UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

DISEÑO E IMPLEMENTACION DE UN MODULO DE PRUEBAS DE SISTEMAS DE VIDEO PARA EQUIPOS COMPATIBLES CON PC IBM

T E S I S

QUE PARA OBTENER EL TITULO DE

INGENIERO MECANICO ELECTRICISTA

PRESENTA

EDGAR MATEOS SANTILLAN

DIRECTOR DE TESIS
ING. JESUS RAMIREZ ORTEGA

CODIRECTOR:
ING. JOSE ANTONIO DOMINGUEZ HERNANDEZ

MEXICO, D. F. 1998

A mi padre por enseñarme lo que es la ingeniería y por su apoyo incondicional

A mi madre por su cariño y consejos

A mis hermanos por su ejemplo.

Al Ing. Jesús Ramírez y a Ing. Antonio Domínguez por sus enseñanzas, paciencia y apoyo para la realización de este trabajo.

Agradezco al personal de la sección de cómputo y en especial al Ing. Alfredo Castillo su apoyo y valiosos comentarios.

CONTENIDO

Introducción

El campo de la ingeniería se hace día con día más amplio y complejo, ello conlleva una serie de retos para los estudiosos de la materia que implican un magno esfuerzo por responder a las variables que la modernización de la sociedad reclama.

Como una modesta colaboración a dicho proceso presentamos nuestra investigación intitulada "Diseño e Implementación de un Módulo de Pruebas de Sistemas de Video para Equipos Compatibles con PC IBM", la cual consta de 4 capítulos, resultados, conclusiones y 4 anexos o apéndices.

El primer capítulo se refiere a los monitores, su clasificación, funcionamiento y estructura.

El segundo capítulo aborda el tema de las tarjetas de video, contiene una breve descripción de su evolución histórica, además nos referimos a la memoria de video, a diversos tipos de tarjetas, y en general a su organización interna.

El tercer capítulo se refiere al bus de la computadora, en él se describen las señales utilizadas por el bus ISA.

El capítulo cuatro lo dividimos en dos partes, en la primera narramos cómo se diseñó nuestro módulo de pruebas y su funcionamiento; en la segunda hacemos una explicación de cómo está organizado el programa así como la jerarquía de clases.

Cabe aclarar que uno de los aspectos mas importantes de nuestra investigación lo describimos en el "manual para el usuario" que presentamos como apéndice No. 1, en él describimos el funcionamiento del programa y en general de todo el sistema, así como su relación con el módulo de pruebas, ello lo hace complemento indispensable del capitulo 4. Así mismo nuestro "Manual para el Usuario" contiene una descripción de los comandos del programa, las instrucciones de uso y especificaciones del sistema,.

Después del capítulo cuarto presentamos los resultados y conclusiones del trabajo. En el Apéndice 2 se incluye el código del software desarrollado debidamente comentado y dentro del Apéndice 3 se presentan los diagramas eléctricos del módulo de pruebas.

Por último en el Anexo 4 se presentan los Diagramas de tiempos de las señales del bus de la PC.

Definición del Problema

En los últimos años el uso de las computadoras personales ha aumentado en forma considerable y con ello han proliferado diferentes equipos de cómputo. Generalmente cada equipo de cómputo cuenta con un monitor que le permite interactuar con el usuario.

Los monitores por ser aparatos electrónicos son susceptibles a fallas, de tal forma que cuando se llegan a dañar se tienen las alternativas de reemplazarlo por otro o el repararlo. En el proceso de reparación de un monitor hay momentos en los que es necesario introducirle un conjunto de señales, las cuales deben de generar en la pantalla del monitor una imagen de referencia.

En muchas ocasiones las diferencias entre la imagen obtenida y la esperada le sirve de información a la persona que está reparando el equipo para saber cúal es el motivo de la falla o bien confirmar el buen funcionamiento del monitor.

Cada tipo de monitor tiene su propio conjunto de señales de entrada por lo que en ocasiones resulta difícil el poder probar algún tipo de monitor. Es por ello que el presente trabajo pretende la construcción de un aparato que conectado a una P.C. proporcione las señales necesarias que sirven de patrón para probar y ajustar algunos de los monitores compatibles con las computadoras personales de IBM, dicho aparato deberá de ser confiable y simple de utilizar.

Alternativas.

Para la realización de un generador de señales que sea posible utilizar en diferentes tipos de monitores se tienen varias alternativas:

a) Generar las señales utilizando un diseño con componentes puramente analógicos.

b) Utilizar un circuito digital para generar las señales y mandarlas directamente al monitor.

c) Construir un sistema analógico-digital que nos genere dichas señales.

En el primer caso, para poder generar las diferentes señales que utiliza un monitor se requeriría de un gran número de componentes además de no ser un diseño sencillo.

La segunda opción es el utilizar circuitos digitales lo cual facilitaría notablemente el diseño. Si éste se realizara con un microprocesador que generase las señales y las transmitiera al monitor, el programa para la generación de dichas señales se podría almacenar en una memoria ROM.

Esta posible solución requeriría de un microprocesador muy rápido en el caso de las señales para los monitores VGA y SVGA, ya que cada uno tiene una resolución de 307,200 y 786,432 pixeles en una pantalla y considerando que la información de la pantalla se tiene que repintar alrededor de 60 veces por segundo resulta que se necesita procesar información con velocidades superiores a los 68 Mhz.

La tercer alternativa , es la construcción de un sistema híbrido (analógico digital), este sistema tiene las mayores ventajas ya que en algunos casos nos facilita el diseño y es más veloz. Estas razones nos motivaron a construir nuestro sistema utilizando un diseño analógico-digital, el cual una vez que fue estructurado, resultó ser muy similar a una tarjeta de video comercial, por lo que no se consideró conveniente el diseñar algo que ya se encontraba en el mercado y se decidió emplear en nuestro diseño tarjetas de video comerciales.

El aparato que se diseñó consiste en un circuito electrónico al cual se conectan varios tipos de tarjetas de video de tal forma que nosotros decidimos cual de ellas debe de trabajar, dicho circuito es conectado con el bus de una computadora personal. Esta conexión le permite a la tarjeta prototipo informar a la P.C. qué clase de patrones de video debe de generar, además de recibir las señales que necesitan las tarjetas de video.

Para el funcionamiento de nuestro sistema de pruebas de equipos de video (SPV), es necesario utilizar un programa que se diseñó ex profeso para generar los patrones adecuados para probar y ajustar los diferentes tipos de monitores (MDA, CGA, EGA, VGA, y SVGA).

1 El Monitor

Un monitor de computadora se puede definir como un dispositivo diseñado específicamente para desplegar datos (digitales) escritos en el buffer de video de una computadora. "Un monitor de computadora en sí no es programable, pero el controlador del subsistema de video si lo es"[1]. Entendiéndose el subsistema de video como el conjunto de componentes involucrados en la transformación de la información contenida en la memoria de video de la computadora en señales apropiadas para un monitor.

La función de un monitor, es el desplegar información en la pantalla. A continuación se presenta una breve descripción de cómo se efectúa este proceso:

La superficie frontal de un monitor de computadora (pantalla del monitor), es en realidad el fin de un tubo de rayos catódicos (CRT) y para desplegar una imagen en ella, es necesario emitir dentro del CRT electrones que sean acelerados y debidamente direccionados contra la superficie interna de la pantalla, para que al incidir sobre ésta, el material que la recubre internamente sea excitado y llevado a altos niveles de energía, que cuando bajan a niveles mas estables, irradian una longitud de onda la cual es visible, así se puede iluminar un punto, pero para iluminar toda la pantalla es necesario

[1] Phoenix Technologies Ltd.
System BIOS for IBM PC/AT computers and compatibles.
Ed. Addison -Wesley
United States of America 1989
pp 164

poder direccionar el haz de electrones de tal manera que barran toda la pantalla. El barrido si nos ubicamos frente a la pantalla, se efectúa de izquierda a derecha y de arriba hacia abajo, barre una línea de izquierda a derecha, regresa a el extremo de la izquierda pero se mueve a una línea abajo y se repite el barrido de izquierda a derecha, así sucesivamente hasta llegar a la esquina inferior de la derecha, de ahí el haz regresa a la esquina superior izquierda para iniciar nuevamente su movimiento.

"El subsistema de video genera señales que provocan el barrido del haz de electrones a través de la pantalla de izquierda a derecha , y de arriba hacia abajo. La información escrita en el buffer de video es representada en la pantalla del CRT como un patrón de puntos iluminados llamados 'pixels'(picture elements)."[2]

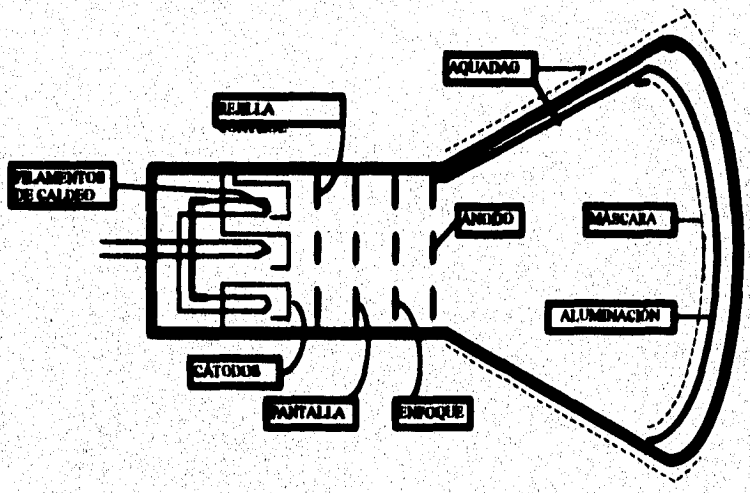
Para direccionar el haz que incide en la pantalla es necesario modificar su trayectoria inicial, esto se logra colocando unas bobinas (yugos) en el cuello del CRT, de tal forma que el campo magnético que emiten, cumpla con la función de direccionar el haz de electrones, es decir para modificar la dirección de los electrones, basta con modificar la corriente que fluye por las bobinas (yugos). El control de las corrientes que fluyen por las bobinas del yugo horizontal y vertical, es realizado por los circuitos de deflexión horizontal y vertical respectivamente.

[2]ibidem

1.1 El Tubo de Rayos Catódicos

El tubo de rayos catódicos (CRT), es "un tubo de vacío con un cañón de electrones en un extremo y en el otro una pantalla cubierta de fósforo, la cual brilla cuando es golpeada por los electrones provenientes del cañón".[3]

Consideremos inicialmente dos tipos de CRT's, los de un solo cañón electrónico, que es empleado para generar imágenes monocromáticas y los de tres cañones, que se utilizan para imágenes de color, Para simplificar sólo se describirán las partes del CRT de tres cañones ya que el monocromático se puede considerar como un caso particular del anterior.



[3]Peter Utz
Today's Video Equipment Setup and Production
Prentice Hall
Ed. Newjersey U.S.A. 1987
pp2

1.1.1 Los filamentos de caldeo

"Los cátodos se calientan indirectamente por medio de un filamento de tungsteno arrollado helicoidalmente para producir la emisión del cilindro que forma el cátodo, para producir la emisión termoiónica. Para aislar el filamento del cátodo se recubre aquel de varias capas de óxido de aluminio".[4]

1.1.2 El Cátodo

"El cátodo es un cilindro hueco para que pueda contener en su interior el filamento de caldeo, cerrado por una de sus bases, que está recubierta con un material de óxido capaz de emitir electrones a baja temperatura"[5]. se suele polarizar positivamente respecto de tierra.

En el caso de los tubos de tres cañones "la señal primaria verde se aplica al cátodo central ya que en una imagen compleja de colores no saturados es el componente verde el que contribuye más a la definición y al contorno de los objetivos porque el ojo es más sensible al verde que al rojo o al azul"[6]

[4] Aurelio Labanda Alonso y Alfonso Martín Marcos
Televisión en Color
Ediciones Auro
Barcelona 1988

[5] ibidem
[6] ibidem

1.1.3 La rejilla de control

"Se construye con forma cilíndrica o plana y está próxima al cátodo, teniendo un pequeño orificio central por el que pasa el haz de electrones". "La tensión de la rejilla de control debe ser negativa respecto al cátodo para poder modular la corriente del haz y, por tanto, obtener variaciones de brillo en función de la tensión aplicada al cátodo.

La diferencia del potencial entre el cátodo y la rejilla de control crea un campo electrostático que hace converger el haz en un punto situado cerca de la rejilla. Haciendo más negativa la rejilla respecto al cátodo se reduce el número de electrones que la atraviesan y por tanto, el brillo, hasta llegar a un valor de tensión de corte para el que se anula la corriente del haz y el brillo". [7]

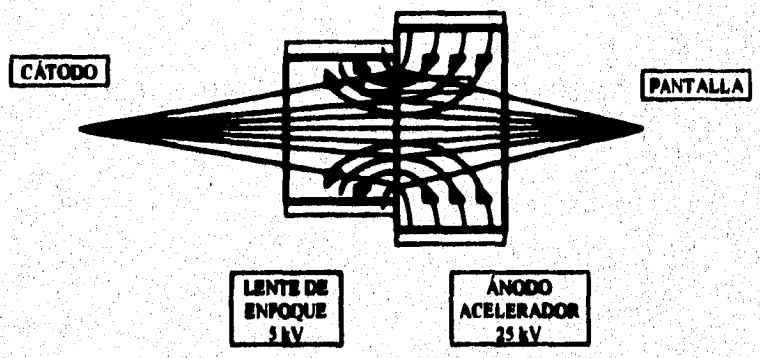
1.1.4 La rejilla de enfoque

La función de la rejilla de enfoque es "enfocar el haz sobre la pantalla ya que tiende a ensancharse debido a las mutuas repulsiones de los electrones que lo forman y si no se actuara sobre él llegaría a la pantalla con unas dimensiones que no permitiría una buena definición.

Todos los tubos de imagen utilizan enfoque electrostático, como consecuencia de uno o más campos electrostáticos que actúan como lentes ópticas. Las lentes electrostáticas producen un efecto convergente, formando un

[7] ibidem

punto de convergencia del haz de electrones, que se originaba cerca de la rejilla de control. como la diferencia de potencial es grande la velocidad de salida del haz es mucho mayor que la de entrada.".[8]



1.1.5 El ánodo acelerador

"El ánodo acelerador está constituido por un cilindro, junto con una capa conductora de aquadag, que es una forma de grafito, que recubre la superficie interior del tubo, llegando hasta la pantalla pero aislada de los luminóforos (fosforos que recubren la pantalla). El cilindro está cerrado en su parte frontal delantera por un disco al que se le han practicado tres orificios circulares, uno para cada cañón , que evitan que los electrones dispersos lleguen a la pantalla".[9]

[8] ibidem

[9] ibidem

La superficie exterior del tubo esta cubierta también de aquadag, aislada de la capa interior por la envoltura de vidrio, de tal manera que se forma un capacitor de unos cuantos picoFarads, el cual es utilizado en la fuente de alto voltaje.

El aquadag externo es conectado a tierra mientras que el interno y con ello el cilindro acelerador es conectado a la denominada "chupeta" de la fuente de alta tensión, siendo un valor típico para un tubo de color el de 25 KV.

1.1.6 La aluminación

"En todos los tubos de imagen actuales se recubren los luminóforos de la pantalla con una fina capa de aluminio de unos 200 nm de espesor"[10], la función de este es:

a) Proteger los luminóforos contra el bombardeo iónico. Realizando la función de una red que permite el paso de electrones pero impidiendo el de los iones que por tener mayor masa generan una mayor cantidad de energía al chocar con la pantalla, lo cual la puede dañar.

b) servir de espejo para la luz emitida por los luminóforos hacia el interior del tubo.

c) "constituye un camino conductor para descargar la pantalla"[11]

[10] Ibidem
[11] Ibidem

1.1.7 La pantalla

La pantalla del tubo de color "está constituido por franjas verticales de fósforos alternativamente rojo verde y azul.

Los fósforos están formados por materiales cristalinos inorgánicos, generalmente en forma de polvo de un tamaño medio aproximado de unas 10 micras. Debido a los defectos e irregularidades de la estructura cristalina, tienen la propiedad de absorber la energía incidente de los electrones que forman el haz y la convierten en luz. El proceso implica la interación de los electrones del haz con los electrones del cristal, estos son excitados por aquellos y llevados a niveles superiores, cuando los electrones excitados vuelven a los niveles de energía más estables se emite una radiación correspondiente a una zona del espectro visible"[12].

Las principales características de los fósforos son:

a) El rendimiento .- Este se puede medir por la relación entre la potencia luminosa emitida y la potencia luminosa incidente.

b) La linealidad Corriente/Luz .- Cuando la corriente es muy alta, su comportamiento se aleja del lineal, saturándose los fósforos y produciendo menos brillo.

c) La cromaticidad.

d) La persistencia.- Es el tiempo necesario para que el brillo máximo del luminóforo descienda al 10% del brillo máximo.

[12] ibídem

En la luminiscencia de todos los fósforos se pueden distinguir la fluorescencia y la fosforescencia, la primera dura mientras hay excitación y la segunda perdura después de la excitación.

"Los luminóforos tienen distinto rendimiento según el color, generalmente es el verde el que tiene mayor rendimiento, seguido del rojo y luego el azul.

Los fósforos más comúnes están constituidos por tierras raras. Un ejemplo típico podría ser:
Fósforo rojo
Trióxido de Ytrio activado con Europio ($_2O_3$ + Eu).
Su rendimiento típico es de: 3 cd/W ó 906 lm /W.
Fósforo verde
Sulfuro de zinc y cadmio activado con cobre y aluminio.
(Szn + aSCd + CU +Al).
Su rendimiento típico es de 9.5 Cd/W o 26.7 lm /W.

1.1.7.1 Tamaño de la pantalla

Los monitores se miden en pulgadas, trazando una línea diagonal de una esquina a otra de la pantalla. Existen varios tamaños por ejemplo 12, 14, 15, 17 20 y 21 pulgadas. Aún y cuando pareciera poca la diferencia no es así, ya que por ejemplo, "un monitor de 21 pulgadas proporciona aproximadamente 60%

más de área de pantalla que un monitor de 17 pulgadas y mucho más del 100% que uno de 15 pulgadas"[13]

1.1.8 La Máscara

La máscara ranurada es una lámina de acero de unas 139 micras de grueso, ligeramente curvada, de acuerdo con la pantalla del tubo de imagen, a la que se le han practicado unas 400,000 ranuras o más dependiendo del tipo de monitor.

El número de ranuras no depende mucho del tamaño de la pantalla, ya que la máscara se diseña para proporcionar suficiente resolución horizontal y vertical, de modo que el número de ranuras no sea el factor limitador comparado con el número activo de líneas de la imagen. El término "Dot Pitch" está asociado a la distancia entre ranuras de la máscara.

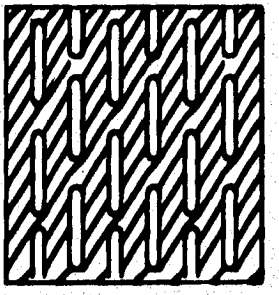
El 'Dot Pitch' "es la distancia entre puntos individuales ('pixels') expresada en milímetros. La máxima resolución depende de la distancia entre espacios."[14]

La separación entre la máscara y la pantalla se optimiza para lograr la mayor mancha de electrones sobre la pantalla sin que se produzca solapamiento, ya que durante la exploración cada haz de electrones debe incidir sobre su luminóforo correspondiente.

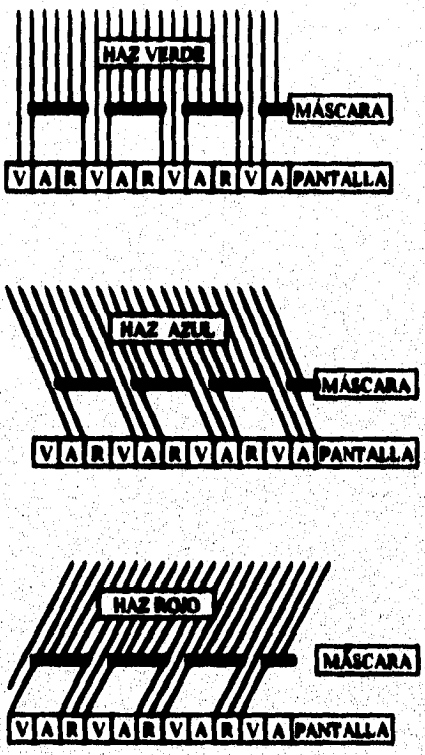
[13]Chris O' Malley, Artículo: "The big Picture", PC Computing, Junio 1996, pp 158.

[14]Rechrig Schueller, Upgrading & Mantaining your PC, Ed. Abacus, Primera Edición, USA 1990.

Para reducir la dilatación se suele recubrir la máscara y el interior del tubo con una capa de sustancia negra conductora que facilita la refrigeración. La sustancia suele ser grafito y dicha máscara se suele denominar grafitada.



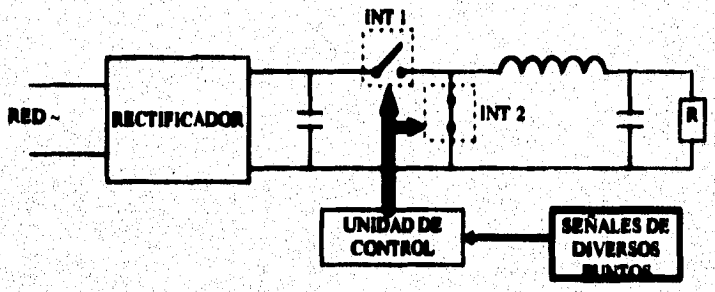
El propósito de la máscara ranurada es asegurar que cada haz de electrones procedente de los cañones incida en los fósforos correspondientes. Esta misión es posible realizarla debido a que cada haz electrónico llega a la máscara con un ángulo diferente, como se puede ver en la siguiente figura.



fusión de la mascara de sombra

1.2 Fuente de Alimentación

En los monitores de computadora actuales se emplean generalmente para su funcionamiento fuentes de alimentación conmutadas. Una fuente de alimentación conmutada básicamente se puede representar de la siguiente manera:



El voltaje a la entrada de la sección conmutada es una tensión continua Ve, la cual contiene un factor de rizo grande, debido a que no ha sido suficientemente filtrada. Los dos interruptores int-1 e int-2 funcionan en oposición, es decir cuando int-1 se encuentra abierto, int-2 estará cerrado y viceversa.

Esta conmutación se realiza con una frecuencia constante y relativamente alta que puede ser de 15 o 25 kHz. Con la fuente de alimentación excitamos a la etapa de salida de línea y por tanto es necesario que ambas frecuencias coincidan.

La unidad de control es la encargada de crear (y modificar cuando sea necesario) el ciclo de trabajo para mantener constante el nivel de voltaje a la salida de la fuente, independientemente del consumo del circuito y por tanto la demanda de corriente.

La fuente conmutada trabaja de la manera siguiente. "Durante el tiempo que int-1 se encuentra cerrado, por la bobina circula una corriente que se reparte entre el condensador y la carga. La tensión de salida varía muy poco de forma creciente, pues al ser un tiempo muy pequeño la tensión en el condensador casi no varía.

Por tanto, durante ese tiempo la tensión de entrada se reparte entre la bobina y la carga, y como la tensión en la bobina es casi constante (decrece muy poco), la corriente que circula por ella crece casi linealmente.

Al abrirse int-1 y cerrase int-2 se invierte la polaridad de la bobina de descargándose ligeramente el condensador, pero manteniendo casi constante la tensión de salida.

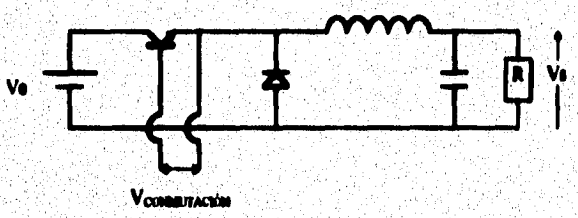
La corriente por la bobina mantiene el mismo sentido aunque decreciendo ligeramente, aportando corriente a la salida mediante la energía que tenía almacenada, a lo que contribuye también el condensador".

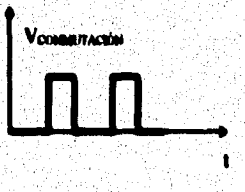
Si la tensión de red o la carga fluctúan de modo que afecte a la tensión Vs, se corrige el ciclo de trabajo por medio de la unidad de control para ajustar el valor de la tensión de salida y así establecerla en su valor nominal.

En la práctica los interruptores int-1 e int-2 no son conmutadores mecánicos sino componentes electrónicos que realizan la misma función.

"El interruptor int-1 está formado por un transistor entre sus terminales de colector y emisor, accionándose su apertura y cierre mediante impulsos eléctricos aplicados entre la base y el emisor.

Para el interruptor int-2 no es necesario colocar otro transistor pues es suficiente con un diodo. Esto debido a que cuando no ha de conducir, el potencial en el extremo de la bobina es más alto que en el otro extremo y cuando ha de conducir sucede justamente lo contrario". El circuito básico de la fuente de alimentación conmutada sería semejante a:





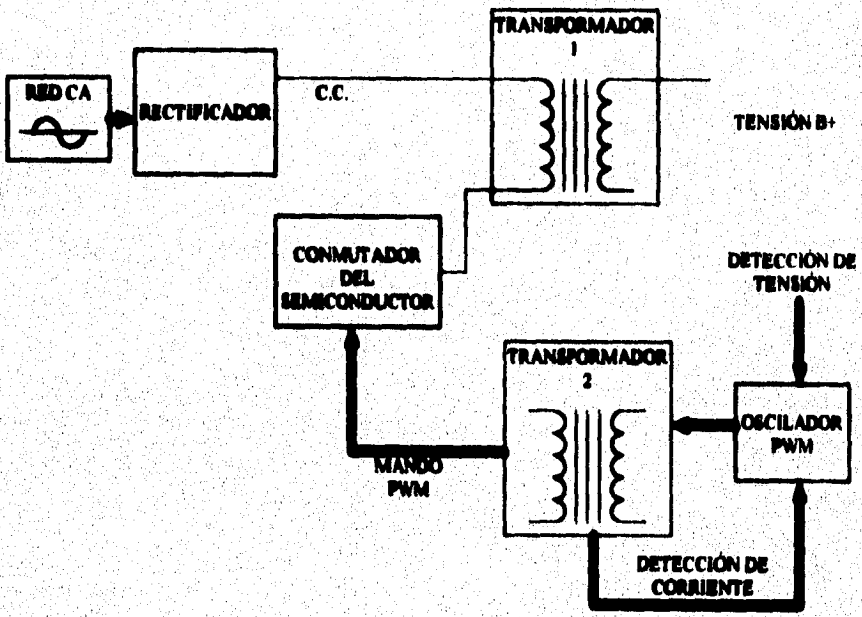
Una desventaja de este sistema es la falta de un transformador. Por ello no hay separación de la fuente principal y el chasis es 'caliente'. Sin el pulso del oscilador de línea, la fuente de alimentación no funcionará.

Las fuentes de alimentación conmutadas se pueden clasificar en dos tipos las de chasis caliente como la que se acaba de describir y las de chasis frío donde el lado secundario del transformador está aislado de la fuente principal.

1.2.1 Fuentes de alimentación del tipo de chasis frío

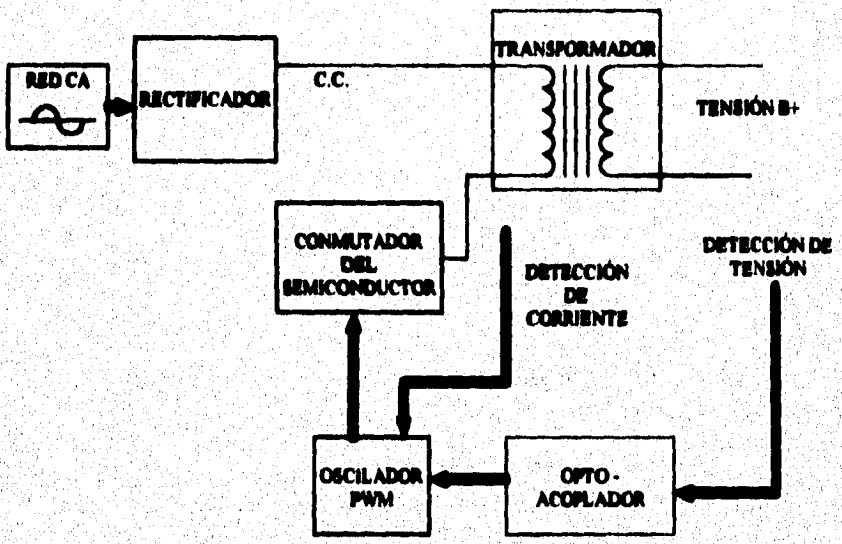
En este apartado tan solo trataremos las fuentes que utilizan como regulador de conmutación un circuito PWM (Modulación por Ancho de Pulso). Dos versiones pueden distinguirse en este tipo de fuentes, unas en donde el circuito PWM está localizado en el secundario del transformador y en las que el circuito PWM esta en el primario.

Las fuentes con el circuito PWM localizado en el secundario tienen la siguiente estructura:



La ventaja de este sistema es que la tensión B+ es detectada fácilmente por el oscilador PWM. Durante el tiempo ON de conmutación, la energía se carga en el transformador I, durante el tiempo OFF se genera una fuerza contra-electromotriz que es transferida al circuito. en consecuencia con la B+ y la detección de corriente, las señales PWM se regulan para obtener una tensión constante de B+.[15]

Una fuente conmutada con circuito PWM en el primario se puede concebir de la siguiente manera:



En este sistema el oscilador PWM está localizado en el primario. esto simplifica el circuito de excitación del conmutador semiconductor y del sensor de corriente (no se requiere separación). Para separar la tensión B+ del

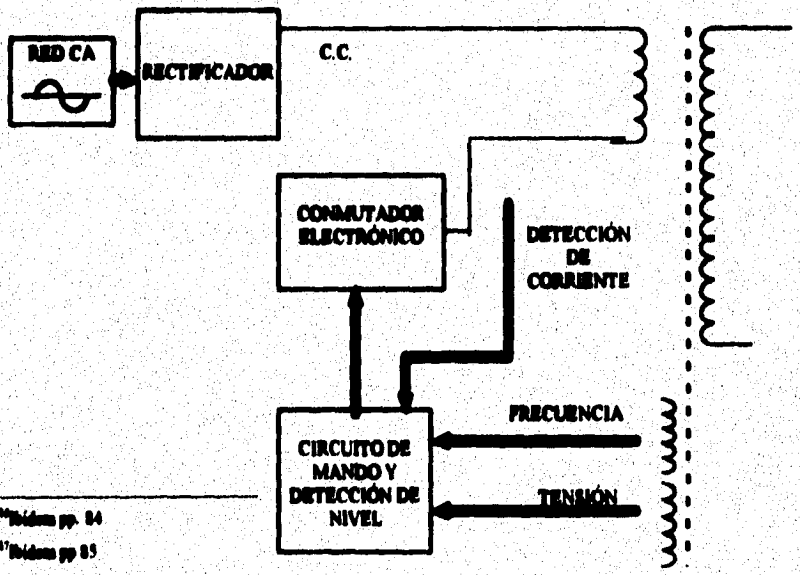
[15]Véase: Sony Service El Trinitron Tecnología y Guía de Reparaciones. Ediciones Aura, Barcelona 1986. pp. 83

secundario de la red para la detección de tensión se utiliza un acoplador óptico.[16]

1.2.2 Regulador de conmutación con oscilación de bloque[17]

En el funcionamiento de este tipo de alimentación, la fuente principal del rectificador se carga en el transformador. Cuando alcanzamos un cierto nivel la retroalimentación de frecuencia desconecta el conmutador electrónico. La magnitud de la fuerza contra-electromotriz y de la corriente es también medida para controlar el ciclo de trabajo ON/OFF del conmutador electrónico.

