, 255
190   NEXT I
200   GOSUB 250
210   FOR I = 64 TO 71
220     PRINT CHR$(I);
230   NEXT I
240   END
250   REM ********************************
260   REM *    RUTINA PARA POSICIONAR      *
270   REM *           EL CURSOR            *
280   REM ********************************
290   PRINT "<SHFT CLR/HOME>"
300   PRINT "<10 CR/AB><15 CR/DE)";
310   RETURN
320   REM ********************************
330   REM *    RUTINA PARA COPIAR LOS      *
340   REM *      CARACTERES  DE ROM        *
350   REM ********************************
```

```
360   POKE 56334,PEEK(56334) AND 254
370   POKE 1,PEEK(1) AND 251
380   FOR I = 0 TO 1023
390     POKE CM+I,PEEK(CX+I)
400   NEXT I
410   POKE 1,PEEK(1) OR 4
420   POKE 56334,PEEK(56334) OR 1
530   RETURN
```

Programa 4.1

El tener la posibilidad de poder cambiar los caracteres y cambiar las imágenes que deseemos es una gran ventaja cuando hablamos de graficación, debido a que podemos ir creando figuras muy parecidas con movimientos pequeños, con el propósito de que cuando se vayan introduciendo sucesivamente los caracteres modificados en una misma localidad o en localidades contiguas de la memoria de la pantalla, se obtenga un efecto de animación.

```
10    REM ****************************
20    REM *    PROGRAMA QUE PONE EN    *
30    REM *    MOVIMIENTO UNA LINEA    *
40    REM * HORIZONTAL Y UNA VERTICAL *
50    REM ****************************
60    REM LOCALIDAD DE INICIO DE :
70    REM CM - NUEVA MEMORIA DE LOS CARACTERES
80    REM CX - NUEVA MEMORIA DE CARACTERES DE ROM
90    REM SM - MEMORIA DE LA PANTALLA
100   REM CL - MEMORIA DEL COLOR
110   CM = 12288
120   CX = 53248
130   SM = 1024
140   CL = 55296
150   GOSUB 490
160   POKE 53272,(PEEK(53272) AND 240) OR 12
170   FOR C = 0 TO 7
180     POKE CM+C*8,255
190     FOR R=1 TO 7
200       BY =C*8+R+CM
210       POKE BY,2^C
220     NEXT R
230   NEXT C
240   FOR C = 8 TO 15
250     FOR R = 0 TO 7
260       BY =C*8+R+CM
```

```
270       POKE BY,128
280       IF R = C-8 THEN POKE BY,255
290     NEXT R
300   NEXT C
310   PRINT "<SHFT CLR/HOME>"
320   POKE CL+500,1
330   FOR I = 0 TO 15
340     POKE SM+500,I
350     GOSUB 430
360   NEXT I
370   FOR I = 15 TO 0 STEP -1
380     POKE SM+500,I
390     GOSUB 430
400   NEXT I
410   GOTO 330
420   END
430   REM ******************************
440   REM * RUTINA DE PERDIDA DE TIEMPO *
450   REM ******************************
460   FOR J = 0 TO 20
470   NEXT J
480   RETURN
490   REM ******************************
500   REM *   RUTINA PARA COPIAR LOS   *
510   REM *     CARACTERES  DE ROM     *
520   REM ******************************
530   PRINT "<SHFT CLR/HOME>"
540   PRINT "<9 CR/AB><7 CR/DE>"
550   PRINT "ESTOY COPIANDO EL CONJUNTO"
560   PRINT SPC(12);"DE CARACTERES"
570   POKE 56334,PEEK(56334) AND 254
580   POKE 1,PEEK(1) AND 251
590   FOR I = 0 TO 1023
600     POKE CM+I,PEEK(CX+I)
610   NEXT I
620   POKE 1,PEEK(1) OR 4
630   POKE 56334,PEEK(56334) OR 1
640   RETURN
```

Programa 4.2

En los programas anteriores se han modificado los caracteres únicamente por medio de fórmulas. El siguiente programa modifica los caracteres usando la proposición DATA, lo cual nos da un mejor recurso para crear figuras más complicadas. En él se modifican los primeros 4 caracteres para formar un paracaidista y un avión. El avión atraviesa toda la pantalla y al oprimirse cualquier tecla diferente de "F" suelta al

paracaidista. Unicamente debe oprimirse una tecla por cada vez que el avión cruce la pantalla. Para terminar el programa debe oprimirse la tecla "F".

```
10   REM ********************************
20   REM *   PROGRAMA QUE MODIFICA LOS    *
30   REM *  PRIMEROS CUATRO  CARACTERES   *
40   REM *  PARA CREAR UN  AVION Y  UN    *
50   REM *         PARACAIDISTA           *
60   REM ********************************
70   REM LOCALIDAD DE INICIO DE:
80   REM CM - NUEVA MEMORIA DE LOS CARACTERES
90   REM CX - MEMORIA DE CARACTERES DE ROM
100  REM SM - MEMORIA DE LA PANTALLA
110  REM CL - MEMORIA DEL COLOR
120  CM = 12288
130  CX = 53248
140  SM = 1024
150  CL = 55296
160  GOSUB 800
170  POKE 53272,PEEK(53272) AND 240) OR 12
180  FOR I = 0 TO 31
190    READ A
200    POKE CM+I,A
210  NEXT I
220  DATA 60,126,255,255,255,129,153,153
230  DATA 90,60,24,24,24,36,66,195
240  DATA 14,17,32,65,127,255,0,0
250  DATA 1,3,135,255,255,255,56,24
260  PRINT "<SHFT CLR/HOME>"
270  FOR I=119 TO 80 STEP -1
280    POKE CL+I,1
290  NEXT I
300  FOR I=119 TO 81 STEP -1
310    POKE SM+I-1,2
320    POKE SM+I,3
330    FOR K=1 TO 30
340    NEXT K
350    POKE SM+I,32
360    POKE SM+I-1,32
370    GET A$
380    IF A$="F" THEN 650
390    IF A$<>"" THEN 420
400  NEXT I
410  GOTO 300
420  J=I
430  FOR M=1 TO 39
440    CO=M
450    IF J<81 THEN 480
460    POKE SM+J-1,2
470    POKE SM+J,3
480    IF CO>12 THEN 510
490    POKE CL+I+40*CO,1
```

```
500     POKE CL+I+40*(CO+1),1
510     FOR K=1 TO 20
520     NEXT K
530     IF CO>12 THEN 560
540     POKE SM+I+40*CO,0
550     POKE SM+I+40*(CO+1),1
560     IF J<81 THEN 590
570     POKE SM+J-1,32
580     POKE SM+J,32
590     IF CO>12 THEN 620
600     POKE SM+I+40*CO,32
610     POKE SM+I+40*(CO+1),32
620     J=J-1
630   NEXT M
640   GOTO 300
650   PRINT "<SHFT CLR/HOME>"
660   END
800   REM ****************************
810   REM *   RUTINA PARA COPIAR LOS   *
820   REM *      CARACTERES DE ROM     *
830   REM ****************************
840   PRINT "<SHFT CLR/HOME>"
850   PRINT "<9 CR/AB><7 CR/DE>";
860   PRINT "ESTOY COPIANDO EL CONJUNTO"
870   PRINT SPC(12) "DE CARACTERES"
880   POKE 56334,PEEK(56334) AND 254
890   POKE 1,PEEK(1) AND 251
900   FOR I=0 TO 1023
910     POKE CM+I,PEEK(CX+I)
920   NEXT I
930   POKE 1,PEEK(1) OR 4
940   POKE 56334,PEEK(56334) OR 1
950   RETURN
```

Programa 4.3

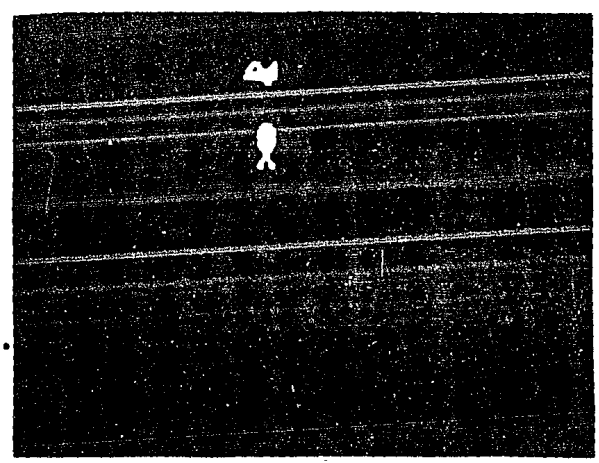


Figura 4.7. Imagen generada por el programa 4.3

Como se podrá observar, en la pantalla, al aparecer la palabra READY la letra A aparece modificada. Para ver los caracteres que se modificaron oprima los caracteres @, A, B y C. Los dos primeros forman al paracaidista y los dos últimos al avión. Los demás caracteres permanecen sin ninguna modificación.

CAPITULO V

S P R I T E S *

Un "sprite" es un tipo especial de caracter que puede desplegarse en cualquier parte de la pantalla. Puede usarse con cualquiera de los modos de gráficas mencionados y tiene las siguientes características:

a) Consta de 504 puntos organizados en una matriz de 24 columnas por 21 renglones.

b) Tiene control de color individual.

c) Puede hacerse en modo multicolor o en alta resolución.

d) Puede expanderse horizontalmente, verticalmente o en ambas direcciones.

e) Puede seleccionarse la prioridad del sprite sobre el fondo de la pantalla o entre sprites, que implica, en caso de intersección de las figuras, cuál de ellas aparece.

f) Pueden detectarse colisiones entre sprites, o bien, detectarse colisiones del sprite sobre lo que se encuentre en la pantalla.

Es posible usar ocho sprites diferentes en un mismo momento en la pantalla y se numeran del 0 al 7.

* Dado que no se encontró una traducción adecuada, a lo largo de estas notas se usará el término en ingles.

Para crear un sprite es necesario hacer lo siguiente:

1) Darle valor al sprite, es decir, dibujar el sprite dentro de la matriz de 24 por 21 renglones.
2) Obtener los datos que lo definen.
3) Proteger las localidades de memoria que van a usarse para guardar los datos del sprite.
4) Prender el sprite.
5) Darle el color que se desee.
6) Dar su posición horizontal (X) en la pantalla.
7) Dar su posición vertical (Y) en la pantalla.
8) Definir si se quiere expander verticalmente o no.
9) Definir si se quiere expander horizontalmente o no.

V.1.- DIBUJO DEL SPRITE.

Como se dijo anteriormente un sprite consta de 24 columnas por 21 renglones, lo cual da un total de 504 puntos. Las columnas se dividen en tres secciones de 8 columnas cada una (cada localidad de memoria está formada por 8 bits, por lo que cada sección se representará por un byte). Debido a lo anterior, cada sprite necesita 63 bytes para poderse representar (504/8=63).

```
BYTE 0     BYTE 1     BYTE 2
BYTE 3     BYTE 4     BYTE 5
  .          .          .
  .          .          .
  .          .          .
BYTE 60    BYTE 61    BYTE 62
```

Además de estos 63 bytes, cada uno de los 8 sprites predefinidos tienen un byte asociado a él llamado apuntador. Las localidades que sirven como apuntadores a los sprites

son de la 2040 a la 2047, comenzando con el sprite 0 . Sin embargo, si se mueve la pantalla, de su ubicación natural en la memoria, los apuntadores se tendrán que mover también.

Cada apuntador puede ser un número entre 0 y 255. Este número apunta a la definición del sprite, es decir, a donde se encuentran los datos que lo definen. Por ejemplo, si el apuntador al sprite 0, que se encuentra en la localidad 2040, tiene un número 14, significa que la definición de dicho sprite comienza en la localidad 14*64=896, que es el buffer del cassette. Visto de otra manera, la fórmula que nos indica la localidad en donde comienza la definición de un sprite sería la siguiente:

LOCALIDAD = (BANCO * 16384) + (APUNTADOR * 64)

Para dibujar un sprite es necesario que se haga una tabla de 21 renglones por 24 columnas. Se numeran por comodidad las columnas dividiéndolas en tres grupos del 0 al 7 y los renglones del 0 al 20. Ya que se tiene la hoja de trabajo simplemente hay que llenar los cuadros que van a formar el sprite, como se muestra en la figura 5.1. Estos cuadros serán los que estarán prendidos cuando el sprite aparezca en la pantalla. Para saber cuáles de ellos deben prenderse y cuáles no, se pondrá un 1 si el cuadro está prendido y un 0 si está apagado. Una vez hecho esto se convierte a decimal el valor de cada byte, obteniendo así los datos que definen al sprite.

En la figura 5.2 se muestra el dibujo de un sprite, que en este caso corresponde al de un avión, así como el valor decimal de cada byte.

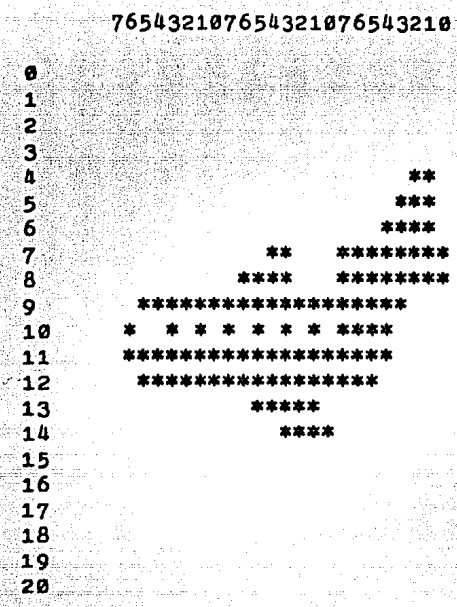


Figura 5.1. Dibujo cuadriculado de un sprite.

	765432107654321076543210	Valor decimal		
0	000000000000000000000000	0	0	0
1	000000000000000000000000	0	0	0
2	000000000000000000000000	0	0	0
3	000000000000000000000000	0	0	0
4	000000000000000000000110	0	0	6
5	000000000000000000001110	0	0	14
6	000000000000000000011110	0	0	30
7	000000000011100011111111	0	24	255
8	000000000111100011111111	0	120	255
9	001111111111111111111000	63	255	248
10	010010101010101011110000	74	170	240
11	011111111111111111110000	127	255	240
12	001111111111111111100000	63	255	240
13	000000000011111000000000	0	62	0
14	000000000000111100000000	0	15	0
15	000000000000000000000000	0	0	0
16	000000000000000000000000	0	0	0
17	000000000000000000000000	0	0	0
18	000000000000000000000000	0	0	0
19	000000000000000000000000	0	0	0
20	000000000000000000000000	0	0	0

Figura 5.2. Dibujo codificado de un sprite y valor decimal de los bytes que lo definen.

El siguiente programa obtiene los datos que definen a un sprite, simplemente dibujando cada renglón como una cadena, usando asteriscos y blancos mediante la proposición DATA.

```
10  REM ************************************************
20  REM *  PROGRAMA QUE OBTIENE LOS DATOS DE UN SPRITE  *
30  REM ************************************************
40  PRINT "<SHFT CLR/HOME>"
50  FOR REN = 0 TO 20
60    READ A$
70    PRINT "DATO  ";
80    FOR COL = 1 TO 24
90      B$ = MID$(A$,COL,8)
100     SUMA = 0
110     FOR J=1 TO 8
120       BIT=8-J
130       C$=MID$(B$,J,1)
140       IF C$<>"*" THEN 160
150       SUMA=SUMA+2^BIT
160     NEXT J
170     PRINT SUMA; "   ";
180   NEXT COL
190   PRINT
200 NEXT REN
210 END
220 DATA "                        "
230 DATA "                        "
240 DATA "                        "
250 DATA "                        "
260 DATA "                      ** "
270 DATA "                     *** "
280 DATA "                    **** "
290 DATA "           **    ********"
300 DATA "          ****   ********"
310 DATA "  ******************    "
320 DATA " *  * * * * * * ****    "
330 DATA " ******************     "
340 DATA "  ****************      "
350 DATA "           *****        "
360 DATA "             ****       "
370 DATA "                        "
380 DATA "                        "
390 DATA "                        "
400 DATA "                        "
410 DATA "                        "
420 DATA "                        "
```

Programa 5.1

Al correr este programa se observará que los datos son los mismos que se obtuvieron anteriormente.

Este programa puede ser utilizado para obtener los datos de cualquier sprite, simplemente modificando de la linea 220 a la 420, las cuales forman el dibujo del sprite.

V.2.- PRENDER UN SPRITE.

Una vez que se ha definido el sprite es necesario prenderlo para que aparezca en la pantalla. La localidad 53269 es un registro para el control de los sprites. Cada uno de ellos tiene un bit en este registro que indica si el sprite está prendido (1) o apagado (0). Para prender un sprite, es necesario prender el bit correspondiente de esta localidad. Por ejemplo, si se quiere prender el sprite 3, se debe prender el bit 3 que corresponde a dicho sprite, lo cual se hace con el procedimiento POKE 53269,8.

Al prender un sprite se debe tener cuidado de no afectar los demás. Como se ve en el ejemplo anterior, si hubiera estado prendido algún otro sprite, al colocar un 8 en la localidad 53269, automáticamente se apagarían los demás. Para prender un sprite sin afectar los demás se tiene la siguiente expresión:

```
POKE 53269,PEEK(53269) OR (2^NS)
```

donde NS es el número del sprite.

Por ejemplo, si se quiere prender el sprite 5 y están prendidos los sprites 0 y 3 se haría lo siguiente:

```
POKE 53269,PEEK(53269) OR (2^5)

    BITS          7 6 5 4 3 2 1 0
PEEK(53269)       0 0 0 0 1 0 0 1
OR (2^5)          0 0 1 0 0 0 0 0
                  ---------------
RESULTADO         0 0 1 0 1 0 0 1
```

Análogamente para apagar cualquier sprite sin afectar a los demás se tiene la expresión:

```
POKE 53269,PEEK(53269) AND (255-2^NS)
```

En los dos ejemplos anteriores se usaron los operadores lógicos AND y OR en la forma en que se explica en el capítulo II.

V.3.- COLOR.

Un sprite puede tener cualquiera de los 16 colores que existen en la Commodore. Sin embargo, se tiene la limitación de que todo el sprite deberá ser de un mismo color.

Las localidades que controlan el color del sprite son de la 53287 a la 53294. Para seleccionar al color de un sprite se tiene la expresión:

```
POKE 53287+NS,C
```

donde NS es el número del sprite y C es el número del color elegido, que debe estar entre 0 y 15.

V.4.- EXPANSION DE SPRITES.

En ocasiones, la figura que dibujamos no tiene las proporciones necesarias o simplemente, se desea que la figura sea más ancha y/o alargada. Para lograr esto los sprites cuentan con la propiedad de poderlos expander horizontal o verticalmente o en ambas direcciones, al doble de su tamaño.

Para expander un sprite horizontalmente, debe prenderse el bit correspondiente al sprite en la localidad 53277.

La expresión para expander un sprite horizontalmente sin afectar los demás es:

```
POKE 53277,PEEK(53277) OR (2^NS)
```

Para regresar al tamaño normal debe apagarse el bit correspondiente a dicho sprite en esta localidad.

```
POKE 53277,PEEK(53277) AND (255-2^NS)
```

Para expander el sprite verticalmente se usa la localidad 53271 y el modo de hacerlo es el mismo que para hacerlo horizontalmente.

V.5.- LOCALIZACION DE UN SPRITE EN LA PANTALLA.

Una vez definidas la forma y el color de un sprite, para poderlo desplegar en la pantalla no es necesario solamente prenderlo, sino que también hay que posicionarlo en la pantalla.

Hay tres registros que controlan la posición de un sprite en la pantalla:

1) Dos registros que controlan la posición horizontal (X).

2) Un registro que controla su posición vertical (Y).

Se puede colocar el sprite en 512 posiciones horizontalmente y en 256 verticalmente.

La posición del sprite se calcula tomando la esquina superior izquierda del área de 24 por 21 puntos, sin importar que el sprite sólo estuviera formado por un punto.

Las localidades que controlan la posición de un sprite son de la 53248 a la 53264. La siguiente tabla muestra qué es lo que controla cada una de estas localidades.

LOCALIDAD	DESCRIPCION	SPRITE
53248	Pos. horizontal	0
53249	Pos. vertical	0
53250	Pos. horizontal	1
53251	Pos. vertical	1
53252	Pos. horizontal	2
53253	Pos. vertical	2
53254	Pos. horizontal	3
53255	Pos. vertical	3
53256	Pos. horizontal	4
53257	Pos. vertical	4
53258	Pos. horizontal	5
53259	Pos. vertical	5
53260	Pos. horizontal	6
53261	Pos. vertical	6
53262	Pos. horizontal	7
53263	Pos. vertical	7
53264	Bit mas significativo	0 a 7
	Posición horizontal (256-511)	

Figura 5.3. Localidades que controlan la posición de un sprite.

V.5.1.- POSICION VERTICAL.

Existen 256 posibles posiciones en la pantalla para colocar un sprite verticalmente, es decir, para moverlo hacia arriba o hacia abajo. Estas posiciones toman valores del 0 al 255 comenzando en la parte superior de la pantalla.

De la posición 50 a la 229 el sprite es totalmente visible en la pantalla. De la posición 0 a la 49, así como de la 230 a la 255 el sprite o una parte de él no es visible en la pantalla, quedando en el primer caso en la parte superior de la pantalla y en el segundo en la parte inferior de la misma.

V.5.2.- POSICION HORIZONTAL.

La posición horizontal de un sprite puede variar entre 0 y 511. Debido a esto son necesarios 9 bits para poder controlar su posición en la pantalla, es decir, es necesario un bit extra para poder tener 512 posiciones.

La localidad 53264 es un registro que sirve para controlar en qué parte de la pantalla (izquierda o derecha) se encuentra un sprite. Cada uno de ellos tiene un bit en esta localidad.

Si su posición horizontal es mayor a 255 es necesario prender el bit correspondiente al sprite en esta localidad. Por el contrario, si su posición es menor a 256 debe apagarse dicho bit.

El siguiente programa muestra un sprite en movimiento expandido horizontalmente. El ejemplo es un avión que atraviesa la pantalla de derecha a izquierda.