A pesar de que esta alimentación es muy fiable, es difícil de reparar. El oscilador en este caso es conmutador, el transformador, el bobinado de frecuencia y el circuito de detección de nivel. Si uno de estos elementos se estropea el oscilador no funcionará. En este caso se estropea fácilmente el transistor de conmutación.



[16] Ibidem pp. 84
[17] Ibidem pp 85

1.2.3 El Circuito de Muy Alta Tensión

Este circuito es el encargado de generar las altas tensiones contínuas, de las que es necesario disponer para polarizar los electrodos del tubo de rayos catódicos, así como para la aceleración de los electrones en su camino desde los cátodos hacia la pantalla del tubo de imagen. A estos circuitos se les conoce como circuitos de MAT (muy alta tensión).

Una tensión contínua de aproximádamente 25,000 volts, que se aplicará al electrodo acelerador final, que recubre el tubo de imagen, aplicándose también interiormente a uno de los electrodos de enfoque. Esta tensión, que es la conocida genéricamente como tensión de MAT no es regulable.

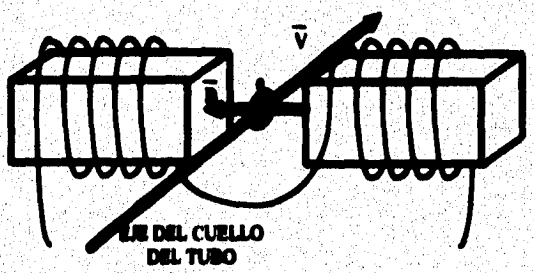
Esta tensión de 25,000 V se aplica al tubo de imagen a través de un cable blindado que termina en una ventosa por lo que es la única tensión que no se le aplica al tubo a través del contacto colocado al principio del cuello, sino en el cuerpo del tubo, cerca de la pantalla.

1.3 Deflexión

Para cubrir toda el área de la pantalla del CRT con los haces de electrones es necesario modificar la trayectoria de éstos tanto en la dirección vertical como en la horizontal.

Para la deflexión se emplean bobinas devanadas en un toroide ferromagnético (yugos de deflexión) y lo que se desea es que el haz se desplace por la pantalla de izquierda a derecha y de arriba a abajo con velocidad uniforme.

El yugo de deflexión se puede considerar como un electroimán colocado horizontalmente en el caso del yugo de deflexión vertical, ya que en el yugo de deflexión horizontal el electroimán se colocaría en sentido vertical.



La fuerza que ejerce el campo magnético sobre el electrón en movimiento de la figura viene dada por la expresión de Lorentz:

$$F = q \cdot V \times B$$

La velocidad del electrón es función de la tensión de aceleración (valor del voltaje en la fuente de muy alta tensión). El campo magnético creado por las bobinas es función de la corriente que las atraviesa, del número de espiras, de la forma y dimensiones de las mismas, por ello para generar el movimiento deseado la corriente que atraviesa las bobinas debe de crecer linealmente y al final de cada trayectoria volver al valor inicial (forma de diente de sierra).

La etapa de salida de líneas es la encargada de generar la corriente en diente de sierra que se ha de aplicar a las bobinas de desviación horizontal, para provocar las desviaciones de los haces de electrones desde la izquierda a la derecha , así como el retorno brusco.

En esta etapa hay que tener en cuenta varios efectos que influyen en ella y se deben de tomar en consideración:

a) La frecuencia del diente de sierra que genere, ha de ser controlada por la frecuencia de los pulsos de sincronía provenientes del exterior.

b) La amplitud del diente de sierra de corriente no ha de ser la misma , cuando se barren las líneas centrales de la pantalla que cuando se está en los bordes superior o inferior. Si la amplitud del barrido fuese siempre del mismo valor, en la zona central de la pantalla, a igualdad de ángulo desviado, provocaría menos desviación física, pues la distancia a recorrer por los electrones es un poco menor, debido a que se encuentran antes con la pantalla, con lo que se originaría una distorsión en forma de "cojín".

c). la amplitud del diente de sierra de corriente, ha de ser función de la corriente del haz de electrones que bombardea la pantalla, pues la tensión de la aceleración de los electrones, inevitablemente será función de la corriente del haz, y la energía necesaria para provocar la misma desviación, depende de la tensión de aceleración.

El problema de sincronismo del diente de sierra se soluciona utilizando como señal de control para la etapa, los impulsos de conmutación de la fuente de alimentación, que en otra parte han sido sincronizados con la frecuencia de línea de la señal de video que está recibiendo en ese instante.

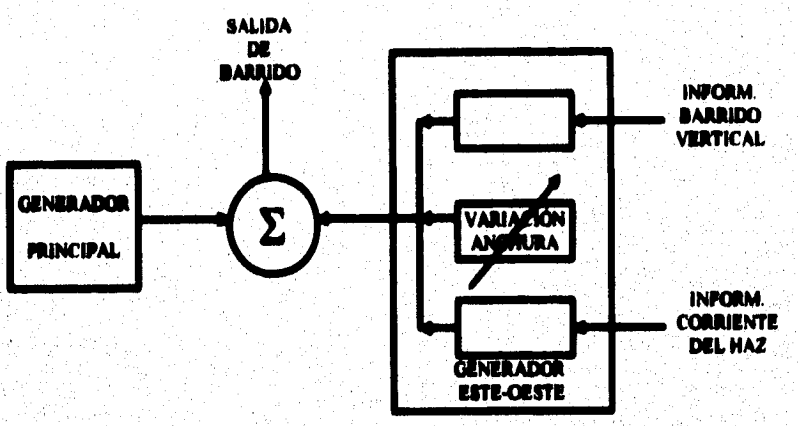
1.3.1 Circuito de Deflexión Horizontal.

Este Circuito, puede considerarse inicialmente dividido en dos circuitos independientes, el Generador Principal y el Generador Secundario.

El Generador Principal aporta aproximadamente el 80% de la energía total que aplica a las bobinas de desviación y genera dientes de sierra con amplitud de corriente constante y sincronizados por los impulsos de conmutación de la fuente de alimentación.

El Generador Secundario también conocido como generador Este-Oeste, es el encargado de aportar el resto de la energía y realizar las correcciones de amplitud del diente de sierra. La misión de este circuito es modificar,

atendiendo a varios factores, la linealidad del diente de sierra de corriente obtenido en el generador principal. A nivel de bloques podemos considerar este generador formado de la siguiente manera:



La tensión generada a la salida es la suma de tres tensiones, cada una de ellas portadora de determinada información. La tensión total se añade a la del generador principal, modificando el valor de la tensión utilizada en el generador principal. Los tres efectos que incluyen el generador Este-Oeste son los siguientes:

a) Variación de la anchura de la imagen. Este efecto es el equivalente a amplificar el diente de sierra de corriente para que tenga mayor o menor amplitud. El efecto se consigue aportando más o menos tensión a la salida, con lo que el diente de sierra de corriente evolucionará con mayor o menor amplitud dando como resultado una modificación en la anchura de la imagen.

b) Información del barrido vertical. La amplitud del diente de sierra ha de ser mayor en el centro de la pantalla que en los extremos, por lo que la

única señal que aporta información sobre la posición del haz es la señal del barrido vertical. Esta información entra al circuito previamente amplificada.

c) Información de la corriente del haz. A mayor corriente de haz corresponde menor corriente de desviación, por lo que es necesario modular la amplitud de la corriente con esta información que accede al circuito.

1.3.2 Circuito de Deflexión Vertical

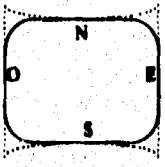
Este circuito se encarga de realizar el barrido de la pantalla de arriba hacia abajo, cuando el haz alcanza la parte inferior de la pantalla retrocede a la parte superior. Esta corriente tiene la forma de un diente de sierra.

El diente de sierra debería de ser completamente lineal si se cumplieran las condiciones de que la pantalla fuese perfectamente esférica desde el cátodo virtual de donde emergen los electrones y si la tensión que acelera los electrones hacia la pantalla fuese siempre constante y no dependiese de la corriente del haz de electrones que la golpea.

1.3.2.1 Corrección de la distorsión Norte-Sur

Debido a la relativa planicidad del tubo de rayos catódicos, cuando el haz se dirige hacia las equinas de la pantalla tiene que recorrer mayor distancia, por lo que se consiguen desviaciones mayores.

Si no se realizara ninguna corrección se produciría la distorsión en cojín. Suponiendo corregida la deformación que se produce con el barrido vertical se tendría el siguiente efecto:



Se puede observar que el efecto consiste en que, cuando nos encontramos entre el principio y la mitad de la pantalla, las líneas dibujadas se curvan hacia arriba, sucediendo lo contrario cuando nos encontramos de la mitad para abajo de la pantalla.

La única forma de corregir este defecto es conseguir que en la mitad superior de la pantalla el barrido vertical sea menos intenso cuando nos encontramos en el centro de la línea respecto de cuando estamos en los extremos a efecto de no curvar la línea en los bordes. En la mitad inferior ha de elevarse el impacto cuando nos encontremos en los extremos respecto de cuando nos encontremos en el centro. Al encontrarnos en el centro de la pantalla no es necesario hacerle corrección, por lo que para conseguir este efecto se puede actuar de dos formas :

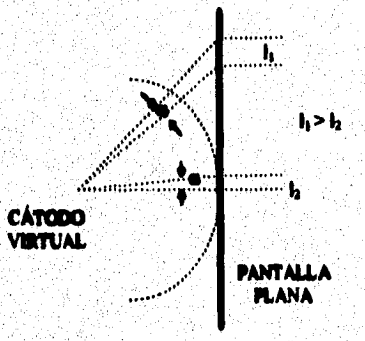
a) Modular la intensidad del diente de sierra de barrido vertical con la frecuencia de líneas de la forma que se indicada en la figura siguiente.

b) Aplicar un diente de sierra sin modular, mediante un diseño adecuado de las bobinas de desviación, en el cual el campo magnético varíe con la posición en forma de barril. "Como esto podría provocar curvatura de las

líneas horizontales en el centro de la pantalla, se introduce una corrección por medio de un anillo imantado, bipolar, situado en el cuello del tubo, coaxial con su eje. Con este anillo se puede modificar la horizontalidad de las líneas centrales".

1.3.2.2 Corrección con la posición vertical

Cuando el barrido de la pantalla es lineal, se considera que el ángulo desviado por unidad de tiempo será igual, por lo que sobre la pantalla se observarán en la parte superior e inferior las figuras más alargadas.



Para corregir ésto, al principio y al final de la pantalla se realizará el barrido menos rápido que en el centro, a efecto de que la desviación sobre la pantalla sea la misma por unidad de tiempo independientemente del punto donde nos encontremos.

La forma práctica de realizar la corrección, consiste en superponerle una tensión parabólica, obtenida de la etapa de salida de cuadro y superpuesta directamente sobre el diente de sierra que se generaba.

1.3.2.3 Corrección de la amplitud con información de la corriente del haz

De manera similar a lo que ocurre con el barrido horizontal, el barrido vertical se ve influenciado por la intensidad del haz de corriente de tal forma que cuando la corriente del haz es muy grande porque se ha de producir una imagen con mucho brillo, la tensión de la fuente de muy alto voltaje disminuye, provocando que con la misma amplitud de barrido la desviación sea mayor pues la velocidad de los electrones será menor.

Para solucionar este problema es necesario que cuando aumente el brillo disminuya proporcionalmente la amplitud del diente de sierra.

1.4 Clasificaciones de los monitores.

Los diferentes tipos de monitores que existen se podrían clasificar de tres formas diferentes, por el tipo de CRT que utiliza, por el tipo de señal que recibe y por el tipo de resolución que ofrece, a continuación pasamos a detallar cada una de estas clasificaciones.

1.4.1 Clasificación de los monitores por el tipo de CRT empleado

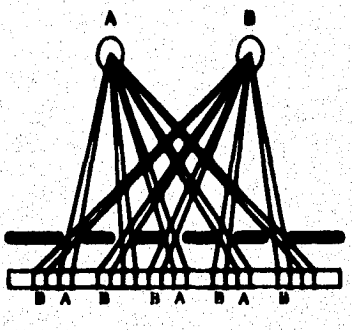
Un monitor de computadora puede emplear para desplegar su información dos tipos de CRT: el monocromático y el de color.

El CRT monocromático tiene un solo cañón de electrones y el color de la pantalla es función de los fósforos utilizados. En los monitores de computadora monocromáticos comúnmente se han empleado fósforos de color ámbar, verde o blanco.

Los monitores a color cuentan con un CRT el cual para generar una pantalla a color emplea tres cañones de electrones, uno para cada color primario de emisión (rojo, verde y azul).

Los fósforos empleados para cubrir el fin del tubo de rayos catódicos es organizado en muchos pequeños arreglos de fósforos que irradian en rojo, verde y azul, cada haz de electrones golpea solo su correspondiente fósforo en cada uno de los pequeños arreglos formándose adecuadamente las luces primarias para su emisión.

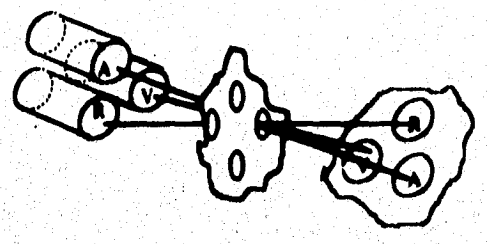
Para que cada haz de electrones incida solo en sus correspondientes fósforos, la máscara de sombra cumple una función primordial, su principio de operación se puede apreciar observando su comportamiento con dos fuentes de luz.

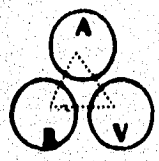


En los CRT de tres cañones, estos se pueden colocar de dos formas distintas, en Delta y en Línea.

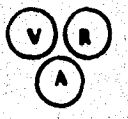
En el modelo en Delta los cañones están en correspondencia con el patrón de puntos rojos verdes y azules de la pantalla, los cuales estarían colocados en los vértices de un triángulo equilátero.[18]

[18]Véase A. Michael Noll Television Technology: Fundamentals and Future Prospects. Artech House. U.S.A. 1986 pp.61



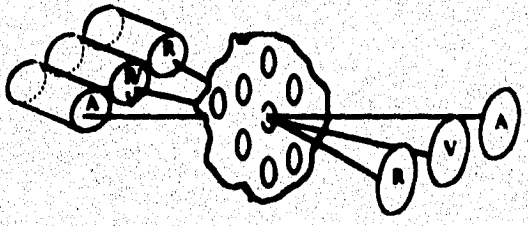


ARREGLO DE CAÑONES EN DELTA

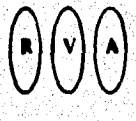


TRÍO DE PUNTOS EN LA PANTALLA

Para el modelo en Línea, los tres cañones están colocados en un mismo plano horizontal, con lo cual los haces al atravesar la máscara inciden sobre su correspondiente fósforo[19]



ARREGLO DE CAÑONES EN LÍNEA



TRÍO DE PUNTOS EN LA PANTALLA

[19]Ibidem pp. 63.

1.4.2 Clasificación de un monitor por el tipo de señal de entrada

Los monitores de computadora se pueden clasificar por el tipo de señal que reciben para su funcionamiento, éstos pueden ser:

1.4.2.1 Video Compuesto

Este tipo de monitores reciben una señal analógica compuesta, semejante a la señal usada para manejar una televisión. En la señal de color compuesta, el rojo, verde y azul además de otras señales de control, son combinadas por el controlador de video y separadas por el monitor, La señal monocromática compuesta es similar, con la excepción de que solo contiene una señal de color. Las señales compuestas pueden ser con facilidad montadas en una sola línea, pero la dificultad que implica el separar exactamente los colores limita la resolución de este tipo de monitores.

1.4.2.2 Video Digital.

Los monitores digitales RGB reciben señales para rojo, verde, o azul en diferentes líneas.

El número de colores desplegados por un monitor digital RGB es variable, por ejemplo, el IBM Enhanced Color Display, es manejado por dos líneas para cada color primario. De cada par de líneas se pueden generar cuatro posibles intensidades dando como resultado 64 colores posibles(4*4*4=64). Un

monitor digital monocromático su señal es idéntica a la del RGB con la excepción que solo tiene líneas de un "color".

1.4.2.3 RGB Analógico y Monocromático.

Los monitores analógicos, aceptan señales para el rojo, verde y azul, en líneas separadas. El número de colores potencialmente desplegables podría ser infinito, variando la intensidad de cada línea con un voltaje apropiado. Las tarjetas de video diseñadas para manejar monitores de color analógicos pueden convertir información de color discreta en voltajes analógicos.

El límite en el número de colores desplegables esta determinado por el número de bits dedicados para cada línea de color. Una tarjeta de video que disponga de 6 bits por color podría generar señales para 262144 diferentes colores ($2^{(3*6)}$= 262144). Para el monocromático, las señales analógicas son idénticas a las señales del de color, con la diferencia de que solo existe una línea de "color". Una tarjeta de video monocromática con 6 bits de color puede generar 64 tonos de color (2^6=64).

1.4.3 Clasificación de los monitores por el tipo de tarjeta controladora

Esta es la clasificación más común. a cada monitor se le nombra de acuerdo a la tarjeta de video que le envia las señales y puede ser MDA, HGC CGA, EGA, VGA, Super VGA, u otra. Esta información es ampliada en el capítulo 2 (Tarjetas de Video).

2 TARJETAS DE VIDEO*

Es indudable que las interfaces gráficas hacen más amena la interacción con la computadora, sin embargo, ello aumenta el trabajo que tiene el procesador central.

La idea original de la tarjeta de video fue la de realizar la agobiante tarea del dibujo de líneas y figuras fuera del procesador central por ello desde algunos años a la fecha, los avances en el desarrollo de las tarjetas de video han sido notables.

Una tarjeta de video es un circuito electrónico que convierte la información contenida en una parte de la memoria de la computadora llamada RAM de video en las señales apropiadas para que el monitor pueda representar en su pantalla la información contenida en la memoria de video.

2.1 Evolución de las Tarjetas de Video

Históricamente las tarjetas de video han cambiado por lo que es conveniente el recordar el desarrollo de dicha tecnología.

* Basado en los siguientes trabajos:
- Michael Tischer, PC Intern. Ed. Abacus, EUA 1992, pp. 85 - 433.
- Michael Tischer, PC System Programming, Ed. Abacus, 5ª Edición, pp. 457 - 562.
- Gunnar Forst, PC Principles, Ed. MIT Press, EUA 1990, pp. 207 - 286.

2.1.1 MDA (Monochrome Display Adapter)

El adaptador de display Monocromático de IBM o llamado también MDA, es el adaptador de video más viejo usado por la PC. El MDA fue el estándar cuando IBM lanzó al mercado la primera PC en el año de 1981.

La tarjeta MDA solo soporta un modo de operación que es el modo texto, este contiene una pantalla de 80 columnas y 25 renglones. A diferencia de otras tarjetas gráficas la MDA contiene una memoria de video muy pequeña, de tal forma que solo puede almacenar en RAM una página de video.

Esta tarjeta no puede desplegar gráficos, no obstante comparada la MDA con la CGA, la primera tiene una mayor resolución en la pantalla con lo cual se reduce el cansancio en la vista.

En la actualidad pocas PC usan tarjetas MDA, ya que IBM suspendió su producción hace algunos años. La tarjeta gráfica Hércules (HGC) reemplazó a la MDA debido a que tenía todos los atributos de la MDA pero ésta si podía desplegar gráficos.

2.1.2 CGA (Color Graphics Adapter)

La tarjeta CGA estándar, fue introducida al mercado en el año de 1981 y ofreció a los usuarios de las PC's una alternativa respecto de las MDA. Algunos usuarios de las tarjetas CGA, podían ahorrar algún dinero si en lugar de conectar un monitor, conectaban un aparato de televisión, siendo ésto

posible ya que la tarjeta CGA tenía un conector especial para este fin, además de contar con una salida RGB. Sin embargo, la calidad del video en el CGA no era tan buena como en el MDA, debido a que generaba tres pixeles de color muy grandes.

Similar a la tarjeta MDA la CGA también tiene un modo texto de 80 columnas por 25 renglones, pero cada carácter está formado por una matriz de carácteres más pequeña. Además una tarjeta CGA puede desplegar gráficos con una resolución de 320 x 200 pixeles en cuatro colores. En el modo monocromático se producen gráficos con una resolución de 600 x 200 pixeles en dos colores. Aún cuando los adaptadores gráficos CGA y MDA son diferentes, ambos están basados en el controlador de video MC6485 de Motorola.

2.1.3 HGC (Hercules Graphics Card)

Un año después de salir al mercado la PC, una compañía llamada Hércules, sacó a la venta una nueva tarjeta gráfica que inmediatamente los hizo famosos, la tarjeta gráfica Hércules (HGC).

Esta tarjeta está basada en el controlador de video de Motorola el MC6485, la tarjeta es completamente compatible con la MDA y puede desplegar dos páginas gráficas de 720 x 348 pixeles. La tarjeta Hércules combina la legibilidad de la MDA y la salida gráfica de las tarjetas CGA con la diferencia de que su resolución tanto gráfica como de texto es de alta calidad.

La tarjeta Hércules fue considerada como un estándar dentro de las tarjetas monocromáticas, desafortunadamente la original HGC tenía un defecto, no soportaba completamente el BIOS de la PC. No obstante el sistema las toleraba ya que eran compatibles con las antiguas MDA y porque la ROM BIOS soportaba el modo texto.

2.1.4 EGA (Enhanced Graphics Adapter)

Después de salir al mercado la tarjeta gráfica Hércules, IBM desarrolló un nuevo diseño de tarjeta, con la intención de superar la capacidad de la tarjeta Hércules, el resultado fue la tarjeta EGA la cual salió al mercado en el año de 1985.

Debido a los numerosos avances tecnológicos que ocurrieron entre 1981 y 1985, la tarjeta EGA inició una pequeña revolución en las computadoras personales. La tarjeta EGA al ser más completa que la CGA y la MDA juntas, cambió el estándar de resolución de la pantalla y con ello del precio, siendo por un lado mayor resolución y menor costo.

La tarjeta EGA tiene sus propios modos de video, que son tan buenos como los del MDA y CGA, con los cuales también es compatible . Esto fue útil para programas que soportaban múltiples modos de video.

La habilidad del las tarjetas EGA para desplegar gráficas monocromáticas o de color dependiendo del monitor que se le conectara fue un atributo importante ya que se convirtió en la primer tarjeta gráfica que pudo manejar pantallas monocromáticas y de color.

La tarjeta EGA tiene un mejor rendimiento cuando se utiliza con un monitor EGA. Este monitor es semejante al monitor CGA con la excepción de que su resolución en modo gráfico es más alta (de 640 x 350 pixeles) y mayor variedad de colores (16 colores ala vez para un total de 64 colores en la paleta). También la tarjeta EGA cuenta con una mayor capacidad en la memoria de video ya que algunas tarjetas pueden tener más de 256 Kb de RAM de video, lo cual les permite desplegar diferentes páginas gráficas en la pantalla, ello le da al usuario la posibilidad de crear animación en la computadora para algunas aplicaciones por ejemplo: los juegos.

En lugar del video controlador MC6485 las tarjetas EGA usan tecnología VLSI para el despliegue del video. Otra diferencia respecto de las MDA y CGA es que la EGA no es soportada por la ROM BIOS de IBM, así que la EGA cuenta con su propia ROM BIOS la cual reemplaza la original de la PC y permite acceder a todas las partes de la tarjeta EGA

2.1.5 VGA (Video Graphics Array)

La tarjeta VGA salió al mercado en el año de 1987, esta tarjeta utiliza tecnología nueva y fundamentalmente es similar a las tarjetas EGA, por lo que se mantiene la compatibilidad. Esta tarjeta ofrece más colores mayor resolución y el mejor despliegue de texto.

Semejante a las tarjetas EGA, una tarjeta VGA tiene su propio BIOS, el cual reemplaza las funciones de salida para video del BIOS estándar de la PC. Muchas veces el hardware de las tarjetas VGA se inclina a la compatibilidad

con el BIOS de las tarjetas EGA, por ello todos los programas diseñados para las tarjetas EGA pueden funcionar sin problemas en una tarjeta VGA.

Originalmente el VGA estándar fue diseñado para las PS/2 de IBM cuyo bus es del tipo microcanal, poco después crearon las tarjetas VGA para bus del tipo ISA, que es el generalmente que utiliza la PC.

Las ventajas de las tarjetas VGA respecto de las EGA es la alta densidad de integración que le permite integrar toda la lógica de control en un solo circuito integrado. A diferencia de las tarjetas EGA, la VGA envía al monitor las señales de color de manera analógica en lugar de señales discretas. Esto significa que las tarjetas VGA pueden generar más de 260,000 colores diferentes cuando son activados los modos 2, 4 ,16 o 256.

La mayor resolución que proporciona una tarjeta VGA es de 640 x 480 pixeles con 2, 4 ó 16 colores dependiendo del modo de video seleccionado. El modo de 320 x 200 pixeles es más versátil, ofreciendo simultáneamente más de 256 colores en la pantalla.

Mayor resolución o más colores significa que en algún lugar debe de ser almacenada dicha información, es por ello que las tarjetas VGA contienen un mínimo de 256 Kb de memoria RAM, la cual puede fácilmente incrementarse a 512 Kb.

2.1.6 SUPER VGA

Una tarjeta Super VGA, tiene casi el mismo hardware que una VGA normal, pero el despliegue de pixeles aumenta con mayor número de colores. estas tarjetas soportan todos los modos del VGA.

Mientras que una tarjeta VGA puede desplegar 256 colores con 320 x 200 pixeles, la tarjeta Super VGA puede desplegar el mismo número de colores en otros tres modos (640 x 200, 640 x 350 y 640 x 480) en otros modos gráficos puede desplegar 800 x 600 y 1024 x 768 pixeles, siempre y cuando disponga de la suficiente RAM de video.

Para este tipo de tarjetas los diferentes fabricantes tienen su propio modo extendido y sus registros para las tarjetas SVGA; sin embargo, los más grandes fabricantes de circuitos integrados VGA (Tseng, Paradise y Video Seven) formaron un consorcio llamado Video Electronic Standard Association (VESA). Todo ello con la finalidad de presentar un estándar para los modos Super VGA y los BIOS de video de las tarjetas que fueran desarrolladas por esos tres fabricantes. Desafortunadamente este consorcio (VESA) fue formado hasta 1990, por lo que los programas que utilizaban estas tarjetas tenían que trabajar directamente con el hardware de las diferentes tarjetas SVGA para poder hacer uso de los modos extendidos del VGA. Actualmente, mediante el uso de programas residentes en memoria (TSR) se pueden sumar nuevas funciones a las antiguas tarjetas SVGA.

"El termino 'Super VGA' ha llegado a ser difícil de ubicar. Al principio Super VGA significó la especificación VESA para las pantallas de 800 x 600 . Sin embargo, a la fecha el comité VESA ha ampliado el término en por lo

menos algunas de sus publicaciones, para incluir cualquier producto gráfico de video que implementa un juego mejorado del estándar del adaptador de video VGA de IBM". "En este contexto Super VGA significa cualquier despliegado gráfico que va más allá de la especificación de VGA de la IBM"[20]

2.1.7 MCGA Memory Controller Gate Array

La tarjeta MCGA (Memory Controller Gate Array) fue diseñada para hacer más económicas las computadoras del tipo PS/2 y para gente que le gustaban textos y gráficos relativamente buenos. Esta tarjeta fue pensada para remplazar casi todos los estándares previos.

El modo texto de la MCGA que es similar al de la tarjeta CGA proporciona una pantalla de 80 x 24 carácteres. los colores del carácter y su fondo pueden ser seleccionados de una paleta de 16 colores. A diferencia de las tarjetas CGA, la paleta del MCGA puede ser seleccionada de un grupo de 264,000 colores (similar al VGA). En modo texto la resolución vertical de las MCGA es de 400 pixeles lo cual proporciona una pantalla de alta calidad.

Para un híbrido las tarjetas MCGA tienen varios modos gráficos, La MCGA soporta bajo el modo CGA resoluciones de 320 x 200 y 640 x 200 pixeles. Debido a que la tarjeta tiene una resolución vertical de 400 pixeles en el modo CGA la iluminación de los pixeles es doble, de otra manera la imagen en la pantalla aparecería a la mitad de su tamaño.

[20] Peter Norton Toda la PC Ed. Prentice Hall 5ª Edición México 1994

La mayor desventaja de los modos VGA en esta tarjeta es la selección del color ya que aun cuando tiene la resolución necesaria para VGA, la tarjeta esta limitada en su paleta de colores debido a que cuenta con una pequeña memoria de video de tan solo 64 Kb.

Las tarjetas MCGA son nombradas así, ya que operan solo en sistemas microcanal, lo cual significa que solo las PS/2 pueden usarlas (las de baja capacidad).

2.1.8 8514/A

IBM presento la tarjeta 8514/A como un sucesor de la tarjeta VGA estándar en el año de 1987. Sin embargo, la tarjeta no fue capaz de reemplazar el estándar VGA. Esto se pudo deber a un pobre desarrollo y/o fallidas decisiones de mercado.

Se menciona un pobre desarrollo ya que la tarjeta solo fue pensada para los modelos PS/2 y sistemas microcanal, con lo cual se redujo inmediatamente el posible mercado, además IBM mantuvo los detalles técnicos de la tarjeta en secreto, con lo cual la tercera parte de los fabricantes de tarjetas no pudo construir copias compatibles con este modelo. Finalmente la tarjeta 8514/A necesitaba un programa desarrollado por IBM para tener un buen desempeño, pero algunas veces, aún y cuando el programa era muy bueno llegaba a interferir con el buen desempeño del hardware.

2.1.9 XGA y XGA-2 Extended Graphics Array

XGA (Extended Graphics Array) es el estándar de video más reciente para la familia de la PC, el cual tiene "características que van más allá de los estándares del VGA estándar y Super"[21]. XGA-2 es una especificación XGA mejorada.

XGA fue desarrollado por IBM como una plataforma de video estratégica, adecuada para las interfaces gráficas de usuario, el XGA contiene todos los modos del VGA y otros tres nuevos.

"El primer nuevo modo es el 14h. Este modo fue diseñado para desplegar 132 caracteres en cada línea, la misma cantidad que tiene una impresora normal de computadora. Este modo es muy útil cuando se está trabajando con una hoja de cálculo grande, por ejemplo, y se quiere tener la capacidad de ver la mayor cantidad de datos posibles al mismo tiempo.

Los otros dos modos no son accedidos mediante el BIOS y, por lo tanto no tienen números de servicio de video. Estos dos modos, así como el primer modo, pueden ser accedidos en tres formas: programando los registros de hardware del XGA directamente, empleando una interfaz de adaptador o usando un manejador del dispositivo.

[21] Ibidem.

Se usa una interfaz de adaptador para ejecutar el XGA con programas escritos para otro diseño de video. Hay por ejemplo, una interfaz de adaptador para el 8514 que permite adaptar programas escritos para el 8514."[22].

[22] Ibidem.

2.2 La Tarjeta de Video

Todas las tarjetas de video tienen el mismo principio de operación, sean las antiguas MDA o las nuevas Super VGA.

En una tarjeta de video el controlador del tubo de rayos catódicos (CRT) realiza muchas tareas para el diseño de pantallas de video pero él no puede trabajar por si solo. Una tarjeta de video además del CRT consta de otras funciones que interactuan con él. Una tarjeta de video se podría representar funcionalmente de la siguiente manera:

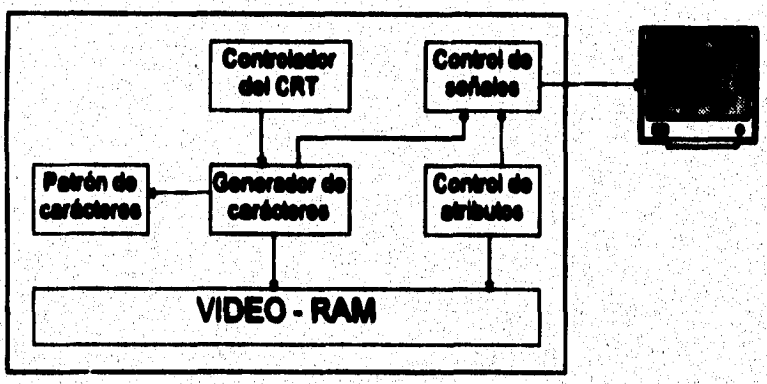


figura 2.1: Diagrama de bloques de una tarjeta de video

La RAM de video es el más importante punto de partida para crear un gráfico. La RAM de video es un rango de la memoria que se conecta directamente a la tarjeta de video. Cuando la tarjeta de video opera en modo texto, la RAM de video contiene los códigos de los carácteres ASCII y los carácteres de color, mientras que en el modo gráfico, la RAM de video es usada para guardar la información de los pixeles individuales. Dentro del modo

texto se puede acceder directamente en la RAM de video tanto al carácter generador como a su atributo. Existen variantes en la forma de almacenar la información dependiendo del tipo de video empleado.

En el modo texto para desplegar un carácter en pantalla, primero una instrucción del controlador del CRT carga un dato en la RAM de video y genera el carácter, después una ROM que contiene todos los patrones de caracteres existentes en el conjunto ASCII convierte el patrón en una tabla, el resultado de esta conversión es pasado directamente al controlador de señales, el cual es manejado por el controlador del CRT. Esta señal de control indica la línea en que se despliega el patrón en la pantalla. En el modo texto el controlador del CRT es responsable de enviar el color del carácter y leer la señal controladora.

El controlador del CRT o CRTC es la cabeza de una tarjeta de video, éste controla la operación de la tarjeta de video y genera las señales que el monitor necesita para crear una imagen. Estas tareas también incluyen controlar los puntos de luz, generar el cursor y controlar la RAM de video.

Para informar al monitor de una nueva línea, el CRTC envía a la pantalla una señal de habilitación. Esta señal activa el cañón de electrones. Miéntras el cañón de electrones se mueve de izquierda a derecha el CRTC controla individualmente las señales del cañón, haciendo aparecer sobre la pantalla el pixel deseado. En el final de la línea el CRTC deshabilita la señal del cañón de electrones.

2.3 La Memoria de video (video RAM)

La memoria RAM de una computadora, generalmente no es necesario que físicamente esté colocada en la tarjeta madre (mother board), ella puede estar colocada en una tarjeta de expansión, carácterística que es usada frecuentemente por las tarjetas de video.

La RAM de video es una localidad de memoria común para todas las tarjetas de video (desde las MDA hasta las Super VGA), sea en modo texto o en modo gráfico. En modo gráfico la RAM de video está organizada de diferente forma para los distintos modelos de tarjetas; sin embargo, en el modo texto su organización es virtualmente idéntica para todas.

2.3.1 La memoria de video dentro del espacio direccionable de la PC

El rango de direccionamiento de la RAM contenida en una PC no es completamente flexible, ésto se debe a que las primeras PC tenían para almacenar información solo un megabyte (en RAM, ROM y/o periféricos), del cual las tarjetas de video solo necesitaban los segmentos A y B, comenzando en la dirección A000H y B000H.

Las tarjetas MDA y Hércules, usan el rango de la memoria de video comprendido entre la dirección B000:0000 y la B000:7FFF; las tarjetas de color como CGA, EGA y VGA usan la memoria de video a partir de la dirección B800:0000. Sin embargo una tarjeta de video no siempre usa 32 Kb de memoria, así la MDA usa solo los primeros 4 KB de la RAM de video, la

tarjeta CGA usa los primeros 16 Kb, la tarjeta Hércules usa 32 Kb y se puede extender a la segunda mitad del *segmento* B. las tarjetas EGA y VGA usan una gran parte de la RAM de video, siendo generalmente más de 256 Kb.

Es posible mejorar la velocidad de escritura y el despliegue de gráficos, si en lugar de utilizar las interrupciones de la ROM BIOS se actúa directamente sobre la memoria de video. Para poder programar directamente la tarjeta de video, es necesario conocer el diseño y posición de la RAM de video, es por ello que a continuación presentamos una explicación de cómo está organizada una tarjeta de video.

Para el modo texto, cada posición en la pantalla del monitor es representada dentro de la RAM de video como dos bytes, donde en el primer byte es colocado el código ASCCI del carácter que se desea desplegar y en la siguiente parte los atributos referentes a ese carácter. El byte de atributos siempre se encuentra en las direcciones impares, está dividido en dos partes, la parte más significativa (bits 4 al 7) contiene el fondo del carácter y la parte menos significativa (bits 0 al 3) describe las características del carácter.

El primer carácter en la pantalla (el de la esquina superior izquierda) también es el primer carácter en la RAM de video y esta localizado con un *offset* de 0000H respecto de la primera localidad de video. el siguiente carácter de la derecha tiene un *offset* de 0002H, etc. El primer carácter de la segunda línea de la pantalla es el siguiente de la última línea del primer renglón y así sucesivamente.

Para encontrar en la memoria de video la posición correspondiente a determinada posición de la pantalla (renglón, columna), es necesario conocer el

máximo de renglones (MaxReng) y el máximo de columnas (MaxCol) del modo de video que se esta utilizando, con esta información se puede calcular la posición deseada empleando la siguiente fórmula:

offset_posición(renglón columna)= renglón*MaxCol*2 + columna*2

El factor de 2 es porque cada posición en la pantalla ocupa dos localidades en la memoria.

En el caso de las tarjetas de video que soportan varias páginas de video, para acceder a la localidad de alguna posición en especial se utiliza la misma fórmula con la adición del número de la página deseada por el tamaño de la página correspondiente a ese modo de video. La primer página contiene el valor de cero.

offset_posición(renglón,columna,página)= renglón*MaxCol*2 + columna*2
+ página*MaxCol*MaxReng*2

Tamaño de una página = MaxCol*MaxReng*2

2.4 MDA

La tarjeta MDA es excelente para desplegar texto ya que cuenta con una matriz de carácteres de 9 x 14 lo cual le permite desplegar carácteres de alta resolución. Esta forma de la matriz es rara ya que el generador de carácteres que contiene el patrón de bits solo puede producir carácteres de 8 pixeles de ancho.

El generador de carácteres requiere de un byte por cada línea de la matriz, (un bit por pixel), por lo que cada carácter necesita 14 bytes. Todo el conjunto de carácteres requiere de cuando menos 4 Kb de memoria, siendo almacenados en un ROM de la tarjeta.

La memoria de video en esta tarjeta comienza en la dirección B000:0000 y abarca 4 KB (4096 bytes); sin embargo, a la tarjeta solo le es posible desplegar a pantalla 2000 carácteres, por lo que requiere solo 4000 bytes de memoria siendo los últimos 96 bytes usados para otras aplicaciones.

El byte que almacena los atributos del carácter en este tipo de tarjetas tiene la estructura mostrada en la figura 2.2.

2.4.1 Registro de control

El registro de control de la tarjeta MDA está localizado en la dirección del puerto 3B8H y como su nombre lo indica controla varias funciones de la tarjeta, solo son importantes de mencionar los bits 0, 3 y 5.

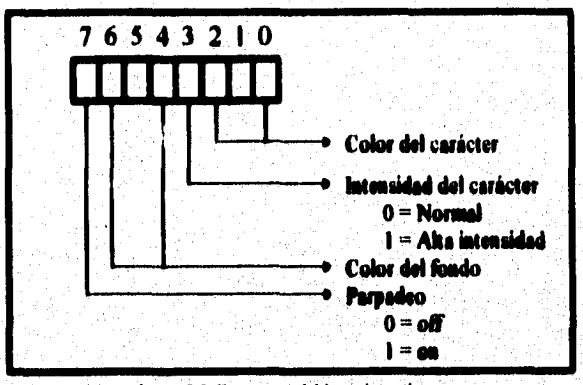


figura 2.2: Estructura del byte de atributos

En el registro de control el bit 0, controla la resolución de la tarjeta, aunque la tarjeta solo soporta una resolución (80 x 25), en este bit se debe de colocar un valor de 1 lógico durante la inicialización del sistema, de lo contrario la computadora entrará en un ciclo de espera sin fin.

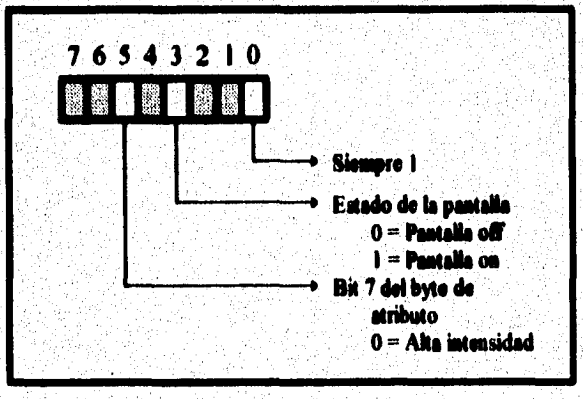


figura 2.3: Registro de Control

El bit 3 controla la creación de una pantalla en el monitor, esto es, si el bit 3 es puesto a 0, la pantalla es negra y el parpadeo del cursordesaparece, si el bit es cambiado a 1, se vuelve a desplegar información a la pantalla. El bit 5 tiene una función similar. Este registro sólo puede ser escrito, lo cual hace imposible para un programa determinar el estado de la pantalla. El valor por default de este registro es 29H esto significa que los tres bits importantes están en 1.

2.4.2 Registro de estado

En el registro de estado solo los bits 0 y 3 son usados, todos los otros bits deben contener el valor de 1. A diferencia del registro de control los programas pueden leer este registro, pero su contenido no puede ser modificado por el código del programa.

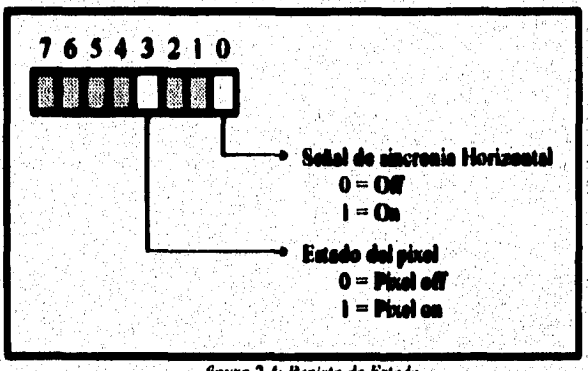


figura 2.4: Registo de Estado

El bit 0 indica si una señal de sincronía horizontal fue mandada a la pantalla, la tarjeta de video manda esta señal después de crear una línea de la pantalla,(una línea de texto es diferente de una línea de la pantalla, ya que la primera consta de 14 líneas de pantalla) esta señal informa al cañón de

electrones que debe retornar al extremo izquierdo de esa línea de la pantalla, para este caso el valor del bit fue 1. El bit 3 contiene el valor del pixel donde el cañón de electrones está localizado, un 1 indica que el pixel es visible en la pantalla y un cero indica que el pixel permanecerá oculto.

Los registros de direccionamiento del 6845 en la tarjeta MDA están en la dirección del puerto 3B4H y el registro del dato esta localizado en la dirección 3B5H.

2.5 HGC

La tarjeta gráfica Hércules es capaz de manejar un monitor monocromático con 80 x 25 carácteres en modo texto y 720 x 348 pixeles en modo gráfico.

El BIOS de la IBM, sólo soportaba las tarjetas MDA y CGA, la tarjeta Hércules no era soportada por el BIOS, sin embargo, este no es un problema en la actualidad para el modo texto, ya que la tarjeta Hércules es completamente compatible con la tarjeta MDA. Esta incompatibilidad es de importancia en el modo gráfico debido a que las funciones del BIOS no pueden cambiar o leer la información de los pixeles en la pantalla.

2.5.1 La RAM de video en un HGC

Las tarjetas de video Hércules contienen 64 Kb de RAM. Esto implica que puede contener algunas páginas de texto (requieren de 4 KB) o una página gráfica.

La memoria puede ser dividida en dos páginas de 32 KB, de las cuales la primer página está localizada en el rango de direcciones de la B000:0000 a la B000:7FFF y . la segunda página inicia en la B000:8000 y termina en la B000:FFFF.

Semejante a las tarjetas MDA y CGA, la tarjeta gráfica Hércules tiene el circuito integrado M6845 como procesador principal del controlador del CRT, la dirección de su registro es la 3B5H.

2.5.2 Registro de configuración

A diferencia de las tarjetas MDA, en las HGC se tiene contacto con el registro de configuración a través del puerto 3BFH, vía software es posible escribir en este puerto pero no se puede leer de él. Este registro tiene dos bits (0 y 1). El bit 0 especifica si el modo gráfico esta habilitado (1) o deshabilitado (0). El bit 1 se utiliza para indicar si la segunda página está habilitada (0 si no se puede, 1 si es posible).

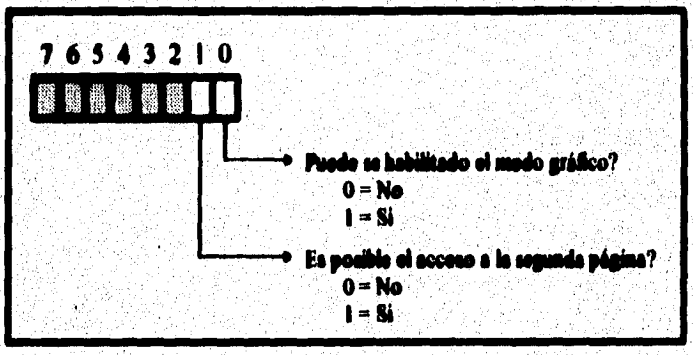


figura 2.5: Registro de configuración

Para evitar conflictos con otras tarjetas de video (tarjetas de color) ambos bits deben ser puestos en cero, así un gráfico no puede ser desplegado y la segunda página no se puede utilizar.

2.5.3 Registro de control

En el registro de control de la tarjeta Hércules a diferencia del adaptador IBM monocromático el bit 0 no se utiliza y no debe ser cambiado a 1 durante el arranque del sistema. El bit 1, determina el modo de video (texto o Gráfico),

La tarjeta Hércules tiene un séptimo bit en el registro de control, el contenido de este bit determina cuál de las dos páginas de video aparecen en la pantalla. Si este bit es 0 aparece la primera página, si es 1 aparecerá la segunda. En cualquier momento es posible leer o escribir de alguna de las páginas.

El registro de control desafortunadamente solo es posible escribirlo ya que cuando se intenta leer retorna el valor de FFH , lo que hace imposible conocer el estado de la tarjeta.

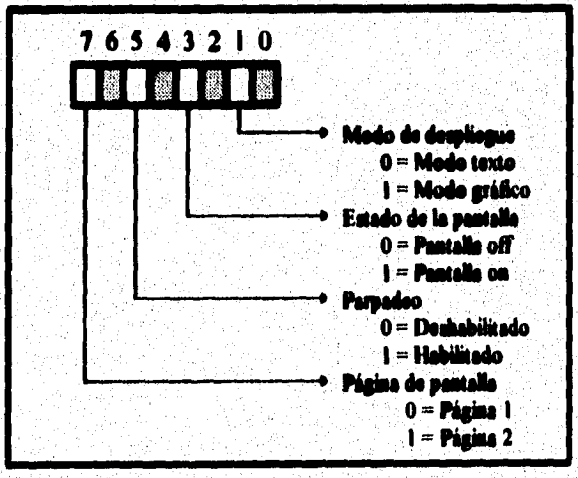


figura 2.6: Registro de control

2.5.4 Inicialización y programación del modo gráfico

No es posible cambiar al modo gráfico usando las funciones del BIOS, no obstante, es factible regresar al modo texto mediante el uso de dichas funciones.

Como se había mencionado, la tarjeta Hércules proporciona una resolución de 720 x 348 pixeles. Cada pixel en la pantalla corresponde a un bit en la RAM de video. Si el bit correspondiente contiene un 1, el punto es visible en la pantalla, caso contrario, este permanece oculto. En la figura 2.7 se muestra la forma en que está distribuida la RAM de video en el modo gráfico.

El patrón de bits está arreglado por líneas, los 32 k de memoria están divididos en cuatro bloques de 8 Kb. El primer bloque contiene el patrón de bits para las líneas divisibles por 4 (0, 4, 8, 12, etc.); el segundo bloque contiene el patrón de bits para las líneas 1, 5, 9, 13, etc.; el tercer bloque contiene el patrón de bits para las líneas 2, 6, 10, 14, etc. y el último bloque contiene las líneas 3, 7, 11, 15, etc.

Cuando el 6845 genera una pantalla, obtiene la información para la línea cero del primer bloque de datos, para la línea uno del segundo bloque etc. La información para la cuarta línea la toma del primer bloque.

Cada línea requiere de 90 bytes ya que una línea de 720 pixeles dividida por 8 bits (un byte) es igual a 90. Los primeros 90 bytes en el primer bloque proporcionan el patrón de bits para la línea cero de la pantalla. El byte cero

dentro de esos 90 bytes, representa las primeras 8 columnas, el siguiente byte corresponde a las columnas de la 8 a la 15, etc.;

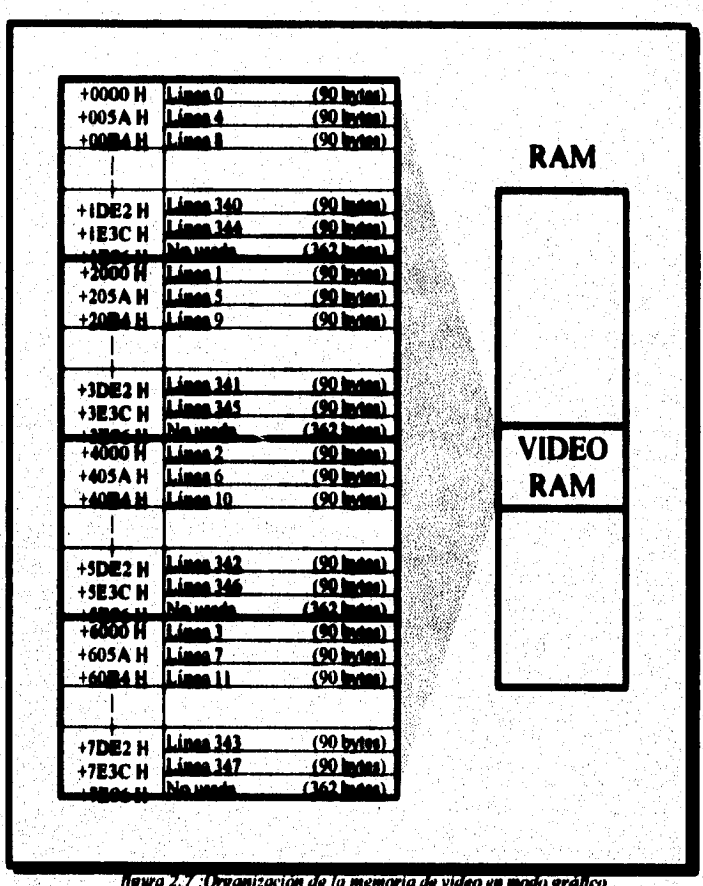


figura 2.7 :Organización de la memoria de video en modo gráfico

Si la pantalla está formada por pixeles ordenados en líneas que van de la 0 a la 347 y columnas de la 0 a la 719, para determinar la dirección del byte correspondiente para un pixel se puede emplear la siguiente fórmula:

Dirección = 2000H*(renglón mod 4)+ 90 * int(renglón/4) + int(columna/8)

y para conocer la posición del bit en esa dirección:

Número del bit = 7 - (columna mod 8)

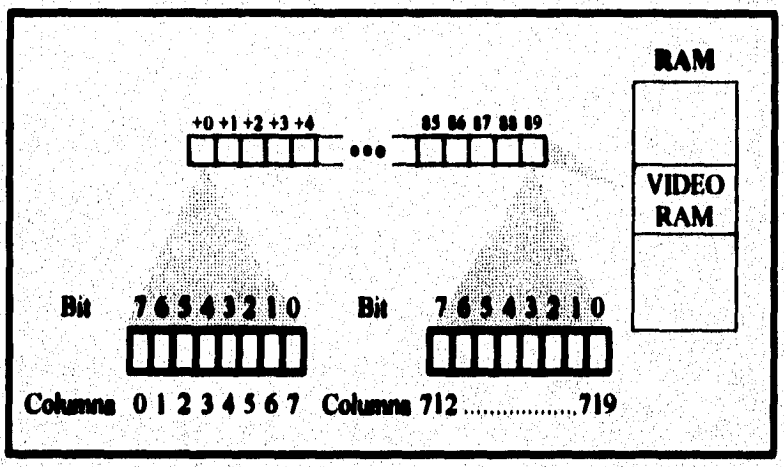


figura 2.8: Relación entre los bytes de la memoria y los pixeles de la pantalla

2.6 CGA

Similar a la tarjeta MDA, la tarjeta CGA fue introducida al mercado en los primeros días de la PC. Esta tarjeta fue el estándar para gráficos durante algunos años hasta que la tarjeta EGA la reemplazó. A diferencia de otras tarjetas de video, en la CGA no es posible acceder a la RAM de video sin la ayuda del controlador del CRT sin provocar "nieve".

La tarjeta CGA reconoce dos diferentes modos de texto cuyas resoluciones son de 40 x 25 y 80 x 25 caracteres. En lo referente a la inicialización mediante las funciones del BIOS, a esos modos se le asignan los códigos 1 para la resolución de 40 x 25 y 3 para la de 80 x 25.

Estos modos incluyen dos variantes con lo que se puede seleccionar el color y fondo del carácter de una paleta de 16 colores. Los modos del BIOS 0 (40 x 25) y 2 (80 x 25) mandan señales monocromáticas al monitor, esto es los colores son convertidos automáticamente a tonos de gris.

Cada página de texto de 80 x 25 requiere 4000 bytes de RAM, 16 Kb permiten un total de cuatro páginas de texto, la primer página empieza en la dirección B800:0000, la segunda en la B800:1000, la tercera en la B8000:2000 y la última en la B8000:3000. El modo de 40 x 25 permite guardar ocho páginas porque en éste modo cada página requiere sólo de 2000 bytes. La primer página comienza en la dirección B800:0000, la segunda en la B800:0800 y la tercera en la B800:1000.

En el byte de atributos, la parte baja indica uno de los 16 colores posibles del carácter. En la parte alta el bit 7 determina si el carácter parpadea y los bits del 4 al 6 indican el color del fondo.

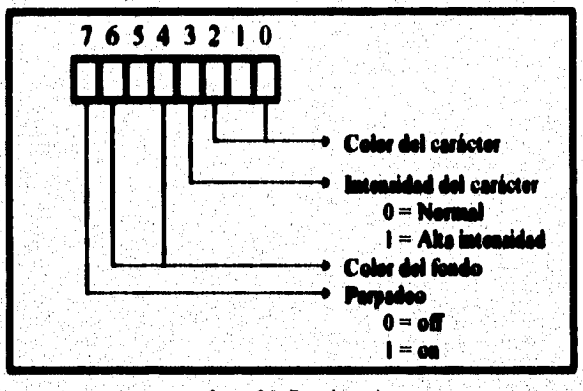


figura 2.9: Byte de atributos

2.6.1 Modo gráfico

El CGA soporta 3 modos gráficos diferentes: de 160 x 100 pixeles con 16 colores, de 320 x 200 pixeles con 4 colores y 640 x 200 pixeles con dos colores. Normalmente sólo son usadas las dos últimas.

2.6.1.1 Resolución de 320 x 200

El CGA usa los 16 Kb de su memoria RAM para desplegar un gráfico con resolución de 320 x 200 pixeles de 4 colores, Por lo que sólo es posible desplegar una pantalla a la vez.

De los 4 colores permitidos, el fondo puede ser seleccionado de los 16 colores disponibles, los otros 3 colores son seleccionados de dos paletas con 3 colores cada una. Como son cuatro los colores permitidos, son necesarios dos bits para representar en la memoria el color del pixel.

La selección de la paleta de colores se puede hacer directamente programando el registro de selección de color o llamando una función especial del BIOS (interrupción 10H función 0BH, subfunción 01H).

La RAM de video para este modo de video está organizada de la siguiente manera:

Las líneas son almacenadas en dos bloques de memoria diferentes. El primer bloque inicia en la dirección B800:0000 y contiene las líneas pares (0, 2, 4, ...); el segundo bloque inicia en la dirección B800:2000 y contiene las líneas impares (1, 3, 5, ...)

Cada línea requiere para almacenar los 320 pixeles 80 bytes, siendo guardados en el primer byte los primeros cuatro puntos de la pantalla, en el segundo byte los siguientes 4 puntos etc. (ver figura 2.10).

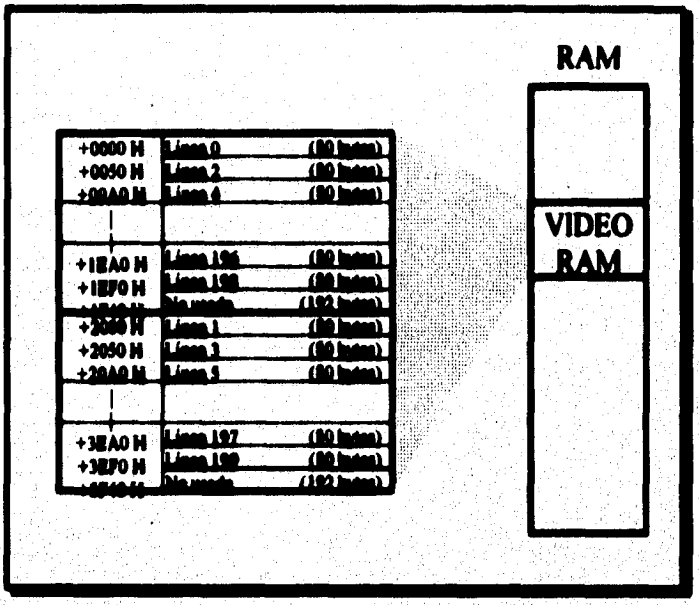


figura 2.10: Organización de la RAM de video en el modo gráfico

Para determinar la dirección en memoria de un punto de la pantalla es posible usar la siguiente fórmula:

Dirección = 2000H*(renglón mod 2) + 80*int(renglón/2) + int (columna/4)

Nota: La dirección es relativa al punto donde inicia la página.