```
1   REM ************************************************
2   REM *  .      SPRITE EN MOVIMIENTO, EXPANDIDO       *
3   REM *                 HORIZONTALMENTE               *
4   REM ************************************************
10    PRINT "<SHFT CLR/HOME>"
20    POKE 53280,0 : REM COLOR DE LA PANTALLA NEGRO
30    POKE 53281,0 : REM COLOR DEL BORDE DE LA PANTALLA
40    CS = 53248
50    POKE 2040,13 : REM APUNTADOR DATOS DEL SPRITE 0
60    DS = 832     : REM LOCALIDAD DE INICIO DATOS SPRITE
70    GOSUB 600    : REM LECTURA DE DATOS
130   POKE 53287,7 : REM COLOR AMARILLO
150   POKE CS,200  : REM POSICION (X) HORIZONTAL
160   POKE CS+1,0  : REM POSICION (Y) VERTICAL
190   POKE 53277,1 : REM EXPANSION HORIZONTAL
200   POKE CS+16,1 : REM ENCIENDE PANTALLA DERECHA
210   POKE CS+21,1 : REM ENCIENDE EL SPRITE 0
220   Y = 100
230   FOR X=343 TO 0 STEP -1
240     POKE CS+1,Y
250     IF X<=255 THEN 280
260     X1 = X - 255
270     GOTO 310
280     IF X>156  THEN Y=Y+0.3
290     X1 = X
300     POKE CS+16,PEEK(CS+16) AND 254:REM APAGA PANT.DER.
310     POKE CS,X1
350   NEXT X
360   POKE CS+21,PEEK(CS+21) AND 254 : REM APAGA SPRITE 0
370   GET B$
380   IF B$="" THEN 370
390   POKE CS+21,0 ; REM APAGA SPRITES
400   PRINT "<SHFT CLR/HOME>"
410   END
500   DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,6,0,0,14,0,0,30,0,
      24,255,0,120,255
510   DATA 63,255,248,74,170,240,127,255,240,63,255,224,
      0,52,0,0,15,0
520   DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
570   REM ****************************
580   REM *      LECTURA DE DATOS      *
590   REM ****************************
600   FOR I = 0 TO 62
610     READ A
620     POKE DS+I,A
630   NEXT I
640   RETURN
```

Programa 5.2

V.6.- PRIORIDAD DE UN SPRITE.

Los sprites tienen prioridades entre ellos, es decir, pueden pasar frente a otro o detrás de él cuando se encuentran en las mismas localidades en la pantalla. Esto permite dar un efecto de tercera dimensión.

La prioridad de cada sprite está determinada de acuerdo al número de sprite que se le asignó: entre mayor sea éste su prioridad es menor. Es decir, el sprite 0 tiene la más alta prioridad, por lo que si éste se cruza con cualquier otro en la pantalla, el que aparecerá adelante será dicho sprite.

Además de tener prioridades entre ellos cada sprite tiene prioridad respecto a cualquier caracter que se encuentre en la pantalla.

La localidad 53275 controla estas prioridades, teniendo cada sprite un bit en dicha localidad. Si el bit correspondiente es un 0, el sprite tiene mayor prioridad que el fondo de la pantalla, en otras palabras el sprite aparecerá adelante de lo que se encuentre en ese momento en la pantalla. Si el bit está encendido, lo que se encuentre en la pantalla tiene prioridad sobre el sprite, por lo que el sprite aparecerá detrás de lo que se encuentre en la pantalla.

La expresión POKE 53275,PEEK(53275) OR (2^NS) prende el bit correspondiente al sprite NS.

Si se desea apagar el bit correspondiente a dicho sprite se tiene la expresión:

```
POKE 53275,PEEK 53275 AND (255-2^NS).
```

El siguiente programa muestra cómo se comportan las prioridades entre dos sprites. En este programa se definirá un nuevo sprite, en forma de luna, pero con una prioridad menor que el avión. Es decir, se definirá como el sprite 1, por lo que automáticamente tendrá menor prioridad. Asimismo, para mostrar la prioridad entre un sprite y cualquier caracter que se encuentre en la pantalla, se dibujará un edificio por medio de los caracteres gráficos y los controles de color del teclado. La prioridad del avión sobre el edificio se controla en la instrucción 120. Si se desea que el avión pase enfrente del edificio, debe cambiarse la expresión por POKE 53275,0. Por el contrario, si se desea que el avión pase atrás del edificio, se debe dejar la expresión como está.

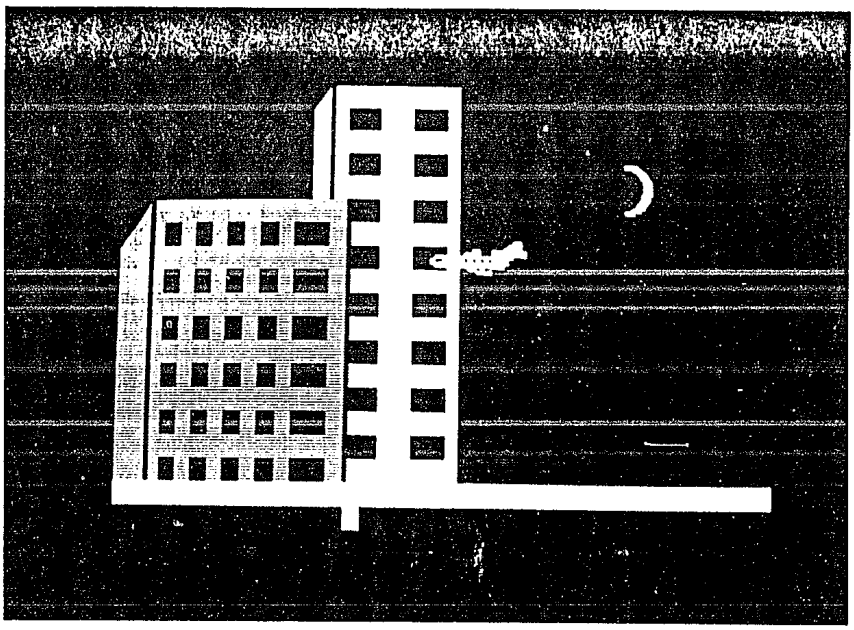


Figura 5.4. Imagen generada por el programa 5.3.

```
1     REM ******************************************
2     REM *   PRIORIDADES ENTRE DOS SPRITES Y ENTRE   *
3     REM *   UN SPRITE Y EL FONDO DE LA PANTALLA     *
4     REM ******************************************
10     PRINT "<SHFT CLR/HOME>"
20     POKE 53280,0 : REM COLOR DE LA PANTALLA NEGRO
30     POKE 53281,0 : REM COLOR DEL BORDE DE LA PANTALLA
40     CS = 53248
50     POKE 2040,13 : REM APUNTADOR A DATOS DEL SPRITE 0
60     POKE 2041,14 : REM APUNTADOR AL SPRITE 1
70     DS = 832     : REM LOCALIDAD DE INICIO DATOS SPRITE
80     GOSUB 600    : REM LECTURA DE DATOS
90     DS = 896      : REM INICIO DATOS SPRITE 1
100    GOSUB 600     : REM LEE DATOS DEL SPRITE 1
110    GOSUB 700     : REM DIBUJO DEL EDIFICIO
120    POKE 53275,1 : REM SPRITE 1< PRIORIDAD QUE EL TEXTO
130    POKE 53287,7 : REM COLOR SPRITE 0 (AMARILLO)
140    POKE 53288,1 : REM COLOR SPRITE 1 (BLANCO)
150    POKE CS,200  : REM POSICION (X) HORIZONTAL
160    POKE CS+1,0  : REM POSICION (Y) VERTICAL
170    POKE CS+2,10 : REM POS X. DEL SP. 1
180    POKE CS+3,98 : REM POS Y. DEL SP. 1
190    POKE 53277,1 : REM EXPANSION HORIZONTAL
200    POKE CS+16,3 : REM ENCIENDE PANTALLA DER. P/AMBOS
210    POKE CS+21,3 : REM ENCIENDE AMBOS SPRITES
220    Y = 100
230    FOR X=343 TO 0 STEP -1
240      POKE CS+1,Y
250      IF X<=255 THEN 280
260      X1 = X - 255
270      GOTO 310
280      IF X>156  THEN Y=Y+0.3
290      X1 = X
300      POKE CS+16,PEEK(CS+16) AND 254:REM APAGA PANT.DER.
310      POKE CS,X1
350    NEXT X
360    POKE CS+21,PEEK(CS+21) AND 254 : REM APAGA SPRITE 0
370    GET B$
380    IF B$="" THEN 370
390    POKE CS+21,0 ; REM APAGA SPRITES
400    PRINT "<SHFT CLR/HOME>"
410    END
500    DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,6,0,0,14,0,0,30,0,
       24,255,0,120,255
510    DATA 63,255,248,74,170,240,127,255,240,63,255,224,
       0,52,0,0,15,0
520    DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
540    DATA 0,0,0,0,0,0,15,192,0,0,240,0,0,60,0,0,30,0,
        0,15,0,0,7,128,0,7,128
550    DATA 0,7,128,0,7,128,0,7,128,0,15,0,0,15,0,0,30,
       0,0,60,0,0,120,0,1,224,0
560    DATA 15,0,0,0,0,0,0,0,0
570    REM ******************************
580    REM *      LECTURA DE DATOS        *
590    REM ******************************
```

```
600   FOR I = Ø TO 62
610     READ A
620     POKE DS+I,A
630   NEXT I
640   RETURN
650 REM ****************************
660 REM *   DIBUJO DE UN EDIFICIO   *
670 REM ****************************
700 PRINT "<7 CR/AB>"
710 PRINT "<12 SPC><CTRL RVSON><COMM 2><SHFT L>
    <COMM H><6 SPC><COMM N>"
720 PRINT "<12 SPC><CTRL RVSON><COMM 2><SPC><COMM H>
    <2 CR/DE><2 SPC><2 CR/DE><COMM N>"
730 PRINT "<12 SPC><CTRL RVSON><COMM 2><SPC><COMM H>
    <6 SPC><COMM N>"
740 PRINT "<12 SPC><CTRL RVSON><COMM 2><SPC><COMM H>
    <2 CR/DE><2 SPC><2 CR/DE><COMM N>"
750 PRINT "<12 SPC><CTRL RVSON><COMM 2><SPC><COMM H>
    <6 SPC><COMM N>"
760 PRINT "<SPC><CTRL RVSON><COMM 1><SHFT L><COMM H>
    <11 SPC><COMM 2><2 CR/DE><2 SPC><2 CR/DE><COMM N>"
765 PRINT "<CTRL RVSON><COMM 1><SHFT L><SPC><COMM H>
    <CR/DE><SPC><CR/DE><SPC><CR/DE><SPC><CR/DE><SPC>
    <2 CR/DE><SPC><COMM 2><COMM H><5 SPC><COMM N>"
770 FOR I=1 TO 5
780   PRINT "<CTRL RVSON><COMM 1><2 SPC><COMM H><11 SPC>
      <COMM 2><2 CR/DE><2 SPC><2 CR/DE><COMM N>"
790   PRINT "<CTRL RVSON><COMM 1><2 SPC><COMM H><CR/DE>
      <SPC><CR/DE><SPC><CR/DE><SPC><CR/DE><SPC><2 CR/DE>
      <SPC><COMM 2><COMM H><5 SPC><COMM N>"
800 NEXT I
810 FOR T=1984 TO 2023
820   POKE T,160
830   POKE T+54272,7
840 NEXT T
850 RETURN
```

Programa 5.3

V.7.- COLISIONES.

Una colisión ocurre cuando alguna parte de un sprite se sobrepone a alguna parte de cualquier otro sprite o a algún caracter de la pantalla.

Una propiedad importante y de gran utilidad es la de poder detectar colisiones entre sprites, ya sea entre ellos mismos o entre ellos y cualquier otro caracter que se encuentre en la pantalla. Detectando colisiones es posible producir ciertos efectos en la animación: por ejemplo, al chocar cualesquiera dos objetos puede producirse una explosión.

La localidad 53278 sirve para detectar colisiones entre sprites. Cada sprite tiene un bit en esta localidad. Si este bit está encendido hay una colisión con ese sprite. El bit permanecerá prendido hasta que sea leído mediante la función PEEK. Las colisiones se detectan aún cuando el sprite se encuentre apagado en la pantalla. Análogamente la localidad 53279 detecta las colisiones entre un sprite y cualquier otro caracter en la pantalla.

Si el lector desea ver un ejemplo de estos dos tipos de colisiones, esto se consigue agregando las siguientes tres instrucciones al programa anterior. (Programa 5.3)

```
320 IF PEEK(53278)=0 AND PEEK(53279) THEN POKE 53287,7
330 IF PEEK(53278)=3 THEN POKE 53278,11
330 IF PEEK(53279)=1 THEN POKE 53278,15
```

Mientras exista una colisión entre el avión y la luna se cambiará el color del avión a gris oscuro, cambiándose a su color original al desaparecer dicha colisión. Análogamente mientras exista una colisión entre el avión y el edificio se cambiará el color del avión a gris claro.

V.8.- SPRITES EN MODO MULTICOLOR.

Este modo permite tener hasta cuatro colores en cada sprite, aunque esto tendrá el inconveniente de que para cada punto del dibujo se requerirán dos bits para representarlo y no sólo uno como en el modo normal. Bajo esta modalidad el dibujo del sprite tendrá que hacerse en una matriz de 21 x 12 puntos, es decir, se reducirá la resolución horizontal de 24 a 12 puntos.

Los cuatro colores que puede tener cada sprite son los siguientes : el color del sprite, el color determinado por la localidad 53285, el determinado por la localidad 53286 o el color del fondo. (Este último se verá en los puntos que esten apagados).

Para seleccionar cuál color tendrá cada punto se da a continuación una tabla de valores de cada par de bits :

VALOR DE LOS BITS	DESCRIPCION
00	COLOR DEL FONDO DE LA PANTALLA
01	COLOR DETERMINADO POR LA LOCALIDAD 53285
10	COLOR DETERMINADO POR EL REGISTRO DEL COLOR
11	COLOR DETERMINADO POR LA LOCALIDAD 53286

Figura 5.5

Para indicar al VIC-II que el sprite está en modo multicolor es necesario encender el bit correspondiente al sprite en la localidad 53276. La siguiente expresión nos premite hacer esto:

```
POKE 53276,PEEK(53276) OR (2^NS)
```

donde NS es el número del sprite (0 al 7).

Para volver al modo normal es necesario apagar el bit correspondiente, mediante la siguiente expresión:

```
POKE 53276,PEEK(53276) AND (255-2^NS)
```

La siguiente figura nos muestra como quedará el sprite multicolor en la pantalla.

```
    765432107654321076543210
 0
 1
 2                    **
 3                    **
 4                   *
 5                 *****
 6                ******
 7               XXXXXXXX
 8              XXXOXXOXXX
 9              XXXXXXXXXX
10              XXXXXXXXXX
11              XXXOXXOXXX
12               XXXOOXXX
13                XXXXXX
14                 XXXX
15
16
17
18
19
20
21
```

Figura 5.6

En donde los cuadros mostrados con una equis (X) tendrán el color determinado por la localidad de control del color del sprite correspondiente. Los cuadros que tienen la letra (O)

tendrán el color que indique la localidad 53286 (color 1). Los cuadros que contienen un asterisco (*) aparecerán del color determinado por la localidad 53285 y por último los cuadros que aparecen en blanco tendrán el color del fondo de la pantalla.

Basándonos en lo anterior se obtiene la siguiente figura del sprite que indica de donde tomará su color cada punto así como el valor decimal de cada byte.

	76543210765432107654321 0	valor decimal		
0	000000000000000000000000	0	0	0
1	000000000000000000010100	0	0	20
2	000000000000000000010100	0	0	20
3	000000000000000001000000	0	0	64
4	000000000101010101000000	0	85	64
5	000000010101010101000000	1	85	64
6	000010101010101010100000	10	170	160
7	001010101110101110101000	42	235	168
8	001010101010101010101000	42	170	168
9	001010101010101010101000	42	170	168
10	001010101110101110101000	42	235	168
11	000010101011111010100000	10	190	160
12	000000101010101010000000	2	170	128
13	000000001010101000000000	0	170	0
14	000000000000000000000000	0	0	0
15	000000000000000000000000	0	0	0
16	000000000000000000000000	0	0	0
17	000000000000000000000000	0	0	0
18	000000000000000000000000	0	0	0
19	000000000000000000000000	0	0	0
20	000000000000000000000000	0	0	0

Figura 5.7. Dibujo codificado y valor decimal de cada byte del sprite de la figura 5.6.

El siguiente programa nos muestra el sprite en la pantalla.

```
10  REM ********************************************
20  REM *         SPRITE EN MODO MULTICOLOR          *
30  REM ********************************************
40  REM
50  PRINT "<SHFT CLR/HOME>"
60  POKE 53280,0  : REM COLOR BORDE PANTALLA = NEGRO
70  POKE 53281,1  : REM COLOR FONDO PANTALLA = BLANCO
80  POKE 2040,13  : REM APUNTADOR A  DATOS DEL SPRITE 0
90  DS = 832      : REM INICIO DE LOS DATOS DEL SPRITE 0
100 GOSUB 600      : REM LECTURA DE LOS DATOS
110 POKE 53276,1   : REM PRENDE MODO MULTICOLOR
120 POKE 53287,7   : REM COLOR DEL SPRITE (AMARILLO)
130 POKE 53285,14  : REM COLOR 1 (AZUL)
140 POKE 53286,2   : REM COLOR 2 (ROJO)
150 POKE 53248,170 : REM POSICION X
160 POKE 53249,130 : REM POSICION Y
170 POKE 53271,1   : REM EXPANSION VERTICAL
180 POKE 53269,1   : REM PRENDE EL SPRITE 0
190 GET A$ : IF A$="" THEN 190
200 POKE 53269,0   : APAGA EL SPRITE 0
210 END
300 DATA 0,0,0,0,0,20,0,0,20,0,0,64,0,85,64,1,85,64,10,
        170,160
310 DATA 42,235,168,42,170,168,42,170,168,42,235,168,10,
        190,160,2,170,128,0
320 DATA 170,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
600 FOR I = 0 TO 62
610   READ A
620   POKE DS+I,A
630 NEXT I
640 RETURN
```

Programa 5.4

Figura 5.8. Imagen generada por el programa 5.4.

V.9.- EDITOR DE SPRITES.

Al correr este programa lo primero que aparece en la pantalla es un cuadro de 24 por 21 puntos en el que será definido un sprite.

Para crear un sprite es necesario moverse con las teclas que controlan el movimiento del cursor hasta el lugar en donde se desee modificar un punto ".". Para borrarlo se debe oprimir la barra espaciadora.

Al ir modificando esta área se observará que del lado derecho de la pantalla, va apareciendo el sprite correspondiente al patrón que se está creando.

Además contamos con otras funciones que nos ayudan a definir los atributos del sprite, para que ste sea el adecuado con las características que se tengan en otro programa al que se vaya a incorporar. Es decir, podemos cambiar el color al sprite o al fondo de la pantalla, para ver cual es la combinación de colores más adecuada. Además es posible expanderlo horizontal o verticalmente o regresarlo a su tamaño original. Otra posibilidad con que cuenta este editor es el poder guardar un sprite ya creado para posteriormente usarlo, mediante este programa. Además el editor proporciona los datos del sprite, el valor decimal de cada byte que lo compone.

A continuación se da una lista de las funciones con las que cuenta este editor, así como de las teclas que producen dichas funciones:

TECLA	FUNCION
<SHFT/CLR>	Borra la pantalla y el sprite
B	Cambia el color del borde de la pantalla
P	Cambia el color del fondo de la pantalla
C	Cambia el color al sprite
H	Expansión horizontal / Tamaño original
V	Expansión vertical / Tamaño original
F1	Carga un sprite del disco
F3	Guarda en disco el sprite
I	Despliega los datos del sprite
<RETURN>	Regresar al modo de edición sin guardar o cargar un sprite.

Figura 5.9. Funciones del editor de sprites.

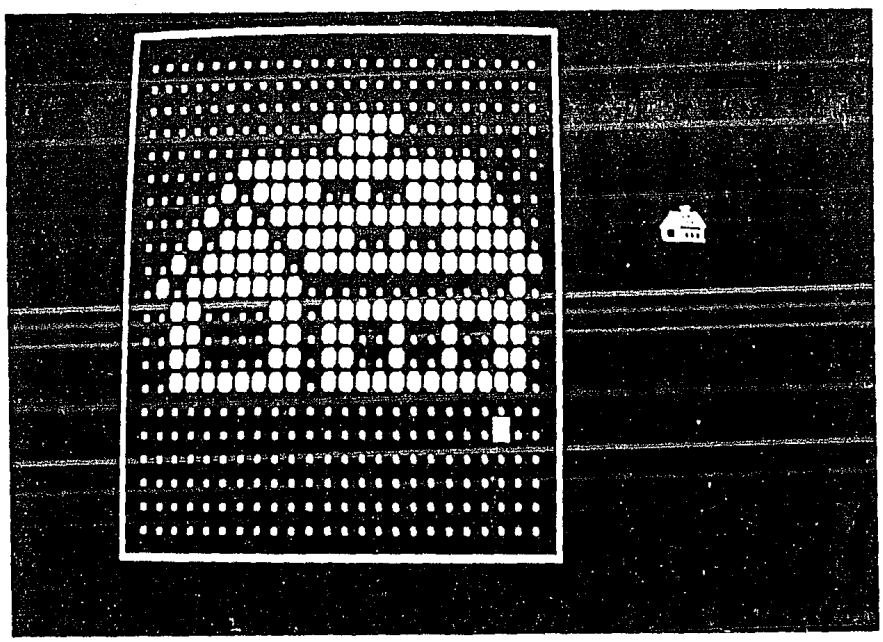


Figura 5.10. Imagen generada por el programa 5.5.

```
10 REM   ****************************
20 REM   *     EDITOR DE SPRITES    *
30 REM   ****************************
40 DIM G(21,24)
50 DIM A(2)
60 GOSUB 100 : REM PANTALLA
70 GOSUB 390 : REM SPRITE
80 GOSUB 560 : REM CICLO
90 END
100 REM ****************************
110 REM *   INICIALIZAR PANTALLA   *
120 REM ****************************
130 PRINT "<CTRL 2>"
140 CB=6
150 CP=6
160 POKE 53280,CB
170 POKE 53281,CP
180 CS=1
190 D=-1
200 EH=-1
210 EV=-1
220 X=0
230 Y=0
240 L=1065
250 PRINT "<SHFT CLR/HOME>";
260 PRINT "<SHIFT O><24 COMM Y><SHIFT P>"
270 FOR R=1 TO 21
280 PRINT "<COMM H>";
290 FOR C=1 TO 24
300 PRINT CHR$(46);
310 G(R,C)=46
320 NEXT C
330 PRINT "<COMM N>"
340 NEXT R
350 PRINT "<SHIFT L><24 COMM P><SHIFT @>"
360 PRINT "<HOME>"
370 RETURN
380 REM
390 REM ****************************
400 REM *   INICIALIZAR SPRITE     *
410 REM ****************************
420 NS=0
430 FOR I=0 TO 62
440 POKE 832+I,0
450 NEXT I
460 POKE 2040,13    : REM LOC. DATOS
470 POKE 53269,1    : REM PRENDER SPRITE 0
480 POKE 53287,CS   : REM COLOR
490 POKE 53248,22   : REM POS. X
500 POKE 53249,110  : REM POS. Y
510 POKE 53264,1    : REM POS. X > 255
520 POKE 53277,0    : REM EXP. HORIZ.
530 POKE 53271,0    : REM EXP. VERTICAL
540 RETURN
```

```
560 REM ****************
570 REM *      CICLO      *
580 REM ****************
590 GET A$ : IF A$<>"" THEN 630
600 D=D*-1 : CA=PEEK(LOC)+128*D : POKE LOC,CA
610 FOR Z=1 TO 100 : NEXT Z
620 GOTO 590
630 IF CA>128 THEN CA=CA-128 : D=D*(-1)
640 POKE LOC,CA
650 IF A$="H" OR A$="V" OR A$="B" OR A$="P" OR A$="C"
    THEN 760
660 IF A$="" THEN 590
670 IF A$="<CR/AR>" THEN Y=Y-1
680 IF A$="<CR/AB>" THEN Y=Y+1
690 IF A$="<CR/DE>" THEN X=X+1
700 IF A$="<CR/IZ>" THEN X=X-1
710 IF X>23 THEN X=0
720 IF X<0 THEN X=23
730 IF Y>20 THEN Y=0
740 IF Y<0 THEN Y=20
750 REM
760 IF A$="H" THEN GOSUB 1190
770 IF A$="V" THEN GOSUB 1260
780 IF A$="B" THEN GOSUB 980
790 IF A$="P" THEN GOSUB 1050
800 IF A$="C" THEN GOSUB 1120
810 IF A$=CHR$(133) THEN GOSUB 1330
820 IF A$=CHR$(134) THEN GOSUB 1550
830 IF A$="I" THEN GOSUB 2040
840 IF A$="<SHFT CLR/HOME>" THEN 60
850 MS=0
860 IF A$="." THEN CA=81 : MS=1
870 IF A$=" " THEN CA=46 : MS=1
880 LOC=1065+Y*40+X
890 IF MS=0 THEN CA=PEEK(LOC)
900 POKE LOC,CA
910 G(Y+1,X+1)=CA
920 IF MS=0 THEN 960
930 GOSUB 1760
940 X=X+1 : IF X>23 THEN X=0
950 LOC=1065 + Y*40 + X
960 GOTO 560
970 RETURN
980 REM  ********************
990 REM  *    COLOR BORDE     *
1000 REM ********************
1010 CB=CB+1
1020 CB=CB AND 15
1030 POKE 53280,CB
1040 RETURN
1050 REM ************************
1060 REM *   COLOR PANTALLA     *
1070 REM ************************
1080 CP=CP+1
1090 CP=CP AND 15
```

```
1100 POKE 53281,CP
1110 RETURN
1120 REM **********************
1130 REM *     COLOR SPRITE     *
1140 REM **********************
1150 CS=CS+1
1160 CS=CS AND 15
1170 POKE 53287,CS
1180 RETURN
1190 REM ****************************
1200 REM *   EXPANSION HORIZONTAL   *
1210 REM ****************************
1220 EH=EH*(-1)
1230 IF EH>0 THEN POKE 53277,1
1240 IF EH<0 THEN POKE 53277,0
1250 RETURN
1260 REM ****************************
1270 REM *   EXPANSION VERTICAL     *
1280 REM ****************************
1290 EV=EV*(-1)
1300 IF EV>0 THEN POKE 53271,1
1310 IF EV<0 THEN POKE 53271,0
1320 RETURN
1330 REM **********************
1340 REM *   CARGAR SPRITE     *
1350 REM **********************
1360 PRINT "<HOME><18 CR/AB><27 CR/DE>";
1370 PRINT "<CARGAR>"
1380 PRINT "<CR/AB><27 CR/DE>";
1390 PRINT "NOMBRE:";
1400 PRINT "<CR/AB><7 CR/IZ>";
1405 NOM$=""
1410 INPUT NOM$
1420 IF NOM$="" THEN 1500
1430 OPEN 2,8,2,NOM$+",SEQ,R"
1440 FOR I=0 TO 62
1450 INPUT #2,DA
1460 POKE 832+I,DA
1470 NEXT I
1480 CLOSE 2
1490 GOSUB 1850
1500 PRINT "<HOME><18 CR/AB><27 CR/DE><8 SPC>";
1510 PRINT "<CR/AB><27 CR/DE><8 SPC>";
1520 PRINT "<CR/AB><7 CR/IZ>";
1530 PRINT "<11 SPC><HOME>"
1540 RETURN
1550 REM ************************
1560 REM *     GRABAR SPRITE     *
1570 REM ************************
1580 PRINT "<HOME><18 CR/AB><27 CR/DE>";
1590 PRINT "<GRABAR>"
1600 PRINT "<CR/AB><27 CR/DE>";
1610 PRINT "NOMBRE:";
1620 PRINT "<CR/AB><7 CR/IZ>";
1625 NOM$=""
```