Para determinar la pareja de bits que representan el pixel use la siguiente fórmula:

Número del bit = 6 - 2*(columna mod 4)

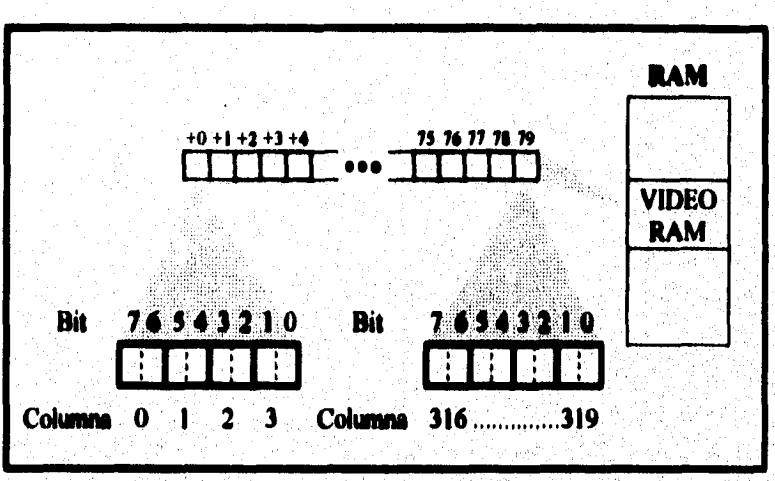


figura 2.11:Representación gráfica de una línea en la resolución de 320 x 200

2.6.1.2 Resolución de 640 x 200

La resolución de 640 x 200 pixeles utiliza el código 6 del BIOS y es similar al modo de 320 x 200. La doble resolución en este modo, hace que sólo sea posible el relacionar un byte por cada pixel, limitando el despliegue de colores en la pantalla a sólo dos. Para conocer la dirección de un pixel es idéntico que en el modo de 320 x 200:

Dirección = 2000H*(renglón mod 2) + 80*int(renglón/2) + int (columna/4)

Número del bit = 7 - (columna mod 4)

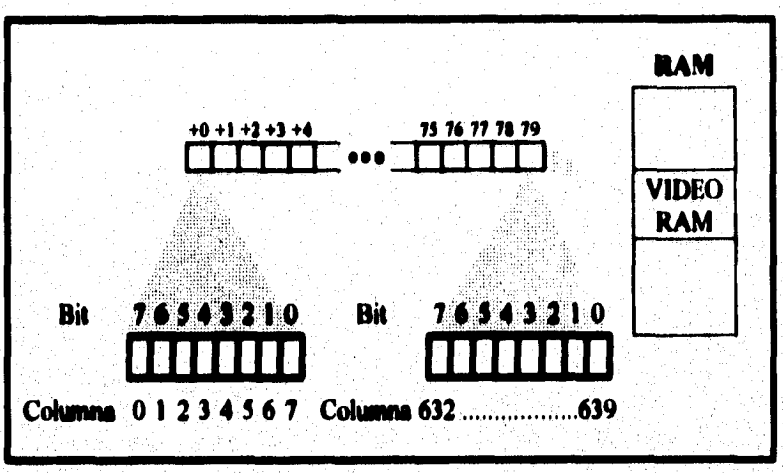


figura 2.12: Representación gráfica de una línea en la resolución de 640 x 200

2.6.2 Registros del CGA

La tarjeta CGA tiene el registro de selección de modo en la dirección 3D8H, el cual es similar al registro de control de la tarjeta MDA. Este registro se puede escribir pero no leer (ver figura 2.13). Los pixeles que se representan con valor de 0 aparecen en este modo como pixeles negros, sin embargo si un 1 es colocado en un bit, el pixel correspondiente aparecerá en la pantalla con el color correspondiente al colocado en el registro de selección de color.

Esta tarjeta cuenta también con un registro de estado que se encuentra en la dirección 3DAH y es de sólo lectura. El bit 0 de éste registro siempre contiene el valor de 1 cuando el 6845 manda la señal de sincronía horizontal a el monitor. Esta señal es transmitida al final de cada línea.

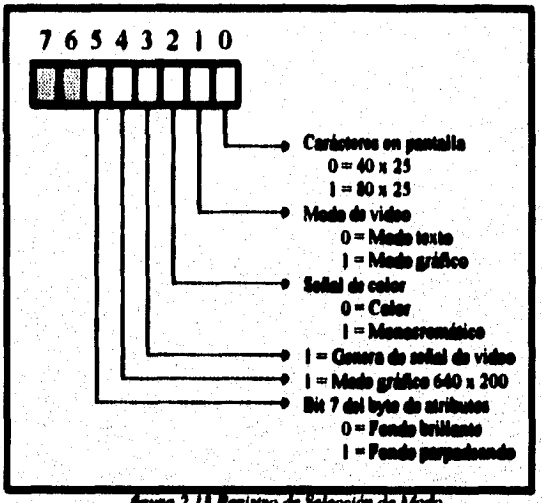


figura 2.13 Registro de Selección de Modo

Registro de Selección de Color

El registro de selección de color está localizado en la dirección 3D9H. Este registro es de sólo escritura (no se puede leer).

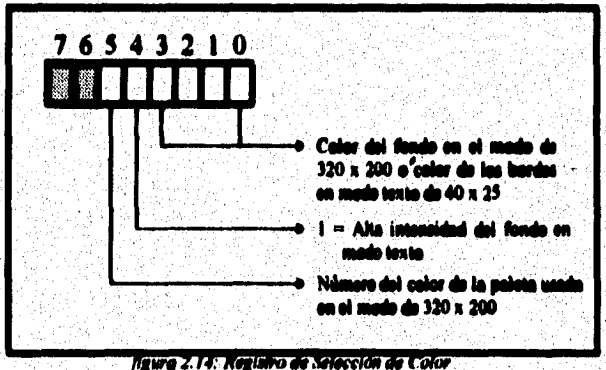


figura 2.14: Registro de Selección de Color

El significado de cada bit en este registro depende del modo de la pantalla. En el modo texto se usan los cuatro bits menos significativos para determinar el color del fondo. En el modo gráfico de 320 x 200 estos 4 bits indican el color de los pixeles codificados como 00(b) (color de fondo).

2.7 EGA y VGA

Las tarjetas EGA y VGA dividen su RAM de video en cuatro grandes áreas llamadas "bitplanes". Estas áreas sirven para diferentes propósitos dependiendo del modo de video utilizado (gráfico o texto). Una tarjeta común con 256 Kb de memoria distribuye 64 KB por bitplane.

2.7.1 MODO TEXTO

Para el modo texto los primeros dos bitplanes (0 y 1) contienen los códigos y atributos de los carácteres. El código ASCII de los carácteres es guardado en el bitplane 0, este se encuentra a partir de la dirección B800H en las localidades pares. Las localidades con un *offset* impar corresponden al bitplane 1, que es el que contiene los atributos.

Las tarjetas EGA y VGA normalmente usan el tercer bitplane para la tabla de carácteres. En el caso de que la tarjeta de video sea de 256 Kb de RAM, éste bitplane dispone de 64 Kb; sin embargo, una tabla de carácteres requiere de sólo 8 Kb, por lo que es posible que almacene 8 diferentes tablas de carácteres. En el caso de la tarjeta EGA sólo se usan 4 tablas, ello significa que quedan 32 Kb sin usar.

En la tabla de carácteres cada byte representa el patrón de bits para una línea de pixeles del carácter, cada byte indica si su correspondiente pixel es encendido o apagado.

Para tener acceso a la tabla de carácteres (bitplane 2) se deben primero de programar los registros de la tarjeta ya que el acceso directo no es

permitido. Es conveniente el uso de los diferentes bitplanes en el modo texto solo si se pretenden utilizar más de un tipo de carácteres en forma simultánea.

2.7.2 MODO GRÁFICO

Cuando se activa el modo gráfico siempre son usados los distintos bitplanes, no obstante que para los distintos modos de video la información es organizada en los bitplanes de diferente forma, la comunicación entre el procesador y la memoria de video utiliza 4 registros de 8 bits llamados registros de 'latch". Cada registro de latch corresponde a uno de los cuatro bitplanes.

Los registros latch no pueden ser directamente accedidos por programas. Si un programa quiere tener acceso a un byte en la RAM de video, entonces los cuatro registros latch reciben éste byte de los cuatro bitplanes correspondientes (utilizando la misma dirección de offset).

Este proceso implica más que una simple lectura o escritura de un registro latch. De un acceso a lectura de la video RAM debe resultar un byte que se manda al procesador y en el acceso a escritura el resultado debe ser un byte que se transfiere del procesador a la RAM de video.

Durante el acceso a lectura sólo un byte a la vez puede ser transferido al procesador. Así, nosotros debemos determinar cual de los cuatro bytes en los registros latch se desea transferir. Algo semejante ocurre en el proceso de escritura de la RAM de video, ya que se debe de indicar cual de los cuatro bytes debe de ser transferido desde el procesador (ver figuras 2.15 y 2.16).

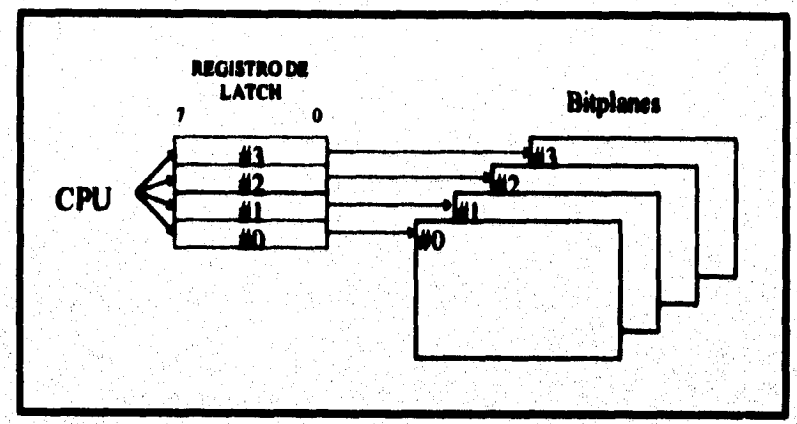


figura 2.15: Acceso a escritura en la RAM por los cuatro registros latch

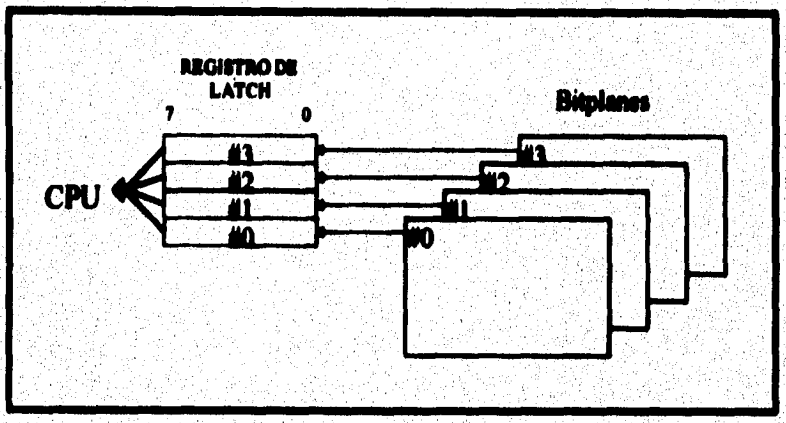


figura 2.16: Acceso a lectura de la RAM por los cuatro registros latch

2.7.2.1 Función del controlador gráfico en la programación de gráficos

El controlador gráfico cuenta con nueve registros que determinan cómo y dónde ocurren los accesos a lectura y escritura en la RAM de video. Estas aplicaciones no son exclusivas de los modos gráficos, también sirven para los modos texto.

Durante la inicialización del modo texto a través de BIOS varios registros son configurados, así que se vuelve innecesario un acceso adicional después de la inicialización. Todas las entradas y salidas de la RAM en modo texto son realizadas a través de los cuatro registros latch.

Los nueve registros del controlador gráfico para la tarjeta EGA son los siguientes:

Registro	Nombre	Valor por Default
00H	Set / Reset	00H
01H	Enable Set / Reset	00H
02H	Color Compare	00H
03H	Function Select	00H
04H	Read Map Select	00H
05H	Mode	00H
06H	Miscellaneous	varios
07H	Color Don't Care	0FH
08H	Bit Mask	FFH

La programación de los registros del controlador gráfico es similar a la de los registros del CRTC de la tarjeta gráfica Hércules.

La dirección del puerto 3CEH, tiene el registro de direcciones, en él se debe de cargar el número del registro que se desea acceder y en el registro de datos (3CFH) se coloca la información que se desea cargar en el registro al que apunta el registro de direcciones.

La escritura de estos dos puertos no es necesario que sea por separado, es posible utilizar el comando de salida "OUT" de 16 bits. El registro AX se manda a la dirección del puerto cargando en el registro AL el número del registro y en el registro AH, el valor que se desea cargar en el registro.

El registro número 5 ó registro de "Mode" es importante para el controlador gráfico durante el acceso de lectura y escritura en la RAM de video, este registro fija uno de los dos modos de lectura y uno de los tres modos de escritura (en el VGA son 4 los modos de escritura). Estos modos determinan la manera en que se van a realizar los accesos de lectura y de escritura. Los otros registros sólo son accesorios donde sus parámetros dependen de lo establecido para los modos de lectura y escritura.

2.7.2.2 Modo 0 de lectura.

El modo 0 de lectura ofrece a los programas la opción de leer un byte de un bitplane específico. Esto es práctico cuando una parte de la RAM debe ser salvada. Para este propósito los cuatro bitplanes son leídos secuencialmente y el área deseada de cada bitplane es cargada y colocada en la memoria principal.

En el modo 0 de lectura, durante un acceso a la memoria los cuatro registros latch son cargados con la información contenida en la dirección

de memoria de cada bitplane; sin embargo, sólo uno de los cuatro bytes obtenidos por los registros latch alcanza el CPU .

El registro "Map Select" determina cuál byte de los diferentes bitplanes llega al CPU. Sólo los dos bits menos significativos de este registro (Map Select) son configurables, estos dos bits dicen el número del registro latch cuyo contenido alcanzará al CPU (recordemos que primer bitplane es el cero).

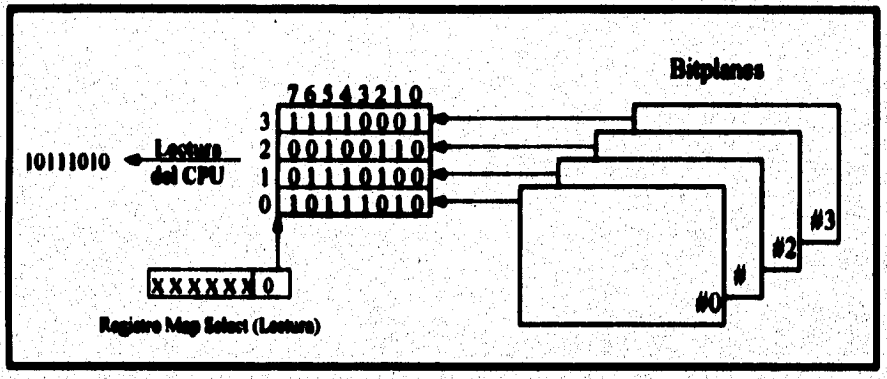


figura 2.17: Acceso a lectura en la RAM de video en modo 0 de lectura

2.7.2.3 Modo 1 de Lectura

Este modo no sólo consiste en mandar el contenido de uno de los registros latch al CPU. En este modo se realizan algunas operaciones lógicas al contenido de los registros latch antes de mandar la información al CPU.

Este modo es responsable de determinar si los bytes de los cuatro registros latch contiene un valor especifico. Este modo puede ser utilizado por

los modos gráficos de las tarjetas EGA y VGA con 16 colores para buscar pixeles con un color determinado.

Después de que los cuatro registros latch son cargados, la información se organiza en ocho grupos, cada uno de estos grupos está formado por 4 bits. Los cuatro bits que ocupan una misma posición en los registros latch forman un grupo, por ejemplo, todos los bits cuya posición es cero forman un grupo, los bits cuya posición es 1 forman otro, etc.

Cada uno de los ocho grupos de cuatro bits es comparado con el valor contenido en el registro de comparación de color, el cual debió de ser cargado antes del acceso a lectura. El resultado de esta comparación determina el byte que es mandado al CPU como resultado de la operación de lectura.

Todos los bits donde la información de su grupo corresponde con el valor en el registro de comparación de color serán colocados en 1 mientras que los otros bits contendrán 0. Así después del acceso a lectura es posible determinar no sólo si uno de los grupos corresponde a un valor de comparación sino también a cual grupo corresponde.

En el proceso de enviar la información de la RAM de video al CPU también interviene el registro de "Color Don't Care". Este registro contiene el valor 00001111b cuando los grupos de cuatro bits se comparan con el registro de color. Cada uno de los cuatro bits menos significativos representan un bitplane, el bit 0 para el primer bitplane, el bit 1 para el segundo , etc.. Solo si uno de estos bits contiene el valor 1, su correspondiente bitplane es incluido en la comparación. Si el valor es 0, es como si el valor de ese bitplane fuera igual

al del correspondiente bit en el registro de comparación de color. Así, si el registro Color Don't Care contiene el valor 0 el valor que se regresa al CPU es el 11111111b (FFH).

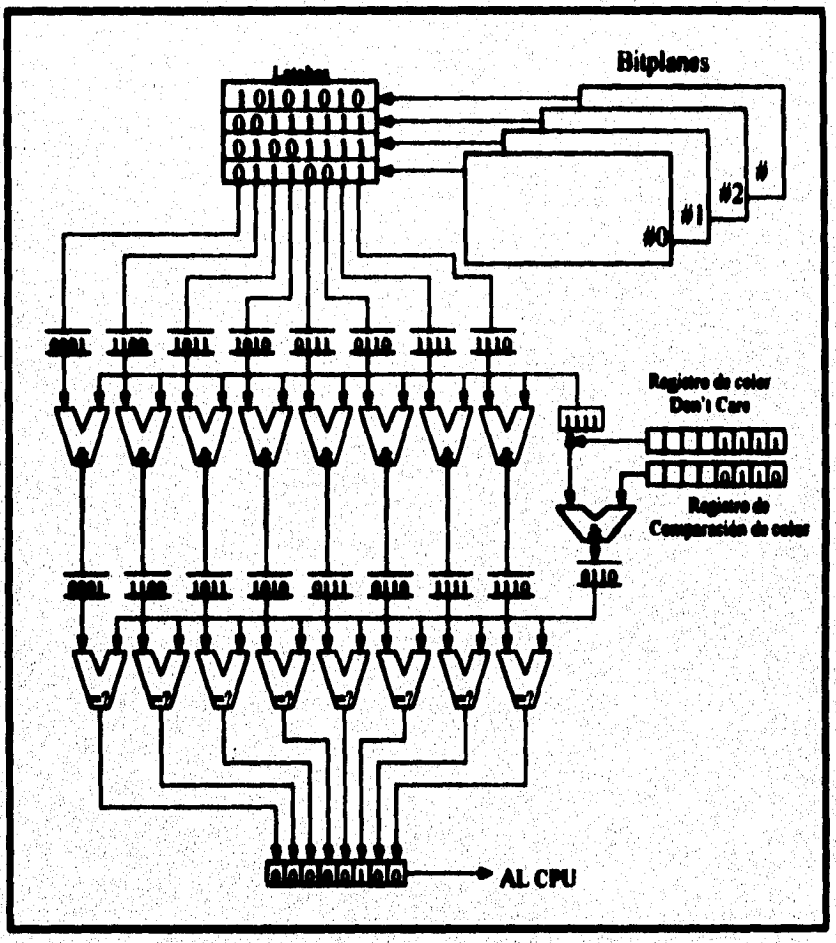


Figura 2.18:Acceso de lectura a laRAM de video mediante el modo 1 de lectura

2.7.2.4 Modo 0 escritura

En el acceso a escritura en modo 0, varias operaciones dependen del contenido de algunos registros. Por ejemplo, el contenido del registro "Bit Mask" decide si el contenido de algún bit de los cuatro latches irá a sus bitplanes sin cambios o serán procesados.

Cada bit del registro Bit Mask corresponde a los bits de cada uno de los cuatro bitplanes. Si un bit en el registro Bit Mask contiene el valor 0, sus correspondientes bits de los cuatro registros latch son transferidos sin cambios a sus bitplanes. Si el bit contiene el valor 1 ocurre una operación la cual depende del contenido del registro "Function Select", el cual pueden reemplazar o manipular los bits con la ayuda de las operaciones lógicas AND, OR y XOR.

El contenido del registro "Enable Set/Reset" determina cuáles bits tomarán parte en la operación. Si los cuatro bits menos significativos del registro Set/Reset contienen el valor 1 en la parte baja del byte proveniente del CPU, cada bit es utilizado en la operación con los registros latch.

Todos los bits del registro latch 0 que serán procesados son ligados con el bit 0 del registro Set/Reset a través del operador de selección. (El byte proveniente del CPU es cargado en el registro Set/Reset). De la misma forma, todos los bits manipulados de los registros latch 1, 2 y 3 son ligados con los bits 1, 2 y 3 del registro Set/Reset.

Si solo el grupo de la posición debe ser modificado y los otros grupos no deben sufrir cambios el valor 0000 0100b (04H) se debe cargar en el registro Bit Mask. Después el valor 0 es escrito en el registro Function Select porque los bits deben ser manipulados y reemplazados por una nueva combinación de bits. Después de ésto, el color del grupo de 4 bits en la posición 2 es cargado en el registro Set/Reset.

Para quitar el color de este registro cuando se escribe en la RAM de video, escriba el valor 1111b (0FH) en el registro Enable Set/Reset en el último acceso al registro del controlador gráfico (ver figura 2.19).

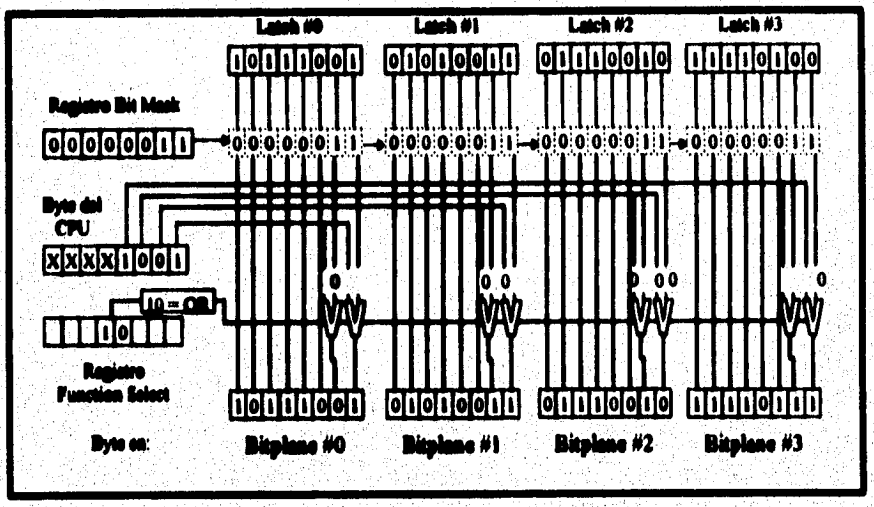


figura 2.19: Modo 0 de escritura cuando el Registro Enable Set/Reset contiene el valor 0FH

Este proceso es diferente cuando el registro Enable Set/Reset contiene el valor 0, en este caso todos los bits de los latch que se manipulan de los cuatro latch son ligados al byte del CPU. Otra vez la operación depende del contenido del registro Function Select (ver figura 2.20).

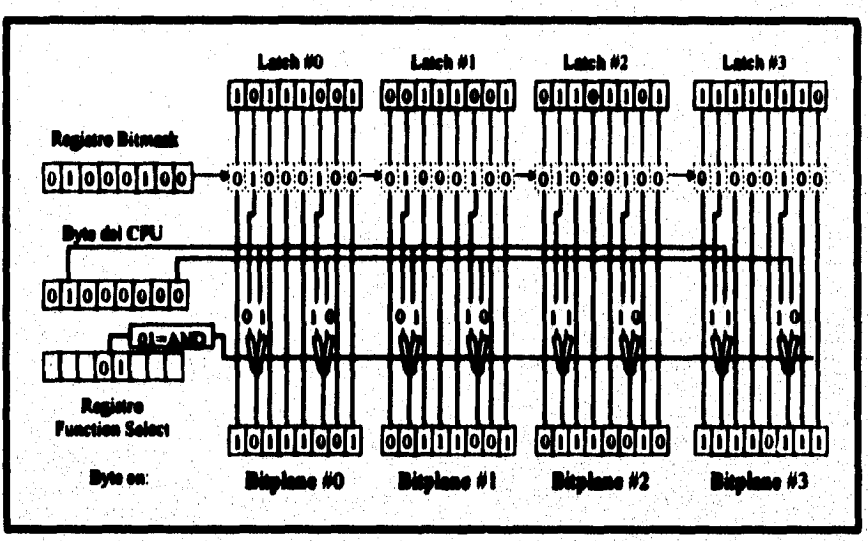


figura 2.20: Modo 0 de escritura cuando el Registro Enable Set/Reset contiene el valor 0011

2.7.2.5 Modo 1 escritura

Este modo es muy simple ya que ni el byte del CPU, ni contenido de varios registros del controlador gráfico son muy importantes. En este modo el contenido de los cuatro registros latch es escrito en la dirección determinada por el offset (dentro de los cuatro bitplanes) sin que ella sea modificada.

Este modo es útil cuando se desea copiar una área específica de la RAM de video en otro lugar, ya que nos permite copiar 4 bytes a la vez, con lo que se ahorra tiempo.

2.7.2.6 *Modo 2 escritura*

El modo 2 de escritura es una combinación de los diferentes modos, a semejanza del modo 0, el registro Bit Mask decide cuales bits serán mandados sin cambio a los registros de latch y cuáles serán modificados. El registro Function Select determina la manera de cómo los bits serán modificados. Sin importar el contenido del registro Enable Set/Reset los 4 bits menos significativos del byte del CPU son ligados con los registros latch. El bit 0 del CPU es ligado con todos los bits que serán modificados en el registro latch 0. Lo mismo se aplica para los bits 1,2 y 3 del CPU los cuales son individualmente ligados con todos los bits por modificar en los registros latch 1, 2 y 3.

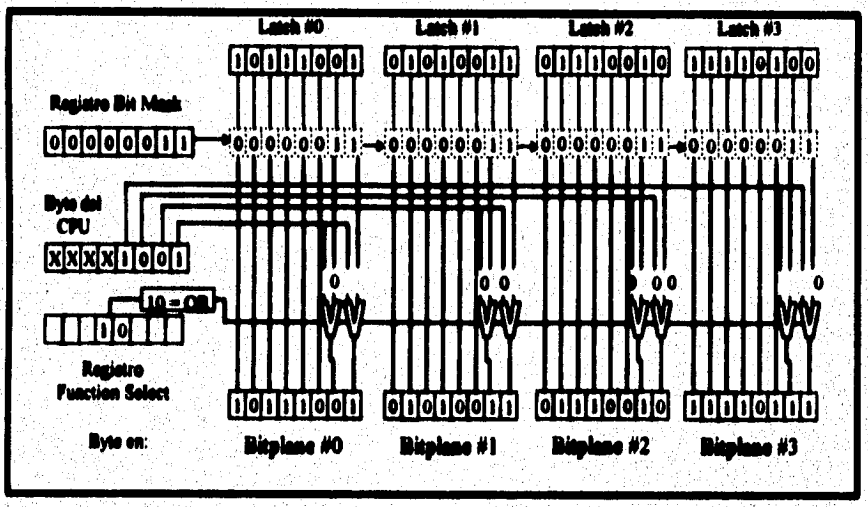


figura 2.21: Acceso a escritura en la RAM dentro del modo 2 de escritura

2.7.2.7 Modo 3 escritura`

En adición a los tres modos de las tarjetas EGA, los desarrolladores de las tarjetas VGA le sumaron a éstas un cuarto modo. Este modo es seleccionado de la misma manera que los otros tres.

En este modo, los cuatro bits menos significativos del registro Set/Reset deben ser unidos con los bits de los cuatro registros latch. A diferencia del modo 0 de escritura, el contenido del registro Enable Set/Reset es incluido en esta operación. Sin embargo, el tipo de unión esta determinada por el registro Function Select.

En este modo una combinación lógica del byte del CPU con el contenido del registro Bit Mask determina cuáles bits de los registros latch son escritos en su bitplane sin cambio y cuales tomarán la combinación del registro Set/Reset y los bits del latch (ver figura 2.22).

El registro Map Mask, con este registro individualmente los bitplanes pueden ser cerrados para evitar una lectura o escritura en ellos. Este registro es útil cuando se desea manipular el contenido de un bitplane específico. Cada bit del registro corresponde a un bitplane, el contenido debe ser 1 para permitir el acceso a su correspondiente bitplane.

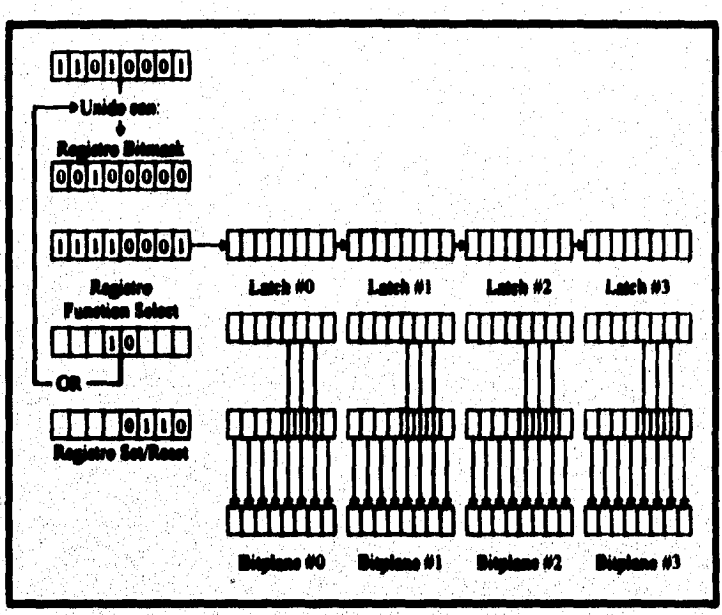


figura 2.22: Acceso a escritura en la RAM dentro del modo 3 de escritura

2.7.3 El modo gráfico de 16 colores

El rasgo más importante de una tarjeta EGA y VGA es el modo gráfico de 16 colores. En éste modo es posible trabajar con resoluciones de 320 x 200 a 640 x 350 pixeles.

Cada byte de la RAM de video contiene la información de color para ocho pixeles consecutivos en la misma línea de la pantalla. Esto significa que cada pixel se representa por un bit. Para el modo de 16 colores es insuficiente este espacio porque cada pixel necesita cuatro bits para identificar el color. Estos bits son obtenidos utilizando un bit de cada uno de los cuatro bitplanes.

Los primeros ocho pixeles de la esquina superior izquierda de la pantalla son representados por los cuatro bytes encontrados en el offset 0000H en cada uno de los cuatro bitplanes. Los cuatro bits requeridos para codificar la información del color son obtenidos por la combinación de bits en la misma posición del bit en el primer byte de cada bitplane. Este procedimiento crea ocho grupos de cuatro pixeles, de los cuales cada uno determina el color de un pixel.