```
1630 INPUT NOM$
1640 IF NOM$="" THEN 1710
1650 OPEN 2,8,2,"@0:"+NOM$+",SEQ,W"
1660 FOR I=0 TO 62
1670 DA=PEEK(832+I)
1680 PRINT #2,DA
1690 NEXT I
1700 CLOSE 2
1710 PRINT "<HOME><18 CR/AB><27 CR/DE><8 SPC>"
1720 PRINT "<CR/AB><27 CR/DE><7 SPC>";
1730 PRINT "<CR/AB><7 CR/IZ>";
1740 PRINT "<11 SPC>"
1750 RETURN
1760 REM ************************
1770 REM *   ACTUALIZAR SPRITE   *
1780 REM ************************
1790 BYTE=Y*3+INT(X/8)
1800 BIT=7-(X AND 7)
1810 LS=832+BYTE
1820 IF A$=" " THEN POKE LS,PEEK(LS) AND (255-2^BIT)
1830 IF A$="." THEN POKE LS,PEEK(LS) OR (2^BIT)
1840 RETURN
1850 REM ***************************
1860 REM *   ACTUALIZAR PANTALLA   *
1870 REM ***************************
1880 FOR R=0 TO 20
1890 FOR I=0 TO 2
1900 LS=832+R*3+I
1910 FOR K=0 TO 7
1920 C=I*8+K
1930 BIT=7-(K AND 7)
1940 VA=2^BIT-(PEEK(LS) AND (2^BIT))
1950 IF VA=0 THEN CR=81
1960 IF VA<>0 THEN CR=46
1970 LP=1065+R*40+I*8+K
1980 G(R+1,C+1)=CR
1990 POKE LP,CR
2000 NEXT K
2010 NEXT I
2020 NEXT R
2030 RETURN
2040 REM ******************************
2050 REM *   IMPRIMIR DATOS SPRITE   *
2060 REM ******************************
2070 LS=832
2080 POKE 53269,0
2090 POKE 53280,1
2100 PRINT "<SHFT CLR/HOME>"
2110 FOR I=0 TO 62 STEP 3
2120 A1=PEEK(LS+I)
2130 A2=PEEK(LS+I+1)
2140 A3=PEEK(LS+I+2)
2150 PRINT TAB(15);A1;TAB(20);A2;TAB(25);A3
2160 NEXT I
2170 PRINT
```

```
2180 PRINT TAB(10)"<RETURN> PARA CONTINUAR"
2190 GET B$ : IF B$<>CHR$(13) THEN 2190
2200 POKE 53280,CB
2210 PRINT "<SHFT CLR/HOME>"
2220 PRINT "<SHIFT O><24 COMM Y><SHIFT P>"
2230 FOR R=0 TO 20
2240 PRINT "<COMM H><24 SPC><COMM N>"
2250 FOR C=0 TO 23
2260 POKE 1065+R*40+C,G(R+1,C+1)
2270 NEXT C
2280 NEXT R
2290 PRINT "<SHIFT L><24 COMM P><SHIFT @>"
2300 POKE 53269,1
2310 RETURN
```

Programa 5.5

NOTA: Debido a que se usa la instrucción GET para determinar que tecla se oprimió, el cursor no aparece en la pantalla, lo cual nos trae el problema de no saber en que posición se encuentra el cursor al editar el sprite. Para evitar este problema nos sirven las instrucciones de la 600 a la 640. Lo que se hace en ellas es estar imitando al cursor, es decir, se está cambiando constantemente el caracter que se encuentra en la pantalla por su caracter inverso correspondiente.

CAPITULO VI

ALTA RESOLUCION

La alta resolución de la pantalla consiste en prender o apagar bits de las localidades correspondientes a la pantalla. Es decir, cada punto (o "pixel") de resolución en la pantalla tiene asignado un bit en la memoria. Si ese bit tiene un 0 el punto asignado a él estará apagado y si el bit es un 1 el punto estará prendido.

La resolución de la Commodore es de 320x200 puntos para toda la pantalla. Esto se debe a que la pantalla es un arreglo de 25 renglones por 40 columnas, lo cual da un total de 1000 caracteres. Como cada caracter está formado por una matriz de 8x8 puntos, ésto da un total de 320 puntos (40x8) horizontalmente y 200 puntos (25x8) verticalmente, es decir, 64,000 puntos; cada uno de los cuales necesita un bit en memoria. Por lo tanto se necesitan 8,000 bytes de memoria (8k). Debido a que esta cantidad de memoria necesaria es considerable, la alta resolución no se usa todo el tiempo.

Esta forma de alta resolución es única en las Commodore. En otras computadoras personales con gráficas en alta resolución, se maneja la pantalla de alta resolución como una sucesión de bits y no como un arreglo de caracteres. La ventaja de tomarlo de esta forma es la velocidad, ya que es más rápido encontrar un bit y prenderlo que con la matriz de caracteres de la Commodore. Por otro lado la ventaja de la Commodore es que es compatible con todas las modalidades de graficación con texto.

Hay dos formas de alta resolución: normal y multicolor. La

diferencia entre ambas es que en el modo normal la resolución es de 320x200 puntos, mientras que en el modo multicolor la resolución es de 160x200 puntos. Es decir, un par de bits controla un solo punto.

VI.1.- MODO NORMAL DE ALTA RESOLUCION.

En este modo, como se mencionó anteriormente, se tienen 200 líneas por 320 puntos, pudiendo seleccionar 2 colores en cada sección de 8 por 8 puntos. Para entrar a este modo de alta resolución es necesario prender el bit 5 de la localidad 53265, lo cual se hace con la siguiente expresión:

```
POKE 53265,PEEK(53265) OR 32
```

Para salir de la alta resolución se apaga este bit con la expresión:

```
POKE 53265,PEEK(53265) AND 223
```

Al entrar a la alta resolución se verán cuadros de diferentes colores y, dependiendo de lo que se estaba haciendo antes de entrar, aparecerán puntos o caracteres esparcidos en la pantalla. Además, dependiendo de las localidades de memoria que se usen para proyectar o acomodar a la pantalla, la basura que haya en dichas localidades aparecerá en la pantalla.

Los bloques de color representan la pantalla de texto. Los colores de aquéllos no se toman de la memoria del color como en el caso del modo de caracteres, sino que se toman de la memoria de la pantalla. Por ejemplo, en lugar de poner un 5 en la localidad 1024 para que aparezca una "E" en la esquina

diferencia entre ambas es que en el modo normal la resolución es de 320x200 puntos, mientras que en el modo multicolor la resolución es de 160x200 puntos. Es decir, un par de bits controla un solo punto.

VI.1.- MODO NORMAL DE ALTA RESOLUCION.

En este modo, como se mencionó anteriormente, se tienen 200 líneas por 320 puntos, pudiendo seleccionar 2 colores en cada sección de 8 por 8 puntos. Para entrar a este modo de alta resolución es necesario prender el bit 5 de la localidad 53265, lo cual se hace con la siguiente expresión:

```
POKE 53265,PEEK(53265) OR 32
```

Para salir de la alta resolución se apaga este bit con la expresión:

```
POKE 53265,PEEK(53265) AND 223
```

Al entrar a la alta resolución se verán cuadros de diferentes colores y, dependiendo de lo que se estaba haciendo antes de entrar, aparecerán puntos o caracteres esparcidos en la pantalla. Además, dependiendo de las localidades de memoria que se usen para proyectar o acomodar a la pantalla, la basura que haya en dichas localidades aparecerá en la pantalla.

Los bloques de color representan la pantalla de texto. Los colores de aquéllos no se toman de la memoria del color como en el caso del modo de caracteres, sino que se toman de la memoria de la pantalla. Por ejemplo, en lugar de poner un 5 en la localidad 1024 para que aparezca una "E" en la esquina

superior izquierda de la pantalla, el valor que haya en esta localidad será el que indique cuales 2 colores podrán tener los bits en ese cuadro de 8 por 8. Los cuatro bits altos, es decir, los bits del 7 al 4 de cada una de las localidades de la memoria de la pantalla, controlarán el color de cada bit que esté prendido dentro del área de 8 por 8 puntos controlada por esta localidad. Los bits del 0 al 3 controlan el color de los bits que estén apagados. A partir del número que tiene asignado cada color, el valor que debe colocarse en cada localidad para elegir los colores con el cual se desea dibujar y el color del fondo de la pantalla se obtiene de la siguiente fórmula:

VALOR = (CD * 16) + CF

Donde CD es el número asignado al color con el que se desea dibujar y CF es el color que tendrá el fondo de la pantalla en ese cuadro.

Por ejemplo, para dibujar con color azul en un fondo de color amarillo se haría lo siguiente:

POKE LOC,103

El número 103 se obtiene de poner el número del color de dibujo (azul), en los bits del 7 al 4 y el número del color del fondo (amarillo) en los bits del 0 al 3.

```
              DIBUJO    FONDO
BITS         7 6 5 4   3 2 1 0
             0 1 1 0   0 1 1 1
               AZUL    AMARILLO
```

El número 01100111, obtenido de estas operaciones, corresponde al 103, es decir, (6x16)+7=103.

Otro problema que debe tomarse en cuenta es el de decidir en que parte de la memoria deben colocarse los 8k que forman la pantalla de alta resolución. Para informarle al VIC-II de donde debe tomar los datos se usa la localidad 53272. Debido a que son necesarios 8k, sólo hay dos posibles localidades en donde puede colocarse la pantalla de alta resolución dentro de cada banco de 16k. Estas localidades son la 0 y la 8192. Como ya se dijo anteriormente, la localidad 53272 también controla en donde se encuentra la memoria del color (memoria de la pantalla). Para indicar al VIC-II donde colocar la pantalla de alta resolución y donde colocar la memoria del color se tiene la siguiente expresión:

```
POKE 53272,MC/64 + AR/1024
```

Donde MC es la localidad de inicio de la memoria del color y AR es la localidad de inicio de la pantalla de alta resolución.

Por ejemplo, para colocar la pantalla de alta resolución en las localidades 8192 a la 16383 sin modificar la dirección de donde ubicar a la memoria del color (1024-2023) se hace lo siguiente:

```
POKE 53272,PEEK(53272) OR 8
```

Ya teniendo las expresiones anteriores se puede empezar a preparar la pantalla para dibujar mediante las siguientes instrucciones:

```
BASE=8192
POKE 53272,PEEK(53272) OR 8
POKE 53265,PEEK(53265) OR 32
```

Como ya se mencionó, al pasar a la alta resolución se tendrá basura en la pantalla por lo que es necesario limpiarla. Esto se hace limpiando la sección de la memoria que se ha elegido para graficar, mediante las siguientes instrucciones:

```
FOR I=BASE TO BASE+7999
POKE I,0
NEXT I
```

Las siguientes instrucciones sirven para elegir el color azul para dibujar y el color blanco para el fondo.

```
FOR I=1024 TO 2023
POKE I,49
NEXT I
```

(Si se ejecutan todas las instrucciones anteriores, primero se verá como se va limpiando la pantalla para después cubrirse toda de blanco).

En seguida la pantalla está lista para prender y apagar bits. Para hacer ésto debe encontrarse el bit correcto que debe prenderse, es decir, hay que encontrar dentro de cuál caracter se encuentra este bit, así como en que renglón de ese caracter está y cual bit de ese renglón (columna) debe prenderse. Las siguientes fórmulas calculan todos estos valores.

```
RENGLON = INT(Y/8)
CARACTER = INT(X/8)
LINEA = Y AND 7
BIT = 7 - (X AND 7)
BYTE=BASE+RENGLON*320+CARACTER*8+LINEA
```

Por último, para prender cualquier punto de coordenadas (X,Y), se tiene la expresión:

```
POKE BYTE,PEEK(BYTE) OR (2^BIT)
```

Por ejemplo, si queremos prender el punto con coordenadas (5,12), es decir, X=5 y Y=12, las dos primeras instrucciones indican en que renglón de la pantalla está ese punto y a que caracter de ese renglón corresponde. En este caso

RENGLON = INT (12/8) = 1

CARACTER = INT(5/8) = 0

Las dos siguientes expresiones nos indican en que línea se encuentra el punto dentro de ese caracter y finalmente, dentro de esa línea, cuál bit es el correspondiente a ese punto.

```
bits    7 6 5 4 3 2 1 0

Y=22    0 0 0 0 1 1 0 0
AND 7   0 0 0 0 0 1 1 1
        -----------------
        0 0 0 0 0 1 0 0   = 4 = LINEA

bits    7 6 5 4 3 2 1 0

X=5     0 0 0 0 0 1 0 1
AND 7   0 0 0 0 0 1 1 1
        -----------------
        0 0 0 0 0 1 0 1   = 5 = BIT
```

Resultando ser el bit 2 de la línea 6.

La instrucción que nos indicará la localidad de memoria que debe ser afectada para prender ese punto es:

BYTE=BASE+RENGLON*320+CARACTER*8+LINEA

que en este caso tendrá el valor de 8516

BYTE = 8192 + 1*320 + 0*8 + 4 = 8516

Por último la expresión:

POKE 8516,PEEK(8516) OR 4

prende el bit 2 de la localidad 8516 sin afectar los demás bits.

8192	8200
8193	8201
8194	8202
8195	8203
8196	8204
8197	8205
8198	8206
8199	8207
8512	8520
8513	8521
8514	8522
8515	8523
8516	8524
8517	8525
8518	8526
8519	8527
.	.
.	.
.	.

Figura 6.1. Localidades correspondientes a los bytes de la figura 6.2, asumiendo que la pantalla de alta resolución comienza en la localidad 8192.

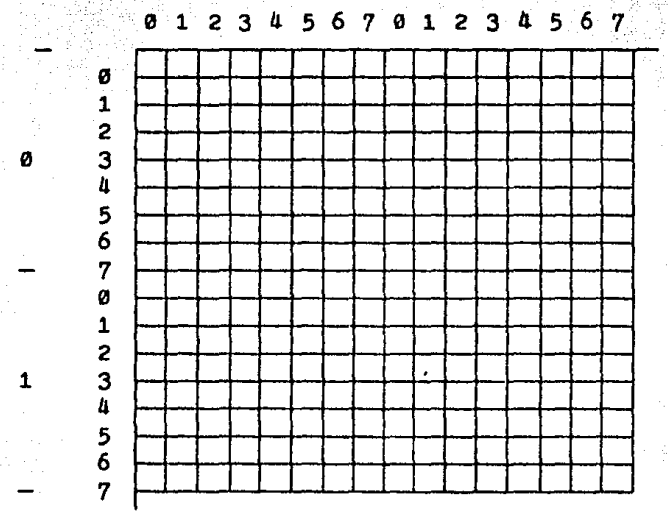


Figura 6.2. Organización del mapa de bits de la pantalla de alta resolución.

```
  1   REM ******************************
  2   REM * CIRCULO EN ALTA RESOLUCION *
  3   REM ******************************
 10   BASE=8192
 20   PANT=1024
 30   PRINT "<SHFT CLR/HOME>"
 40   PRINT "<HOME><5 CR/AB> COORDENADAS DEL CENTRO (X,Y):";
 50   INPUT XC,YC
 55   IF XC>320 OR YC>200 THEN 40
 60   PRINT "<HOME><8 CR/AB> RADIO HORIZONTAL:";
 70   INPUT RX
 75   IF XC+RX>320 OR XC-RX<0 THEN 60
 80   PRINT "<HOME><10 CR/AB> RADIO VERTICAL  :";
 90   INPUT RY
 95   IF YC+RY>200 OR YC-RY<0 THEN 80
140   POKE 251,0
150   POKE 252,32
160   POKE 53272,PEEK(53272) OR 8
170   POKE 53265,PEEK(53265) OR 32
180   FOR I=BASE TO BASE+7999
182    POKE I,0
184   NEXT I
190   FOR I=PANT TO PANT+999
200     POKE I,1
210   NEXT I
220   POKE 53280,6
230   POKE 53281,0
```

```
240 GOSUB 570
250 GET T$
260 IF T$="" THEN 250
270 PRINT "<SHFT CLR/HOME>"
280 POKE 53265,PEEK(53265) AND (255-32)
290 POKE 53272,PEEK(53272) AND (255-8)
300 END
310 REM ******************************
320 REM *   IMPRESION DE LOS PUNTOS  *
330 REM ******************************
500 RE=INT(Y/8)
510 CAR=INT(X/8)
520 LIN=Y AND 7
530 BIT=7-(X AND 7)
540 BYTE=BASE+RE*320+CAR*8+LIN
550 POKE BYTE,PEEK(BYTE) OR (2^BIT)
560 RETURN
570 REM ******************************
580 REM *        C I R C U L O       *
590 REM ******************************
600 IF RX>=RY THEN RA=INT(RX/20)+1
610 IF RY>RX THEN RA=INT(RY/20)+1
620 FOR P=0 TO 2*π  STEP (π/(32*RA))
630   X=XC+RX*COS(P)
640   Y=YC+RY*SIN(P)
650   Y=Y*0.8
660   GOSUB 310
670 NEXT P
680 RETURN
```

Programa 6.1

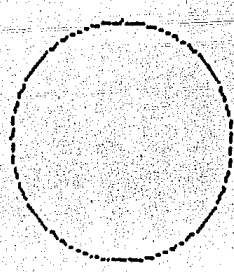


Figura 6.3. Gráfica de una circunferencia con centro en 160,100 y sus radios horizontal y vertical igual a 50 (Programa 6.1).

El programa anterior dibuja una circunferencia, dando las coordenadas del centro, la longitud de su radio horizontal y la longitud de su radio vertical. Como se podrá observar la velocidad a la cual se dibujan los puntos no es muy alta. Pero sin duda lo más tardado del programa es la inicialización de la pantalla de alta resolución, debido a que se limpian 8,000 localidades de memoria. A continuación daremos dos subrutinas en lenguaje de maquina, que se encargan de preparar la pantalla de alta resolución pero con una mayor velocidad. La primera de ellas (49152) se encarga de inicializar la pantalla. La segunda (40705) se encarga de asignar el color con el que se dibuja y el color que tendrá el fondo de la pantalla. Para incorporar estas subrutinas al programa anterior sólo es necesario teclear las siguientes líneas.

```
100 FOR I=0 TO 20
110   READ T
120   POKE 49152+I,T
130 NEXT I
180 SYS 49152
190 FOR I=40705 TO 40730
195 READ D
200 POKE I,D
205 NEXT I
210 SYS 40705
900 DATA 169,00,170,168,145,251,200,240,02,208,249,230
910 DATA 252,232,224,32,240,02,208,240,96
920 DATA 169,1,162,0,134,150,162,4,134,151,160,0,145,150
930 DATA 200,208,251,230,151,166,151,224,8,208,243,96
```

La instrucciones 100,110,120,130 y 180 sustituyen a las instrucciones 180, 182 y 184 del programa anterior. La instrucción 180 llama a la subrutina en lenguaje de máquina.

Los datos de las líneas 900 y 910 corresponden a los códigos de las instrucciones en lenguaje de máquina.

Las instrucciones 190,195,200,205 y 210 sustituyen a las instrucciones 190,200 y 210. Los datos de las instrucciones 920 y 930 son los códigos de las instrucciones en lenguaje de máquina de dicha subrutina (40705).

Al correr este programa se verá que el incremento en la velocidad es considerablemente mayor.

Cambiando las siguientes líneas al programa 6.1 se podrá dibujar un rectángulo, dando las coordenadas de los extremos superior izquierdo e inferior derecho.

```
1   REM ***********************************************
2   REM *          GRAFICA DE UN RECTANGULO           *
3   REM ***********************************************
40  PRINT  "<5 CR/AB> DAME LAS  SIGUIENTES  COORDENADAS
    (X,Y):"
50  PRINT "<HOME><8 CR/AB> ESQUINA SUPERIOR IZQUIERDA:";
60  INPUT X1,Y1
65  IF X1>320 OR Y1>200 THEN 50
70  PRINT "<HOME><10 CR/AB> ESQUINA INFERIOR DERECHA: ";
80  INPUT X2,Y2
85  IF X2>320 OR Y2>200 THEN 70
90
570 REM **********************
580 REM *     RECTANGULO     *
590 REM **********************
600 FOR X=X1 TO X2
610   Y=Y1
620   GOSUB 310
630   Y=Y2
640   GOSUB 310
650 NEXT X
660 FOR Y=Y1 TO Y2
670   X=X1
680   GOSUB 310
690   X=X2
700   GOSUB 310
710 NEXT Y
720 RETURN
```

Programa 6.2

Si nuevamente se cambian las siguientes instrucciones al programa 6.2 se podrá dibujar cualquier recta, si se dan las coordenadas de sus extremos.

```
1   REM ****************************************
2   REM *              GRAFICA DE UNA RECTA      *
3   REM ****************************************
40  PRINT "<5 CR/AB> DAME LAS COORDENADAS DE LOS EXTREMOS:"
50  PRINT "<HOME><8 CR/AB> EXTREMO 1 (X,Y):";
60  INPUT X1,Y1
65  IF X1>320 OR Y1>200 THEN 50
70  PRINT "<HOME><10 CR/AB> EXTREMO 2 (X,Y):";
80  INPUT X2,Y2
85  IF X2>320 OR Y2>200 THEN 70
570 REM ****************
580 REM *      RECTA     *
590 REM ****************
600 IF X1>=X2 THEN XI=X2 : XF=X1 : GOTO 620
610 XI=X1 : XF=X2
620 IF Y1>=Y2 THEN YI=Y2 : YF=Y1 : GOTO 640
630 YI=Y1 : YF=Y2
640 IF X1=X2 THEN 730
650 IF Y1=Y2 THEN 780
660 M=(Y2-Y1)/(X2-X1)
670 IF (M>0 AND M<1) OR (M<0 AND M>-1) THEN 830
680 FOR Y=YI TO YF
690   X=INT((Y-Y1)/M+X1)
700   GOSUB 310
710 NEXT Y
720 RETURN
730 FOR Y=YI TO YF
740   X=X1
750   GOSUB 310
760 NEXT Y
770 RETURN
780 FOR X=XI TO XF
790   Y=Y1
800   GOSUB 310
810 NEXT X
820 RETURN
830 FOR X=XI TO XF
840   Y=M*(X-X1)+Y1
850   GOSUB 310
860 NEXT X
870 RETURN
```

Programa 6.3

VI.2.- MODO MULTICOLOR DE ALTA RESOLUCION.

Este modo de alta resolución permite tener hasta cuatro colores diferentes, dentro de cada sección de 8x8 puntos en las que es dividida la alta resolución para el control del color. Como se mencionó anteriormente, en este modo la resolución es de 160 por 200 puntos.

Al igual que en el modo multicolor de "sprites" (Capitulo 5), cada punto de la pantalla necesita 2 bits para poderse representar. Por esta razón la resolución horizontal se reduce de 320 a 160 puntos.

Las distintas combinaciones que resultan de prender o apagar un bit de cada pareja de bits (00,01,10,11), nos indicarán que color se ha elegido para cada punto.

Los colores de cada sección de 8x8 se seleccionan de:

- La localidad que controla el color del fondo de la pantalla.
- Los 4 bits altos de la localidad correspondiente en la memoria de la pantalla.
- Los 4 bits bajos de la localidad correspondiente en la memoria de la pantalla.
- La memoria del color (RAM).

BITS	LOCALIDAD
00	53281
01	1024+LOC (los 4 bits altos)
10	1024+LOC (los 4 bits bajos)
11	55296+LOC

Figura 6.4. Localidades de control del color del modo multicolor de alta resolución.
LOC = localidad correspondiente al área de 8x8 dentro de la cual se encuentra el punto que se va a dibujar.

Estando dentro de la alta resolución, para pasar al modo multicolor es necesario prender el bit 4 de la localidad 53270, lo cual se hace con la expresión:

```
POKE 53270,PEEK(53270) OR 16
```

Para regresar al modo normal de alta resolución se tiene la expresión:

```
POKE 53270,PEEK(53270) AND 239
```

VI.3.- EDITOR DE IMAGENES EN ALTA RESOLUCION

El siguiente programa (Programa 6.4) es un editor de imágenes en alta resolución en el cual se utiliza la palanca de juegos para dibujar.

Los movimientos del cursor se realizarán de acuerdo a los movimientos de la palanca de juegos, ésta tiene 8 posibles direcciones que son :

VALOR DE LA PALANCA DE JUEGOS	D I R E C C I O N
0	NO HAY MOVIMIENTO
1	ARIBA
2	ABAJO
3	----------
4	IZQUIERDA
5	ARRIBA IZQUIERDA
6	ABAJO IZQUIERDA
7	----------
8	DERECHA
9	ARRIBA DERECHA
10	ABAJO DERECHA

Además la palanca de juegos cuenta con un botón que tiene dos estados, estar oprimido o no.

VALOR DEL BOTON	E S T A D O
0	SIN OPRIMIR
16	OPRIMIDO

En este programa existen dos modos en los que puede ser desplazado el cursor, que son:

MOVER.- En este modo es posible desplazar el cursor sin dibujar.

DIBUJAR.-En este modo es posible prender los puntos al desplazar el cursor, logrando de esta manera dibujar.

Las funciones que pueden usarse en este editor y las teclas que las producen son:

F1 : GUARDA LA IMAGEN EN DISCO

F3 : LEE LA IMAGEN DEL DISCO Y LA DESPLIEGA EN PANTALLA

M : DESPLAZA EL CURSOR SIN PINTAR (MODO MOVER)

D : DESPLAZA EL CURSOR PINTANDO (MODO DIBUJAR)

+ : INCREMENTA EN UN PUNTO EL DESPLAZAMIENTO DEL CURSOR

- : DECREMENTA EN UN PUNTO EL DESPLAZAMIENTO DEL CURSOR

Cuando se inicia el Programa 6.4 el cursor aparecerá en el centro de la pantalla, al desplazar el cursor éste no pintará ningún punto ya que se encuentra en el modo mover. En el momento en el que se desee dibujar se tendrá que oprimir la tecla "D" para cambiar al modo dibujar y ahora al desplazar el cursor empezaremos a dibujar. Para regresar al modo mover tendremos que oprimir la tecla "M".

Si deseamos borrar algun(os) punto(s) tendremos que hacer lo

mismo que para dibujar, sólo que ahora al desplazarnos oprimiremos simultáneamente el botón de de la palanca de juegos, obteniendo el borrado de los puntos por los que se está desplazando el cursor. Esto se logra al cambiar el color de los puntos al color del fondo de la pantalla obteniendo un efecto de borrado.

```
10   REM ****************************************
20   REM * EDITOR DE IMAGENES EN ALTA RESOLUCION *
30   REM ****************************************
40   INC = 1
50   X = 160
60   Y = 100
70   BASE = 8192
80   PANT = 1024
90   POKE 251,0
100  POKE 252,32
110  FOR I = 0 TO 20
120    READ T
130    POKE 49152+I, T
140  NEXT I
150  DATA 169,00,170,168,145,251,200,240,02,208,249,230
160  DATA 252,232,224,32,240,02,208,240,96
170  POKE 53272,PEEK(53272) OR 8
180  POKE 53265,PEEK(53265) OR 32
190  SYS 49152
200  FOR I = 40705 TO 40730
210    READ D
220    POKE I,D
230  NEXT I
240  DATA 169,31,162,0,134,150,162,4,134,151,160,0,145,150
250  DATA 200,208,251,230,151,166,151,224,8,208,243,96
260  SYS 40705
270  GOSUB 550 : REM DIBUJA CURSOR
280  GOSUB 720 : REM LEE OPCIONES DE LA PALANCA DE JUEGOS
290  A$ = ""
300  GET A$
310  IF A$ = "M" THEN PI = 0
320  IF A$ = "D" THEN PI = 1
330  IF A$ = "+" OR A$ = "-" THEN GOSUB 1470
340  IF A$ = CHR$(133) THEN GOSUB 1580
350  IF A$ = CHR$(134) THEN GOSUB 1810
360  IF PI = 1 THEN 380
370  GOTO 420
380  IF JO <> 0 THEN GOSUB 790
390  IF JO=0 AND A$="" THEN 280 : REM NO HAY LECTURA DE
     DATOS
```

```
400  GOSUB 1360 : REM DIBUJA PUNTO
410  GOTO 280 : REM REGRESA A LEER DATOS
420  IF JO <> 0 THEN GOSUB 790
430  GOTO 280
440  REM ************************************
450  REM *     VERIFICA SI X > 255            *
460  REM ************************************
470  IF X+13 >=255 THEN POKE 53264,1:X1=X+13-255:GOTO 500
480  X1 = X + 13
490  POKE 53264,0
500  IF X < 0 THEN X1 = X1 + INC : X = X + INC
510  IF X > 319 THEN X1 = X1 - INC : X = X - INC
520  IF Y < 0 THEN Y = Y + INC
530  IF Y > 199 THEN Y = Y - INC
540  RETURN
550  REM ************************************
560  REM *           C U R S O R              *
570  REM ************************************
580  FOR I = 0 TO 62
590    READ A
600    POKE 832+I,A
610  NEXT I
620  DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
630  DATA 0,16,0,0,16,0,0,0,0,0,198,0,0,0,0,0,16,0,0,16,0
640  DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
650  POKE 2040,13     : REM LOC. DATOS SPRITE 0
660  POKE 53269,1     : REM PRENDER SPRITE 0
670  POKE 53287,0     : REM COLOR SPRITE 0
680  POKE 53248,X+13 : REM POSICION X
690  POKE 53249,Y+40 : REM POSICION Y
700  POKE 53264,0     : REM POSICION X > 255
710  RETURN
720  REM ****************************************
730  REM *      LEE OPCIONES DE LA PALANCA       *
740  REM ****************************************
750  JO = PEEK(53261)
760  BO = JO AND 16
770  JO = 15 - ( JO AND 15 )
780  RETURN
790  REM ****************************************
800  REM *       MOVIMIENTOS DEL CURSOR          *
810  REM ****************************************
820  IF BO = 16 THEN DI = 1 : GOTO 840
830  DI = 0
840  ON JO GOTO 850,900,950,960,1010,1080,1150,1160,1210,
     1280
850  REM ********************  ARRIBA
860  Y = Y - INC
870  GOSUB 440
880  POKE 53249,Y+40
890  GOTO 1350
900  REM ********************  ABAJO
910  Y = Y + INC
920  GOSUB 440
930  POKE 53249,Y+40
```

```
940  GOTO 1350
950 REM ******************  NO HAY MOVIMIENTO : GOTO 6150
960  REM ***********************   IZQUIERDA
970  X = X - INC
980  GOSUB 440
990  POKE 53248,X1
1000 GOTO 1350
1010 REM ********************** ARRIBA IZQUIERDA
1020 X = X - INC
1030 Y = Y - INC
1040 GOSUB 440
1050 POKE 53249,Y+40
1060 POKE 53248,X1
1070 GOTO 1350
1080 REM ********************** ABAJO IZQUIERDA
1090 X = X - INC
1100 Y = Y + INC
1110 GOSUB 440
1120 POKE 53249,Y+40
1130 POKE 53248,X1
1140 GOTO 1350
1150 REM ******************  NO HAY MOVIMIENTO : GOTO 6150
1160 REM ******************  DERECHA
1170 X = X + INC
1180 GOSUB 440
1190 POKE 53248,X1
1200 GOTO 1350
1210 REM ********************  ARRIBA DERECHA
1220 X = X + INC
1230 Y = Y - INC
1240 GOSUB 440
1250 POKE 53249,Y+40
1260 POKE 53248,X1
1270 GOTO 1350
1280 REM ********************  ABAJO DERECHA
1290 X = X + INC
1300 Y = Y + INC
1310 GOSUB 440
1320 POKE 53249,Y+40
1330 POKE 53248,X1
1340 GOTO 1350
1350 RETURN
1360 REM ********************************
1370 REM *  DIBUJA O DESPLAZA UN PUNTO   *
1380 REM ********************************
1390 RE    = INT(Y/8)
1400 CARAC = INT(X/8)
1410 LINEA = Y AND 7
1420 BIT   = 7 - ( X AND 7 )
1430 BYTE  = BASE + RE * 320 + CARAC * 8 + LINEA
1440 IF DI=1 THEN POKE BYTE,PEEK(BYTE) OR (2^BIT) : GOTO
     1460
1450 POKE BYTE,PEEK(BYTE) AND (255-2^BIT)
1460 RETURN
```

```
1470 REM ************************************************
1480 REM * INCRE. O DECRE. EL DESPLAZAMIENTO DEL CURSOR *
1490 REM ************************************************
1500 IF A$ = "+" THEN 1520
1510 GOTO 1540
1520 IF INC <= 19 THEN INC = INC + 1
1530 GOTO 1570
1540 IF A$ = "-" THEN GOTO 1560
1550 GOTO 1570
1560 IF INC >= 2 THEN INC = INC - 1
1570 RETURN
1580 REM ****************************
1590 REM * CARGA IMAGEN EN PANTALLA *
1600 REM ****************************
1610 POKE 53265,PEEK(53265) AND 223
1620 POKE 53272,PEEK(53272) AND 247
1630 POKE 53269,0
1640 PRINT "<SHFT CLR/HOME><3 CR/AB><2 CR/DER>"
1650 INPUT "NOMBRE : ";NOM$
1660 IF NOM$ = "" THEN 1750
1670 OPEN 2,8,2,NOM$+",SEQ,R"
1680 FOR I = 0 TO 999
1690   INPUT #2,DA
1700   POKE 8192+I,DA
1730 NEXT I
1740 CLOSE 2
1750 SYS 40705
1760 POKE 2040,13
1770 POKE 53269,1
1780 POKE 53272,PEEK(53272) OR 8
1790 POKE 53265,PEEK(53265) OR 32
1800 RETURN
1810 REM ********************************
1820 REM *  GRABAR IMAGEN EN DISCO      *
1830 REM ********************************
1840 POKE 53265,PEEK(53265) AND 223
1850 POKE 53272,PEEK(53272) AND 247
1860 POKE 53269,0
1870 PRINT "<SHFT CLR/HOME><3 CR/AB>GRABAR<2 CR/DE>"
1880 INPUT "NOMBRE : ";NOM$
1890 IF NOM = "" THEN 1970
1900 OPEN 2,8,2,"@0:"+NOM$+",SEQ,W"
1910 FOR I = 0 TO 999
1920   DA = PEEK(8192+I)
1930   PRINT #2,DA
1950 NEXT I
1960 CLOSE 2
1970 SYS 40705
1980 POKE 2040,13
1990 POKE 53269,1
2000 POKE 53272,PEEK(53272) OR 8
2010 POKE 53265,PEEK(53265) OR 32
2020 RETURN
```

Programa 6.4

CAPITULO VII

TRANSFORMACIONES DE ESCALAS TRASLACIONES Y ROTACIONES

Una de las principales ventajas de la graficación por computadora es la facilidad con que pueden hacerse alteraciones a las imágenes ya creadas. Puede verse una imagen desde diferentes ángulos sin tener que crear una imagen nueva. Por ejemplo, un arquitecto puede ver un edificio desde diferentes puntos de observación sin tener que dibujar cada una de las vistas. Puede también cambiarse la escala a una imagen o gráfica, o en la animación puede cambiarse continuamente la posición de cualquier figura.

Estos cambios no son tan difíciles de realizar, debido a que las imágenes por computadora son generadas por una serie de segmentos de recta, los cuales son representados por sus puntos extremos. De esto, un cambio en la imagen lo podemos hacer alterando estos puntos, que son almacenados como números en la computadora. Para alterarlos pueden aplicárseles operaciones matemáticas llamadas transformaciones.

Dichas transformaciones nos permiten alterar la figura completamente o sólo una parte de ella. Esto nos proporciona una herramienta muy útil para cuando se tienen que hacer gráficas o imágenes a mano, ya que mediante la computadora no es necesario volver a dibujar la imagen, únicamente se tiene que definir la transformación que se quiere hacer y aplicarla a la imagen que se tenía anteriormente para obtener una nueva.

A continuación explicaremos las transformaciones geométricas de proyección, traslación y rotación.

VII.1.- PROYECCION.

Un punto se puede representar por medio de una matriz de 1x2 que al multiplicarla por una matriz de 2x2 sufrirá una transformación. Entre las distintas transformaciones tenemos la idéntica en la que los puntos no sufren ninguna alteración:

$$PT = (x,y) \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = (x,y)$$

Otra posible transformación es la proyección (cambio de escala) que se logra si T1 es de la siguiente forma:

$$T1 = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}$$

entonces tendríamos que

$$P\ T1 = (x,y) \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix} = (2x,y)$$

Por lo que ahora en cada punto el valor de la primera coordenada será dos veces mayor que el valor original. La nueva figura tendrá el mismo largo, pero el ancho será el doble que el de la figura inicial.

Si tenemos ahora la matriz de transformación T2

$$T2 = \begin{bmatrix} 1 & 0 \\ 0 & 0.5 \end{bmatrix} \qquad P\ T2 = (X,Y) \begin{bmatrix} 1 & 0 \\ 0 & 0.5 \end{bmatrix} = (X,0.5)$$

la altura de la imagen se reducirá a la mitad, conservándose el ancho de la misma.

De lo anterior tenemos que la matriz de transformación para modificar la escala horizontal (ancho) de una figura es la

siguiente:

$$T3 = \begin{bmatrix} Sx & 0 \\ 0 & 1 \end{bmatrix}$$

en donde Sx es el factor de escala que modifica el ancho de la imagen.

Análogamente si queremos modificar la escala vertical de una imagen (altura) tendremos la matriz de transformación T4:

$$T4 = \begin{bmatrix} 1 & 0 \\ 0 & Sy \end{bmatrix}$$

donde Sy es el factor que modifica la altura de la imagen.

Si se aplican las transformaciones T3 y T4, se modificará tanto la altura como el ancho de la imagen. Multiplicando estas dos matrices se obtiene una sola matriz de transformación que permite cambiar la escala horizontal (x) y la vertical (y) mediante una sola operación.

$$T5 = T3\ T4 = \begin{bmatrix} Sx & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & Sy \end{bmatrix} = \begin{bmatrix} Sx & 0 \\ 0 & Sy \end{bmatrix}$$

donde Sx es el factor de escala para las "X" y Sy el de las "Y".

Ejemplo:

Si tenemos un cuadrado cuyos vértices son los puntos (1,1), (1,2), (2,2), (2,1) y se quiere cambiar ambas escalas al doble de su valor original se tendría que aplicar la siguiente matriz de transformación:

$$T = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$$

Aplicándola a cada uno de los puntos, se obtendrían las nuevas coordenadas que son respectivamente: (2,2), (2,4),

(4,4), (4,2) . La siguiente figura ilustra este ejemplo.

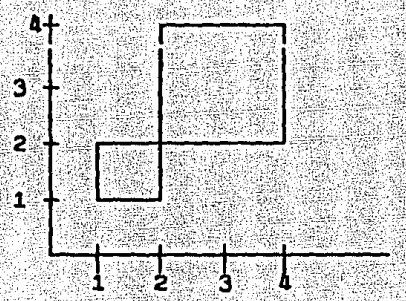


Figura 7.1

VII.2.- ROTACION.

Al igual que en las proyecciones, para rotar una figura lo que se hace es modificar la ubicación de la frontera de la misma. Las rotaciones se harán con respecto al origen.

Si tenemos un punto A con coordenadas (x,y), sabemos que la distancia de este punto al origen será:

$$D = X^2 + Y^2$$

además

$$sen(\sigma) = y / D$$

$$cos(\sigma) = x / D$$

Donde σ es el ángulo formado por la línea que une al punto A con el origen y la línea formada por el eje de las abscisas. Si rotamos el punto A θ grados en sentido contrario a las manecillas del reloj, obtendremos un nuevo punto con coordenadas (x1,y1)

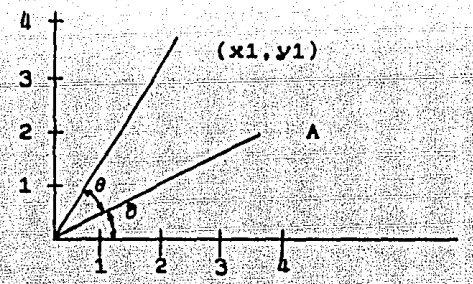


Figura 7.2

en donde

$$y1 = D * sen(\sigma+\theta) \quad y \quad x1 = D * cos(\sigma+\theta)$$

Por otro lado sabemos que

$$sen(\sigma+\theta) = sen(\sigma)cos(\theta) + cos(\theta)sen(\theta)$$

$$cos(\sigma+\theta) = cos(\sigma)cos(\theta) - sen(\sigma)sen(\theta)$$

por lo que

$$y1 = D * sen(\sigma)cos(\theta) + D * cos(\sigma)sen(\theta)$$

$$y1 = y * cos(\theta) + x * sen(\theta)$$

Análogamente

$$x1 = D * cos(\sigma)cos(\theta) - D * sen(\sigma)sen(\theta)$$

$$x1 = x * cos(\theta) + y * sen(\theta)$$

Por lo tanto

$$(x1,y1) = (x,y) \begin{bmatrix} cos(\theta) & sen(\theta) \\ -sen(\theta) & cos(\theta) \end{bmatrix} =$$

$$= (xcos(\theta)-ysen(\theta) \ , \ xsen(\theta)+ycos(\theta))$$

Partiendo de lo anterior, la matriz de transformación que se debe utilizar para rotar un punto θ grados en sentido contrario a las manecillas del reloj es la siguiente:

$$R = \begin{bmatrix} cos(\theta) & sen(\theta) \\ -sen(\theta) & cos(\theta) \end{bmatrix}$$

Ejemplo:

Rotar el punto (6,1) con un ángulo de $\pi/4$ radianes en sentido contrario de las manecillas del reloj.

$$(6,1)\begin{bmatrix} \cos(\pi/4) & \text{sen}(\pi/4) \\ -\text{sen}(\pi/4) & \cos(\pi/4) \end{bmatrix} =$$

$$= (6\cos(\pi/4)-\text{sen}(\pi/4)\ ,\ 6\text{sen}(\pi/4)+\cos(\pi/4))$$

$$= (3.5355,\ 4.949)$$

Si por el contrario deseamos rotar un punto en el sentido de las manecillas del reloj tendríamos que hacerlo utilizando la siguiente matriz de rotación:

$$R = \begin{bmatrix} \cos(-\theta) & \text{sen}(-\theta) \\ -\text{sen}(-\theta) & \cos(-\theta) \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\text{sen}(\theta) \\ \text{sen}(\theta) & \cos(\theta) \end{bmatrix}$$

VII.3.- TRASLACION.

Mover la imagen de un lugar a otro es a lo que llamamos traslación. Para efectuar las traslaciones, sólo tendremos que sumar o restar el desplazamiento que deseamos obtener a cada punto. Esto significa que si trasladamos una imagen a una nueva posición (x1,y1) tendríamos que afectar cada coordenada (x,y) de la siguiente manera:

$$x1 = x + Tx$$

$$y1 = y + Ty$$

Como se verá mas adelante nos resultaría más útil poder ejecutar estas traslaciones por medio de una matriz.

Para obtener dicha matriz tenemos que agregar una coordenada ficticia (Método de Coordenadas Homogéneas) y de esta manera estaremos multiplicando una matriz de 1x3 por una de 3x3.

obteniendo una de 1x3. Si nuestra coordenada ficticia es w tendríamos que multiplicar X e Y por w, por lo cual cada punto (x,y) se transforma en (Xw,Yw,w). Para volver a obtener el valor original de X e Y tendremos que dividir dichas coordenadas entre w.

Por lo tanto por el método de la matriz de Coordenadas Homogéneas tendríamos que la matriz de traslación para Tx y Ty sería la siguiente:

$$T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ Tx & Ty & 1 \end{bmatrix}$$

aplicándola al punto (x,y) tenemos

$$(Xw, Yw, W) \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ Tx & Ty & 1 \end{bmatrix} = (Xw+Txw)\ (Yw+Tyw)\ w$$

dividiendo entre w obtenemos las coordenadas del nuevo punto

$$(x1, y1) = (x+Tx\ ,\ y+Ty)$$

VII.4.- ROTACION SOBRE UN PUNTO DISTINTO AL ORIGEN.

Si deseamos rotar una imagen alrededor de un punto que no sea el origen, tendremos que hacerlo ejecutando los siguientes pasos:

1.- Trasladar la imagen hasta que el punto sobre el que queremos rotar coincida con el origen.

2.- Rotar la imagen.

3.- Trasladar la imagen a su posición original.

Como es necesario ejecutar tres transformaciones a la imagen, lo más conveniente es encontrar una sola matriz

equivalente a estos tres pasos, lo que nos dará una mayor eficiencia. Una manera de hacer esto es usando el Método de Coordenedas Homogéneas, en el cual, como ya se explicó, cada punto está determinado por tres coordenadas en lugar de dos, agregándose una coordenada ficticia w.

Para efectuar el paso 1 tenemos la siguiente matriz

$$T1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -Tx & -Ty & 1 \end{bmatrix}$$

Para el segundo paso obtenemos la siguiente matriz de rotación

$$R = \begin{bmatrix} \cos\theta & \text{sen}\,\theta & 0 \\ -\text{sen}\,\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Análogamente que para el paso 1 tenemos la matriz

$$T2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ Tx & Ty & 1 \end{bmatrix}$$

Para obtener la matriz única que efectué los tres pasos antes mencionados tenemos que multiplicar las tres matrices antes mencionadas:

$$T1\ R\ T2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -Tx & -Ty & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & \text{sen}\,\theta & 0 \\ -\text{sen}\,\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ Tx & TY & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -Tx & -Ty & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & \text{sen}\,\theta & 0 \\ -\text{sen}\,\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$T1\ R\ T2 = \begin{bmatrix} \cos\theta & \text{sen}\,\theta & 0 \\ -\text{sen}\,\theta & \cos\theta & 0 \\ -Tx\cos\theta + Ty\,\text{sen}\,\theta + Tx & -Tx\,\text{sen}\,\theta + Ty\,\text{sen}\,\theta + Ty & 1 \end{bmatrix}$$

El siguiente programa rota un cuadrado en sentido contrario a las manecillas del reloj, que se encuentra fuera del origen. Para ello es necesario trasladarlo al origen, hacer la rotación y posteriormente trasladarlo a su lugar original.

```
1   REM ********************************************
2   REM *  ROTACION DE UN CUADRADO SOBRE UN PUNTO   *
3   REM *           DISTINTO AL ORIGEN              *
4   REM ********************************************
10  BASE=8192
20  PANT=1024
30  PRINT "<SHIFT CLR/HOME>"
35  PRINT "<5 CR/AB>  ANGULO DE ROTACION (RADIANES):";
40  INPUT ANG
50  FOR D=1 TO 4
60    READ X(D),Y(D)
65    Y(D)=Y(D)*.8
70  NEXT D
80  DATA 10,10,50,10,50,50,10,50
90  T(1,1)=COS(ANG)
100 T(1,2)=SIN(ANG)
110 T(2,1)=-SIN(ANG)
120 T(2,2)=COS(ANG)
130 POKE 251,0
140 POKE 252,32  : REM TOPE DE BASIC=8192
150 POKE 53272,PEEK(53272) OR 8 :REM PANT.DE A.R. EN 8192
160 POKE 53265,PEEK(53265) OR 32 : REM PRENDE ALTA RESOL.
170 FOR I=0 TO 20
180   READ T
190   POKE 49152+I,T : REM LIMPIA LA MEMORIA DE LA PANTALLA
200 NEXT I
210 SYS 49152
220 DATA 169,00,170,168,145,251,200,240,02,208,249,230
230 DATA 252,232,224,32,240,02,208,240,96
240 FOR I=40705 TO 40730
245   READ D
250   POKE I,D
255 NEXT I
260 SYS 40705
265 DATA 169,1,162,0,134,150,162,4,134,151,160,0,145,150
270 DATA 200,208,251,230,151,166,151,224,8,208,243,96
275 POKE 53280,6
280 POKE 53281,1
290 FOR D=1 TO 4
300   YA=Y(D)-Y(4)
310   XA=X(D)-X(4)
```

```
320   X1(D)=XA*T(1,1)+YA*T(1,2)+X(4)
330   Y1(D)=XA*T(2,1)+YA*T(2,2)+Y(4)
340   XD=XD+160
350   X1(D)=X1(D)+160
360   Y(D)=Y(D)+100
370   Y1(D)=Y1(D)+100
380 NEXT D
390 FOR D=1 TO 4
400   D1=D+1
405   IF D=4 THEN D1=1
410   GOSUB 700 : REM GRAFICA EL CUADRADO
420 NEXT D
430 GET T$
440 IF T$="" THEN 430
450 PRINT "<CLR/HOME>"
460 POKE 53265,PEEK(53265) AND (255-32)
470 POKE 53272,PEEK(53272) AND (255-8)
480 END
485 REM *********************************
490 REM *   IMPRESION DE LOS PUNTOS     *
495 REM *********************************
500 REN=INT(Y/8)
510 CARAC=INT(X/8)
520 LINEA=Y AND 7
530 BIT=7-(X AND 7)
540 BYTE=BASE+REN*320+CARAC*8+LINEA
550 POKE BYTE,PEEK(BYTE) OR (2^BIT)
560 RETURN
580 REM *******************************
590 REM *            RECTA            *
600 REM *******************************
700 IF X1(D)>=X1(D1) THEN XI=X1(D1) : XF=X1(D) :GOTO 720
710 XI=X1(D) : XF=X1(D1)
720 IF Y1(D)>=Y1(D1) THEN YI=Y1(D1) : YF=Y1(D) : GOTO 740
730 YI=Y1(D) : YF=Y1(D1)
740 IF X1(D)=X1(D1) THEN 830
750 IF Y1(D)=Y1(D1) THEN 880
760 M=(Y1(D1)-Y1(D))/(X1(D1)-X1(D))
770 IF (M>0 AND M<1) OR (M<0 AND M>-1) THEN 930
780 FOR Y=YI TO YF
790   X=INT((Y-Y1(D))/M+X1(D))
800   GOSUB 500 : REM IMPRIME EL PUNTO EN LA PANTALLA
810 NEXT Y
820 RETURN
830 FOR Y=YI TO YF
840   X=X1(D)
850   GOSUB 500
860 NEXT Y
870 RETURN
880 FOR X=XI TO XF
890   Y=Y1(D)
900   GOSUB 500
910 NEXT X
920 RETURN
```