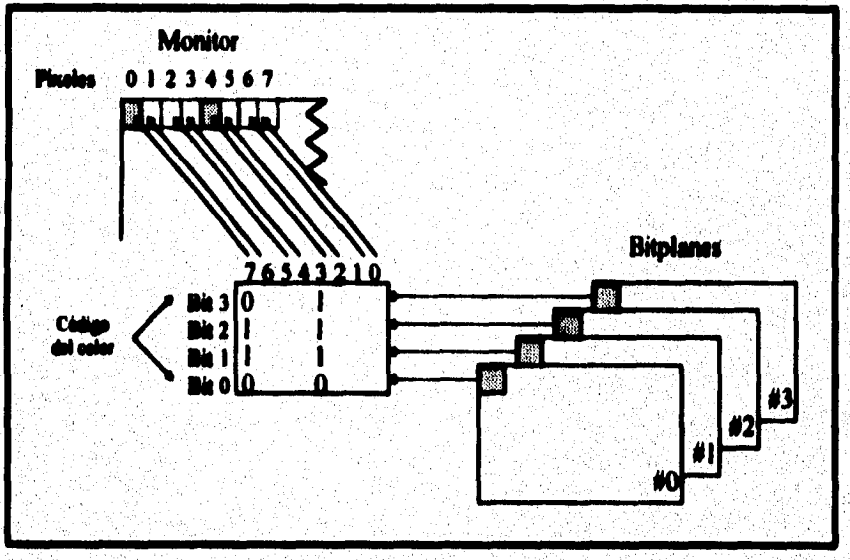


figura 2.23: Estructura de la RAM de video en el modo gráfico de 16 colores

El bit para el bitplane 0 viene de el bit 0 del código de color. El bit del bitplane 1 es el bit 1 del código de color y así para los bits 2 y 3. Los códigos de colores son interpretados como en las tarjetas CGA: negro (0), azul (1), verde (2), etc.

La programación en el modo gráfico de 16 colores es muy simple. Comenzando con un offset de 0000H en la RAM de video, cada línea de

pixeles es representada por un cierto número de bytes consecutivos. Para el modo gráfico con resolución de 640 pixeles es de 80 bytes.

Para calcular el offset de direccionamieto del byte que contiene algún pixel se utiliza la siguiente expresión:

Offset = renglón * (resolución_horizontal / 8) + int (columna/8)

El byte encontrado en esta dirección contiene información para ocho pixeles consecutivos. Para encontrar el bit específico que contiene la información de color para el pixel deseado se utiliza la siguiente fórmula:

Bit = 7 - (X mod 8)

2.7.4 El modo VGA de 256 colores.

Una de las más grandes ventajas de la tarjeta VGA respecto de las EGA es la capacidad de desplegar 256 colores diferentes en forma simultánea en la pantalla. Este conjunto de 256 colores es seleccionado de una paleta de 262,000 colores, pero el modo de 256 colores tiene una resolución de 320 x 200 que es mucho menor que la resolución de 640 x 480 pixeles del modo de 16 colores.

Para habilitar este modo se puede utilizar la interrupción de video con la función 00H del BIOS . El número para este modo de video es el 13H, siendo ésto suficiente para colocar el modo gráfico de 256 colores.

En este modo, de manera semejante al modo texto, cada pixel es representado por un byte en la RAM de video (ver figura 2.24). Este byte

representa el color como un valor entre 0 y 255. Este valor de color actúa directamente como un índice en la tabla de color del DAC.

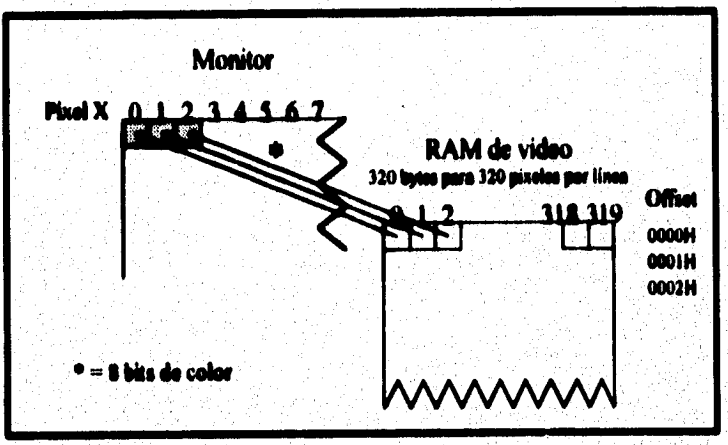


figura 2.24: Estructura de la RAM de video para el modo gráfico de 320 x 200 pixeles y 256 colores

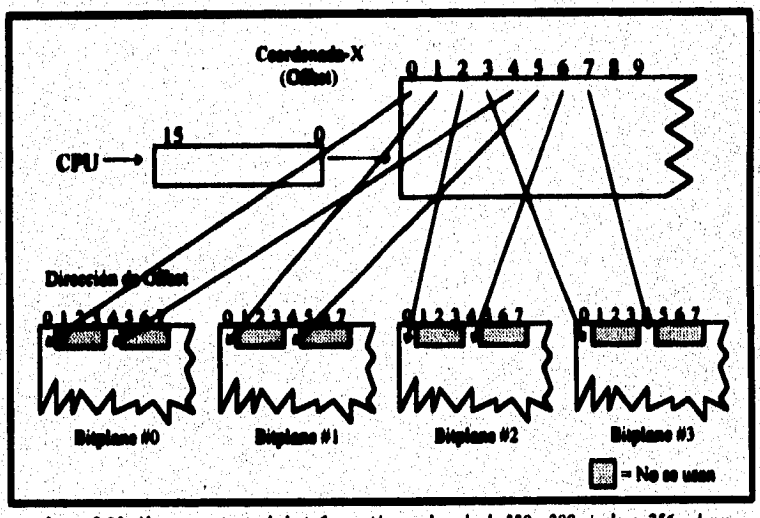


figura 2.25: Almacenamiento de la información en el modo de 320 x 200 pixeles y 256 colores

El cálculo de la posición en memoria para un pixel está dado por:

Offset=renglón * 320 + columna

Como cada pixel es representado por un byte y se tiene una resolución de 320 x 200 solo se requieren 64 Kb de memoria para almacenar una página.

2.7.4.1 Cuatro páginas gráficas en una.

Para poder almacenar cuatro páginas de video en el modo gráfico de 256 colores, es necesario utilizar algunos trucos. Esto significa que los pixeles son manejados en cuatro bitplanes, utilizando una extensión del modo odd/even que utiliza la tarjeta VGA en el modo texto para mover información de la RAM de video en los Bitplanes 0 y 1.

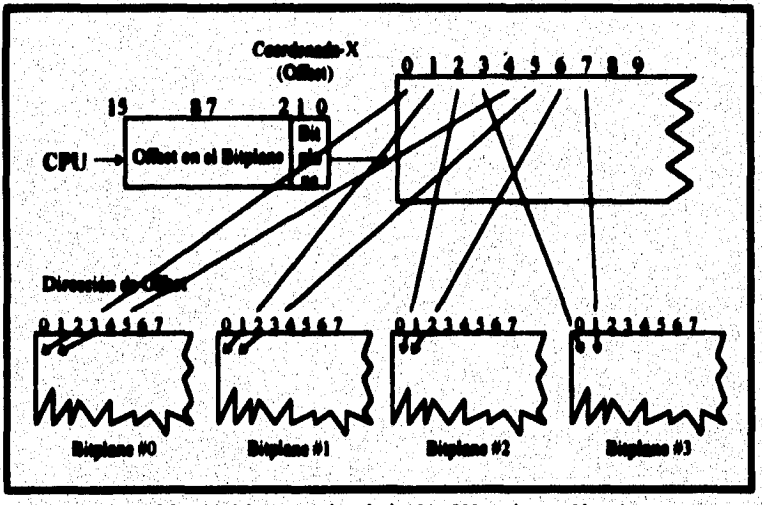


figura 2.26:: Modificaciones al modo de 320 x 200 pixeles con 256 colores vistas por el controlador del CRT

En este modo los dos bits menos significativos de un offset específico determinan el número del bitplane al cual se manda el valor. Estos dos bits son

puestos en cero y usan la dirección de offset para acceder al bitplane seleccionado.

2.7.4.2 Dos páginas de video de 320 x 400 pixeles

Aunque los 256 colores del modo de 320 x 200 son impresionantes, la resolución de este modo no se compara con otros modos gráficos del VGA. Sin embargo, los numerosos colores del modo 256, dan la impresión de ser una alta resolución.

La tarjeta puede ser programada para desplegar 400 líneas por pantalla en lugar de 200. En el modo de 320 x 200 pixeles la tarjeta VGA tiene 400 líneas horizontales en la pantalla, sólo que las líneas son unidas en pares para aparentar que son solo 200. Así que si se desea tener 400 líneas en lugar de tener el doble debemos direccionar cada pixel por separado.

Los registros del CRT responsables de los tiempos de barrido horizontal y vertical no deben de ser reprogramados para cambiar de 320 x 200 al modo de 320 x 400.

Los 128,000 pixeles de una pantalla no pueden ser direccionados por el camino usual, es por ello que se debe de usar una técnica de direccionamiento similar a la que se utiliza en el modo de cuatro páginas de 320 x 200.

2.7.5 Libre selección de los colores.

Una importante diferencia entre las tarjetas EGA/VGA y sus predecesoras es la habilidad para trabajar con una cantidad superior a los 16 colore básicos. La tarjeta EGA tiene 64 colores para escoger su paleta de color, mientras que las tarjetas VGA exceden de los 262,000 colores. No todos los colores pueden ser desplegados en la pantalla al mismo tiempo.

Los 16 registros de la paleta de color son parte del controlador del CRT, estos registros son importantes para el despliegue de colores en la pantalla ya sea que se trate del modo texto o gráfico de 16 ó menos colores.

Cuando el controlador del CRT construye una pantalla, éste recibe la información para cada pixel en forma de un valor entre 0 a 15. Este valor es utilizado como un índice en los registros de la paleta. La información del color es tomada del índice del registro de la paleta y es mandada directamente al monitor por las tarjetas EGA.

Las tarjetas EGA y VGA tienen una fuerte relación entre sus códigos de color y los colores que aparecen en la pantalla, característica que facilita al programador la elección de los colores que aparecen en la pantalla. La selección de los colores ocurre sin modificar el contenido de la RAM de video.

Los registros de la paleta de color pueden ser controlados mediante el uso de la función 12H del BIOS. Después de haber inializado el modo de video con la función 00H, la función número 12H es colocada en el registro AH y la subfunción número 31H en el registro BL. El registro AL es usado para

determinar si los registros de la paleta son automáticamente inicializados o no. Si contiene el valor 1, los registros de la paleta no son inicializados con las subsecuentes llamadas de la función 00H. Con este elemento activo, se puede, mediante la programación de los registros, usar la capacidad de color expandido de las tarjetas EGA y VGA.

2.7.5.1 La tabla de colores del DAC

El contenido de los registros de la paleta pueden pasar directamente de una tarjeta EGA a el monitor a través de un cable de seis hilos para el color mas las señales de control. Sin embargo, ésto es imposible para las tarjetas VGA ya que éstas producen una señal analógica que es mandada al monitor.

Así el contenido de los 16 registros de la paleta es sumada a la tabla de colores del DAC antes de que la información sea mandada a el monitor. DAC son las siglas de "Digital to Analog Converter"(convertidor Digital - Analógico).

El DAC contiene 256 registros, cada uno almacena la información para un color, éste es seleccionado de un total de 262,000 colores que ofrece la paleta del VGA. Este número de colores es resultado de los 18 bits de código de color $2^{18} = 262144$.

Cada entrada en el DAC consta de tres entradas de 6 bits de color, una para cada componente de los colores rojo, verde y azul. Para seleccionar un registro en la tabla de colores del DAC el controlador de video interpreta el contenido de los registros de la paleta como un índice para la tabla de colores del DAC en lugar de considerarlo como un valor de color.

El registro de control de Modo del controlador de video juega un papel importante en ese proceso. Si el registro contiene el valor de 0, entonces el índice para la tabla de colores del DAC es formado tomando en cuenta el contenido de los bit 0 al 5 del correspondiente registro de la paleta y de los bits 2 y 3 del registro de Selección de Color. Esto significa que la tabla de colores del DAC es dividida en 4 grupos de 64 registros y que los bits 2 y 3 del registro de Selección de Color determinan cual de los cuatro grupos es el que está activo.

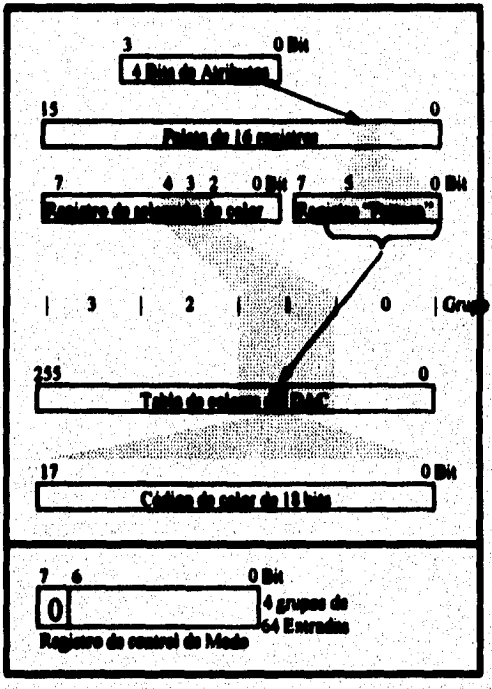


figura 2.27: Creación de colores con la tarjeta VGA cuando es cero el registro Mode Control

Si el bit 7 del registro de control de modo contiene el valor de 1, en este caso la tabla de colores del DAC es dividida en 16 grupos de 16 registros. El

índice para la tabla es creado con los bits del 0 al 3 del registro de la paleta y de los bits 0 al 3 del registro de selección de color. Otra vez el valor del registro de la paleta es un índice y el valor en el registro de selección de color determina el grupo que permanecerá activo en la tabla del DAC.

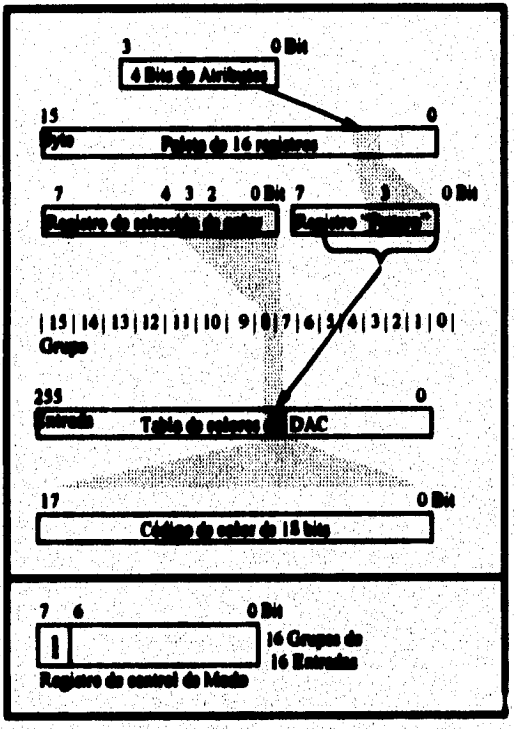


figura 2.28: Creación de colores con la tarjeta VGA cuando es uno el registro Mode Control

Este tipo de codificación realiza rápidos y continuos cambios de color para grupos enteros de carácteres o pixeles en la pantalla.

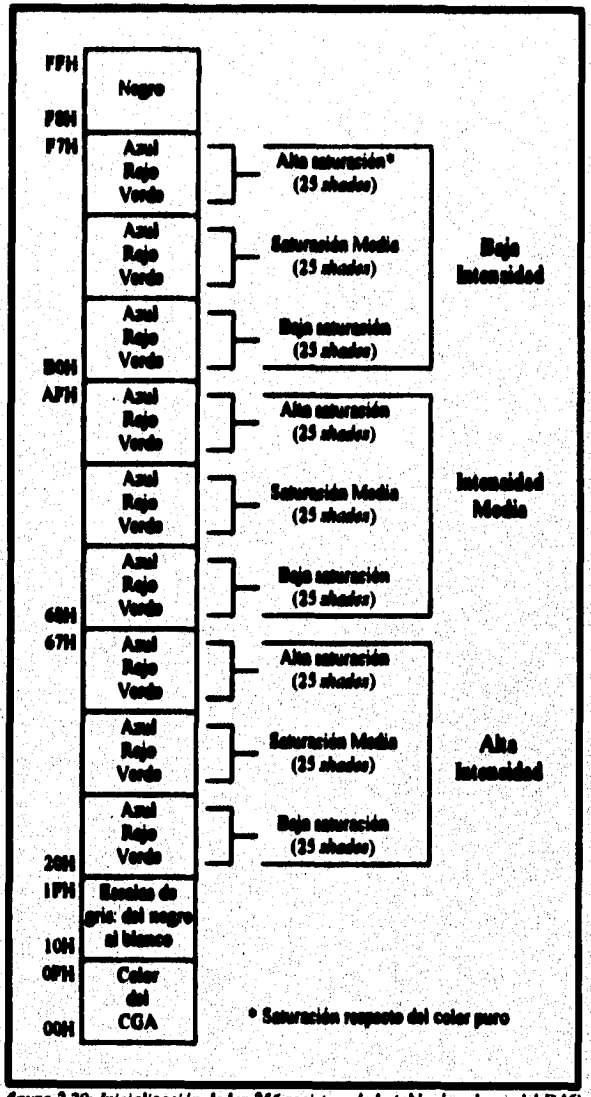


figura 2.29: Inicialización de los 256 registros de la tabla de colores del DAC

Para emular los colores del CGA, la tarjeta VGA inicializa los 16 registros de la paleta apuntando a los primeros 16 registros de la tabla de colores del DAC. Estos registros en la tabla de colores del DAC son cargados con los estándares de información de color para la tarjeta CGA ("0" para el negro hasta "15" para el blanco). Los otros registros de la tabla de colores del DAC no son modificados cuando el modo texto es inicializado con la función 00H del BIOS. Generalmente para los modos gráficos los 256 registros de la tabla de colores del DAC son inicializados (ver figura 2.29).

3 El bus de la PC

Un bus se puede definir como "una colección de alambres que conducen información entre componentes del sistema"[23], otra acepción dice que es "el conjunto de líneas compartidas por los elementos del sistema, las cuales son usadas para comunicarse entre esos elementos"[24].

Los elementos que constituyen las computadoras personales utilizan para comunicarse el concepto de bus; sin embargo, en los diversos modelos de computadoras existen diferentes diseños de bus, por lo que en el presente trabajo sólo nos referiremos al bus ISA (Industry Standard Architecture) debido a que es "el más popular y común de los diseños de bus de la PC"[25].

En el bus ISA sus líneas se unen a los conectores de tarjetas de expansión ('slots'), dichos slots tienen la distribución que se muestra en la figura 3.1.

3.1 Señales del BUS ISA

A0 - A19. (Address Line 0-19) Las señales de A0 a la A19 son de salida, se usan para direccionar a la memoria y los periféricos. Con estas 20 líneas es posible direccionar 1 Mb de memoria. A0 es el bit menos significativo y A19 el más significativo. Las señales son controladas por el CPU durante los ciclos de lectura y escritura de memorias y puertos o por el controlador del DMA durante un ciclo DMA.

[23] Robert J. Baron y Lee Higbie Computer Architecture Addison-Wesley USA 1992 pp480.

[24] A J Van de Goor Computer Architecture and Design Addison-Wesley USA 1990.

[25] Peter Norton Todo la PC Ed Prentice Hall 5a Edición México 1994 pp36.

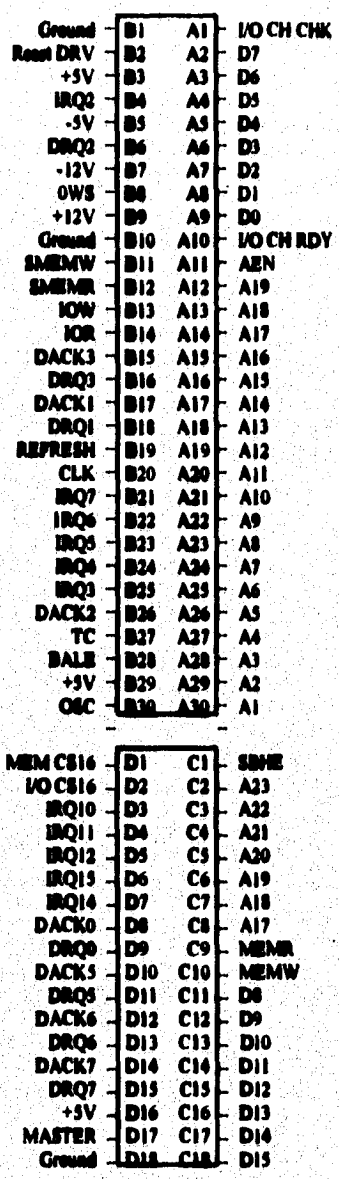


figura3.1: El bus ISA

A17-A23. (Address Line 17-23) Son señales de salida usadas para proporcionar a la memoria las direcciones del presente ciclo del bus. Estas señales a diferencia de las señales A0- A19 no son válidas durante todo el ciclo de bus. Estas señales son generadas de esta forma para reducir el tiempo de direccionamiento cuando son usadas para decodificar un bloque de memoria.

D0 a D7. (Data Line 0-7) Son señales bidireccionales que se utilizan para transmitir datos entre el microprocesador, memorias y puertos.

D8- D15. (Data Line 8-15) Son señales bidireccionales que transmiten el byte superior en la transferencia de datos de 16 bits. Estas señales son semejantes a D0 - D7, pero los periféricos y adaptadores deben indicar al sistema si son capaces de soportar transferencias de 16 bits, para lo cual utilizan las señales MEM CS16 e I/O CS16. La primera para dispositivos de memoria y la segunda para dispositivos de entrada/ salida.

IOR (I/O Read). La señal IOR es una señal de salida manejada por el controlador del bus y es activada con nivel bajo. Es usada para indicar a los puertos de entrada/salida que se trata de un ciclo de lectura a puertos y la dirección del bus es un puerto.

IOW. (I/O Write) Es una señal de salida que se activa baja y es manejada por el controlador del bus. Indica que la dirección del bus es la dirección de un puerto de entrada/salida, y que el dato del bus será escrito en el puerto indicado.

SMEMR. (Memory Read) Señal de salida que es activada con nivel bajo, la cual indica que la localidad de memoria que en ese momento está direccionada por el canal, debe copiar su contenido al bus de datos.

SMEMW. (Memory Write) Es una señal de salida que se activa con nivel bajo, es usada para escribir datos del bus del sistema a la memoria. La señal indica que el bus de direcciones contiene la dirección de la localidad de memoria a la cual se va a transferir la información contenida en el bus de datos.

MEMR. (Memory Read) La señal MEMR es idéntica a la señal SMEMR en el slot del bus de las XT, excepto que esta es activada en todos los ciclos de lectura de la memoria incluyendo aquellos arriba de 1Mb. Si SMEMR está inactiva y MEMR activa el ciclo de lectura de memoria es para direcciones arriba de 1Mb . Esta señal se activa baja.

MEMW. (Memory Write) Esta señal es idéntica a la señal SMEMW del bus de las XT excepto que esta es activada en todos los ciclos de escritura a memoria, incluyendo aquellos arriba del rango de 1 Mb. Si SMEMW está inactiva y MEMW activa, el ciclo de escritura a memoria es para direcciones superiores a 1 Mb. Esta señal es activada baja e indica que el microprocesador ha colocado en el bus de datos información que desea almacenar en la memoria. Los datos son escritos con el flanco de subida de esta señal .

MEM CS16 (Memory Chip Salect 16) . Esta señal se activa con nivel bajo, es usada para indicar que el periférico en el presente ciclo de bus puede soportar un tiempo de espera para su transferencia de bus de 16 bits.

DRQ. 1-DRQ 3 (DMA Request 1-3). Las líneas DRQ1 a la DRQ3 son entradas que se activan con nivel alto. Los dispositivos periféricos usan estas líneas para solicitar un acceso directo a la memoria. Las líneas llegan al controlador del DMA, donde son comparadas con otras peticiones antes de que el ciclo de DMA sea cedido.

DACK 0-3 (Direct Memory Access Acknowledge 0 -3). Son señales de salida que se activan con nivel bajo proporcionadas por el controlador de DMA que indican que la petición de un acceso directo a memoria es cedida y que el ciclo está por iniciarse.

DRQ 5- DRQ7. (DMA Request 5-7) Estas líneas son similares a las de DRQ1 - DRQ3, también son usadas para solicitar un acceso directo a memoria, la diferencia es que los ciclos del DMA en estos canales, es para transferencias de 16 bit en el bus.

DACK 5 - DACK 7. (Direct Memory Access Acknowledge 5-7) Estas señales son similares a las DACK 0- DACK 3.
Estas señales conceden las peticiones de un acceso directo a memoria para los niveles del 5 al 7.

IRQ 2-7. (Interrupt Request 2-7) Estas 6 señales son de entrada, se usan para generar peticiones de interrupción al microprocesador y van directamente al controlador de interrupciones. El programa de la ROM BIOS, debe inicializar el controlador de interrupciones para que IRQ2 tenga la mayor prioridad e IRQ7 sea la de menor. Para solicitar una interrupción se lleva la

señal de un nivel bajo a uno alto y se mantiene en alto hasta que es aceptada por el CPU.

IRQ 10,11,12,14,15. (Interrupt Request 10-15) Estas señales son activadas alto, son casi idénticas a las señales IRQ2- IRQ7 y son usadas para solicitar interrupciones.

I/O CH CHK. (I/O Channel Check) Señal de entrada, activada baja, que se usa para reportar errores en los conectores de las tarjetas. Cuando esta señal se activa se genera una interrupción no mascarable en el microprocesador.

I/O CH RDY. (I/O channel Ready) Esta señal se usa para extender los ciclos de reloj usados en un ciclo del bus. Cuando una memoria o puerto no es lo suficientemente rápido para realizar una lectura o escritura en 4 ciclos de reloj, deberá mantener esta señal baja para tener tiempos adicionales y terminar su operación.

RESET DRV. Esta es una señal de salida, se mantiene activa durante las secuencias de encendido del sistema. Esta señal permanecerá activa hasta que todos los niveles hayan alcanzado sus rangos de operación. Esta señal es generalmente usada para proporcionar una señal de inicialización a la interfaz lógica. La señal es activada alta y está sincronizada con el flanco de bajada del reloj.

TC. (Terminal Count) Es una señal de salida que se activa con nivel alto y es manejada por el controlador del DMA, para indicar que uno de los canales DMA ha alcanzado el número de ciclos de transferencia preprogramados.

BALE (ALE Address Latch Enable). Es una señal de salida y es manejada por el controlador del bus, es utilizada para indicar que la dirección que se encuentra en el bus es válida para comenzar un ciclo. Esta señal se activa poco antes de que la dirección esté en el bus y se desactiva poco después de que se ha fijado la dirección.

CLK. (Clock). Es una señal de salida que refleja la velocidad del reloj del procesador, la cual puede ser de un medio, un tercio o un cuarto de la velocidad de éste, incluso en algunos diseños es independiente del reloj y se puede fijar a una frecuencia determinada

0WS (0 Wait State). La señal es activada con nivel bajo, es usada para forzar el presente ciclo de bus de 16 bits a proceder sin tiempos de espera. Si el ciclo del bus es de 8 bits sólo son añadidos dos tiempos de espera.

I/O CS16. (I/O Chip Select 16 Bits) Esta señal es de entrada y se activa con nivel bajo, es usada para indicar que el periférico (dispositivo conectado), puede soportar un tiempo de espera en la entrada/salida actual (ésta debe ser de 16 bits).

SBHE. (System Bus High Enable) . Es una señal de salida del sistema ésta indica que "The bus - attached device " y el presente ciclo de bus espera una transferencia de datos en las líneas de la 8 a la 15. Esta señal y la menos significativa de las de direccionamiento (A0) pueden ser usadas para decodificar el tipo de ciclo de bus.

SBHE	Valor de A0	Función
1	0	Transferencia de 16 bits.
1	1	Transferencia del byte superior.
0	0	Transferencia de la parte baja de los bits.
0	1	no válida

MASTER. Es una señal de entrada que se activa con nivel bajo, es usada con las señales DRQ5 - DRQ7 para permitir a un bus maestro tomar el control del bus del sistema. Cuando una señal DACK es regresada por la activación de las señales DRQ5- DRQ7, la señal MASTER es activada, con lo cual se cancelan los canales de operación del DMA, y se coloca en alta impedancia el bus de direcciones, datos y señales de control. Esto permite al adaptador tomar el control del sistema. El nuevo dueño del bus debe obedecer todos los requerimientos de tiempo del bus, y regresar el control al procesador de la PC y canales DMA después de 15 micro segundos, para evitar perdidas en la memoria debido a falta de ciclos de refresco.

VOLTAJES Y TIERRA. Existen también líneas que son de voltaje constante.

Voltaje de + 5 V dc $\pm$ 5%

Voltaje de +12V dc $\pm$ 5%

Voltaje de - 5 V dc $\pm$ 10%

Voltaje de - 12 V dc $\pm$ 10%

Voltaje de 0 Vdc (GND)

Las señal anteriormente descritas trabajan en conjunto para realizar algunos de los procesos mas importantes de las PC. Por este motivo en el Apendice IV se muestran los diagramas de tiempos de las señales que intervienen en dichos procesos.

4 Diseño de un Sistema de Pruebas para Equipos de Video

En la fabricación del sistema de pruebas para equipos de video compatibles con PC de IBM, se llevaron acabo dos procesos principalmente: El Electrónico y La Programación.

4.1 Diseño Electrónico.

El diseño electrónico consistió en la elaboración de un circuito que se conecta con el bus de una PC, en dicho circuito se insertan 3 tarjetas de video, las cuales generan las señales necesarias para los diferentes tipos de monitor. Este circuito se encarga de informar a la PC qué tipo de monitor se desea conectar y en consecuencia determina cual tarjeta de video permanecerá activa.

De acuerdo a lo anterior, el circuito electrónico elaborado se puede dividir en dos partes funcionales :

1.- Un selector de tarjetas de video.

2.- Un puerto de entrada (port*) el cual sirve para que la PC reciba datos.

4.1.1 El selector de Tarjetas de Video

El selector de la tarjeta de video consiste en un controlador que maneja un conjunto de interruptores, de tal forma que cierra las líneas entre el bus de la PC y la tarjeta que se desea activar y abre los demás interruptores.

A continuación se presenta el diagrama que ejemplifica este proceso:

* Port .- Canal a través del cual los datos son transferidos entre un dispositivo de entrada/ salida y el micro procesador.. Tomado de Computer Dictionary, Microsoft Press U.S.A. 1991.