```
930 FOR X=XI TO XF
940   Y=M*(X-X1(D))+Y1(D)
950   GOSUB 500
960 NEXT X
970 RETURN
```

Programa 7.1

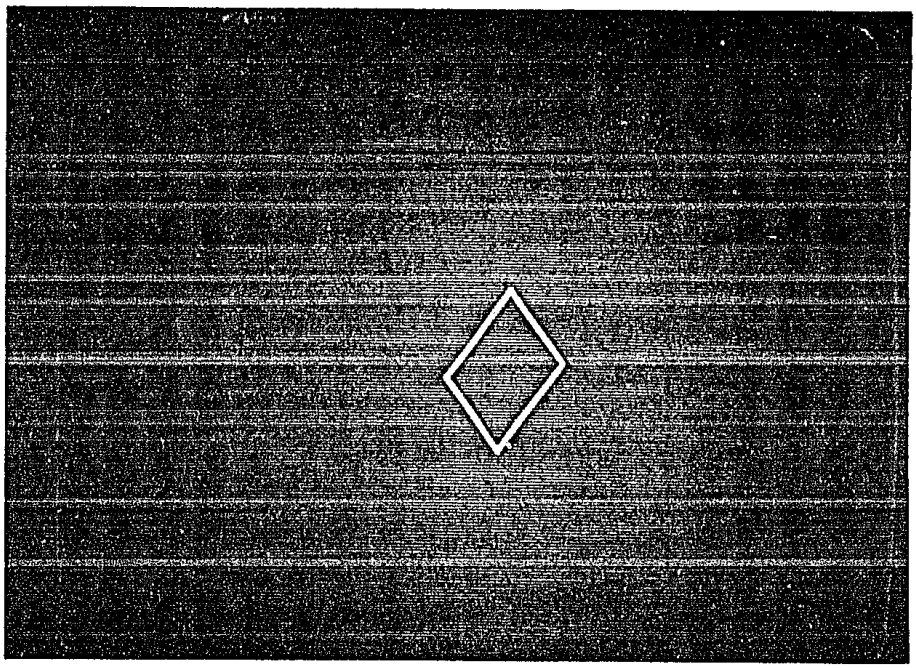


Figura 7.3. Imagen generada por el programa 7.1 cuando el ángulo de rotación es $\pi/4$ radianes.

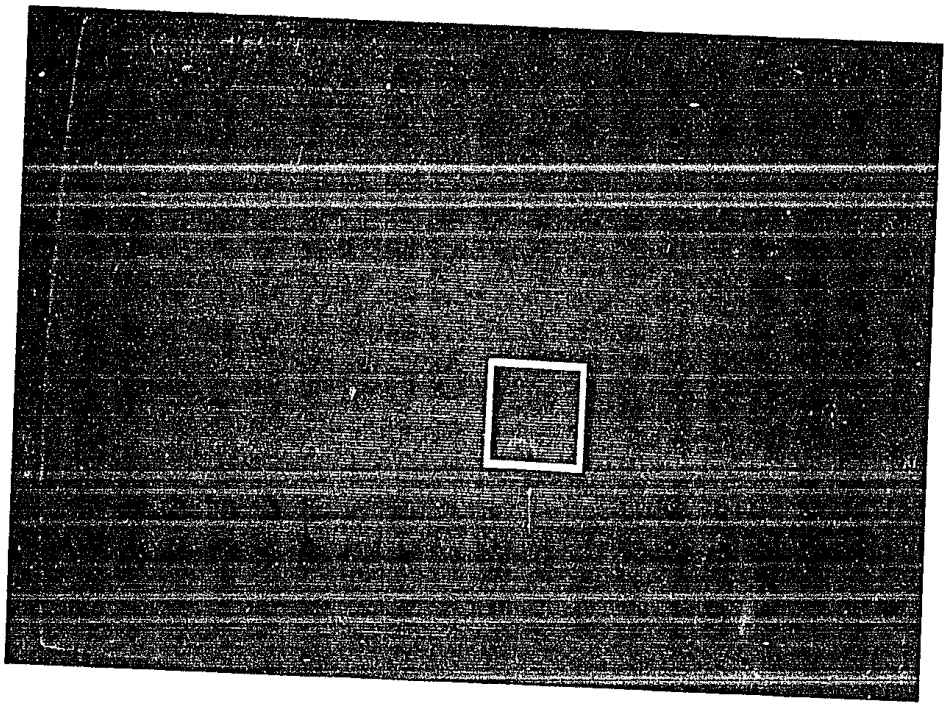


Figura 7.4. Imagen generada por el programa 7.1 cuando el ángulo de rotación es cero.

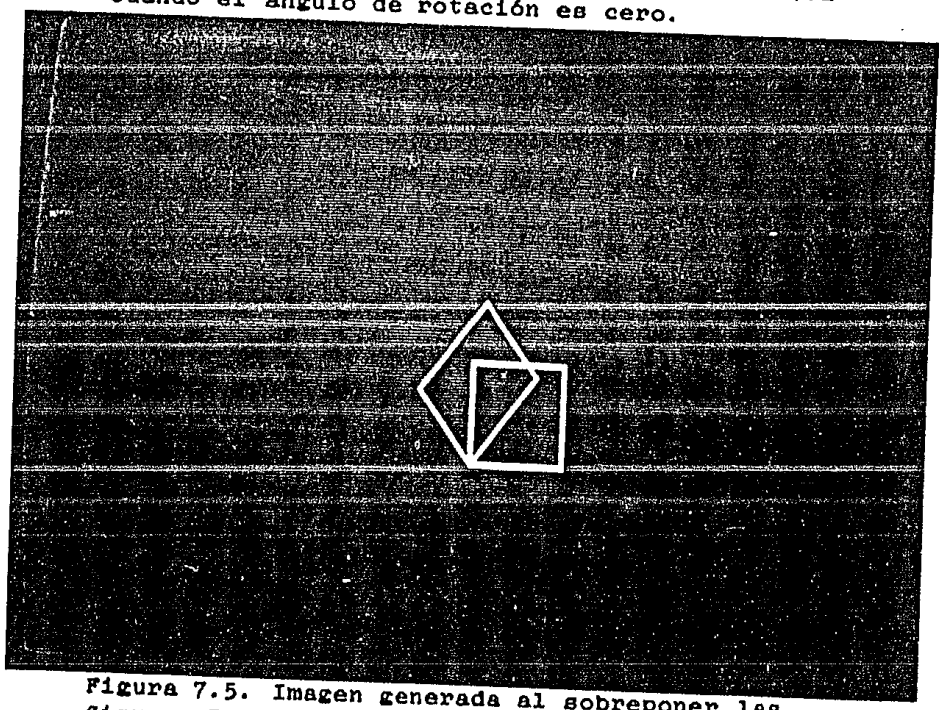


Figura 7.5. Imagen generada al sobreponer las figuras 7.3 y 7.4.

CAPITULO VIII

APLICACIONES DE GRAFICACION FINA Y GRAFICACION CON MOVIMIENTO

VIII.1.- GRAFICAS CON EL SIMON BASIC.

Para los siguientes programas usaremos un cartucho llamado SIMON BASIC, el cual tiene 114 comandos adicionales al BASIC estándar de la Commodore 64. Dichos comandos facilitan el uso de la computadora, ya que no es necesario accesar localidades de memoria, además de ser más rápidos en su ejecución debido a que están programados en firmware.
Dentro de los 114 comandos existen 21 comandos para el uso de gráficas, de los cuales sólo usaremos algunos de ellos. A continuación mostraremos dichos comandos, así como las instrucciones que equivalen a ellos en el BASIC estándar, que ya fueron explicadas en el capítulo 6.

A) COLOUR cp,cb

Función: Define los colores del fondo de la pantalla y del borde de ésta, en el modo de baja resolución. El parámetro cp determina el color del fondo de la pantalla y el parámetro cb el color del borde.

Equivalencia:

```
POKE 53281,CP
POKE 53280,CB
```

B) HIRES cd,cf

Función: Inicializa el modo gráfico de alta resolución. El parámetro cd, determina el color de dibujo (0-15). El

parámetro cf, determina el color del fondo de la pantalla, es decir, de cada cuadrado de 8x8 puntos donde se dibuja.

Equivalencia:

```
BA=8192
VD=CD*16+CF
POKE 53265,PEEK(53265) OR 32
POKE 53272,PEEK(53272) OR 8
FOR I=BA TO BA+7999
POKE I,0
NEXT I
FOR I=1024 TO 2023
POKE I,VD
NEXT I
```

C) PLOT x,y,pg

Función: Dibuja un punto en la pantalla. Los parámetros x e y especifican las coordenadas horizontal y vertical del punto en la pantalla. Como se vió en el capítulo 6 las coordenadas de la esquina superior izquierda de la pantalla son (0,0) y las de la esquina inferior derecha son (320,200) .El parámetro pg es un parámetro gráfico que puede tener 3 posibles valores:

0 - Borra un punto
1 - Dibuja un punto en la pantalla
2 - Invierte el punto

Equivalencia:

```
RENG=INT(Y/8)
CARAC=INT(X/8)
LINEA=Y AND 7
BIT=7 - (X AND 7)
BYTE=BA+RENG*320+CARAC*8+LINEA
```

Si PG es igual a:

0 - POKE BYTE,PEEK(BYTE) AND (255-2^BIT)

1 - POKE BYTE,PEEK(BYTE) OR (2^BIT)

2 - VB=PEEK(BYTE) AND (2^BIT)
IF VB=0 THEN POKE BYTE,PEEK(BYTE) OR (2^BIT)
IF VB=1 THEN POKE BYTE,PEEK(BYTE) AND(255-2^BIT)

D) NRM

Función: Regresar al modo de baja resolución.

Equivalencia:

```
POKE 53265,PEEK(53265) AND 223
POKE 53272,PEEK(53272) AND 247
```

E) LINE x1,y1,x2,y2,pg

Función: Dibujar una recta. Los parámetros x1,y1 son las coordenadas del punto de inicio de la recta. Los parámetros x2,y2 son las coordenadas del extremo final de la recta. Pg es el parámetro gráfico explicado en el inciso C.

Equivalencia:

Programa 6.3 (Capítulo 6).

F) REC x1,y1,x2,y2,pg

Función: Dibujar un rectángulo. Los parámetros x1,y1, son las coordenadas de la esquina superior izquierda del rectángulo. Los parámetros x2,y2 son las coordenadas de la esquina inferior derecha. Pg es el parámetro gráfico explicado en el inciso C.

Equivalencia:

Programa 6.2 (Capítulo 6).

G) CIRCLE x,y,xr,yr,pg

Función: Dibujar un círculo. Los parámetros x e y son las coordenadas del centro del círculo. Los parámetros xr e yr determinan los radios horizontal y vertical respectivamente. Variando estos radios es posible dibujar circunferencias y elipses de diferentes tamaños. Pg es el parámetro gráfico explicado en el inciso C.

Equivalencia:

Programa 6.1 (Capítulo 6).

VIII.1.2.- GRAFICAS DE LAS FUNCIONES TRIGONOMETRICAS.

El siguente programa grafica las funciones SENO(x) y COSENO(x). El intervalo donde se grafica es (-2π,2π).

```
10  REM ************************************************
20  REM * GRAFICAS DE LAS FUNCIONES:SENO(X) Y COSENO(X) *
30  REM ************************************************
40  PRINT "<SHFT CLR/HOME><6 CR/AB><7 CR/DE>SELECCIONA LA
    OPCION :"
50  PRINT "<2 CR/AB><10 CR/DE>1.- SENO"
60  PRINT "<10 CR/DE>2.- COSENO"
70  PRINT "<10 CR/DE>3.- FIN"
80  PRINT "<HOME><6 CR/AB><28 CR/DE>";
90  INPUT OP
100 IF OP>3 OR OP<1 THEN 80
110 IF OP=3 THEN 230
120 ES = 25
130 PRINT "<SHFT CLR/HOME>"
140 PRINT "<HOME><7 CR/AB><4 SPC>LIMITE INFERIOR (-2π,2π):
    <10 SPC><10 CR/IZ>";
150 INPUT LI
160 IF LI<-2*π OR LI>2*π THEN 140
170 PRINT "<HOME><9 CR/IZ><4 SPC>LIMITE SUPERIOR (-2π,2π):
    <10 SPC><10 CR/IZ>";
180 INPUT LS
190 IF LS<-2*π OR LS>2*π THEN 170
200 REM
210 HIRES 1,0
220 GOSUB 440
230 ON OP GOSUB 310,390,570
240 GET A$: IF A$="" THEN 240
250 NRM
260 OP=0 : LI=7 : LS=7
270 GOTO 40
280 REM ************************
290 REM *        SENO(X)         *
300 REM ************************
310 FOR X=LI TO LS STEP π/64
320   Y=ES*SIN(X)
330   PLOT X*ES+160,200-(100+Y),1
340 NEXT
350 RETURN
360 REM ************************
370 REM *       COSENO(X)        *
380 REM ************************
390 FOR X=LI TO LS STEP π/64
400   Y=ES*COS(X)
410   PLOT X*ES+160,200-(100+Y),1
420 NEXT X
```

```
430 RETURN
440 REM ****************
450 REM *      EJES      *
460 REM ****************
470 XS=INT(200/ES)
480 FOR K=1 TO 7
490   LINE 158,25*K,162,25*K,1
500 NEXT K
510 FOR K=0 TO 12
520   LINE 25*K+10,98,25*K+10,102,1
530 NEXT K
540 LINE 160,0,160,200,1
550 LINE 0,100,320,100
560 RETURN
570 PRINT "<SHFT CLR/HOME>"
580 END
```

Programa 8.1

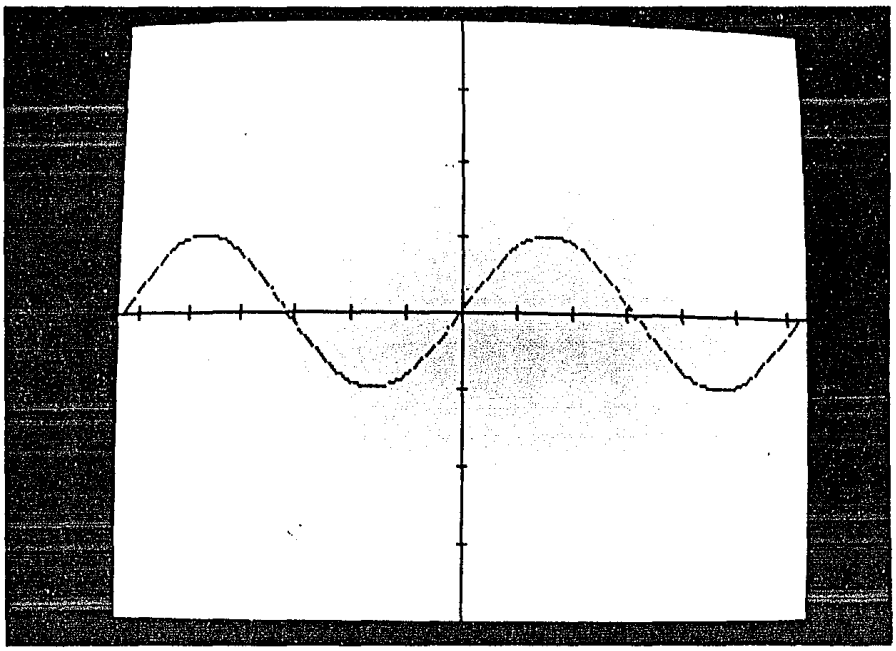


Figura 8.1. Gráfica de la función Y=sen(X).

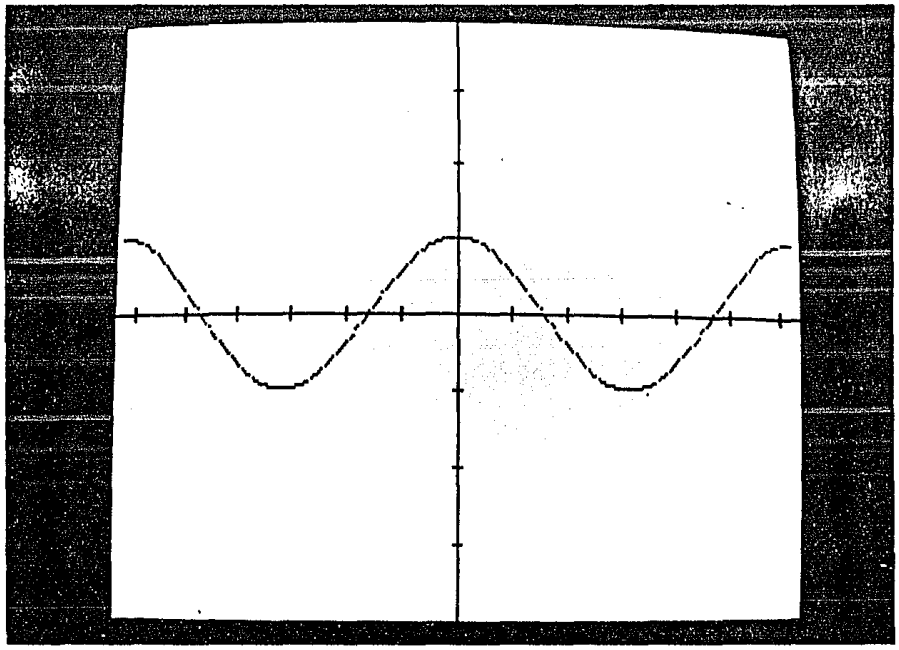


Figura 8.2. Gráfica de la función Y=cos(x).

El siguiente programa grafica las funciones SECANTE(x) y COSECANTE(x).

```
10  REM ******************************************************
20  REM *  GRAFICAS DE LAS FUNCIONES: SEC(X)  Y CSC(X)     *
30  REM ******************************************************
40  PRINT "<SHFT CLR/HOME><6 CR/AB><7 CR/DE>SELECCIONA LA
    OPCION :"
50  PRINT "<2 CR/AB><10 CR/DE>1.- SECANTE"
60  PRINT "<10 CR/DE>2.- COSECANTE"
70  PRINT "<10 CR/DE>3.- FIN"
80  PRINT "<HOME><6 CR/AB><28 CR/DE>";
90  INPUT OP
100 IF OP>3 OR OP<1 THEN 80
110 IF OP=3 THEN 230
120 ES = 3
130 PRINT "<SHFT CLR/HOME>"
140 PRINT "<HOME><7 CR/AB><4 SPC>LIMITE INFERIOR (-2π,2π):
    <10 SPC><10 CR/IZ>";
150 INPUT LI
160 IF LI<-2*π OR LI>2*π THEN 140
170 PRINT "<HOME><9 CR/IZ><4 SPC>LIMITE SUPERIOR (-2π,2π):
    <10 SPC><10 CR/IZ>";
180 INPUT LS
190 IF LS<-2*π OR LS>2*π THEN 170
210 HIRES 1,0
220 GOSUB 440
230 ON OP GOSUB 310,390,530
240 GET A$: IF A$="" THEN 240
250 NRM
260 OP=0 : LI=7 : LS=7
270 GOTO 40
280 REM ***********************
290 REM *    COSECANTE(X)     *
300 REM ***********************
310 FOR X=LI TO LS STEP π/64
315   ER = .01
320   Y=SIN(X)
325   IF ABS(Y)<=ER THEN 340
330   Y=ES*1/Y
335   PLOT X*10+160,200-(100+Y),1
340 NEXT X
350 RETURN
360 REM ************************
370 REM *      SECANTE(X)      *
380 REM ************************
390 FOR X=LI TO LS STEP π/64
395   ER=.01
400   Y=COS(X)
405   IF ABS(Y)<=ER THEN 410
410   Y=ES*1/Y
415   PLOT X*10+160,200-(100+Y),1
420 NEXT X
```

```
430 RETURN
440 REM ***********************
450 REM *        E J E S        *
460 REM ***********************
470 FOR K=0 TO 32
480   LINE 10*K,98,10*K,102,1
490 NEXT K
500 LINE 160,0,160,200,1
510 LINE 0,100,320,100,1
520 RETURN
530 PRINT "<SHFT CLR/HOME>"
540 END
```

Programa 8.2

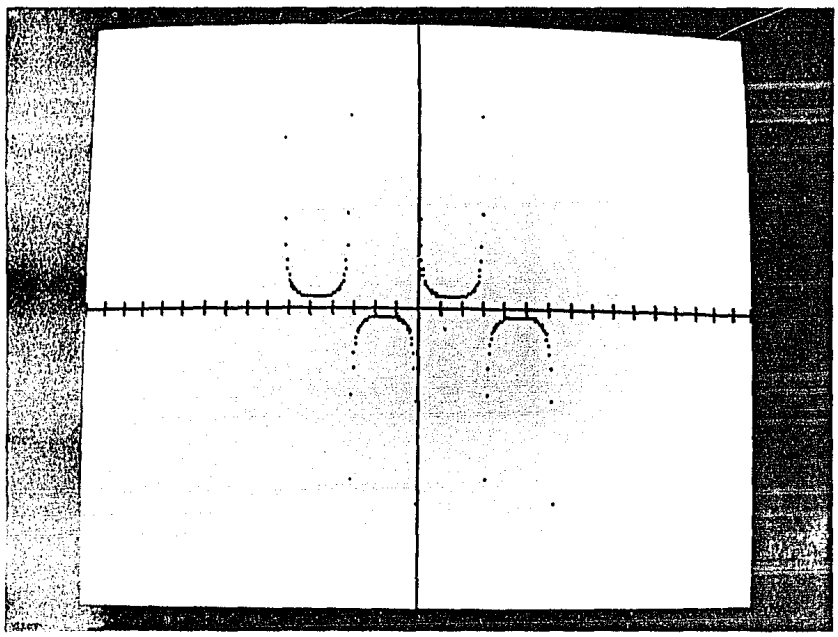


Figura 8.3. Gráfica de la función Y=sec(x).

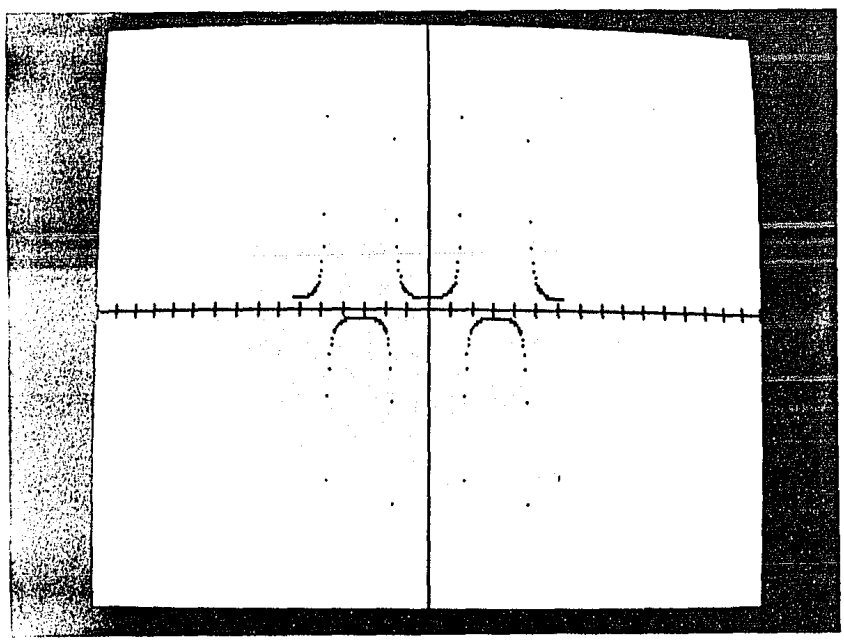


Figura 8.4. Gráfica de la función Y=csc(x).

El siguiente programa grafica las funciones TANGENTE(X) y COTANGENTE(X).

```
10  REM ************************************************
20  REM *  GRAFICAS DE LAS FUNCIONES: TAN(X)  Y COT(X)  *
30  REM ************************************************
40  PRINT "<SHFT CLR/HOME><6 CR/AB><7 CR/DE>SELECCIONA LA
    OPCION :"
50  PRINT "<2 CR/AB><10 CR/DE>1.- TANGENTE"
60  PRINT "<10 CR/DE>2.- COTANGENTE"
70  PRINT "<10 CR/DE>3.- FIN"
80  PRINT "<HOME><6 CR/AB><28 CR/DE>";
90  INPUT OP
100 IF OP>3 OR OP<1 THEN 80
110 IF OP=3 THEN 230
120 ES = 3
130 PRINT "<SHFT CLR/HOME>"
140 PRINT "<HOME><7 CR/AB><4 SPC>LIMITE INFERIOR (-2π,2π):
    <10 SPC><10 CR/IZ>";
150 INPUT LI
160 IF LI<-2*π OR LI>2*π THEN 140
170 PRINT "<HOME><9 CR/IZ><4 SPC>LIMITE SUPERIOR (-2π,2π):
    <10 SPC><10 CR/IZ>";
180 INPUT LS
190 IF LS<-2*π OR LS>2*π THEN 170
210 HIRES 1,0
220 GOSUB 440
230 ON OP GOSUB 310,390,530
240 GET A$: IF A$="" THEN 240
250 NRM
260 OP=0 : LI=7 : LS=7
270 GOTO 40
280 REM ***********************
290 REM *     TANGENTE(X)     *
300 REM ***********************
310 FOR X=LI TO LS STEP π/64
315   ER=.01
320   IF ABS(X-π/2)<=ER OR ABS(X-3*π/2)<=ER THEN 340
325   IF ABS(X+π/2)<=ER OR ABS(X+3*π/2)<=ER THEN 340
330   Y=ES*TAN(X)
335   PLOT X*10+160,200-(100+Y),1
340 NEXT X
350 RETURN
360 REM ***********************
370 REM *    COTANGENTE(X)    *
380 REM ***********************
390 FOR X=LI TO LS STEP π/64
395   ER=.01
397   IF ABS(X-2*π)<=ER OR ABS(X+2*π)<=ER THEN 420
400   IF ABS(X-π/2)<=ER OR ABS(X-3*π/2)<=ER THEN Y=0 : GOTO
      415
405   IF ABS(X+π/2)<=ER OR ABS(X+3*π/2)<=ER THEN Y=0 : GOTO
      415
410   Y=TAN(X)
```

```
412    IF ABS(Y)<=ER THEN 420
414    Y=ES*1/Y
415    PLOT X*10+160,200-(100+Y),1
420 NEXT X
430 RETURN
440 REM ********************
450 REM *      E J E S      *
460 REM ********************
470 FOR K=0 TO 32
480    LINE 10*K,98,10*K,102,1
490 NEXT K
500 LINE 160,0,160,200,1
510 LINE 0,100,320,100,1
520 RETURN
530 PRINT "<SHFT CLR/HOME>"
540 END
```

Programa 8.3

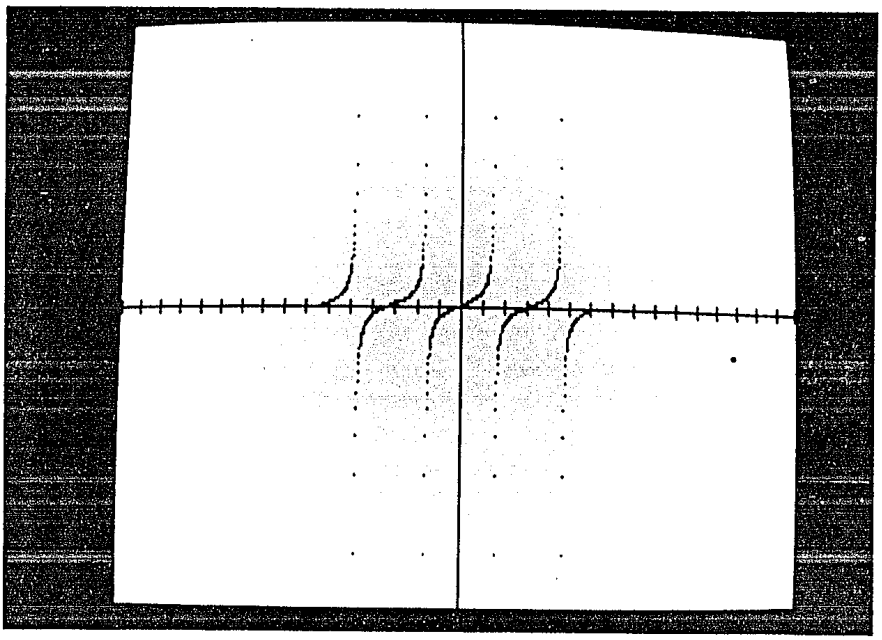


Figura 8.5. Gráfica de la función Y=tan(X).

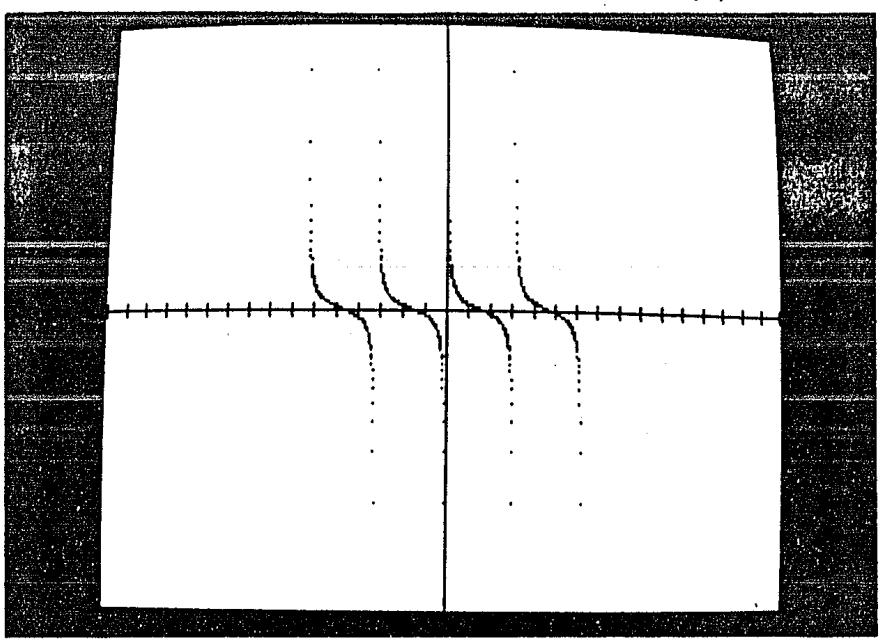


Figura 8.6. Gráfica de la función Y=ctg(X).

VIII.1.3.- GRAFICAS DE LAS CONICAS.

El siguiente programa grafica una parábola dando únicamente las coordenadas de su vértice y de su foco.

```
10  REM ************************************************
20  REM * GRAFICA DE UNA PARABOLA DADO SU VERTICE Y FOCO*
30  REM ************************************************
50  PRINT "<SHFT CLR/HOME>"
60  CENTRE "P A R A B O L A"
70  PRINT
80  CENTRE "DAME LAS COORDENADAS DEL:"
90  PRINT
100 INPUT "<CR/AB><2 CR/DE>VERTICE (H,K):";H,K
110 INPUT "<CR/AB><2 CR/DE>FOCO    (X,Y):";FX,FY
120 HIRES 0,1
130 GOSUB 500 : REM DIBUJAR EJES
140 IF H=FX THEN 190
150 IF K=FY THEN 340
160 GET A$ : IF A$="" THEN 160
170 NRM
180 END
190 REM ******************************
200 REM *     PARABOLA VERTICAL      *
210 REM ******************************
220 IF K<FY THEN Y1=125  : S=1
230 IF K>FY THEN Y1=-125 : S=-1
240 P = ABS(K-FY)
250 FOR Y=K TO Y1 STEP S
260   X=SQR(4*P*S*(Y-K))+H
270   IF ABS(X)>160 THEN Y=Y1+S : GOTO 320
280   YY=Y*.8+100
290   PLOT X+160,200-YY,1
300   X=-SQR(4*P*S*(Y-K))+H
310   PLOT X+160,200-YY,1
320 NEXT Y
330 GOTO 160
340 REM ******************************
350 REM *    PARABOLA HORIZONTAL     *
360 REM ******************************
370 P = ABS(H-FX)
380 IF H<FX THEN X1=160  : S=1
390 IF H>FX THEN X1=-160 : S=-1
400 FOR X=H TO X1 STEP S
410   Y=SQR(4*P*S*(X-H))+K
420   IF ABS(Y)>100 THEN X=X1+S : GOTO 480
430   YY=Y*.8+100
440   PLOT X+160,200-YY,1
450   Y=-SQR(4*P*S*(X-H))+K
460   YY=Y*.8+100
470   PLOT X+160,200-YY,1
480 NEXT X
```

```
490 GOTO 160
500 REM ************************
510 REM *        E J E S         *
520 REM ************************
530 LINE 160,0,160,200,1
540 LINE 0,100,320,100,1
550 RETURN
```

Programa 8.4

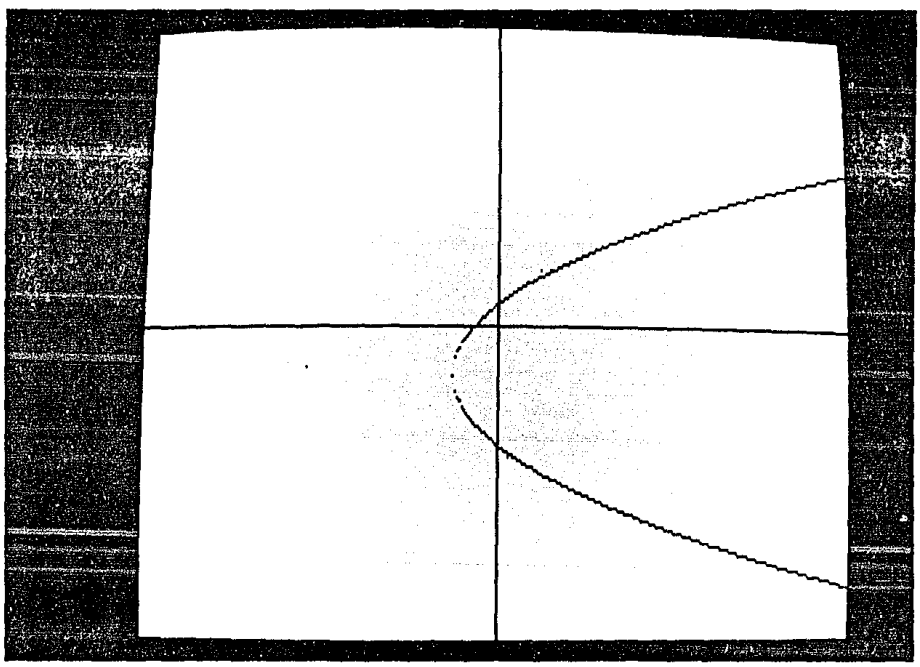


Figura 8.7. Gráfica de la parábola con vértice en (-20,20) y foco en (-10,-20).

El siguiente programa grafica una elipse dando las coordenadas de su centro, de uno de sus vértices y la longitud de su eje mayor.

```
10  REM ************************************************
20  REM *  GRAFICA DE UNA ELIPSE DADO SU CENTRO, FOCO   *
30  REM *        Y  LA LONGITUD DE SU EJE MAYOR          *
40  REM ************************************************
50  PRINT "<SHFT CLR/HOME>"
60  CENTRE "E L I P S E"
70  PRINT
80  CENTRE "<CR/AB>DAME LAS COORDENADAS DEL:"
90  PRINT
100 INPUT "<CR/AB><2 CR/DER>CENTRO (H,K):";H,K
110 INPUT "<2 CR/AB><2 CR/DE>FOCO     (X,Y):";FX,FY
120 INPUT "<2 CR/AB><2 CR/DE>LONGITUD DEL EJE MAYOR:";LM
130 A = LM/2
140 HIRES 0,1
150 GOSUB 900 : REM DIBUJAR EJES
160 IF H=FX THEN 220
170 IF K=FY THEN 540
180 GET A$ : IF A$="" THEN 180
190 PRINT "<SHFT CLR/HOME>"
200 NRM
210 END
220 REM ************************
230 REM *    ELIPSE VERTICAL     *
240 REM ************************
250 C=ABS(K-FY)
260 B=SQR(A^2-C^2)
270 LR=2*B^2/A
280 X1=H-B^2/A
290 FOR X=X1 TO X1+LR
300   Z=A^2*B^2-A^2*(X-H)^2
310   IF Z<=0 THEN Z=0
320   Y=-SQR(Z/B^2) + K
330   YY=Y*.8+100
340   PLOT X+160,200-YY,1
350 NEXT X
360 Y1=K-C
370 FOR Y=Y1 TO Y1+2*C
380   Z=(A^2*B^2)-(B^2*(Y-K)^2)
390   X=SQR(Z/A^2)+H
400   IF ABS(X)>160 THEN Y=200 : GOTO 450
410   YY=Y*.8+100
420   PLOT X+160,200-YY,1
430   X=-SQR(Z/A^2)+H
440   PLOT X+160,200-YY,1
450 NEXT Y
460 FOR X=X1 TO X1+LR
470   Z=(A^2*B^2)-(A^2*(X-H)^2)
480   IF Z<=0 THEN Z=0
```

```
490    Y=SQR(Z/B^2)+K
500    YY=Y*.8+100
510    PLOT X+160,200-YY,1
520  NEXT X
530  GOTO 180
540  REM ***************************
550  REM *     ELIPSE HORIZONTAL     *
560  REM ***************************
570  C=ABS(H-FX)
580  B=SQR(A^2-C^2)
590  LR=2*B^2/A
600  Y1=K-LR/2
610  FOR Y=Y1 TO Y1+LR
620    Z=(A^2*B^2)-(A^2*(Y-K)^2)
630    IF Z<=0 THEN Z=0
640    X=-SQR(Z/B^2) + H
650    YY=Y*.8+100
660    PLOT X+160,200-YY,1
670  NEXT Y
680  X1=H-C
690  FOR X=X1 TO X1+2*C
700    Z=(A^2*B^2)-(B^2*(X-H)^2)
710    IF Z<=0 THEN Z=0
720    Y=SQR(Z/A^2) + K
730    IF ABS(Y)>100 THEN X=320 : GOTO 790
740    YY=Y*.8+100
750    PLOT X+160,200-YY,1
760    Y=-SQR(Z/A^2) + K
770    YY=Y*.8+100
780    PLOT X+160,200-YY,1
790  NEXT X
800  FOR Y=Y1 TO Y1+LR
810    Z=(A^2*B^2)-(A^2*(Y-K)^2)
820    IF Z<=0 THEN Z=0
830    X=SQR(Z/B^2) + H
840    YY=Y*.8+100
850    PLOT X+160,200-YY,1
860  NEXT Y
870  GOTO 180
880  REM ************************
890  REM *        E J E S       *
900  REM ************************
910  LINE 160,0,160,200,1
920  LINE 0,100,320,100,1
930  RETURN
```

Programa 8.5

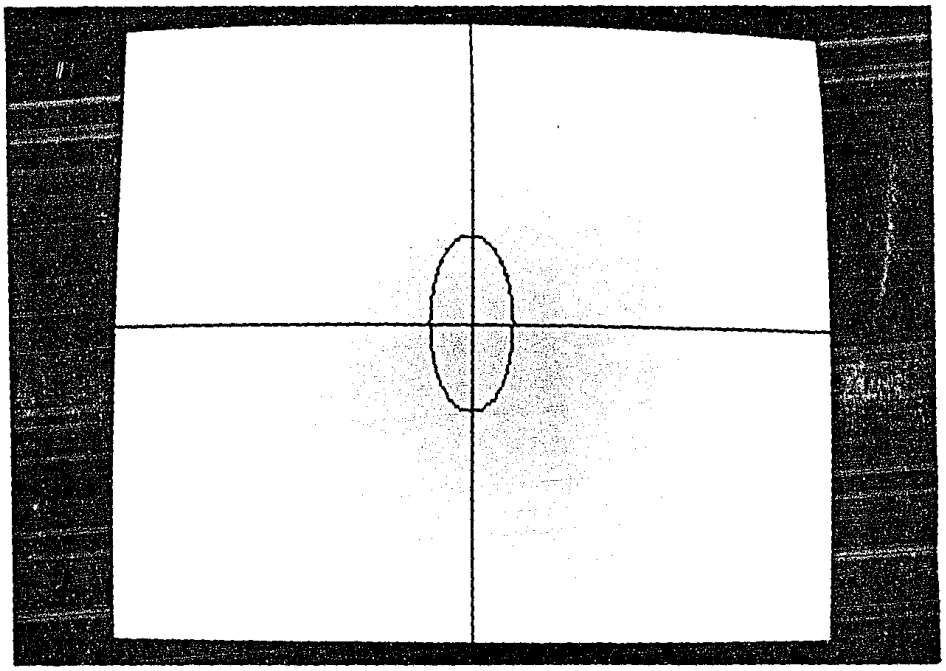


Figura 8.8. Gráfica de la elipse con centro en el origen, foco en (0,30) y longitud del eje mayor igual a 70.

El siguiente programa grafica una hipérbola, dando las coordenadas de su centro, de uno de sus focos y del vértice correspondiente.

```
10   REM ************************************************
20   REM *   GRAFICA DE UNA HIPERBOLA DADO SU CENTRO,     *
30   REM *            SU FOCO Y SU VERTICE                *
40   REM ************************************************
50   PRINT "<SHFT CLR/HOME>"
60   CENTRE "HIPERBOLA"
70   PRINT
80   CENTRE "<CR/AB>DAME LAS COORDENADAS DEL:"
90   PRINT
100  INPUT "<CR/AB><2 CR/DE>CENTRO (H,K):";H,K
110  INPUT "<2 CR/AB><2 CR/DE>FOCO    (X,Y):";FX,FY
120  INPUT "<2 CR/AB><2 CR/DE>VERTICE (X,Y):";VX,VY
130  HIRES 0,1
140  GOSUB 900 : REM DIBUJAR EJES
150  IF H=FX THEN 210
160  IF K=FY THEN 550
170  GET A$ : IF A$="" THEN 170
180  PRINT "<SHFT CLR/HOME>"
190  NRM
200  END
210  REM ************************
220  REM *       VERTICAL       *
230  REM ************************
240  A=ABS(K-VY)
250  C=ABS(K-FY)
260  B=SQR(C^2-A^2)
270  GOSUB 960 : REM ASINTOTAS
280  Y1=K+A
290  FOR Y=Y1 TO 100
300    Z=-(A^2*B^2)+B^2*(Y-K)^2
310    X=SQR(Z/A^2)+H
320    IF (X-INT(X))>.5 THEN X=INT(X)+1 : GOTO 340
330    X=INT(X)
340    IF ABS(X)>160 THEN Y=200 : GOTO 400
350    YY=Y*.8+100
360    PLOT X+160,200-YY,1
370    X=-SQR(Z/A^2) + H
380    IF ABS(X)>160 THEN Y=200 : GOTO 400
390    PLOT X+160,200-YY,1
400  NEXT Y
410  Y1=K-A
420  FOR Y=Y1 TO -100 STEP -1
430    Z=-(A^2*B^2)+B^2(Y-K)^2
440    X=SQR(Z/A^2) + H
450    IF (X-INT(X))>.5 THEN X=INT(X)+1 : GOTO 470
460    X=INT(X)
470    IF ABS(X)>160 THEN Y=-100 : GOTO 530
480    YY=Y*.8+100
490    PLOT X+160,200-YY,1
```

```
500    X=-SQR(Z/A^2) + H
510    IF ABS(X)>160 THEN Y=-100 : GOTO 530
520    PLOT X+160,200-YY,1
530  NEXT Y
540  GOTO 170
550  REM **********************
560  REM *     HORIZONTAL      *
570  REM **********************
580  A=ABS(H-VX)
590  C=ABS(X-FX)
600  B=SQR(C^2-A^2)
610  GOSUB 1100 : REM ASINTOTAS
620  X1=H+A
630  FOR X=X1 TO 160
640    Z=-(A^2*B^2)+B^2*(X-H)^2
650    Y=SQR(Z/A^2) + K
660    IF (Y-INT(Y))>.5 THEN Y=INT(Y)+1 : GOTO 680
670    Y=INT(Y)
680    IF ABS(Y)>100 THEN X=200 : GOTO 750
690    YY=Y*.8+100
700    PLOT X+160,200-YY,1
710    Y=-SQR(Z/A^2) + K
720    IF ABS(Y)>100 THEN X=200 : GOTO 750
730    YY=Y*.8+100
740    PLOT X+160,200-YY,1
750  NEXT X
760  X1=H-A
770  FOR X=X1 TO -160 STEP -1
780    Z=-(A^2*B^2)+B^2*(X-H)^2
790    Y=SQR(Z/A^2) + K
800    IF (Y-INT(Y))>.5 THEN Y=INT(Y)+1 : GOTO 820
810    Y=INT(Y)
820    IF ABS(Y)>100 THEN X=-160 : GOTO 890
830    YY=Y*.8+100
840    PLOT X+160,200-YY,1
850    Y=-SQR(Z/A^2) + K
860    IF ABS(Y)>100 THEN X=-160 : GOTO 890
870    YY=Y*.8+100
880    PLOT X+160,200-YY,1
890  NEXT X
895  GOTO 170
900  REM **********************
910  REM *       E J E S       *
920  REM **********************
930  LINE 160,0,160,200,1
940  LINE 0,100,320,100,1
950  RETURN
960  REM **********************
970  REM *     ASINTOTAS       *
980  REM **********************
990  FOR X=-160 TO 160
1000   Y=A/B*(X-H)+K
1010   IF ABS(Y)>100 THEN 1080
1020   YY=Y*.8+100
1030   PLOT X+160,200-YY
```

```
1040    Y=-A/B*(X-H)+K
1050    IF ABS(Y)>100 THEN 1080
1060    YY=Y*.8+100
1070    PLOT X+160,200-YY
1080 NEXT X
1090 RETURN
1100 REM **********************
1110 REM *      ASINTOTAS      *
1120 REM **********************
1130 FOR X=-160 TO 160
1140    Y=B/A*(X-H)+K
1150    IF ABS(Y)>100 THEN 1220
1160    YY=Y*.8+100
1170    PLOT X+160,200-YY
1180    Y=-B/A*(X-H)+K
1190    IF ABS(Y)>100 THEN 1220
1200    YY=Y*.8+100
1210    PLOT X+160,200-YY
1220 NEXT X
1230 RETURN
```

Programa 8.6

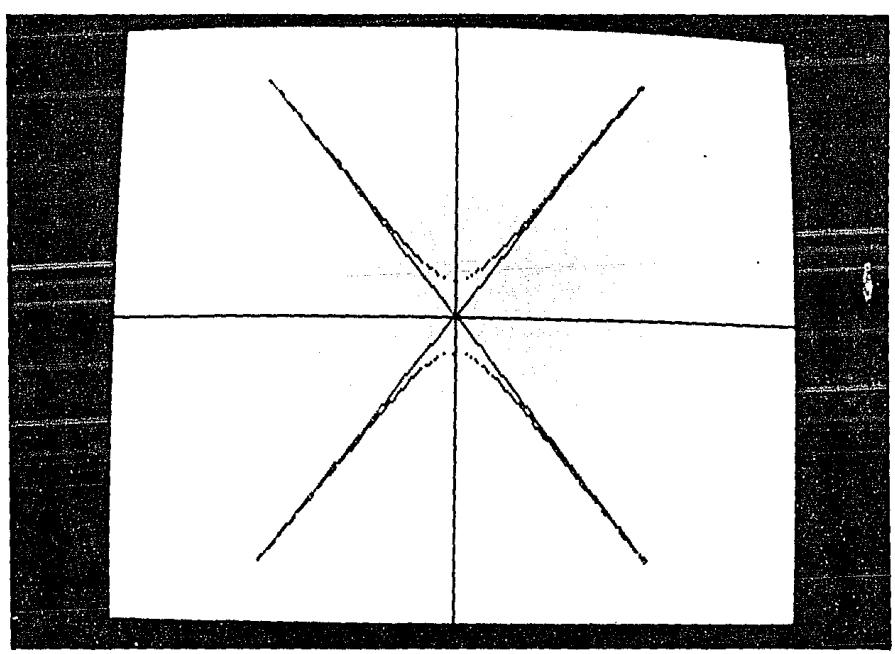


Figura 8.9. Gráfica de la hipérbola con centro en el origen, foco en (0,20) y vértice en (0,20).

VIII.1.4.- GRAFICAS EN DOS DIMENSIONES.

Un espirografo consiste de un disco dentado dentro de un círculo dentado, el cual es colocado en una hoja de papel. Supongamos que el círculo exterior tiene radio A y el disco un radio igual a B. El disco está siempre en contacto con el círculo. En el disco hay un pequeño orificio a una distancia D del centro del disco. En el orificio se coloca una pluma. El disco es rotado en sentido contrario a las manecillas del reloj, pero siempre debe tocar el círculo exterior. La pluma traza un patrón, el cual es completado al regresar la pluma a su posición original.

Se asume, que inicialmente los centros del círculo y del disco así como el orificio se encuentran alineados en la parte positiva del eje X, siendo el centro del círculo el origen.

Para poder simular el movimiento del espirografo es necesario encontrar una función que describa los puntos en la trayectoria de la pluma. Sea β el ángulo formado con el eje de las X, por la linea que une al origen con el punto en donde el círculo y el disco tienen contacto.

El punto de contacto es por lo tanto (A cos(β), A sen(β)) y el centro es el punto ((A-B)cos(β),(A-B)sen(β)). Sea $-\delta$ el ángulo formado por la línea que une al orificio con el centro del disco y el eje X, entonces las coordenadas del orificio están dadas por:

$$(A-B)\cos(\beta) + D\cos(\delta) \ , \ (A-B)\sen(\beta) - D\sen(\delta)$$

El punto de contacto entre el disco y el círculo se habrá movido una distancia Aβ alrededor del círculo, y una distancia Bθ alrededor del disco. Como no hay ningún desplazamiento estas distancias deben ser iguales y de esto resulta la siguente ecuación:

$$\theta = (A/B)\ \beta$$

La pluma regresa a su posición original cuando ambas β y θ son enteros y múltiplos de 2π. Cuando β = 2πN entonces θ = 2πN (A/B) entonces, la pluma regresa a su posición original por primera vez cuando N*(A/B) es un entero por primera vez, es decir, cuando N es igual a B dividido entre el máximo factor común de B y A.

El siguiente programa calcula el valor de N y después varía β entre 0 y 2πN en incrementos de π/100. Para cada β, el valor de θ es calculado y por ello es dibujada la trayectoria. Otro de los parámetros que usa este programa es la escala, lo cual nos permite usar parámetros mayores.

La subrutina 500 calcula el máximo factor común de dos enteros.