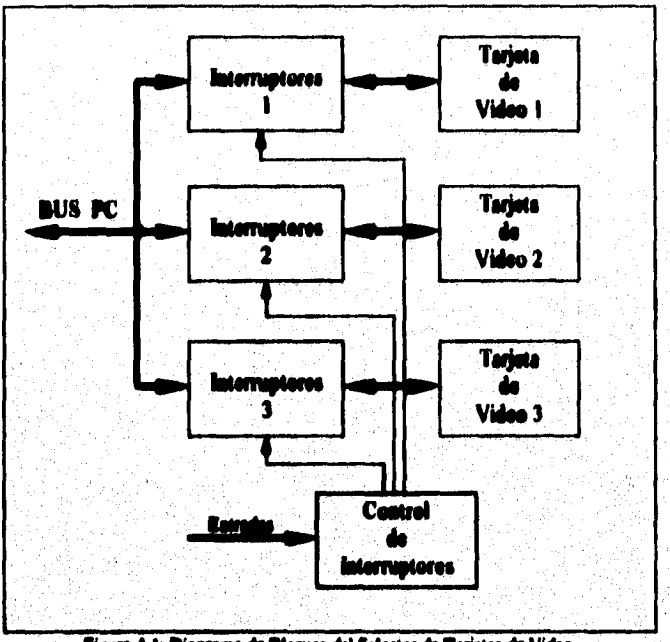


Figura 4.1: Diagrama de Bloques del Selector de Tarjetas de Video.

Es conveniente mencionar que en nuestro caso para desactivar una tarjeta no es necesario el interrumpir todas las líneas entre el bus de la PC y la tarjeta de video, basta con desconectar las líneas de datos y las líneas que llevan señales de control de la tarjeta de video a la PC. (ver cap. referente a el bus).

Para la implementación de este selector de tarjetas se utilizaron como interruptores los circuitos integrados CD4016B, dichos integrados contienen cuatro "switches bilaterales", y para elaborar el control de los interruptores se empleó un decodificador 2x4. Se consideraron como entradas al decodificador los dos bits más significativos del dato que se introduce al sistema mediante 4

interruptores tipo cola de rata. Tres de las salidas del decodificador entran a inversores del tipo colector abierto para después conectare a las líneas de control de los interruptores analógicos (ver Apéndice III).

4.1.2 El Puerto

Aún cuando un microprocesador puede tener tantos puertos como número de localidades pueda direccionar, generalmente en los modelos de PC. no se aprovecha completamente el espacio de direccionamiento ya que sólo los 10 bits menos significativos de bus de direcciones son utilizados para direccionamientos a puertos. [26]

"Cuando la novena línea del bus de direcciones de la PC está inactiva, no se pueden recibir datos de los conectores de expansión (sólo de la tarjeta principal) y cuando el bit 9 está activo habilita la capacidad de recibir datos de los conectores de expansión"[27]. Esto significa que para colocar un puerto sólo es posible utilizar una dirección en el intervalo de la 0200 a la 02FF, razón por la cual se buscó una dirección que no estuviera ocupada por algún dispositivo, siendo seleccionada como dirección de nuestro puerto la localidad 0270.

Una vez que se eligió esta dirección y tomando en cuenta cuáles señales intervienen en el proceso de lectura de un dato del exterior (ver capítulo referente al bus), se diseñó un circuito acorde.

[26] Lewis. C. Eggebrecht. Interfacing to the IBM Personal computer. Ed. SAMS. Segunda edición. USA 1990.
[27] Ibidem.

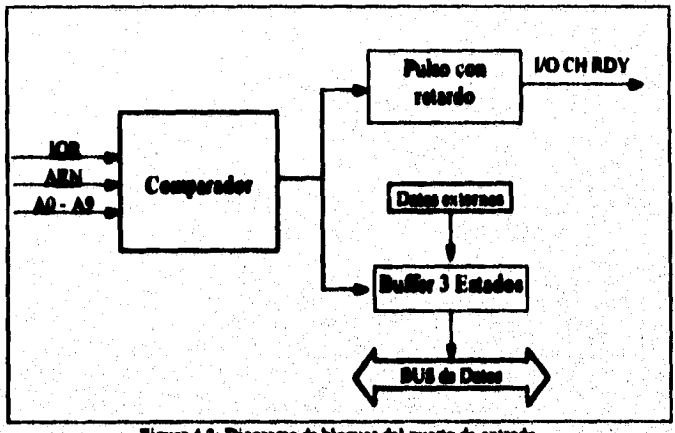


Figura 4.2: Diagrama de bloques del puerto de entrada

Para la implementación de este puerto se utilizaron en la etapa de comparación los circuitos integrados 74S260 y 74LS30. El circuito 74S260 es una compuerta NOR de 5 entradas y la compuerta 74LS30 es una NAND de 8 entradas.

En la etapa de comparación las señales que deben de ser cero (A0, A1, A2, A3, A7, A8 y AEN) entran a las compuertas NOR y las que deben de ser 1 lógico entran a la NAND (A4, A5, A6 y A9) (ver Apéndice III).

Cuando las líneas de direcciones y de control en la transferencia de datos coinciden con la dirección 0270 y las señales IOR y AEN son cero, se activa la salida de la NAND con lo que se habilita el "Buffer" 3 estados (74LS2440) permitiendo que el dato que se quiere transmitir a la PC quede presente en el bus de datos. Al mismo tiempo que se activa el Buffer 3 estados, se genera un pulso de 35 nanosegundos el cual pasa a través de 4 inversores para que sea retardado un instante, dicho pulso se retorna como señal de I/O CH RDY.

Adicionalmente a estas partes funcionales que previamente se han explicado fue necesario el poder controlar de manera automática la configuración de la tarjeta CGA, ésto debido a que ella es capaz de trabajar como CGA, emular una tarjeta MDA o una tarjeta Hércules. Es importante mencionar que en la tarjeta principal (Mother Board) de las PC. existe un interruptor para configurar el tipo de video, el cual se debe de colocar en Mono cuando se quiere utilizar una tarjeta CGA como la nuestra (Magic Combo CT-6190S) y se debe desactivar en el caso de las otras tarjetas. Estos interruptores utilizan para activarse 'jumpers' y tienen la siguiente forma:

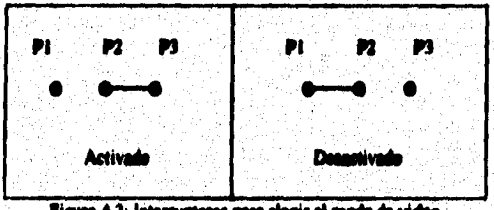


Figura 4.3: Interruptores para elegir el modo de video

Para implementar estos interruptores se utilizó el circuito CD4016B y el 74LS04 los cuales se conectaron de la siguiente manera.

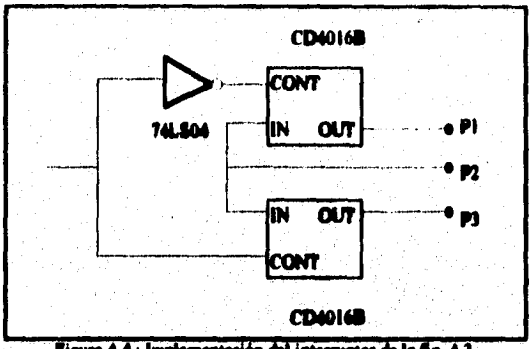


Figura 4.4 : Implementación del interruptor de la fig. 4.3

En algunas ocasiones cuando se desea ajustar un monitor y se posee otro del mismo tipo en buen estado podría ser útil el conectar ambos monitores a la misma tarjeta de video. Esto es técnicamente posible debido a que la tarjeta de video cuenta con una baja impedancia de salida y los monitores tienen una alta impedancia de entrada. Para realizar ésto, es suficiente con fabricar un cable de extensión en el que cada línea del cable se bifurque hacia un conector del mismo tipo que aquel en el que vamos a conectar esta extensión.

4.2 Diseño del Software

El software es una parte fundamental en el funcionamiento del sistema de pruebas, es el encargado de interpretar cual es la prueba que debe ejecutar y de generar la forma de las imágenes que se presentarán en cada una de las pruebas.

Para la realización del software del sistema se tomaron como directrices las siguientes características:

1.- Ser un programa fácil de manejar.

2.- Contener un código que sea posible utilizar en un diseño futuro, por ejemplo el de extender las pruebas a diferentes tipos de monitores que puedan surgir o el integrar el programa en un sistema de pruebas para otros periféricos.

Por las razones antes mencionadas se eligió como metodología para diseñar el software de nuestro sistema la Programación Orientada a Objetos ya que "es un importante conjunto de técnicas que se pueden utilizar para hacer el desarrollo de programas más eficiente mientras mejora la fiabilidad de los programas resultantes". [28]

Algunos de los beneficios de la tecnología orientada a objetos son:

[28] Joyanes Aguilar Luis. C++ un enfoque orientado a objetos. Editorial Mc Graw Hill. España 1994. p.p.2.

"Reutilización. Un objetivo fundamental de las técnicas orientadas a objetos es lograr la reutilización masiva al construir un software"[29]

"Confiabilidad. Es probable que el software construido a partir de clases estables ya probadas tenga menos fallas que el software elaborado a partir de cero".[30]

"Diseño de mayor calidad. Los diseños suelen tener mayor calidad, puesto que se integran a partir de componentes probados que han sido verificados y pulidos varias veces."[31]

"Mantenimiento más sencillo. El programador encargado del mantenimiento cambia un método de clase a la vez. Cada clase efectúa sus funciones independientemente de las demás."[32]

4.2.1 Jerarquía de Clases

El software que se desarrolló tiene la jerarquía de clases: de la figura 4.5

En la clase VIDEOBIOS están contenidas las funciones más importantes de la interrupción de video 10H. Los nombres de las funciones de esta clase están basados en el libro IBM ROM BIOS[33].

---

[29] James Martin. James j. Odell. Análisis y Diseño Orientado a objetos. Editorial Prence Hall. México .1994. pp.36.

[30] Ibidem pp37

[31] Ibidem pp37

[32] Ibidem pp37

[33] Ray Duncan IBM ROM BIOS Microsoft Press USA 1988.

La clase PANTALLA se deriva (hereda) de la clase VIDEOBIOS y cuenta con algunas funciones que nos facilitan trabajar con la pantalla real como ocultar el cursor, limpiar la pantalla, pintar líneas, pintar círculos, pintar superficies rectangulares de algún color, etc..

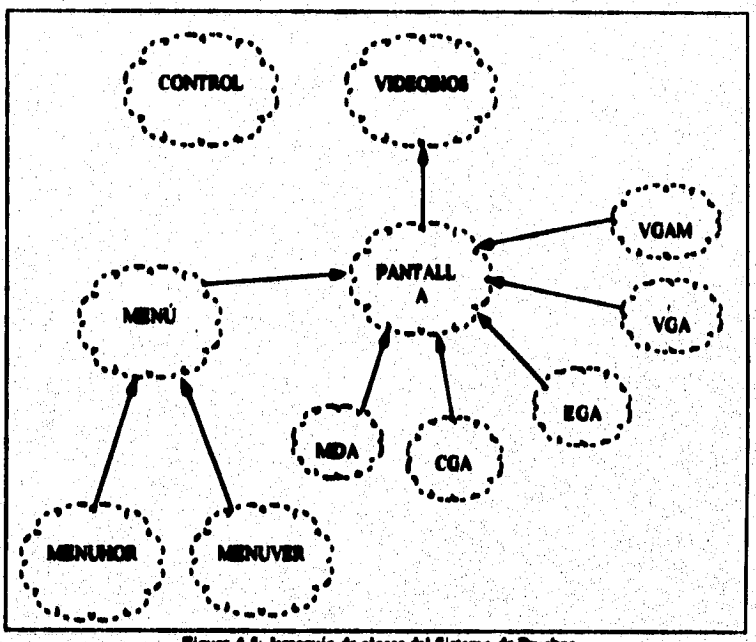


Figura 4.5: Jerarquía de clases del Sistema de Pruebas

La clase MENÚ es una clase abstracta[34], cuyo objetivo es el contener las características generales de un menú como son el título, sus opciones, su posición, etc.. Las instancias de esta clase no serían visibles ni podrían trabajar como menú, por éste motivo se crearon las clases MENUHOR y MENUVER, las cuales se heredan de la clase MENÚ.

[34] Se denominan clases abstractas, las clases diseñadas con el objetivo principal de ser heredadas. Normalmente no se crean instancias a partir de clases abstractas, aunque esto posible. JOYANES, Luis Op.cit. pp. 313

Las instancias de la clase MENUHOR son menúes donde las opciones se encuentran en un mismo renglón, mientras que en las instancias de MENUVER las opciones se encuentran una arriba de la otra. En ambos casos los menúes regresan un número asociado a cada opción.

Las clases MDA, CGA, EGA, VGAM Y VGA heredan de la clase VIDEOBIOS. Cada una de las funciones contenidas en estas clases, es capaz de generar uno de los patrones de prueba relativos a algún tipo de monitor.

La clase CONTROL, entre otras cosas, es la encargada de canalizar los mensajes entre los objetos que están activos, además de llevar la secuencia de las pruebas que se realizan.

La función "LeePuerto" es la encargada de leer la información que se colocó en el gabinete del sistema de pruebas mediante los interruptores.

La función "Presentación" es la encargada de mostrar en la pantalla los datos generales acerca del programa como nombre, fecha de elaboración, versión, autor, etc..

La implementación de estas clases y funciones se realizó utilizando el compilador de Borland C++ versión 3.1. En el apéndice II se incluye el listado del programa así como las clases y funciones implementadas.

RESULTADOS

Como resultado del presente trabajo se obtuvo un aparato que conectado al bus de una computadora nos permite evaluar el estado de un monitor, así como el poder realizarle los ajustes necesarios.

A nivel de Hardware, los resultados son un selector de tarjetas de computadora y un puerto de entrada de datos, ambos implementados en una tarjeta de baquelita usando técnica "wire wrap". La tarjeta perforada podría llegar a presentar fallas debido al ruido que capta; sin embargo, con un circuito impreso no sucederá ésto ya que en él se eliminan contactos poco firmes y una gran cantidad de alambres.

En lo relativo al Software se realizó un programa para generar patrones de video acordes a las características de los diferentes tipos de monitor. Es importante mencionar que la mayor parte del software desarrollado para este trabajo es posible reutilizarlo en una gran variedad de aplicaciones

CONCLUSIONES

El Sistema de pruebas que se desarrolló aporta un avance en el proceso de reparación a equipos de video como monitores, ya que nos permite realizar pruebas y ajustes a casi cualquier tipo de monitor compatible con PC de IBM, lo cual lo convierte en un aparato de gran utilidad.

A diferencia del software comercial que se utiliza para probar monitores pero que son propios para el diagnóstico de las tarjetas de video, el programa que se realizó es más facil de manejar además de contar con los patrones de video nesesarios para probar y ajustar monitores. Sin embargo, en algunas de las pruebas el programa es lento en comparación con el manejo de pantallas del software comercial.

Creemos que el presente trabajo puede llegar a ser útil para la capacitación de personal en el área de Mantenimiento de Equipo de Cómputo.

BIBLIOGRAFIA

1. A. J. Van de Goor, Computer Architecture and Design, Ed. Addison-Wesley, USA 1990.

2. A. Michael Noll, Television Technology: Fundamentals and Future Prospects., Ed. Artech House, U.S.A. 1986.

3. Aurelio Labanda Alonso y Alfonso Martín Marcos, Televisión en Color, Ediciones Aura, Barcelona 1988.

4. Brooks Charles J.,IBM PC Peripheral Troubleshooting and Repair, Ed. Howard W. SAMS, Primera Edición,USA 1987.

5. Computer Dictionary, Ed. Microsoft Press, U.S.A. 1991.

6. Ed Mitchell, and other Masters, Secrets of the Borland C++ Masters, Ed. SAMS Publishing,, USA 1992.

7. Gunnar Forst, PC Principles, Ed. MIT Press, EUA 1990.

8. IBM Corp., Technical Reference, Personal Computer Hardware Reference Library, USA 1984.

9. Joyanes Aguilar Luis, C++ un enfoque orientado a objetos, Ed. Mc Graw Hill, España 1994.

10. Lewis. C. Eggebrecht, Interfacing to the IBM Personal computer, Ed. SAMS, Segunda edición, USA 1990.

11. Margolis ART, Troubleshooting and repairing the new personal computers, Ed. Tab Books, USA 1987.

12. Martin, James. y Odell.James J., Análisis y Diseño Orientado a Objetos, Ed. Pretice Hall, México 1994.

13. Michael Tischer, PC Intern , Ed. Abacus, EUA 1992.

14. Michael Tischer, PC System Programming, Ed. Abacus, 5ª Edición.

15. Minansi Mark, The Complete PC upgrade and Maintenance guide, Ed Sybex, Primera Edición, USA 1993.

16. Pappas Chris H, y Murray William H, Manual de Borland C++ 4.0, Ed. Osborne Mc Graw Hill, España 1994.

17. Peter Norton, Toda la PC , Ed. Prentice Hall, 5ª Edición, México 1994.

18. Peter Utz, Today's Video Equipment Setup and Production, Ed. Prentice Hall, Newjersey U.S.A. 1987.

19. Phoenix Technologies Ltd., System BIOS for IBM PC/AT computers and compatibles, Ed. Addison -Wesley, United States of America 1989.

20. Ray Duncan, IBM ROM BIOS, Ed. Microsoft Press, USA 1988.

21. RCA Corporation Solid State Division, RCA COS/MOS/Integrated Circuits, USA 1980.

22. RoehrigSchueller, Upgrading & Mantaining your PC, Ed. Abacus, Primera Edición, USA 1990.

23. Robert J. Baron y Lee Higbie, Computer Architecture, Ed. Addison-Wesley, USA 1992.

24. Solari Edward, ISA and EISA: theory and operation, Ed. Annabooks, USA 1992.

25. Sony Service, El Trinitron Tecnologia y Guía de Reparaciones, Ediciones Aura, Barcelona 1986.

26. Taylor Billy P., C/C++ Programmer's Guide to Using PC BIOS, Ed. Lande A. Leventhal, USA 1993.

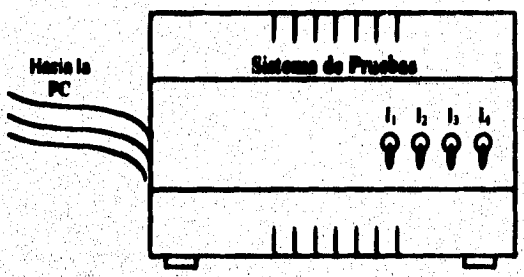
27. Texas Instruments, TTL Logic Data Book, USA 1988.

APÉNDICE I

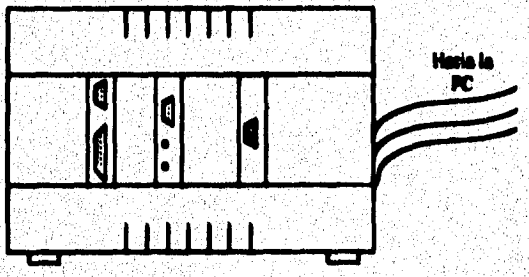
Manual del Sistema de Pruebas de Equipos de Vídeo Compatibles con PC de IBM

Situación y función de los controles

Parte frontal:



Parte posterior.



Para poder reconocer las funciones que tiene el sistema de pruebas es necesario describir los elementos que lo forman. A continuación se explican sus partes fundamentales.

Comando:

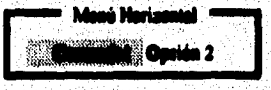
Palabra o frase que es una instrucción para que el sistema de pruebas puede realizar una acción. Existen comandos que al activarlos se ejecutan; sin embargo, puede ser que algún comando requiera de más información para lo cual se desplegará un menú donde se solicitará información adicional para ejecutar el comando.

Menúes:

El sistema de pruebas agrupa todos los comandos disponibles en menúes. La agrupación se realiza en base a la acción que realiza cada comando.

Menú Horizontal:

Existen menúes en los que los comandos (opciones) se encuentran uno a la derecha de otro.



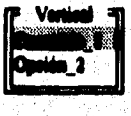
En este tipo de menú para seleccionar un comando existen varias alternativas:

1.- Pulsar la tecla que contiene el carácter resaltado en ese comando.

2.- Con ayuda de las teclas: Flecha a la derecha o Flecha a la izquierda, se coloca uno sobre la opción deseada, la cual estará resaltada o parpadeando, después se presiona la tecla Flecha hacia abajo o <ENTER>.

Menú Vertical:

Los menúes Verticales son aquellos en los que los comandos se encuentran uno arriba de otro.

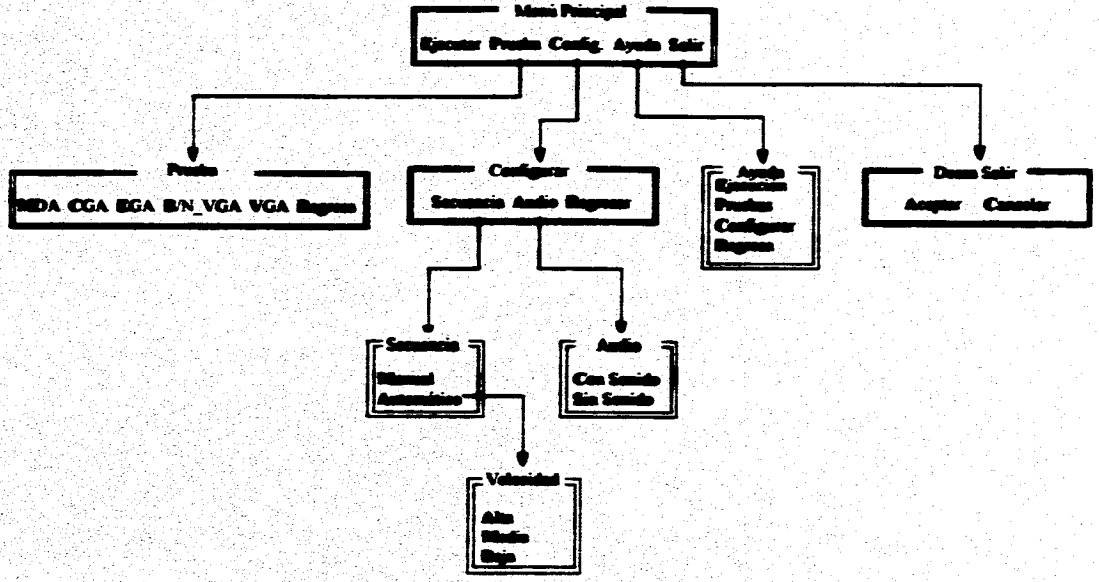


Para seleccionar un comando en esta clase de menúes hay dos posibilidades:

1.- Presionar la tecla que contiene el carácter resaltado de ese comando

2.- Colocarse con ayuda de las teclas Flecha hacia Arriba o Flecha hacia abajo sobre la opción deseada y después pulsar la tecla <ENTER>.

Organigrama del Sistema de Pruebas



Descripción de los Comandos

Ejecutar

Inicia la generación de los diferentes patrones de vídeo seleccionados.

Prueba

Abre el menú Prueba. En este menú se selecciona el tipo de prueba que puede o no ser el mismo que se eligió en el modulo de pruebas con los interruptores.

Config.

Activa el menú Configurar.

Ayuda

Despliega el menú de ayuda.

Salir

Muestra un menú para confirmar la salida.

Comandos del menú Prueba.

MDA

Selecciona las pruebas de vídeo para un monitor de resolución similar a un MDA.

CGA

Selecciona las pruebas de vídeo para un monitor de resolución CGA.

EGA

Selecciona las pruebas de vídeo para un monitor de resolución EGA.

B/N_VGA

Selecciona las pruebas de vídeo para un monitor de resolución VGA monocromático.

VGA

Selecciona las pruebas de vídeo para un monitor de resolución VGA o SVGA.

Comandos del menú Configurar

Secuencia

Activa el menú Secuencia.

Audio

Despliega el menú Audio.

Regresar

Retorna al menú principal.

Comandos del menú Ayuda

Ejecución

Muestra la ayuda referente a la ejecución del programa.

Pruebas

Despliega la ayuda correspondiente a la selección de las pruebas.

Configurar

Presenta información relativa a la configuración del sistema.

Regresa

Retorna al menú Principal.

Comandos del menú Desea Salir?

Aceptar

Termina la ejecución del programa del sistema de pruebas.

Cancelar

Regresa al menú Principal.

Comandos del menú Secuencia

Manual

Este comando configura la secuencia de las pruebas de tal forma que cuando se ejecutan éstas, nosotros indicamos al programa mediante el teclado si queremos que genere el siguiente patrón o el anterior. Para seleccionar el siguiente patrón se puede pulsar alguna de estas teclas: S, s. Barra espaciadora, Flecha hacia abajo, Flecha hacia la derecha o <ENTER>.

Para mostrar el patrón anterior se pueden utilizar las teclas A, a, Flecha hacia arriba o Flecha hacia la izquierda.

En este modo de secuencia después de que se mostró el último patrón, si se pulsa alguna tecla para ver el siguiente patrón, el sistema regresará al menú Principal.

Automático

Este comando configura las pruebas de tal forma que se generan los patrones en forma secuencial ya al llegar al final se vuelve a comenzar con el primer patrón.

Cuando se elige esta opción, automáticamente se despliega un menú para elegir la velocidad con la que cambiarán los patrones de la prueba.

Para terminar la ejecución de una prueba automática basta con pulsar la tecla <ESC>, con lo que se regresa al menú principal

Comandos del menú Velocidad

Alta

Configura el tiempo entre que se termina de pintar un patrón y se comienza el siguiente en aproximadamente 1 segundo.

Media

Configura el tiempo entre que se termina de pintar un patrón y se comienza el siguiente en aproximadamente 5 segundos.

Baja

Configura el tiempo entre que se termina de pintar un patrón y se comienza el siguiente en aproximadamente 10 segundos.

Comandos del menú Audio

Con Sonido

Configura el sistema para que antes de pintar cualquier patrón emita un sonido por el puerto de audio.

Sin Sonido

Configura el sistema para suprimir el sonido que se emite antes de generar algún patrón de pruebas.

Procedimientos más comunes en el sistema

Entrar al sistema de pruebas

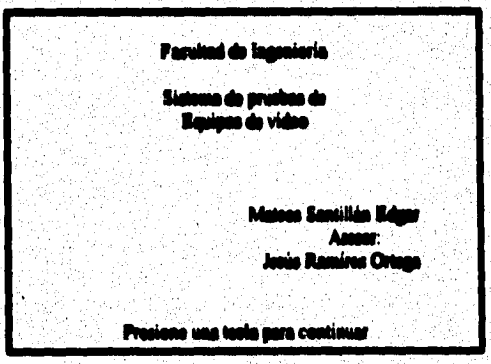
1.- Asegúrese de que la computadora en la que esta conectado el Módulo de Pruebas está apagada.

2.- Seleccione con ayuda de los interruptores (I_1 a I_4)el tipo de prueba que desea realizar.

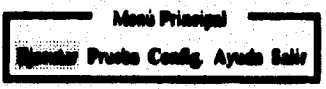
Tipo de prueba	I_1	I_2	I_3	I_4
MDA	0	0	0	1
CGA	0	0	1	1
EGA	0	1	0	0
VGA mono	1	0	0	0
SVGA o VGA	1	0	0	1

Nota: 0 Significa Interruptor abierto (abajo).
1 Significa Interruptor cerrado (arriba).

3.- Encienda la computadora. En la pantalla del monitor que se desea probar puede aparecer(dependiendo del estado del monitor)un mensaje similar al de siguiente figura



4.- Presione alguna tecla. En el centro de la pantalla aparecerá un menú como el siguiente:



Ejecutar Pruebas

1.- Para iniciar las pruebas relativas a algún tipo de monitor es suficiente con seleccionar en el menú principal el comando **Ejecutar**.

Cada vez que se activa el sistema de pruebas se configura una secuencia manual, con sonido y el tipo de monitor seleccionado (Ver comandos "Manual" y "Con Sonido").

Seleccionar un tipo de prueba

1.- Regrese al menú principal.

2.- Elija el comando **Prueba**. Aparecerá el menú Prueba.

3.- De las opciones que se muestran en el menú seleccione una. Es importante que considere que el tipo de tarjeta que seleccionó al entrar al sistema de pruebas soporta esta nueva elección.

Romper una prueba que se ejecuta en secuencia automática

1.- Para terminar una secuencia automática presione la tecla <ESC> y espere unos segundos. Poco después aparecerá el menú principal.

Terminar una sesión de pruebas

1.- Regrese al menú principal.

2.- Seleccione el comando **Salir**. Se abrirá un menú para confirmar la terminación del programa.

3.- Escoja la opción **Aceptar** y espere a que aparezca el 'prompt' del sistema operativo

Configurar secuencia

1.- Regrese al menú principal.

2.- Seleccione el comando **Configurar**. Deberá aparecer el menú Configurar.

3.- Elija la opción **Secuencia.** Aparecerá el menú secuencia.

4.- Escoja la forma de secuencia deseada.

5.- En caso de escoger **Automático.** Determine cual será la velocidad de ejecución de las pruebas.

Configurar sonido

1.- Regrese al menú principal.

2.- Seleccione el comando **Configurar**.

3.- Elija la opción **Audio**. Aparecerá el menú Audio.

4.- Escoja la forma en que desea que se realicen las pruebas.

Especificaciones

Dimensiones del Módulo de Prueba:

19 x 26 x 17 cm.