```
10  REM ****************************
20  REM *        ESPIROGRAFO         *
30  REM ****************************
40  PRINT "<SHFT CLR/HOME>"
50  PRINT "<HOME><2 CR/AB><2 CR/DE>RADIO DEL CIRCULO EXTER
    IOR:";
60  INPUT A
70  PRINT "<HOME><4 CR/AB><2 CR/DE>RADIO DEL DISCO INTERIO
    R :";
80  INPUT B
90  IF A<B THEN 70
100 PRINT "<HOME><6 CR/AB><2 CR/DE>DISTANCIA DEL CENTRO D
    EL DISCO AL ORIFICIO:";
110 INPUT D
120 IF B<D THEN 100
130 PRINT "<HOME><9 CR/AB><2 CR/DE>ESCALA (1-10):";
```

```
140 INPUT SC
150 IF SC<1 OR SC>10 THEN 130
160 HIRES 0,1
170 RD = D*SC
180 RA = (A-B)*SC
190 BE=0
200 BD = π * .02
210 A0 = A/B
220 GOSUB 500
230 N=B/KK
240 N0 = 100*N
250 FOR I=1 TO N0
260   BE = BE + BD
270   DE = TE * A0
280   X=RA*COS(BE)+RD*COS(DE)
290   Y=RA*SIN(BE)-RD*SIN(DE)
300   PLOT X+160,Y*.8+100,1
310 NEXT I
320 GET A$
330 IF A$="" THEN 320
340 PRINT "<SHFT CLR/HOME>"
350 END
500 REM ************************
510 REM * MAXIMO FACTOR COMUN *
520 REM ************************
530 II=A
540 JJ=B
550 M=MOD(II,JJ)
560 IF M=0 THEN 600
570 II=JJ
580 JJ=M
590 GOTO 550
600 KK=JJ
610 RETURN
```

Programa 8.7

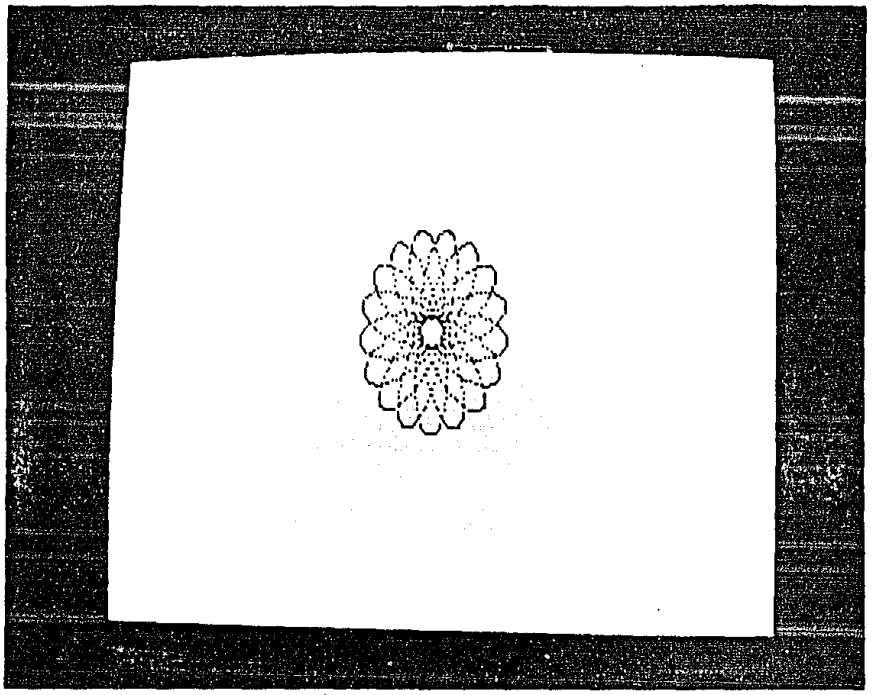


Figura 8.10. Imagen generada por el programa 8.7. Los parámetros usados son: 10,7,4,5.

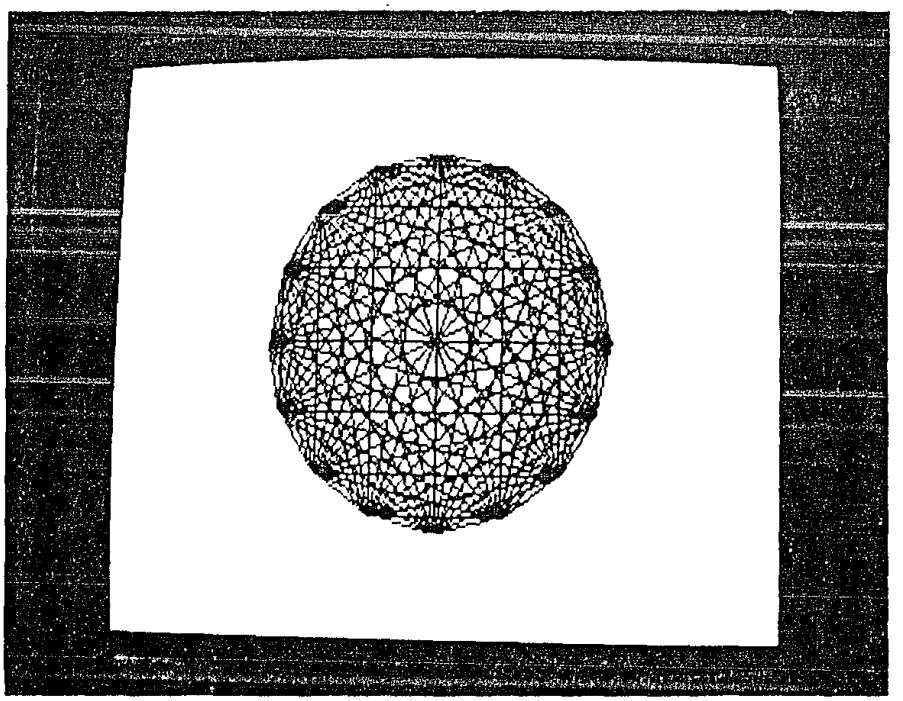


Figura 8.11. Imagen generada por el programa 8.8.

El siguente programa coloca N puntos espaciados a igual distancia sobre una circunferencia de radio R y posteriormente une cada punto con los demás.

```
10  REM ************************************
20  REM *    CIRCULO FORMADO POR LINEAS    *
30  REM ************************************
40  DIM X(50),Y(50)
50  PRINT "<SHFT CLR/HOME><10 CR/AB><6 CR/DE>NUMERO DE PU
    NTOS:";
60  INPUT N
70  X=1
80  J=1
90  R=80 : REM RADIO
100 HIRES 0,1
110 FOR I=0 TO 2*π STEP π*2/N
120   X(J)=R*COS(I)
130   Y(J)=R*SIN(I)
140   PLOT X(J)+160,Y(J)*.8+100,1
150   J=J+1
160 NEXT I
170 FOR I=1 TO N
180   FOR J=I TO N
190     LINE X(I)+160,Y(I)*.8+100,X(J)+160,Y(J)*.8+100,1
200   NEXT J
210 NEXT I
220 GET A$
230 IF A$="" THEN 220
240 PRINT "<SHFT CLR/HOME>"
250 END
```

Programa 8.8

La siguiente imagen está formada por 16 figuras intercaladas. Cada figura consiste de un cuadrado dentro del cual hay N cuadrados, que además de irse haciendo cada vez más pequeños se van rotando α grados.

El primer cuadrado es rotado en el sentido de las manecillas del reloj. El siguiente se rota en sentido contrario al primer cuadrado, y así sucesivamente van rotándose en sentido contrario al cuadrado anterior.

```
10  REM ***************************
20  REM *      CUADROS ROTADOS      *
30  REM ***************************
40  DIM X(4),Y(4),XD(4),YD(4),X1(4),Y1(4)
50  HIRES 0,1
60  DATA -90,-90,-150,-150
70  DATA -120,-60,-60,-120
80  FOR J=1 TO 4
90    READ X1(J)
100   READ Y1(J)
110   Y1(J)=Y1(J)*.8
120 NEXT J
130 RM=.1
140 SM=.9
150 FOR R=0 TO 3
160   FOR C=0 TO 4
170     FOR J=1 TO 4
180       X(J)=X1(J)+C*60
190       Y(J)=Y1(J)+R*48
200     NEXT J
210     RM=SM
220     SM=1-RM
230     FOR I=1 TO 21
240       FOR J=1 TO 4
250         NJ=MOD(J,4)+1
260         LINE X(J)+160,Y(J)+100,X(NJ)+160,Y(NJ)+100,1
270         XD(J)=RM*X(J)+SM*X(NJ)
280         YD(J)=RM*Y(J)+SM*Y(NJ)
290       NEXT J
300       FOR J=1 TO 4
310         X(J)=XD(J)
320         Y(J)=YD(J)
330       NEXT J
340     NEXT I
350   NEXT C
```

```
360 NEXT R
370 GET A$ : IF A$="" THEN 370
380 NRM
390 PRINT "<SHFT CLR/HOME>"
390 END
```

Programa 8.9

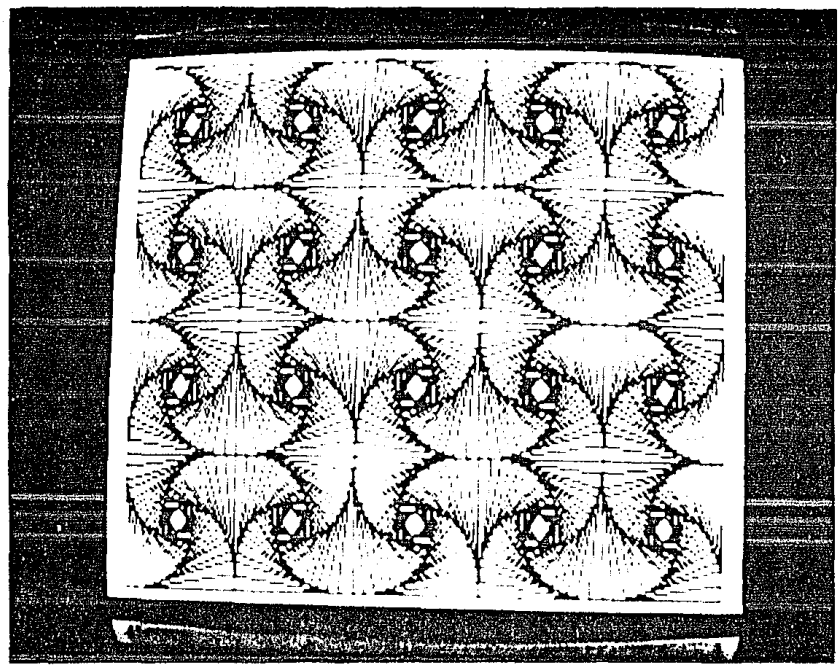


Figura 8.12. Imagen generada por el programa 8.9

VIII.2.- JUEGO DE TABLAS DE MULTIPLICAR

En este juego hay dos modos: practicar y competir.

Practicar: En este modo no se lleva ningún marcador, además de que es posible seleccionar en que rango de tablas se desea practicar. Por ejemplo, si únicamente se desea practicar con las tablas del 5 al 7 debe darse como rango inferior 5 y como rango superior 7. Además de no llevarse un marcador en este modo solamente aparecen tres respuestas posibles.

Competir : En este modo se lleva un marcador, se tienen 5 respuestas a escoger y las tablas con las que se juega son del 1 al 15 no pudiendo seleccionar un rango como en el caso anterior.

El juego consiste en lo siguiente:

Se tienen 5 casas sobre las cuales un avión soltará una bomba al mismo tiempo que aparece una pregunta (5 x 3 = ?) y 5 respuestas sobre las cuales tiene que seleccionarse la correcta, utilizando la palanca de juegos. En caso de seleccionar una respuesta equivocada explotará la bomba destruyendo la casa sobre la cual se dejó caer y además aparece el letrero: "LA RESPUESTA CORRECTA ERA: 5 x 3 = 15". En caso de seleccionar la respuesta correcta se darán N puntos dependiendo de la rápidez de la respuesta.

El juego termina al ser destruídas las cinco casas. Al inicio del juego la bomba cae a una velocidad mínima, sin embargo, a medida que el juego avanza esta velocidad va incrementándose en uno cada 10 oportunidades. Así mismo los puntos por cada respuesta correcta serán cada vez mayores.

Al terminar cada oportunidad debe oprimirse el botón de la palanca de juegos para continuar.

Al finalizar el juego en caso de que el marcador obtenido se encuentre entre los diez mejores marcadores registrados, se pedirá el nombre del competidor para que sea registrado entre ellos.

NOTA : La palanca de juegos deberá estar en el puerto 1.

La siguientes figuras muestran el desarrollo del juego.

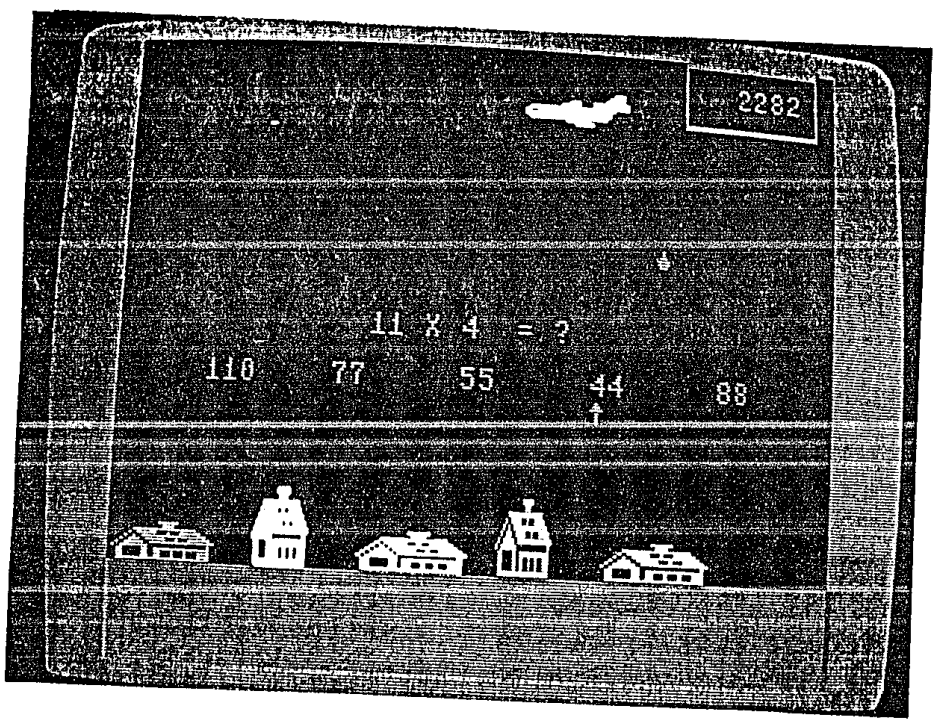


Figura 8.13

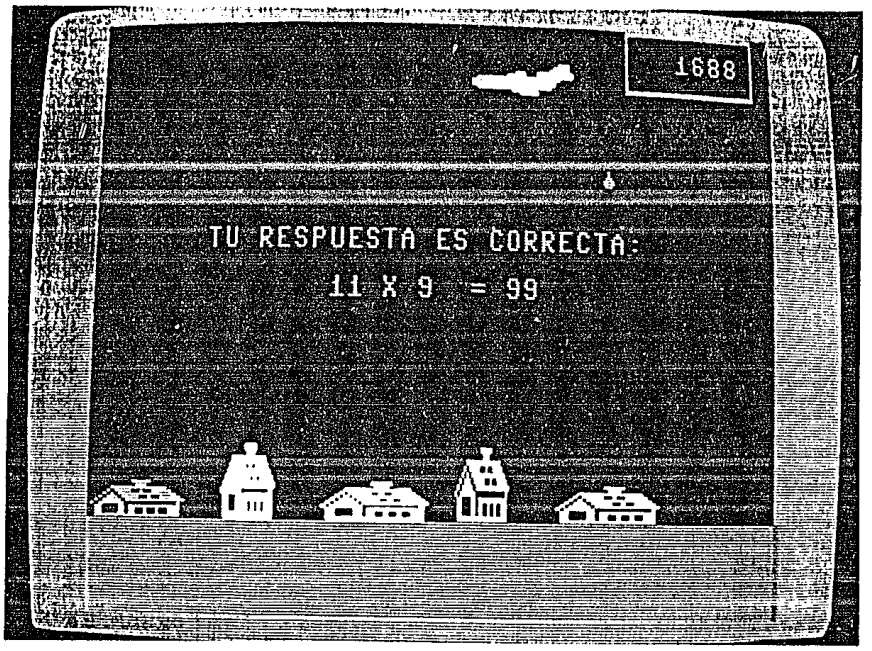


Figura 8.14

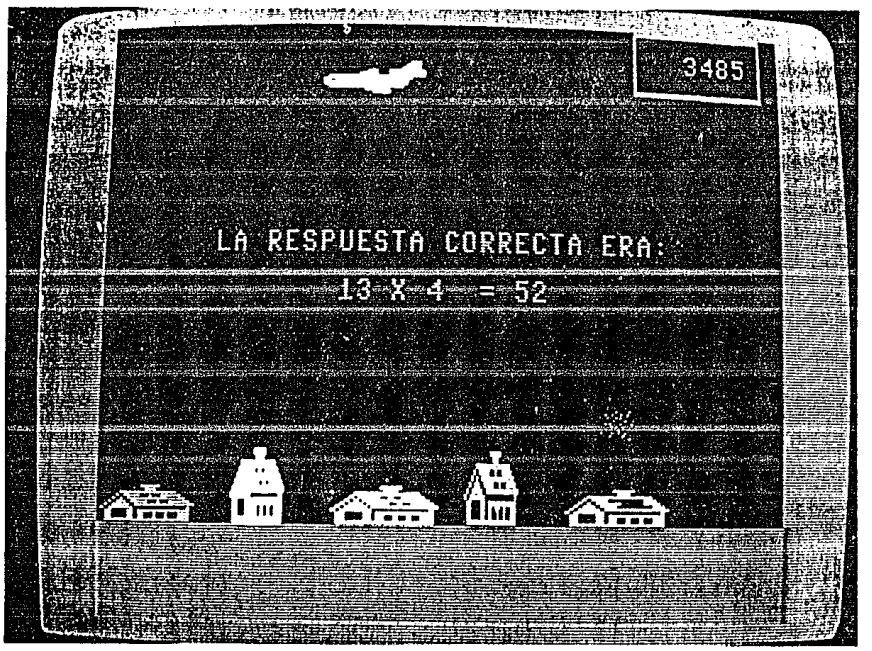


Figura 8.15

```
10  REM ********************************************
20  REM *        JUEGO DE TABLAS DE MULTIPLICAR      *
30  REM *          USANDO SPRITES, JOYSTICK          *
40  REM ********************************************
60  GOSUB 480 : REM DATOS DE LOS SPRITES
70  GOSUB 210 : REM MENU
80  IF NI=3 THEN PRINT "<SHFT CLR/HOME>" : GOTO 200
90  GOSUB 1010 : REM DIBUJA SPRITES
100 GOSUB 890  : REM INICIAR
110 IF CA>=6 THEN 160
120 GOSUB 1390 : REM INICIALIZA
130 GOSUB 1450 : REM PREGUNTAS Y RESPUESTAS
140 GOSUB 1670 : REM CICLO
150 IF CA<6 THEN 120
160 GOSUB 3020
170 PRINT "<SHFT CLR/HOME>"
180 POKE 53269,0 : REM APAGA SPRITES
190 GOTO 70
200 END
210 REM ********************
220 REM *        MENU        *
230 REM ********************
240 PRINT "<SHFT CLR/HOME>"
250 PRINT "<4 CR/AB><8 CR/DE>SELECCIONE UNA OPCION:"
260 PRINT "<CR/AB><10 CR/DE>1) PRACTICAR"
270 PRINT "<CR/AB><10 CR/DE>2) COMPETIR"
280 PRINT "<CR/AB><10 CR/DE>3) TERMINAR"
290 PRINT "<HOME><5 CR/AB><30 CR/DE>";
300 INPUT NI
310 IF NI<1 OR NI>3 THEN 300
320 IF NI=3 THEN 470
330 IF NI=1 THEN 350
340 LI=1 : LS=15 : GOTO 450
350 PRINT "<8 CR/AB><8 CR/DE>RANGO DE TABLAS (1-15):"
360 PRINT "<CR/AB><10 CR/DE>INFERIOR:"
370 PRINT "<CR/AB><10 CR/DE>SUPERIOR:"
380 PRINT"<HOME><16 CR/AB><21 CR/DE>";
390 INPUT LI
400 IF LI<1 OR LI>15 THEN 380
410 PRINT "<HOME><18 CR/AB><21 CR/DE>";
420 INPUT LS
430 IF LS<LI OR LS>15 THEN 410
440 IF NI=1 THEN R=3 : VE=1
450 IF NI=2 THEN R=5 : VE=2
460 INC=7
470 RETURN
480 REM ***********************************
490 REM *       INICIALIZA SPRITES        *
500 REM ***********************************
510 REM CASA
520 FOR I=0 TO 62
530   READ A
540   POKE 832+I,A
550 NEXT I
```

```
560  DATA 0,0,0,0,0,0,0,0,0,0,0,31,0,0,14,0,7,255,240
570  DATA 11,228,248,21,255,252,46,249,62,95,127,255,191,
     128
580  DATA 2,97,191,254,97,178,78,97,178,78,127,255,254
590  DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
600  REM CASA CON AGUJERO
610  FOR I=0 TO 62
615    READ A
620    POKE 896+I,A
625  NEXT I
630  DATA 0,0,0,0,0,0,0,0,0,0,0,31,0,0,14,0,7,255,240
640  DATA 11,192,56,21,248,60,46,240,14,95,127,255,191,128
650  DATA 2,97,191,254,97,178,78,97,178,78,127,255,254
660  DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
670  REM AVION
680  FOR I=0 TO 62
690    READ A
700    POKE 960+I,A
710  NEXT I
720  DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,6,0,0,14,0,0,30,0,24
730  DATA 255,0,120,255,63,255,248,74,170,240,127,255,240
     ,63
740  DATA 255,224,0,62,0,0,15,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
     ,0,0,0,0,0
750  REM BOMBA
760  FOR I=0 TO 62
770    READ A
780    POKE 12288+I,A
790  NEXT I
800  DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
810  DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
820  DATA 0,0,0,0,48,0,0,48,0,0,120,0,0,252,0,0,252,0,0,
     120,0
825  REM EXPLOSION
830  FOR I=0 TO 62
835    READ A
840    POKE 12352+I,A
845  NEXT I
850  DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
860  DATA 0,0,0,32,128,4,137,0,2,40,0,0,66,0,5,8,64,0,164,0,0
870  DATA 16,0,16,146,0,1,4,0,0,0,128,0,0,0,0,0,0
880  RETURN
890  REM ***************
900  REM *    INICIAR    *
910  REM ***************
920  CA=1   :   REM NO. DE CASA A EXPLOTAR
930  NC=5   :   REM NO. DE CASAS
940  BE=170 : REM ALTURA A LA QUE EXPLOTA LA BOMBA
950  XB(5)=25
960  XB(4)=82
970  XB(3)=136
980  XB(2)=190
990  XB(1)=241
1000 RETURN
```