Peso:

2.5 Kg.

Consumo de energía:

1.6 Watts.

Tamaño del programa de pruebas:

84 Kbytes.

APÉNDICE II

```
//*****************************************************************************
//*                                                                           *
//*                              Clase CGA                                    *
//*                                                                           *
//*************************************CGA.H***********************************

#ifndef CGA_H
#define CGA_H 1


#include "PANTALLA.H"

class CGA:public PANTALLA
{
public:
        CGA() {};
        // Pinta en pantalla algunos carácteres(40 x 25) en forma monocromática
        void CaracteresMono40();
        // Pinta en pantalla algunos carácteres (40 x 25) a colores
        void CaracteresColor40();
        // Pinta en pantalla carácteres (40 x 25) destacando sus atributos
        void Atributos40();
        // Pinta en pantalla algunos carácteres(80 x 25) en forma monocromática
        void CaracteresMono80();
        // Pinta en pantalla algunos carácteres (80 x 25) a colores
        void CaracteresColor80();
        // Pinta en pantalla carácteres (80 x 25) destacando sus atributos
        void Atributos80();
        // Pinta la pantalla de color rojo con resolución de 320 x 200
        void PaletaRojo();
        // Pinta la pantalla de color amarillo con resolución de 320 x 200
        void PaletaAmarillo();
        // Pinta la pantalla de color verde con resolución de 320 x 200
        void PaletaVerde();
        // Pinta la pantalla de color blanco con resolución de 320 x 200
        void PaletaBlanco();
        // Pinta la pantalla con los colores rojo,verde y amarillo
        // con resolución de 320 x 200
        void Paleta0();
        // Pinta la pantalla con los colores cyan,magenta y blanco
        // con resolución de 320 x 200
        void Paleta1();
        // Pinta en la pantalla una cuadrícula con puntos en el centro
        void Convergencia();
        // Pinta una cuadrícula y un círculo en el centro de la pantalla
        void Apariencia();
```

```
        //Dibuja en la pantalla un cuadrado
        void PintaRojo();
};

#include "CGA.CPP"
#endif
```

```
//*****************************************************************************
//*                                                                           *
//*                              Clase CGA                                    *
//*                                                                           *
//*************************************CGA.CPP*********************************
// Pinta en pantalla algunos carácteres(40 x 25) en forma monocromática
        void CGA::CaracteresMono40()
                {
                int modo = GetModoVideo();
                if(modo != 0x01) SetVideoModo(0x01);
                PintaCaracteresM();
                }

// Pinta en pantalla algunos carácteres (40 x 25) a colores
        void CGA::CaracteresColor40()
                {
                int modo = GetModoVideo();
                if(modo != 1) SetVideoModo(1);
                PintaCaracteresC();
                }

// Pinta en pantalla carácteres (40 x 25) destacando sus atributos
        void CGA::Atributos40()
                {
                int modo = GetModoVideo();
                if(modo != 1) SetVideoModo(1);
                AtributosCaracter();
                }

// Pinta en pantalla algunos carácteres(80 x 25) en forma monocromática
        void CGA::CaracteresMono80()
                {
                int modo = GetModoVideo();
                if(modo != 3) SetVideoModo(3);
                PintaCaracteresM();
                }

// Pinta en pantalla algunos carácteres (80 x 25) a colores
        void CGA::CaracteresColor80()
                {
                int modo = GetModoVideo();
                if(modo != 3) SetVideoModo(3);
                PintaCaracteresC();
                }

// Pinta en pantalla carácteres (80 x 25) destacando sus atributos
```

```
        void CGA::Atributos80()
                {
                int modo = GetModoVideo();
                if(modo != 3) SetVideoModo(3);
                AtributosCaracter();
                }

// Pinta la pantalla de color rojo con resolución de 320 x 200
        void CGA::PantallaRojo()
                {
                int modo = GetModoVideo();
                if(modo != 4) SetVideoModo(4);
                SetPaleta(0x01,0x00);
                Superficie(0,0,0,320,200,0x02);
                }

// Pinta la pantalla de color amarillo con resolución de 320 x 200
        void CGA::PantallaAmarillo()
                {
                int modo = GetModoVideo();
                if(modo != 4) SetVideoModo(4);
                SetPaleta(0x01,0x00);
                Superficie(0,0,0,320,200,0x03);
                }

// Pinta la pantalla de color verde con resolución de 320 x 200
        void CGA::PantallaVerde()
                {
                int modo = GetModoVideo();
                if(modo != 4) SetVideoModo(4);
                SetPaleta(0x01,0x00);
                Superficie(0,0,0,320,200,0x01);
                }

// Pinta la pantalla de color blanco con resolución de 320 x 200
        void CGA::PantallaBlanco()
                {
                int modo = GetModoVideo();
                if(modo != 4) SetVideoModo(4);
                SetPaleta(0x01,0x01);
                Superficie(0,0,0,320,200,0x03);
                }

// Pinta la pantalla con los colores verde, rojo y amarillo
// con resolución de 320 x 200
        void CGA::Paleta0()
```

```
        {
        int modo = GetModoVideo();
        if(modo != 4) SetVideoModo(4);
        SetPaleta(0x01,0x00);
        Superficie(0,0,0,160,130,0x01);
        Superficie(0,160,0,320,130,0x02);
        Superficie(0,0,130,320,200,0x03);
        }

// Pinta la pantalla con los colores cyan,magenta y blanco
// con resolución de 320 x 200
    void CGA::Paleta1()
        {
        int modo = GetModoVideo();
        if(modo != 4) SetVideoModo(4);
        SetPaleta(0x01,0x01);
        Superficie(0,0,0,160,130,0x01);
        Superficie(0,160,0,320,130,0x02);
        Superficie(0,0,130,320,200,0x03);
        }

// Pinta en la pantalla una cuadricula con puntos en el centro
    void CGA::Convergencia()
        {
        int modo = GetModoVideo();
        if(modo != 6) SetVideoModo(6);
        double aspecto=Aspecto();
        int i=(int)5*aspecto;
        int Xmax=GetMaxCol();
        int Ymax=GetMaxRow();
        LimpiaPantalla();
        Raja(10);
        for(i;i<Xmax;i=i+(int)10*aspecto)
            for(int j=5;j<Ymax;j+=10)
                WritePixel(0x0F,0,j,i);
        }

// Pinta una cuadricula y un circulo en el centro de la pantalla
    void CGA::Apariencia()
        {
        int modo = GetModoVideo();
        if(modo != 6) SetVideoModo(6);
        LimpiaPantalla();
        Raja(10);
        PintaCirculo(0,320,100,100);
        }
```

```
//*************************************************************************
//*                                                                       *
//*                          Clase CONTRTOL                               *
//*                                                                       *
//*************************CONTROL.H*************************************
#ifndef CONTROL_H
#define CONTROL_H 1

#include "MENUVER.H"
#include "MENUHOR.H"
#include "MDA.H"
#include "CGA.H"
#include "EGA.H"
#include "VGAM.H"
#include "VGA.H"

class CONTROL
{
private:
        int secuencia,velocidad,sonido,prueba;
public:
        CONTROL(){secuencia=0,velocidad=1,sonido=1;};
        // Determina el tipo de prueba que se seleccionó en el módulo de pruebas
        void LeePuerto();
        // Retorna el valor asociado con alguna tecla
        int Respuesta(void);
        //Pinta la pantalla de manera uniforme para colocar un menú
        void FondoMenu(MENUHOR *,short int);
        // Selecciona la manera en que se presentaran las pruebas
        // (manual o automático)
        void ConfigSecuencia(MENUVER *MENUSEC,MENUVER *MVELO);
        //Configura la velocidad en modo automático
        void ConfigVelocidad(MENUVER *);
        // Configura la presencia o ausencia de sonido
        void ConfigSonido(MENUVER *);
        // Despliega un menú para configurar la secuencia, velocidad y sonido
        void ConfigSistema(MENUHOR *MENUSIS,MENUVER *MSECU,
                                        MENUVER *MVELO,MENUVER *MAUDI);
        // Pide confirmar la salida
        int  SalirSistema(MENUHOR *MSALI);
        // Emite un sonido por el puerto de audio
        void Suena();
        // Realiza las pruebas respectivas de un monitor MDA
        void EjecutaMDA(MDA*);
        // Realiza las pruebas respectivas de un monitor CGA
        void EjecutaCGA(CGA*);
```

```
        // Realiza las pruebas respectivas de un monitor EGA
        void EjecutaEGA(EGA*);
        // Realiza las pruebas respectivas de un monitor VGA Monocromático
        void EjecutaVGAM(VGAM*);
        // Realiza las pruebas respectivas de un monitor VGA
        void EjecutaVGA(VGA*);
        // Ejecuta la prueba del monitor que se haya seleccionado
        void EjecutaPrueba(MDA *MDAEXE,CGA *CGAEXE,
                                EGA *EGAEXE,VGAM *VGAMEXE,VGA *VGAEXE);
        // Despliega el menú de ayuda
        void Ayuda(MENUVER *);
        // Despliega la ayuda Ejecución
        void AyuEjecucion(MENUVER *MENUAYU);
        // Despliega la ayuda de selección de pruebas
        void AyuPrueba(MENUVER *MENUAYU);
        // Despliega la ayuda configuración
        void AyuConfig(MENUVER *MENUAYU);
        // Selecciona en un menú una de las pruebas de monitor
        void SelecPrueba(MENUHOR *MENUPRU);
        // Despliega el menú principal del sistema
        int Principal(MENUHOR *MPRIN,MENUHOR *MPRUE,MENUHOR
*MCONFI,MENUHOR *SALI,
                MENUVER *MSECU,MENUVER *MVELO,MENUVER *MAUDI,MENUVER
*MAYUD,
                MDA *MDAEXE,CGA *CGAEXE,EGA *EGAEXE,VGAM *VGAMEXE,VGA
*VGAEXE,
                short int modoini);
};

#include "CONTROL.CPP"
#endif
```

```
//*************************************************************************
//*                                                                       *
//*                          Clase CONTRTOL                               *
//*                                                                       *
//*****************************CONTROL.CPP*********************************

// Determina el tipo de prueba que se seleccionó en el modulo de pruebas
void CONTROL::LeePuerto()
{
        unsigned char dato;
        int acumulador;
        dato= inportb(0x0270);
        acumulador=int(dato & 0x0F);
        switch (acumulador)
                {
                case 0x01: prueba=1; break;
                case 0x03: prueba=2; break;
                case 0x04: prueba=3; break;
                case 0x08: prueba=4; break;
                case 0x09: prueba=5; break;
                default: prueba=0; break;
                }
}

// Retorna el valor asociado con alguna tecla
int CONTROL::Respuesta()
{
        union inkey {char ch[2];int i; } c;
        int tecla;
        if(secuencia==1)
                {
                for (;;)
                        {
                        while(!bioskey(1))
                                {
                                delay(velocidad*1000);
                                return 0;
                                }
                        c.i=bioskey(0);
                        if(c.ch[0]==27) return -1;
                        }
                }
        else
        {
        for (;;)
                {
```

```
                while(!biosKey(1));
                tecla=tecla;
                c.i=biosKey(0);
                if (c.ch[0])
                        {
                        switch(c.ch[0])
                                {
                                case '\r': return 0;
                                case 's' : return 0;
                                case 'S' : return 0;
                                case ' ' : return 0;
                                case 'a' : return 2;
                                case 'A' : return 2;
                                case 27  : return -1;
                                }
                        }
                        else
                                {
                                switch(c.ch[1])
                                        {
                                        case 72 : return 2;
                                        case 75 : return 2;
                                        case 77 : return 0;
                                        case 80 : return 0;
                                        }
                                }
                }
        }

}

//Pinta la pantalla de manera uniforme para colocar un menú
void CONTROL::FondoMenu(MENUHOR *MENUPRIN, short int modoini)
        {
        short int maxcol,mod,pag;
        int niv3,color,atributo;
        MENUPRIN->GetVideoMode(maxcol,mod,pag);
        if(mod!=modoini)MENUPRIN->SetVideoMode(modoini);
        MENUPRIN->GetVideoMode(maxcol,mod,pag);
        if(mod==0x07 || mod==0x06 || mod==0x11)color=0x00;
        else      color=0x10;
        atributo=color+0x07;
        for(int i=0;i<24;i++)
                {
                MENUPRIN->SetCursorPosition(0,i,0);
                MENUPRIN->WriteCharAttribute(0,178,atributo,maxcol);
```

```
            }
        MENUPRIN->SetCursorPosition(0,24,0);
        MENUPRIN->WriteCharAttribute(0,32,atributo,maxcol);
        }


// Selecciona la manera en que se presentaran las pruebas
// (manual o automático)
void CONTROL::ConfigSecuencia(MENUVER *MENUSEC,MENUVER *MVELO)
        {
        short int maxcol,mod,pag;
        int niv3;
        char *menusec[]={
        "^Manual    ",
        "^Automático ",
        "\0");
        MENUSEC->Carga_Datos(menusec);
        MENUSEC->Titulo(" SECUENCIA");
        MENUSEC->GetVideoMode(maxcol,mod,pag);
        pag=(maxcol-15)/2;
        MENUSEC->Mueve(pag,10);
        niv3=MENUSEC->Activa_Menu();
        switch (niv3)
                {
                case 0:secuencia=0; MENUSEC->OcultaMenu(); break;
                case 1:secuencia=1; MENUSEC->OcultaMenu();
                        ConfigVelocidad(MVELO); break;
                }
        }

//Configura la velocidad en modo automático
void CONTROL::ConfigVelocidad(MENUVER *MENUVEL)
        {
        short int maxcol,mod,pag;
        int niv4;
        char *menuvel[]={
        "^Alta  ",
        "^Media ",
        "^Baja  ",
        "\0");
        MENUVEL->Carga_Datos(menuvel);
        MENUVEL->Titulo(" VELOCIDAD");
        MENUVEL->GetVideoMode(maxcol,mod,pag);
        pag=(maxcol-15)/2;
        MENUVEL->Mueve(pag,10);
        niv4=MENUVEL->Activa_Menu();
```

```
        switch (niv4)
                {
                case 0: velocidad=1; MENUVEL->OcultaMenu(); break;
                case 1: velocidad=5; MENUVEL->OcultaMenu(); break;
                case 2: velocidad=10;MENUVEL->OcultaMenu(); break;
                }
        }

// Configura la presencia o ausencia de sonido
void CONTROL::ConfigSonido(MENUVER *MENUSON)
        {
        short int maxcol,mod,pag;
        int niv3;
        char *menuson[]={
        "^Con Sonido ",
        "^Sin Sonido ",
        "\0"};
        MENUSON->Carga_Datos(menuson);
        MENUSON->Titulo(" AUDIO");
        MENUSON->GetVideoMode(maxcol,mod,pag);
        pag=(maxcol-15)/2;
        MENUSON->Mueve(pag,10);
        niv3=MENUSON->Activa_Menu();
        switch (niv3)
                {
                case 0: sonido=1; MENUSON->OcultaMenu(); break;
                case 1: sonido=0; MENUSON->OcultaMenu(); break;
                }
        }
// Despliega un menú para configurar la secuencia, velocidad y sonido
void CONTROL::ConfigSistema(MENUHOR *MENUSIS,MENUVER *MSECU,
                                        MENUVER *MVELO,MENUVER *MAUDI)
        {
        short int maxcol,mod,pag;
        int niv2;
        char *menusis[]={
        "^Secuencia ",
        "^Audio ",
        "^Regresar",
        "\0"};
        MENUSIS->Carga_Datos(menusis);
        MENUSIS->Titulo(" CONFIGURAR");
        MENUSIS->GetVideoMode(maxcol,mod,pag);
        pag=(maxcol-25)/2;
        MENUSIS->Mueve(pag,10);
        niv2=MENUSIS->Activa_Menu();
```

```
        switch (niv2)
                {
                case 0: MENUSIS->OcultaMenu(); ConfigSecuencia(MSECU,MVELO); break;
                case 1: MENUSIS->OcultaMenu(); ConfigSonido(MAUDI); break;
                case 2: MENUSIS->OcultaMenu(); break;
                }
        }

// Pide confirmar la salida
int CONTROL::SalirSistema(MENUHOR *MENUSAL)
        {
        short int maxcol,med,pag;
        int niv2;
        char *manual[]={
        "^Aceptar ",
        "^Cancelar",
        "\0"};
        MENUSAL->Carga_Datos(manual);
        MENUSAL->Titulo(" Desea Salir");
        MENUSAL->GetVideoMode(maxcol,med,pag);
        pag=(maxcol-19)/2;
        MENUSAL->Mueve(pag,10);
        niv2=MENUSAL->Activa_Menu();
        switch (niv2)
                {
                case 0: MENUSAL->OcultaMenu(); return -1;
                case 1: MENUSAL->OcultaMenu(); return 1;
                }
        }


// Emite un sonido por el puerto de audio
void CONTROL::Suena()
        {
        sound(sonido*800);
        delay(100);
        nosound();
        }


// Realiza las pruebas respectivas de un monitor MDA
void CONTROL::EjecutaMDA(MDA *MDAEXE)
        {
        int seleccion=0;
        int respuesta,n=3;
```

```
        while(seleccion > -1 && seleccion <= n )
                {
                switch(seleccion)
                        {
                        case 0: Suena(); MDAEXE->PintaCaracteres();; seleccion++; break;
                        case 1: Suena(); MDAEXE->AtributosCaracter(); seleccion++; break;
                        case 2: Suena(); MDAEXE->PintaRaja(); seleccion++; break;
                        case 3: Suena(); MDAEXE->PintaPantalla(); seleccion++; break;
                        }
                respuesta=Respuesta();
                if (respuesta == -1)
                        seleccion = -1;
                else
                        seleccion= seleccion - respuesta;
                        if(secuencia==1 && seleccion==n+1)seleccion=0;

                }
}

// Realiza las pruebas respectivas de un monitor CGA
void CONTROL::EjecutaCGA(CGA *CGAEXE)
        {
        int seleccion=0;
        int respuesta,n=13;
        while(seleccion > -1 && seleccion <= n )
                {
                switch(seleccion)
                        {
                        case 0: Suena(); CGAEXE->CaracteresModo40(); seleccion++; break;
                        case 1: Suena(); CGAEXE->CaracteresColor40(); seleccion++; break;
                        case 2: Suena(); CGAEXE->Atributos40(); seleccion++; break;
                        case 3: Suena(); CGAEXE->CaracteresModo80(); seleccion++; break;
                        case 4: Suena(); CGAEXE->CaracteresColor80(); seleccion++; break;
                        case 5: Suena(); CGAEXE->Atributos80(); seleccion++; break;
                        case 6: Suena(); CGAEXE->PaletaRaja(); seleccion++; break;
                        case 7: Suena(); CGAEXE->PaletaAmarillo(); seleccion++; break;
                        case 8: Suena(); CGAEXE->PaletaVerde(); seleccion++; break;
                        case 9: Suena(); CGAEXE->PaletaBlanco(); seleccion++; break;
                        case 10: Suena(); CGAEXE->Paleta0(); seleccion++; break;
                        case 11: Suena(); CGAEXE->Paleta1(); seleccion++; break;
                        case 12: Suena(); CGAEXE->Convergencia(); seleccion++; break;
                        case 13: Suena(); CGAEXE->Apariencia(); seleccion++; break;
                        }
                respuesta=Respuesta();
                if (respuesta == -1)
                        seleccion = -1;
```

```
            else
                seleccion= seleccion - respuesta;
                if(secuencia==1 && seleccion==n+1)seleccion=0;

            }
}

// Realiza las pruebas respectivas de un monitor EGA
void CONTROL::EjecutaEGA(EGA *EGAEXE)
    {
    int seleccion=0;
    int respuesta,n=12;
    while(seleccion > -1 && seleccion <= n )
        {
        switch(seleccion)
            {
            case 0: Suena(); EGAEXE->CaracteresMono40(); seleccion++; break;
            case 1: Suena(); EGAEXE->CaracteresColor40(); seleccion++; break;
            case 2: Suena(); EGAEXE->Atributos40(); seleccion++; break;
            case 3: Suena(); EGAEXE->CaracteresMono80(); seleccion++; break;
            case 4: Suena(); EGAEXE->CaracteresColor80(); seleccion++; break;
            case 5: Suena(); EGAEXE->Atributos80(); seleccion++; break;
            case 6: Suena(); EGAEXE->PurezaRoja(); seleccion++; break;
            case 7: Suena(); EGAEXE->PurezaAzul(); seleccion++; break;
            case 8: Suena(); EGAEXE->PurezaVerde(); seleccion++; break;
            case 9: Suena(); EGAEXE->PurezaBlanco(); seleccion++; break;
            case 10: Suena(); EGAEXE->ColoresPrim(); seleccion++; break;
            case 11: Suena(); EGAEXE->Convergencia(); seleccion++; break;
            case 12: Suena(); EGAEXE->Apariencia(); seleccion++; break;
            }
        respuesta=Respuesta();
        if (respuesta == -1)
            seleccion = -1;
        else
            seleccion= seleccion - respuesta;
            if(secuencia==1 && seleccion==n+1)seleccion=0;

        }
}

// Realiza las pruebas respectivas de un monitor VGA
void CONTROL::EjecutaVGA(VGA *VGAEXE)
    {
    int seleccion=0;
    int respuesta,n=12;
    while(seleccion > -1 && seleccion <= n )
```

```
        {
        switch(seleccion)
                {
                case 0: Suma(); VGAEXE->CaracteresMono40(); seleccion++; break;
                case 1: Suma(); VGAEXE->CaracteresColor40(); seleccion++; break;
                case 2: Suma(); VGAEXE->Atributos40(); seleccion++; break;
                case 3: Suma(); VGAEXE->CaracteresMono80(); seleccion++; break;
                case 4: Suma(); VGAEXE->CaracteresColor80(); seleccion++; break;
                case 5: Suma(); VGAEXE->Atributos80(); seleccion++; break;
                case 6: Suma(); VGAEXE->PuroRojo(); seleccion++; break;
                case 7: Suma(); VGAEXE->PuroAzul(); seleccion++; break;
                case 8: Suma(); VGAEXE->PuroVerde(); seleccion++; break;
                case 9: Suma(); VGAEXE->PuroBlanco(); seleccion++; break;
                case 10:Suma();  VGAEXE->ColoresPrim(); seleccion++; break;
                case 11:Suma();  VGAEXE->Convergencia(); seleccion++; break;
                case 12:Suma();  VGAEXE->Apariencia(); seleccion++; break;
                }
        respuesta=Respuesta();
        if (respuesta == -1)
                seleccion = -1;
        else
                seleccion= seleccion - respuesta;
                if(secuencia==1 && seleccion==n+1)seleccion=0;

        }
}

// Realiza las pruebas respectivas de un monitor VGA Monocromático
void CONTROL::EjecutaVGAM(VGAM *VGAMEXE)
        {
        int seleccion=0;
        int respuesta,n=4;
        while(seleccion > -1 && seleccion <= n )
                {
                switch(seleccion)
                        {
                        case 0: Suma(); VGAMEXE->CaracteresMono80(); seleccion++; break;
                        case 1: Suma(); VGAMEXE->Atributos80(); seleccion++; break;
                        case 2: Suma(); VGAMEXE->PuroBlanco(); seleccion++; break;
                        case 3:Suma();  VGAMEXE->Convergencia(); seleccion++; break;
                        case 4:Suma();  VGAMEXE->Apariencia(); seleccion++; break;
                        }
                respuesta=Respuesta();
                if (respuesta == -1)
                        seleccion = -1;
                else
```

```
                    seleccion= seleccion - respuesta;
                    if(secuencia==1 && seleccion==n+1)seleccion=0;

                }
}

// Ejecuta la prueba del monitor seleccionado
void CONTROL::EjecutaPrueba(MDA *MDAEXE,CGA *CGAEXE,EGA *EGAEXE,
                                                VGAM *VGAMEXE,VGA *VGAEXE)
        {
        switch(prueba)
                {
                case 1: EjecutaMDA(MDAEXE); break;
                case 2: EjecutaCGA(CGAEXE); break;
                case 3: EjecutaEGA(EGAEXE); break;
                case 4: EjecutaVGAM(VGAMEXE); break;
                case 5: EjecutaVGA(VGAEXE); break;
                default: break;
                }
        }


// Despliega el menú de ayuda
void CONTROL::Ayuda(MENUVER *MENUAYU)
        {
        short int maxcol,mod,pag;
        int niv2;
        char *menuayu[]={
        "^Ejecución  ",
        "^Pruebas    ",
        "^Configurar ",
        "^Regresa    ",
        "\0"};
        MENUAYU->Carga_Datos(menuayu);
        MENUAYU->Titulo(" AYUDA");
        MENUAYU->GetVideoMode(maxcol,mod,pag);
        pag=(maxcol-11)/2;
        MENUAYU->Mueve(pag,10);
        niv2=MENUAYU->Activa_Menu();
        switch (niv2)
                {
                case 0: MENUAYU->OcultaMenu();
                        AyuEjecucion(MENUAYU); break;
                case 1: MENUAYU->OcultaMenu();
                        AyuPrueba(MENUAYU); break;
                case 2: MENUAYU->OcultaMenu();
```

```
                AyuConfig(MENUAYU); break;
            case 3: MENUAYU->OcultaMenu(); break;
        }
        }
// Despliega la ayuda de Ejecución
void CONTROL::AyuEjecucion(MENUVER *MENUAYU)
        {
        short int maxcol,mod,pag;
        MENUAYU->GetVideoMode(maxcol,mod,pag);
        pag=(maxcol-38)/2;
        MENUAYU->OcultaCursor();
        MENUAYU->EscribeCaracteres(0,pag,5,"□""""""""""""""""""""""""""""""""""""""",0x1F);
        MENUAYU->EscribeCaracteres(0,pag,6,"†Para iniciar las pruebas relativas a†",0x1F);
        MENUAYU->EscribeCaracteres(0,pag,7,"†algún tipo de monitor es  suficiente†",0x1F);
        MENUAYU->EscribeCaracteres(0,pag,8,"†con seleccionar  el comando Ejecuta.†",0x1F);
        MENUAYU->EscribeCaracteres(0,pag,9,"†                                    †",0x1F);
        MENUAYU->EscribeCaracteres(0,pag,10,"†Presione alguna tecla para continuar†",0x1F);
        MENUAYU->EscribeCaracteres(0,pag,11,"□""""""""""""""""""""""""""""""""""""""",0x1F);
        getch();
        }

// Despliega la ayuda de Selección de prueba
void CONTROL::AyuPrueba(MENUVER *MENUAYU)
        {
        short int maxcol,mod,pag;
        MENUAYU->GetVideoMode(maxcol,mod,pag);
        pag=(maxcol-38)/2;
        MENUAYU->OcultaCursor();
        MENUAYU->EscribeCaracteres(0,pag,5,"□""""""""""""""""""""""""""""""""""""""",0x1F);
        MENUAYU->EscribeCaracteres(0,pag,6,"† 1.- Regrese al menú Principal.     †",0x1F);
        MENUAYU->EscribeCaracteres(0,pag,7,"† 2.- Elija  el comando  Prueba.     †",0x1F);
        MENUAYU->EscribeCaracteres(0,pag,8,"† 3.- Seleccione  una  Opción        †",0x1F);
        MENUAYU->EscribeCaracteres(0,pag,9,"†                                    †",0x1F);
        MENUAYU->EscribeCaracteres(0,pag,10,"†Presione alguna tecla para continuar†",0x1F);
        MENUAYU->EscribeCaracteres(0,pag,11,"□""""""""""""""""""""""""""""""""""""""",0x1F);
        getch();
        }

// Despliega la ayuda de Configurar
void CONTROL::AyuConfig(MENUVER *MENUAYU)
        {
        short int maxcol,mod,pag;
        MENUAYU->GetVideoMode(maxcol,mod,pag);
        pag=(maxcol-38)/2;
        MENUAYU->OcultaCursor();
        MENUAYU->EscribeCaracteres(0,pag,4,"□""""""""""""""""""""""""""""""""""""""",0x1F);
```

```
        MENUAYU->EscribeCaracteres(0,pag,5,"† 1.- Regresa al menú Principal.  †",0x1F);
        MENUAYU->EscribeCaracteres(0,pag,6,"† 2.- Elija el comando Configurar.  †",0x1F);
        MENUAYU->EscribeCaracteres(0,pag,7,"† 3.- Seleccione una Opción.       †",0x1F);
        MENUAYU->EscribeCaracteres(0,pag,8,"† Si Elige Secuencia, escoja la †",0x1F);
        MENUAYU->EscribeCaracteres(0,pag,9,"†   forma de secuencia deseada, †",0x1F);
        MENUAYU->EscribeCaracteres(0,pag,10,"†   y en caso de ser automática †",0x1F);
        MENUAYU->EscribeCaracteres(0,pag,11,"†   determine la velocidad.       †",0x1F);
        MENUAYU->EscribeCaracteres(0,pag,12,"† Si Selecciona Audio escoja alguna †",0x1F);
        MENUAYU->EscribeCaracteres(0,pag,13,"†   de las opciones.              †",0x1F);
        MENUAYU->EscribeCaracteres(0,pag,14,"†                                 †",0x1F);
        MENUAYU->EscribeCaracteres(0,pag,15,"†Presione alguna tecla para continuar†",0x1F);
        MENUAYU->EscribeCaracteres(0,pag,16,"□""""""""""""""""""""""""""""""""""""""""",0x1F);
        getch();
        }


// Selecciona en un menú una de las pruebas de monitor
void CONTROL::SelecPrueba(MENUHOR *MENUPRU)
        {
        short int maxcol,mod,pag;
        int niv2;
        char *menupru[]={
        "^Regresa",
        "^MDA",
        "^CGA",
        "^EGA",
        "^B/N_VGA",
        "^VGA",
        "\0"};
        MENUPRU->Carga_Datos(menupru);
        MENUPRU->Titulo(" PRUEBA");
        MENUPRU->GetVideoMode(maxcol,mod,pag);
        pag=(maxcol-33)/2;
        MENUPRU->Mueve(pag,10);
        prueba=MENUPRU->Activa_Menu();
        MENUPRU->OcultaMenu();
        }

// Despliega el menú principal del sistema
int CONTROL::Principal(MENUHOR *MENUPRIN,MENUHOR *MPRUE,MENUHOR
*MCONFI,MENUHOR *MSALI,
                       MENUVER *MSECU,MENUVER *MVELO,MENUVER
*MAUDI,MENUVER *MAYUD,
                       MDA *MDAEXE,CGA *CGAEXE,EGA *EGAEXE,VGAM
*VGAMEXE,VGA *VGAEXE,
                       short int modoini)
```

```
{
short int maxcol,mod,pag;
int niv1,salida=1;
char *menuprin[]={
"^Ejecuta",
"^Pruebas",
"^Config.",
"^Ayuda",
"^Salir",
"\0");
MENUPRIN->Carga_Datos(menuprin);
MENUPRIN->Titulo(" MENU PRINCIPAL");
MENUPRIN->GetVideoMode(maxcol,mod,pag);
pag=(maxcol-38)/2;
MENUPRIN->Mueve(pag,10);
FondoMenu(MENUPRIN,modoini);
niv1=MENUPRIN->Activa_Menu();

switch (niv1)
        {
        case 0: MENUPRIN->OcultaMenu();
                EjecutaPrueba(MDAEXE,CGAEXE,EGAEXE,VGAMEXE,VGAEXE);
                FondoMenu(MENUPRIN,modoini); return 1;
        case 1: MENUPRIN->OcultaMenu(); SelecPrueba(MPRUE);
                FondoMenu(MENUPRIN,modoini); return 1;
        case 2: MENUPRIN->OcultaMenu();
ConfigSistema(MCONFI,MSECU,MVELO,MAUDI);
                FondoMenu(MENUPRIN,modoini); return 1;
        case 3: MENUPRIN->OcultaMenu(); Ayuda(MAYUD);
                FondoMenu(MENUPRIN,modoini); return 1;
        case 4: MENUPRIN->OcultaMenu();salida=SalirSistema(MSALI);
                if(salida==1) return 1;
                else return -1;

        }
}
```

```
//*****************************************************************************
//*                                                                           *
//*                              Clase EGA                                    *
//*                                                                           *
//*************************************EGA.H***********************************

#ifndef EGA_H
#define EGA_H 1
#include "PANTALLA.H"

class EGA:public PANTALLA
{
public:
        EGA() {};
        // Pinta en pantalla algunos carácteres(40 x 25) en forma monocromática
        void CaracteresMono40();
        // Pinta en pantalla algunos carácteres (40 x 25) a colores
        void CaracteresColor40();
        // Pinta en pantalla carácteres (40 x 25) destacando sus atributos
        void Atributos40();
        // Pinta en pantalla algunos carácteres(80 x 25) en forma monocromática
        void CaracteresMono80();
        // Pinta en pantalla algunos carácteres (80 x 25) a colores
        void CaracteresColor80();
        // Pinta en pantalla carácteres (80 x 25) destacando sus atributos
        void Atributos80();