```
1010 REM ******************************
1020 REM *    DIBUJA CASAS Y AVION     *
1030 REM ******************************
1040 PRINT "<SHFT CLR/HOME>"
1050 NS=0: CS=10: EX=1: EY=0: LX=25 : LY=203: DS=13 : GOS
     UB 1250
1060 NS=1: CS=1 : EX=0: EY=1: LX=90 : LY=188: DS=13 : GOS
     UB 1250
1070 NS=2: CS=7 : EX=1: EY=0: LX=135: LY=203: DS=13 : GOS
     UB 1250
1080 NS=3: CS=3 : EX=0: EY=1: LX=195: LY=188: DS=13 : GOS
     UB 1250
1090 NS=4: CS=15: EX=1: EY=0: LX=240: LY=203: DS=13 : GOS
     UB 1250
1100 FOR I=1864 TO 2023
1110   POKE I,160
1120   POKE 54272+I,5
1130 NEXT I
1140 NS=6 : CS=1 : EX=1 : EY=0 : LX=0: LY=0 : DS=15 : GOS
     UB 1250
1150 NS=5 : CS=12: EX=0 : EY=0 : LX=0: LY=0 : DS=192: GOS
     UB 1250
1160 IF NI=1 THEN 1240
1170 PRINT "<HOME>";
1180 FOR I=1 TO 31 : PRINT "<CR/DE>"; : NEXT I
1190 PRINT "<COMM 7><SHFT O><6 COMM Y><SHFT P>"
1200 FOR I=1 TO 31 : PRINT "<CR/DE>"; : NEXT I
1210 PRINT "<COMM 7><COMM H><6 SPC><COMM N>"
1220 FOR I=1 TO 31 : PRINT "<CR/DE>"; : NEXT I
1230 PRINT "<COMM 7><SHFT L><6 COMM P><SHFT @>"
1240 RETURN
1250 REM ********************
1260 REM *      SPRITES      *
1270 REM ********************
1280 POKE 2040+NS,DS
1290 POKE 53287+NS,CS
1300 POKE 53248+NS*2,LX
1310 POKE 53249+NS*2,LY
1320 POKE 53269,PEEK(53269) OR 2^NS
1340 IF EX=1 THEN POKE 53277,PEEK(53277) OR 2^NS : GOTO 1360
1350 POKE 53277,PEEK(53277) AND (255-2^NS)
1360 IF EY=1 THEN POKE 53271,PEEK(53271) OR 2^NS : GOTO 1380
1370 POKE 53271,PEEK(53271) AND (255-2^NS)
1380 RETURN
1390 REM ******************
1400 REM *    INICIALIZA    *
1410 REM ******************
1420 IF NI=1 THEN X=6  : XI=12 : XS=27
1430 IF NI=2 THEN X=-2 : XI=5  : XS=33
1440 RETURN
1450 REM ******************************
1460 REM *    PREGUNTAS Y RESPUESTAS    *
1470 REM ******************************
1480 N1=INT(RND(0)*(LS-LI))+LI
1490 N2=INT(RND(0)*10)+1
```

```
1500 RC=N1*N2 : REM REPUESTA CORRECTA
1510 I1=2 : I2=4
1520 IF NI>1 THEN I1=1 : I2=5
1530 FOR I=1 TO R
1540   N=INT(RND(0)*10)+1
1550   IF N=0 THEN 1540
1560   RE(I)=N1*N
1570   IF RE(I)=RC THEN 1540
1580   IF I=1 THEN 1620
1590   FOR J=1 TO I-1
1600     IF RE(I)=RE(J) THEN 1540
1610   NEXT J
1620 NEXT I
1630 LR=INT(RND(0)*R)+1
1640 IF NI=1 THEN LR=LR+1
1650 RE(LR)=RC
1660 RETURN
1670 REM ******************
1680 REM *     CICLO      *
1690 REM ******************
1700 YA=60 : YB=60
1710 XA=343 : X1=343-255
1715 POKE 53264,64
1720 POKE 53269,127
1730 POKE 53248+12,X1
1740 POKE 53249+12,YA
1750 X1=XA
1760 IF XA>255 THEN X1=XA-255
1790 IF XA=253 THEN POKE 53269,63
1800 IF XA=253 THEN POKE 53264,0
1810 POKE 53248+12,X1
1820 IF X1=253 THEN POKE 53269,127
1830 IF XA<>XB(CA) THEN 1870
1840 GOSUB 2130 : REM DIBUJA RESPUESTA
1850 POKE 53248+10,XB(CA)
1860 POKE 53249+10,YB
1870 IF XA<XB(CA) THEN 1890
1880 GOTO 2010
1890 YB=YB+VE
1900 XA=XA-3
1910 GOSUB 2280 : REM PALANCA DE JUEGOS
1920 IF FR=16 THEN 1950
1930 IF XJ=LR THEN GOSUB 2570  : GOTO 2030
1940 IF XJ<>LR THEN GOSUB 2710 : GOTO 2030
1950 IF XA>0 THEN POKE 53248+12,XA
1960 IF XA<0 THEN POKE 53269,PEEK(53269) AND (255-64)
1970 POKE 53249+10,YB
1980 POKE 53248+10,XB(CA)+12
1990 IF YB>BE THEN GOSUB 2710 : GOTO 2030
2000 GOTO 1890
2010 XA=XA-3
2020 IF XA>0 THEN 1750
2030 PRINT "<HOME><9 CR/AB><7 CR/DE><29 SPC>"
2040 PRINT "<CR/AB><37 SPC>"
2050 POKE 53258,0
```

```
2060 POKE 53259,0
2070 POKE 2045,192
2080 POKE 53260,0
2090 POKE 53261,0
2100 OP=OP+1
2110 IF OP=11 THEN OP=1 : VE=VE+1
2120 RETURN
2130 REM **********************
2140 REM *  DIBUJA RESPUESTA  *
2150 REM **********************
2160 PRINT "<HOME>"
2170 FOR K=1 TO 10 : PRINT "<CR/AB>"; : NEXT K
2180 FOR K=1 TO 13 : PRINT "<CR/DE>"; : NEXT K
2190 PRINT N1"X"N2" = ?" : PRINT
2200 I1=1 : I2=R
2210 IF NI=1 THEN I1=2 : I2=4
2220 FOR J=I1 TO I2
2230   L=J*INC-3
2250   PRINT TAB(L);RE(J);
2260 NEXT J
2270 RETURN
2280 REM ************************
2290 REM *    PALANCA DE JUEGOS  *
2300 REM ************************
2310 IF YB>60+VE THEN 2370
2320 XJ=0
2330 IF NI=1 THEN XJ=1
2340 PRINT
2350 IF NI=1 THEN PRINT "<8 SPC>"CHR$(94);: GOTO 2370
2360 PRINT " "CHR$(94);
2370 JS=PEEK(56321)
2380 FR=JS AND 16
2390 JS=15 - (JS AND 15)
2400 IF (JS=8 OR JS=4) AND X<XI THEN PRINT "<CR/IZ><SPC>
     <3 CR/IZ>";
2410 IF JS<>8 THEN 2500
2420 X=X+INC : XJ=XJ+1
2430 IF X>XS THEN X=XS : XJ=XJ-1 : GOTO 2560
2440 PRINT "<CR/IZ><SPC>";
2450 FOR J=1 TO INC-1
2460   PRINT "<CR/DE>";
2470 NEXT J
2480 PRINT CHR$(94);
2490 GOTO 2560
2500 IF JS<>4 THEN 2560
2510 X=X-INC : XJ=XJ-1
2520 IF X<XI THEN X=XI : XJ=XJ+1 : GOTO 2560
2530 PRINT "<CR/IZ><SPC>";
2540 FOR J=1 TO INC+1 : PRINT "<CR/IZ>"; : NEXT J
2550 PRINT CHR$94);
2560 RETURN
2570 REM ************************
2580 REM *  RESPUESTA CORRECTA  *
2590 REM ************************
2600 PRINT "<HOME><9 CR/AB><7 CR/DE>TU RESPUESTA ES CORRE
```

```
     CTA:"
2610 PRINT
2620 PRINT "<23 CR/DE>";RC
2630 PRINT "<CR/AB><37 SPC>"
2640 PRINT "<37 SPC>"
2650 FOR I=1 TO 50 : NEXT I
2660 IF NI=2 THEN GOSUB 2890 : REM MARCADOR
2670 FR=PEEK(56321) AND 16
2680 IF FR<>0 THEN 2670
2700 RETURN
2710 REM ***************************
2720 REM *   RESPUESTA INCORRECTA  *
2730 REM ***************************
2740 POKE 2040+5,193
2750 FOR I=0 TO 16
2760   POKE 53281,I
2770 NEXT I
2780 POKE 53281,6
2790 NC=5-CA
2800 POKE 2040+5-CA,14
2810 PRINT "<HOME><9 CR/AB><7 CR/DE>LA RESPUESTA CORRECTA
      ERA:"
2820 PRINT "<CR/AB><23 CR/DE>";RC
2830 PRINT "<CR/AB><37 SPC>"
2840 PRINT "<37 SPC>"
2850 FR=PEEK(56321) AND 16
2860 IF FR<>0 THEN 2850
2870 CA=CA+1
2880 RETURN
2890 REM ****************
2900 REM *   MARCADOR   *
2910 REM ****************
2920 PT=VE*(BE-YB)
2930 SC=SC+PT
2940 PRINT "<HOME><CR/AB>";
2950 LO=LEN(STR$(SC)) - 1
2960 L1=6-LO
2970 FOR I=1 TO 31+L1
2980   PRINT "<CR/DE>";
2990 NEXT I
3000 PRINT "<COMM 7>";SC
3010 RETURN
3020 REM *************************
3030 REM *     GRABAR RECORD     *
3040 REM *************************
3050 PRINT "<SHFT CLR/HOME>" : POKE 53269,0
3060 FOR I=1 TO 12 : PRINT "<CR/AB>"; : NEXT I
3070 PRINT "      OPRIMA <RETURN> PARA CONTINUAR";
3080 GET Z$ : IF Z$<>CHR$(13) THEN 3080
3090 OPEN 2,8,2,"SCORE,SEQ,R"
3100 FOR I=1 TO 10
3110   INPUT #2,NM$,MA$
3120   MC(I)=VAL(MA$)
3130   NO$(I)=NM$
3140 NEXT I
```

```
3150 CLOSE 2
3160 FOR I=10 TO 1 STEP -1
3170   IF MC(I)>=SC THEN LL=I+1 : I=0
3180 NEXT I
3190 IF LL=0 THEN LL=1
3200 IF LL=11 THEN 3420
3210 FOR I=10 TO LL+1 STEP -1
3220   MC(I)=MC(I-1)
3230   NO$(I)=NO$(I-1)
3240 NEXT I
3250 POKE 53269,0
3260 PRINT "<SHFT CLR/HOME>"
3270 PRINT "<4 CR/AB><7 CR/DE>FELICIDADES!   ESTAS ENTRE"
3280 PRINT "<CR/AB><7 CR/DE>LOS 10 MEJORES MARCADORES"
3290 PRINT "<2 CR/AB><7 CR/DE>DAME TU NOMBRE:";
3300 INPUT NO$(LL)
3310 MC(LL)=SC
3320 PRINT "<SHFT CLR/HOME><2 CR/AB><10 CR/DE>MEJORES MAR
     CADORES"
3330 PRINT "<2 CR/AB>"
3340 OPEN 2,8,2,"@0:SCORE,SEQ,W"
3350 C$=","
3360 FOR I=1 TO 10
3370   PRINT TAB(5)I;")"TAB(10)NO$(I)TAB(30)MC(I)
3380   MA$=STR$(MC(I))
3390   PRINT#2,NO$(I)C$MA$
3400 NEXT I
3410 CLOSE 2
3420 RETURN
```

Programa 8.10

APENDICE A

CODIGOS DE LA PANTALLA

La siguiente tabla muestra todos los caracteres que existen en la Commodore 64. En ella se muestra que número debe introducirse en la memoria de la pantalla (localidad 1024-2023) para obtener el caracter deseado.

Hay dos conjuntos de caracteres, disponibles, pero sólo uno está disponible en un momento dado. Puede cambiarse de un conjunto de caracteres a otro oprimiendo las teclas de <SHIFT> y <COMMODORE> simultáneamente.

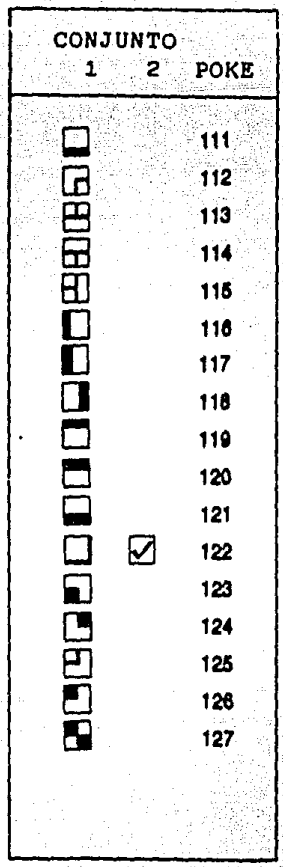
Desde BASIC esto se hace con la función POKE 53272,21 que cambia al conjunto de mayúsculas (conjunto 1) y con la expresión poke 53272,23 se cambia al de minúsculas.

Además de cualquier caracter de esta tabla puede desplegarse su correspondiente caracter inverso. El código del caracter inverso correspondiente se obtiene sumando 128 al valor que se tiene en tabla.

CONJUNTO 1	CONJUNTO 2	POKE
@		0
A	a	1
B	b	2
C	c	3
D	d	4
E	e	5
F	f	6
G	g	7
H	h	8
I	i	9
J	j	10
K	k	11
L	l	12
M	m	13
N	n	14
O	o	15
P	p	16
Q	q	17
R	r	18
S	s	19
T	t	20
U	u	21
V	v	22
W	w	23
X	x	24
Y	y	25
Z	z	26
[		27
£		28
]		29
↑		30
←		31
ESPACIO		32
!		33
"		34
#		35
$		36

CONJUNTO 1	CONJUNTO 2	POKE
%		37
&		38
'		39
(		40
)		41
*		42
+		43
,		44
-		45
.		46
/		47
0		48
1		49
2		50
3		51
4		52
5		53
6		54
7		55
8		56
9		57
:		58
;		59
<		60
=		61
>		62
?		63
		64
	A	65
	B	66
	C	67
	D	68
	E	69
	F	70
	G	71
	H	72
	I	73

CONJUNTO 1	CONJUNTO 2	POKE
	J	74
	K	75
	L	76
	M	77
	N	78
	O	79
	P	80
	Q	81
	R	82
	S	83
	T	84
	U	85
	V	86
	W	87
	X	88
	Y	89
	Z	90
		91
		92
		93
		94
		95
ESPACIO		96
		97
		98
		99
		100
		101
		102
		103
		104
		105
		106
		107
		108
		109
		110



Del código 128 al 255 los caracteres son los inversos correspondientes.

APENDICE B

CODIGO ASCII

Este apéndice muestra que caracteres aparecerán si se imprime CHR$(X) para X de 1 a 255.

ASCII	IMPRIME	ASCII	IMPRIME	ASCII	IMPRIME
	0	AZUL	31	=	61
	1	<ESPACIO>	32	>	62
	2	!	33	?	63
	3	"	34	@	64
	4	#	35	A	65
BLANCO	5	$	36	B	66
	6	%	37	C	67
	7	&	38	D	68
CONJUNTO 1	8	'	39	E	69
CONJUNTO 2	9	(	40	F	70
	10	)	41	G	71
	11	*	42	H	72
	12	+	43	I	73
<RETURN>	13	,	44	J	74
MINUSCULAS	14	-	45	K	75
	15	.	46	L	76
	16	/	47	M	77
<CR/AB>	17	0	48	N	78
ENC.INVERSOS	18	1	49	O	79
<CLR/HOME>	19	2	50	P	80
<INS/DEL>	20	3	51	Q	81
	21	4	52	R	82
	22	5	53	S	83
	23	6	54	T	84
	24	7	55	U	85
	25	8	56	V	86
	26	9	57	W	87
	27	:	58	X	88
ROJO	28	;	59	Y	89
<CR/DE>	29	<	60	Z	90
VERDE	30				

ASCII	IMPRIME
[	91
£	92
]	93
↑	94
←	95
	96
	97
	98
	99
	100
	101
	102
	103
	104
	105
	106
	107
	108
	109
	110
	111
	112
	113
	114
	115
	116
	117
	118
	119
	120
	121
	122
	123
	124
	125

ASCII	IMPRIME
	126
	127
	128
NARANJA	129
	130
	131
	132
f1	133
f3	134
f5	135
f7	136
f2	137
f4	138
f6	139
f8	140
<SHFT\RET>	141
MAYUSCULAS	142
	143
NEGRO	144
<CR/AR>	145
APAG.INVERS	146
<CLR/HOME>	147
<INS/DEL>	148
CAFE	149
ROJO CLARO	150
GRIS CLARO	151
GRIS MEDIO	152
VERDE CLARO	153
AZUL CLARO	154
GRIS OSCURO	155
PURPURA	156
<CR/IZ>	157
AMARILLO	158
CYAN	159
<ESPACIO>	160

ASCII	IMPRIME
	161
	162
	163
	164
	165
	166
	167
	168
	169
	170
	171
	172
	173
	174
	175
	176
	177
	178
	179
	180
	181
	182
	183
	184
	185
	186
	187
	188
	189
	190
	191

Los códigos del 192 al 223 son los mismos que del 96 al 127.

Los códigos del 224 al 254 son los mismos que del 160 al 190.

El código 255 es el mismo que el 126.

APENDICE C

SUBRUTINAS EN LENGUAJE DE MAQUINA

A continuación se explican las subrutinas usadas en el Capítulo 6, página 94.

A) SUBRUTINA QUE LIMPIA LA PANTALLA.

Instrucción en lenguaje de máquina	Código de la instrucción	Código Decimal
LDA #Ø0	A9 00	169, 00
TAX	AA	170
TAY	A8	168
STA ($FB),Y	91 FB	145, 251
INY	C8	200
BEQ $C00B	F0 02	240, 02
BNE $C004	D0 F9	208, 249
INC $FC	E6 FC	230, 252
INX	E8	232
CPX #$20	E0 20	224, 32
BEQ $C014	F0 02	240, 02
BNE $C004	DO F0	208, 240
RTS	60	96

Esta subrutina fué copiada del libro "COMMODORE MAGIC"

B) SUBRUTINA QUE INDICA EL COLOR DEL FONDO DE LA PANTALLA Y EL COLOR DE DIBUJO.

Instrucción en lenguaje de máquina	Código de la instrucción	Código Decimal
LDA #$1F	A9 1F	169, 0
LDX #$0	A2 00	162, 0
STX $96	86 96	134, 150
LDX #$04	A2 04	162, 4
STX $97	86 97	134, 151
LDY #$0	A0 00	169, 00
STA ($96),Y	91 96	145, 150
INY	C8	200
BNE $9F0D	D0 FB	208, 251
INC $97	E6 97	230, 151
LDX $97	A6 97	166, 151
CPX #$08	E0 08	224, 8
BNE $9F0D	D0 F3	208, 243
RTS	60	96

APENDICE D

LOCALIDADES DE MEMORIA UTILIZADAS

LOCALIDAD	BITS	DESCRIPCION
		CAPITULO I
56295-55296		MEMORIA DEL COLOR
57343-53248		MEMORIA DE LOS CARACTERES
2030-1024		MEMORIA DE LA PANTALLA
		CAPITULO II
53280		CONTROLA EL COLOR DEL BORDE DE LA PANTALLA
53281		CONTROLA EL COLOR DEL FONDO DE LA PANTALLA
56320	0 Y 3	CONTROLA LA DIRECCION QUE INDICA LA PALANCA DE JUEGOS "JOYSTICK"
53265	3	CONTROLA EL NUMERO DE RENGLONES DE LA PANTALLA VALOR DEL BIT — MODO 0 — 24 RENGLONES 1 — 25 RENGLONES
56576	0 Y 1	SELECCIONA EL BANCO DE DATOS DEL QUE VA A LEER EL VIC-II (DEFAULT 11) BANCO — VALOR DE LOS BITS 0 — 11 1 — 10 2 — 01 3 — 00
829- 1019		BUFFER DEL CASSETTE
40960-49151		INTERPRETE DE BASIC
679- 767		NO TIENEN USO
820- 827		NO TIENEN USO
1020- 1023		NO TIENEN USO

LOCALIDAD	BITS	DESCRIPCION
		CAPITULO IV
53272		LE INDICA AL VIC-II DONDE ENCONTRAR:
	7-4	LA MEMORIA DE LA PANTALLA
	3-1	LE MEMORIA DE LOS CARACTERES
56		LIMITE SUPERIOR DE BASIC
		CAPITULO V
2040-2047		APUNTADORES A LOS SPRITES
53269		PRENDE UN SPRITE (BIT I - SPRITE I)
53287-53294		COLOR DE LOS SPRITES (0-7)
53277		EXPANSION HORIZONTAL DE SPRITES (0-7)
53271		EXPANSION VERTICAL DE SPRITES (0-7)
53248-53264		POSICION DE UN SPRITE
53275		PRIORIDAD DEL SPRITE SOBRE EL FONDO DE LA PANTALLA (0-7)
53278		COLISION ENTRE SPRITES
53279		COLISION DEL SPRITE SOBRE EL FONDO
53276		SPRITES EN MODO MULTICOLOR
53285-53286		CONTROLAN DOS DE LOS POSIBLES CUATRO COLORES DE UN SPRITE EN MODO MULTICOLOR
		CAPITULO VI
53265	5	PRENDE ALTA RESOLUCION VALOR ESTADO 0 APAGA LA ALTA RESOLUCION 1 PRENDE LA ALTA RESOLUCION
53272		CONTROLA DONDE SE ENCUENTRA LA MEMORIA DEL COLOR Y LA PANTALLA DE ALTA RESOLUCION
53270	4	CONTROLA MODO MODO MULTICOLOR DE ALTA RESOLUCION

GLOSARIO DE TERMINOS.

RAM.

(Random Acces Memory). Memoria de acceso aleatorio. Es la parte de la memoria de la computadora en la que se puede guardar información en forma de datos o programas.

ROM.

(Read Only Memory). Es un tipo de memoria que ha sido programada de fábrica. A diferencia de la RAM no puede ser modificada por el programador, ni se pierde la información almacenada en ella al apagarse la computadora.

BIT.

Unidad mínima de información. Es un variable binaria.

BYTE.

Unidad de memoria que consta de 8 bits. En la Commodore 64 un byte equivale a una palabra. Un Kb equivale a 1024 bytes.

VIC - II.

Unidad de interface de video 6567. Este microprograma permite tener varios modos de graficación. Se encarga de todo el manejo de la pantalla, en él radica la capacidad de graficación de la Commodore.

SID.

(Sound Interface Device) Unidad de Interface para el sonido. Se encarga de manejar el sonido de la Commodore, permitiendo utilizar 3 voces simultáneamente.

CIA.

(Complex Interface Adaptors) Interfaces para adaptadores que se encargan de manejar las entradas y salidas, como la palanca para juegos, el teclado, la pluma luminosa, etc.

CHIP.

Microprograma hecho mediante sistemas electrónicos montados en una pequeña placa de silicio formándose circuitos integrados.

MICROPROGRAMA.

Programa hecho en lenguaje de máquina.

ASCII.

(American Standard Code for Information Interchange) Código standard americano de intercambio de información. Es un código de 7 bits por lo que sólo pueden representarse 128 caracteres. También existe una variación del código en 8 bits.

CPU.

(Central Processing Unit) Unidad central de proceso. Se llama así a la parte del procesador que engloba a la unidad aritmética lógica y a la unidad de control.

STACK.

Parte de la memoria que tiene la característica de que la primera información en entrar a esta parte de la memoria es la última en salir. El microprocesador 6510 tiene su propio stack (STACK 6510). Se usa por ejemplo cuando se encuentra un GOSUB en un programa, para recordar en donde fue llamada la subrutina, de tal manera que al encontrarse la proposición RETURN, el intérprete de BASIC, sabe a donde regresar para continuar con la ejecución del programa.

SPRITE.

Es un tipo especial de caracter definido por el usuario que puede desplegarse en cualquier parte de la pantalla. Se diferencian de los caracteres normales o gráficas producidas por la computadora, en que ellos son creados separadamente y después mezclados a la señal de video antes de desplegarse en la computadora. En otras palabras los sprites son sobrepuestos en la pantalla, no son una parte de ella. Esto resuelve el problema de las animaciones que destruyen el fondo. También simplifica el problema de memoria y velocidad debido a que son generados por el VIC-II y no por el procesador central.

B I B L I O G R A F I A

PROGRAMER'S REFERENCE GUIDE COMMODORE 64
FIRST EDITON
NINTH PRINTING 1984
PUBLISHED BY COMMODORE BUSINES MACHINES, INC.

COMPUTE!'S FIRST BOOK OF COMMODORE 64 SOUND AND GRAPHICS
GREENSBORO, NORTH CAROLINA
PRINTED IN U.S.A.
COMPUTE! PUBLICATIONS, INC.

CREATING ARCADE GAMES ON THE COMMODORE 64
ROBERT CAMP
PRINTED IN U.S.A.
COMPUTE! PUBLICATIONS, INC.

MACHINE LANGUAGE FOR BEGINNERS
RICHARD MANSFIELD
GREENSBORO, NORTH CAROLINA
PRINTED IN U.S.A. 1983
COMPUTE! PUBLICATIONS, INC.

COMMODORE MAGIC
MICHAEL CALLERY
E.P. DUTTON, INC. NEW YORK
PRINTED IN U.S.A.
FIRST EDITION

A PRACTICAL INTRODUCTION TO COMPUTER GRAPHICS
IAN O. ANGEL

COMPUTER GRAPHICS
A PROGRAMMING APPROACH
STEVEN HARRINGTON
INTERNATIONAL STUDENT EDITION