        // Pinta la pantalla de color rojo con resolución de 640 x 200
        void PurezaRojo();
        // Pinta la pantalla de color azul con resolución de 640 x 200
        void PurezaAzul();
        // Pinta la pantalla de color verde con resolución de 640 x 200
        void PurezaVerde();
        // Pinta la pantalla de color blanco con resolución de 640 x 200
        void PurezaBlanco();
        // Pinta la pantalla con los colores rojo,verde, azul y blanco
        // con resolución de 640 x 200
        void ColoresPrim();
        // Pinta en la pantalla una cuadrícula con puntos en el centro
        void Convergencia();
        // Pinta una cuadrícula y un círculo en el centro de la pantalla
        void Apariencia();

};

#include "EGA.CPP"
#endif
```

```
//*************************************************************************
//*                                                                       *
//*                              Clase EGA                                *
//*                                                                       *
//*********************************EGA.CPP*********************************

// Pinta en pantalla algunos carácteres(40 x 25) en forma monocromática
        void EGA::CaracteresMono40()
                {
                int modo = GetModoVideo();
                if(modo != 1) SetVideoMode(1);
                PintaCaracteresM();
                }

// Pinta en pantalla algunos carácteres (40 x 25) a colores
        void EGA::CaracteresColor40()
                {
                int modo = GetModoVideo();
                if(modo != 1) SetVideoMode(1);
                PintaCaracteresC();
                }

// Pinta en pantalla carácteres (40 x 25) destacando sus atributos
        void EGA::Atributos40()
                {
                int modo = GetModoVideo();
                if(modo != 1) SetVideoMode(1);
                AtributosCaracter();
                }


// Pinta en pantalla algunos carácteres(80 x 25) en forma monocromática
        void EGA::CaracteresMono80()
                {
                int modo = GetModoVideo();
                if(modo != 3) SetVideoMode(3);
                PintaCaracteresM();
                }

// Pinta en pantalla algunos carácteres (80 x 25) a colores
        void EGA::CaracteresColor80()
                {
                int modo = GetModoVideo();
                if(modo != 3) SetVideoMode(3);
                PintaCaracteresC();
                }
```

```
// Pinta en pantalla carácteres (80 x 25) destacando sus atributos
        void EGA::Atributos80()
                {
                int modo = GetModoVideo();
                if(modo != 3) SetVideoModo(3);
                AtributosCaracter();
                }

// Pinta la pantalla de color rojo con resolución de 640 x 200
        void EGA::PurezaRoja()
                {
                int modo = GetModoVideo();
                if(modo != 0x0E) SetVideoModo(0x0E);
                Superficie(0,0,0,640,200,0x04);
                }

// Pinta la pantalla de color azul con resolución de 640 x 200
        void EGA::PurezaAzul()
                {
                int modo = GetModoVideo();
                if(modo != 0x0E) SetVideoModo(0x0E);
                Superficie(0,0,0,640,200,0x01);
                }

// Pinta la pantalla de color verde con resolución de 640 x 200
        void EGA::PurezaVerde()
                {
                int modo = GetModoVideo();
                if(modo != 0x0E) SetVideoModo(0x0E);
                Superficie(0,0,0,640,200,0x02);
                }

// Pinta la pantalla de color blanco con resolución de 640 x 200
        void EGA::PurezaBlanco()
                {
                int modo = GetModoVideo();
                if(modo != 0x0E) SetVideoModo(0x0E);
                Superficie(0,0,0,640,200,0x0F);
                }

// Pinta la pantalla con los colores verde, rojo,azul y blanco
// con resolución de 640 x 200
        void EGA::ColoresPrim()
                {
                int modo = GetModoVideo();
```

```
        if(modo != 0x0E) SetVideoMode(0x0E);
        Superficie(0,0,0,320,50,0x04);
        Superficie(0,320,0,640,50,0x02);
        Superficie(0,160,50,480,100,0x0F);
        Superficie(0,0,150,640,200,0x01);
        Superficie(0,0,50,160,130,0x04);
        Superficie(0,480,50,640,130,0x02);
        Superficie(0,0,130,160,150,0x01);
        Superficie(0,480,130,640,150,0x01);
        }

// Pinta en la pantalla una cuadrícula con puntos en el centro
    void EGA::Convergencia()
        {
        int modo = GetModoVideo();
        if(modo != 0x10) SetVideoMode(0x10);
        double aspecto=Aspecto();
        int i=(int)5*aspecto;
        int Xmax=GetMaxCol();
        int Ymax=GetMaxRow();
        LimpiaPantalla();
        Reja(10);
        for(i;i<Xmax;i=i+(int)10*aspecto)
            for(int j=5;j<Ymax;j+=10)
                WritePixel(0x0F,0,j,i);
        }

// Pinta una cuadrícula y un círculo en el centro de la pantalla
    void EGA::Apariencia()
        {
        int modo = GetModoVideo();
        if(modo != 0x10) SetVideoMode(0x10);
        LimpiaPantalla();
        Reja(10);
        PintaCirculo(0,320,175,175);
        }
```

```
//*************************************************************************
//*                                                                       *
//*                          Clase MDA                                    *
//*                                                                       *
//*************************************MDA.H*********************************

#ifndef MDA_H
#define MDA_H 1

#include "PANTALLA.H"

class MDA:public PANTALLA
{
public:
        MDA() {};
        //Despliega en la pantalla los carácteres ASCII
        void PintaCaracteres();
        //Active un texto con los tipos de atributos disponibles
        void AtributosCaracter();
        //Dibuja en la pantalla un cuadrado en modo texto
        void PintaRaya();
        // Pinta la pantalla de color blanco
        void PintaPantalla();
};

#include "MDA.CPP"
#endif
```

```
//*****************************************************************************
//*                                                                           *
//*                              Clase MDA                                    *
//*                                                                           *
//*********************************MDA.CPP*************************************


//Despliega en la pantalla los carácteres ASCII
void MDA::PintaCaracteres(void)
        {
        int modo = GetModoVideo();
        if(modo != 7) SetVideoModo(7);
        OcultaCursor();
        PintaMargen(0);
        for(int j=1;j<24;j++)
                {
                for(int i=2;i<78;j++)
                        {
                        SetCursorPosition(0,j,i);
                        WriteChar(0,i+46,1);
                        }
                }
        }


//Activa un texto con los tipos de atributos disponibles
void    MDA::AtributosCaracter()
        {
        int modo = GetModoVideo();
        if(modo != 7) SetVideoModo(7);
        PintaMargen(0);
        int i=22;

        EscribeCaracteres(0,i,5,"Texto Normal",0x02);
        EscribeCaracteres(0,i,6,"Texto Normal que Parpadea",0x82);
        EscribeCaracteres(0,i,7,"Texto Normal Subrrayado",0x01);
        EscribeCaracteres(0,i,8,"Texto Subrayado que Parpadea",0x081);
        EscribeCaracteres(0,i,10,"Texto Brillante",0x0A);
        EscribeCaracteres(0,i,11,"Texto Brillante que Parpadea",0x8A);
        EscribeCaracteres(0,i,12,"Texto Brillante y Subrrayado",0x09);
        EscribeCaracteres(0,i,13,"Texto Brillante que Parpadea y Subrrayado",0x89);
        EscribeCaracteres(0,i,15,"Texto Inverso",0x10);
        EscribeCaracteres(0,i,17,"Texto Inverso que Parpadea",0x90);
        }
```

```
//Dibuja en la pantalla un enrejado en modo texto
void    MDA::PintaReja()
        {
        int i,j;
        int modo = GetModoVideo();
        if(modo != 7) SetVideoMode(7);
        PintaMargen(0);
        for (i=3;i<76;i=i+9)
                {
                LineaVertical(0,i,3,21);
                }
        for (i=2;i<23;i=i+4)
                {
                LineaHorizont(0,i,3,76);
                }
        for (i=2;i<20;i=i+4)
                {
                SetCursorPosition(0,i,3);
                WriteChar(0,195,1);
                SetCursorPosition(0,i,75);
                WriteChar(0,180,1);
                }
        for (i=2;i<20;i=i+4)
        for (int j=12;j<75;j=j+9)
                {
                SetCursorPosition(0,i,j);
                WriteChar(0,197,1);

                }
        for (i=3;i<76;i=i+9)
                {
                SetCursorPosition(0,2,i);
                WriteChar(0,194,1);
                SetCursorPosition(0,22,i);
                WriteChar(0,193,1);
                }
                SetCursorPosition(0,2,3);
                WriteChar(0,218,1);
                SetCursorPosition(0,2,75);
                WriteChar(0,191,1);
                SetCursorPosition(0,22,3);
                WriteChar(0,192,1);
                SetCursorPosition(0,22,75);
                WriteChar(0,217,1);
        }
```

```
// Pinta la pantalla de color blanco
void MiDA::PintaPantalla(void)
        {
        int modo = GetModoVideo();
        if(modo != 7) SetVideoModo(7);
        OcultaCursor();
        for(int j=0;j<25;j++)
                {
                SetCursorPosition(0,j,0);
                WriteCharAttribute(0,219,0x0A,80);
                }
        }
```

```
//*************************************************************************
//*                                                                       *
//*                              Clase MENU                               *
//*                                                                       *
//*************************************MENU.H*************************************

#ifndef MENU_H
#define MENU_H 1

#include "PANTALLA.H"
#include <bios.h>
#include <conio.h>
#include <ctype.h>
#include <string.h>

class MENU:public PANTALLA
{
protected:
        short int  X,Y;
        int  longitud,fondo_norm, fondo_inv,caracter_norm,caracter_inv,
                sombra,letras_esp,elementos;
        int  posicion[30];
        char *menu[30];
        char *titulo;

public:
        // Constructor por defecto
        MENU();
        // Constructor con argumento
        MENU::MENU(int,int,int,int,int,int,int,int,char *);
        // Pinta una sombra al lado derecho y abajo del menú
        void Coloca_Sombra(short int ,short int ,int);
        // Pinta el extremo superior o inferior de un marco
        void Margen(int,int,int,int,int);
        // Pinta un marco del tamaño del menú
        void Marco(int,int);
        //Escribe una cadena de carácteres en la pantalla
        int  Escribe_Menu(int,int,char *, int);
        // Busca un carácter en una cadena
        int  Esta_en(char *, char);
        // Cambia la posición del menú a otro lugar dado
        void Mueve(int,int);
        // Configura el color de los carácteres que aparecen en el menú
        void Color_Fondo_Norm(int);
        // Cambia el color del fondo inverso
        void Color_Fondo_Inv(int);
```

```
        //Modifica el color del carácter inverso normal
        void Color_Carac_Norm(int);
        // Configura el color del carácter inverso
        void Color_Carac_Inv(int);
        // Configura la opción de tener sombra
        void Sombra(int);
        // Modifica la longitud del menú
        void Longitud(int);
        // Modifica el color de las letras especiales
        void Color_Letra_Esp(int);
        // Cambia el título del menú
        void Titulo(char*);
        //Carga los nombres de las opciones del menú
        void Carga_Datos(char*pato[]);
        //Modifica el nombre de alguna de las opciones del menú
        void Cambia_Dato(int,char*);
};

#include "menu.cpp"
#endif
```

```
//*************************************************************************
//*                                                                       *
//*                          Clase MENU                                   *
//*                                                                       *
//*************************************MENU.CPP****************************

// Constructor por defecto
        MENU::MENU()
        {
        X=0;Y=0;longitud=1;fondo_norm=112;fondo_inv=32;
        caract_norm=0;caract_inv=4;sombra=1;letras_esp=4;
        elementos=0;
        titulo=" Sin Titulo";
        menu[0]="^Opcion 1   ";
        menu[1]="^Salir      ";
        menu[2]="\0";
        }
// Constructor con argumento
        MENU::MENU(int x,int y,int f_norm,int f_inv,int c_norm,
                   int c_inv,int somb,int let_esp,char *titul)
        {
        X=x; Y=y;
        fondo_norm=f_norm;fondo_inv=f_inv;
        caract_norm=c_norm;caract_inv=c_inv;
        sombra=somb;letras_esp=let_esp;
        elementos=0;
        titulo=titul;
        menu[0]="^Opcion 1   ";
        menu[1]="^Salir      ";
        menu[2]="\0";
        }


// Pinta una sombra al lado derecho y abajo del menú
void MENU::Coloca_Sombra(short int x1,short int y1,int longit)
        {
        register int i;
        char caracter,atributo=0;
        for (i=0; i<longit; i++)
                {
                SetCursorPosition(0,y1,x1++);
                caracter=ReadCharAttribute(0,(short)caracter,atributo);
                WriteCharAttribute(0,caracter,7,1);
                }
        }
```

```
// Pinta el extremo superior o inferior de un marco
void MENU::Margen(int x1,int y1,int c1,int c2,int c3)
        {
        SetCursorPosition(0,y1,x1);
        WriteCharAttribute(0,c1,carac_norm+fondo_norm,1);
        SetCursorPosition(0,y1,x1+1);
        WriteCharAttribute(0,c2,carac_norm+fondo_norm,longitud-2);
        SetCursorPosition(0,y1,x1+longitud-1);
        WriteCharAttribute(0,c3,carac_norm+fondo_norm,1);
        }

// Pinta un marco del tamaño del menú
void MENU::Marco(int x1,int y1)
        {
        int i,compara;
        longitud=x1-X+1;
        Margen(X,Y,201,205,187);
        for(i=0;i<y1-Y-1;i++)
                {
                SetCursorPosition(0,Y+i+1,X);
                WriteCharAttribute(0,186,carac_norm+fondo_norm,1);
                SetCursorPosition(0,Y+i+1,X+1);
                WriteCharAttribute(0,32,carac_norm+fondo_norm,x1-X-1);
                SetCursorPosition(0,Y+i+1,x1);
                WriteCharAttribute(0,186,carac_norm+fondo_norm,1);
                if(sombra) Coloca_Sombra(x1+1,Y+i+1,1);
                }
        Margen(X,y1,200,205,188);
        if(sombra)
                {
                Coloca_Sombra(X+longitud,y1,1);
                Coloca_Sombra(X+1,y1+1,longitud);
                }
        compara=strlen(titulo);
        i=(((x1-X)>compara)/2)+X;
        if(compara>longitud)longitud=compara;
        Escribe_Menu(i+1,Y,titulo,fondo_norm);
        }

//Escribe una cadena de caracteres en la pantalla
int  MENU::Escribe_Menu(int x1,int y1,char *mensaje, int atributo)
        {
        register short int i,j,k;
        for(i=x1;*mensaje;i++)
                {
                SetCursorPosition(0,y1,i);
```

```
            if(*mensaje=='^')
                {
                k=atributo+letras_esp;
                j=*++mensaje;
                WriteCharAttribute(0,(short)*mensaje,k,1);
                }
            else
                {
                k=atributo+caract_norm;
                *mensaje++;
                WriteCharAttribute(0,(short)*mensaje,k,1);
                }
            }
        return(j);
        }

// Busca un carácter en una cadena
int  MENU::Esta_en(char *s, char c)
        {
        register int i;
        for(i=0; *s;i++)if (toupper(*s++)==toupper(c)) return(i+1);
        return 0;
        }

// Cambia la posición del menú a otro lugar dado
void MENU::Mueve(int x1,int y1)
        {
        X=x1;
        Y=y1;
        }

// Configura el color de los caracteres que aparecen en el menú
void MENU::Color_Fondo_Norm(int color)
        {
        fondo_norm=color;
        }

// Cambia el color del fondo inverso
void MENU::Color_Fondo_Inv(int color)
        {
        fondo_inv=color;
        }
```

```
//Modifica el color del carácter inverso normal
void MENU::Color_Carac_Norm(int color)
        {
        carac_norm=color;
        }

// Configura el color del carácter inverso
void MENU::Color_Carac_Inv(int color)
        {
        carac_inv=color;
        }
// Configura la opción de tener nombre
void MENU::Nombre(int color)
        {
        nombre=color;
        }

// Modifica la longitud del menú
void MENU::Longitud(int large)
        {
        longitud=large;
        }

// Modifica el color de las letras especiales
void MENU::Color_Letra_Esp(int color)
        {
        letras_esp=color;
        }

// Cambia el título del menú
void MENU::Titulo(char *NuevoTitulo)
        {
        titulo=NuevoTitulo;
        }

//Carga los nombres de las opciones del menú
void MENU::Carga_Datos(char *opciones[])
        {
        int i;
        for(i=0;i<30;i++)
                {
                menu[i]=opciones[i];
                if (opciones[i]=="\0") i=30;
                }
        }
```

```
//Modifica el nombre de alguna de las opciones del menú
void MENU::Cambio_Dato(int i,char *nuevo_mensaje)
        {
        menu[i]=nuevo_mensaje;
        }
```

```
//*************************************************************************
//*                                                                       *
//*                          Clase MENUHOR                                *
//*                                                                       *
//*********************************MENUHOR.H*******************************

#ifndef MENUHOR_H
#define MENUHOR_H 1

#include "MENU.H"

class MENUHOR:public MENU
{
private:
        int bandera;
        short int *salvado;

public:
                MENUHOR();
                MENUHOR::MENUHOR(int,int,int,int,int,int,int,int,char *);
        //Modifica atributos de las opciones del menú en modo monocromático
        void AjustaFondo();
        // Retorna el valor asociado a la opción elegida
        int  Respuesta(int *, int,char *);
        // Investiga la posicion del selector
        void Obten_Posicion();
        // Hace visible el menú en la pantalla
        int  Activa_Menu();
        // Oculta el menú y coloca en la pantalla el texto que se encontraba antes
        void OcultaMenu();

};

#include "MENUHOR.CPP"
#endif
```

```
//*************************************************************************
//*                                                                       *
//*                          Clase MENUHOR                                *
//*                                                                       *
//*************************************MENUHOR.CPP*************************

//Modifica atributos de las opciones del menú en modo monocromático
MENUHOR::MENUHOR()
        {
        X=0;Y=0;longitud=1;fondo_norm=112;fondo_inv=32;
        caract_norm=0;caract_inv=4;nombre=1;letras_esp=4;
        elementos=0;
        bandera=0;
        titulo=" Sin Titulo";
        menu[0]="^Opcion 1   ";
        menu[1]="^Salir      ";
        menu[3]="\0";
        }

// Retorna el valor asociado a la opción elegida
MENUHOR::MENUHOR(int x,int y,int f_norm,int f_inv,int c_norm,
                 int c_inv,int nomb,int let_esp,char *titul)
        {
        X=x; Y=y;
        fondo_norm=f_norm;fondo_inv=f_inv;
        caract_norm=c_norm;caract_inv=c_inv;
        nombre=nomb;letras_esp=let_esp;
        elementos=0;
        bandera=0;
        titulo=titul;
        menu[0]="^Opcion 1   ";
        menu[1]="^Salir      ";
        menu[2]="\0";
        }

// Investiga la posicion del selector
void MENUHOR::AjusteFondo(void)
        {
        short int maxcol,mod,pag;
        GetVideoModo(maxcol,mod,pag);
        if(mod==0x07 || mod==0x11 || mod==0x06)
                {
                fondo_norm=0x00;
                fondo_inv=0x80;
                caract_norm=0x07;
                caract_inv=0x07;
```

```
                sombra=1;
                letras_esp=0x07;
                }
        }


// Hace visible el menú en la pantalla
int  MENUHOR::Respuesta(int *x, int y,char *teclas)
        {
        union inkey {
        char ch[2];
        int i; } c;
        int  tecla_elegida,flecha=0,b,fl_ant;
        Escribe_Menu(x[flecha],y,menu[flecha],fondo_inv);
        for(;;)
                {
                while(!biokey(1));
                fl_ant=flecha;
                c.i=biokey(0);
                if(c.ch[0])
                        {
                        tecla_elegida=Esta_en(teclas,toupper(c.ch[0]));
                        if(tecla_elegida)
                                {
                                flecha=tecla_elegida-1;
                                Escribe_Menu(x[fl_ant],y,menu[fl_ant],fondo_norm);
                                Escribe_Menu(x[flecha],y,menu[flecha],fondo_inv);
                                return(flecha);
                                }
                        switch(c.ch[0])
                                {
                                case '\r':
                                        Escribe_Menu(x[flecha],y,menu[flecha],fondo_inv);
                                        return flecha;
                                case ' ' : flecha++; break;
                                case 27  : return -1;
                                }
                        }
                        else
                                {
                                switch(c.ch[1])
                                        {
                                        case 75: flecha--; break;
                                        case 77: flecha++; break;
                                        case 80:
                                                Escribe_Menu(x[flecha],y,menu[flecha],fondo_inv);
```

```
                                        return flecha;
                                    }
                                }
                            if(flecha==elementos) flecha=0;
                            if(flecha<0) flecha=elementos-1;
                            if(fl_ant!=flecha)
                                {
                                Escribe_Menu(x[fl_ant],y,menu[fl_ant],fondo_norm);
                                Escribe_Menu(x[flecha],y,menu[flecha],fondo_inv);
                                }
                        }
                }

// Asigna un valor a cada elemento del menú
void MENUHOR::Obten_Posicion()
        {
        register int i;
        posicion[0]=X+1;
        for(i=1;i<elementos;i++)
                {
                posicion[i]=posicion[i-1]+strlen(menu[i-1]);
                }
        }


//Activa el menú horizontal
int  MENUHOR::Activa_Menu()
        {
        int n;
        char teclas[30];
        while(strcmp(menu[elementos],"\0")!=0) elementos++;
        Obten_Posicion();
        longitud=posicion[elementos-1]+strlen(menu[elementos-1]);
        salvado=new short int[(longitud+2-X)*4];
        SalvaPantalla(X,Y,longitud+1,Y+3,salvado);
        bandera=1;
        AjustaFondo();
        Marco(longitud,Y+2);
        for (n=0;n<elementos;n++)
                {
                teclas[n]=Escribe_Menu(posicion[n],Y+1,menu[n],fondo_norm);
                }
  teclas[n+1]='\0';
  return(Respuesta(posicion,Y+1,teclas));
}
```

```
// Oculta el menú y coloca en la pantalla el texto que se encontraba antes
void MENUHOR::OcultaMenu()
        {
        int xi=X+longitud;
        int yi=Y+3;
        if(bandera==1)
                {
                ColocaPantalla(X,Y,xi,yi,salvado);
                delete salvado;
                bandera=0;
                }
        }
```

```
//*************************************************************************
//*                                                                       *
//*                          Clase MENUVER                                *
//*                                                                       *
//*********************************MENUVER.H*******************************

#ifndef MENUVER_H
#define MENUVER_H 1

#include "MENU.H"

class MENUVER:public MENU
{
private:
        int bandera;
        short int *salvado;
public:

                MENUVER();
                MENUVER::MENUVER(int,int,int,int,int,int,int,int,char *);
        //Modifica atributos de las opciones del menú en modo monocromático
        void AjustaFondo(void);
        // Retorna el valor asociado a la opción elegida
        int  Respuesta(int,int,char *);
        // Calcula el tamaño del menú
        void Localiza_Longitud();
        // Hace visible el menú en la pantalla
        int  Activa_Menu();
        // Oculta el menú y coloca en la pantalla el texto que se encontraba antes
        void OcultaMenu();

};

#include "MENUVER.CPP"
#endif
```

```
//*****************************************************************************
//*                                                                           *
//*                              Clase MENUVER                                *
//*                                                                           *
//*****************************MENUVER.CPP*************************************

MENUVER::MENUVER()
        {
        X=0;Y=0;longitud=1;fondo_norm=112;fondo_inv=32;
        carac_norm=0;carac_inv=4;sombra=1;letras_esp=4;
        elementos=0;
        bandera=0;
        titulo=" Sin Titulo";
        menu[0]="^Opcion 1  ";
        menu[1]="^Salir     ";
        menu[3]="\0";
        }

MENUVER::MENUVER(int x,int y,int f_norm,int f_inv,int c_norm,
                 int c_inv,int somb,int let_esp,char *titul)
        {
        X=x; Y=y;
        fondo_norm=f_norm;fondo_inv=f_inv;
        carac_norm=c_norm;carac_inv=c_inv;
        sombra=somb;letras_esp=let_esp;
        elementos=0;
        bandera=0;
        titulo=titul;
        menu[0]="^Opcion 1  ";
        menu[1]="^Salir     ";
        menu[2]="\0";
        }

//Modifica atributos de las opciones del menú en modo monocromático
void MENUVER::AjustaFondo(void)
        {
        short int maxcol,mod,pag;
        GetVideoMode(maxcol,mod,pag);
        if(mod==0x07 || mod==0x11 || mod==0x06)
                {
                fondo_norm=0x00;
                fondo_inv=0x80;
                carac_norm=0x07;
                carac_inv=0x07;
                sombra=1;
```

```
                letras_esp=0x07;
                }
        }


// Retorna el valor asociado a la opción elegida
int MENUVER::Respuesta(int x, int y,char *teclas)
        {
        union inkey {
        char ch[2];
        int i; } c;
        int  tecla_elegida,flecha=0,b,fl_ant;
        Escribe_Menu(x,y+flecha,menu[flecha],fondo_inv);
        for(;;)
                {
                while(!bioskey(1));
                fl_ant=flecha;
                c.i=bioskey(0);
                if(c.ch[0])
                        {
                        tecla_elegida=Esta_en(teclas,toupper(c.ch[0]));
                        if(tecla_elegida)
                                {
                                flecha=tecla_elegida-1;
                                Escribe_Menu(x,y+fl_ant,menu[fl_ant],fondo_norm);
                                Escribe_Menu(x,y+flecha,menu[flecha],fondo_inv);
                                return(tecla_elegida-1);
                                }
                        switch(c.ch[0])
                                {
                                case '\r':
                                        Escribe_Menu(x,y+flecha,menu[flecha],fondo_inv);
                                        return flecha;
                                case ' ': flecha++; break;
                                case 27  : return -1;
                                }
                        }
                        else
                                {
                                switch(c.ch[1])
                                        {
                                        case 72: flecha--; break;
                                        case 80: flecha++; break;
                                        }
                                }
                        if(flecha==elementos) flecha=0;
```

```
                    if(flecha<0) flecha=elementos-1;
                    if(fl_ant!=flecha)
                        {
                        Escribe_Menu(x,y+fl_ant,menu[fl_ant],fondo_norm);
                        Escribe_Menu(x,y+flecha,menu[flecha],fondo_inv);
                        }
                }
        }

// Calcula el tamaño del menú
void MENUVER::Localiza_Longitud()
        {
        int i,mayor=0;
        for(i=0;i<elementos;i++)
                {
                if(strlen(menu[i])>mayor)
                        {
                        mayor=strlen(menu[i]);
                        longitud=mayor;
                        }
                }
        }


// Hace visible el menú en la pantalla
int  MENUVER::Activa_Menu()
        {
        int n;

        char teclas[30];
        while(strcmp(menu[elementos],"\0")!=0) elementos++;
        Localiza_Longitud();
        if(longitud<strlen(titulo)) longitud=strlen(titulo);
        salvado=new short int[(longitud+5)*2*(elementos+3)];
        SalvaPantalla(X,Y,X+longitud+3,Y+elementos+3,salvado);
        fondovs=1;
        AjustaFondo();
        Marco(X+longitud+1,Y+elementos+1);
        for (n=0;n<elementos;n++)
                {
                teclas[n]=Escribe_Menu(X+1,Y+n+1,menu[n],fondo_norm);
                }
  teclas[n+1]='\0';
  return(Respuesta(X+1,Y+1,teclas));
  }
```

```
// Oculta el menú y coloca en la pantalla el texto que se encontraba antes
void MENUVER::OcultaMenu()
        {
        int xi=X+longitud+1;
        int yi=Y+elementos+2;
        if(bandera==1)
                {
                ColocaPantalla(X,Y,xi,yi,salvado);
                delete salvado;
                bandera=0;
                }
        }
```

```
//*****************************************************************************
//*                                                                           *
//*                              Clase PANTALLA                               *
//*                                                                           *
//*****************************************PANTALLA.H**************************

#ifndef PANTALLA_H
#define PANTALLA_H 1

#include "videobio.h"
#include <math.h>

class PANTALLA : public VIDEOBIOS
{
public:
        PANTALLA() {};
/*  Funciones de Pantalla en modo texto     */

        // Oculta el cursor
        void OcultaCursor();
        // Muestra el cursor
        void MuestraCursor();
        // Limpia la pantalla de video
        void LimpiaPantalla();
        // Guarda en memoria un segmento de la pantalla
        void SalvaPantalla(int x1,int y1,int x2,int y2,short int pant[]);
        //Coloca en la pantalla la información que se guardo con SalvaPantalla()
        void ColocaPantalla(int x1,int y1,int x2,int y2,short int pant[]);
        // Pinta una doble linea al rededor de la pantalla visible
        void PintaMargen(int pagina);
         //Despliega en la pantalla los carácteres ASCII en modo monocromático
        void PintaCaracteresM();
        //Despliega en la pantalla los carácteres ASCII en color
        void PintaCaracteresC();
        //Activa textos con los tipos de atributos disponibles en modo monocromático
        void AtributosCaracter();
        // Escribe una cadena de carácteres en la posición indicada
        void EscribeCaracteres(int pagina,int columna,int renglon,
                                        char *mensaje,int atributo);
        // Pinta una linea vertical en la posición indicada
        void LineaVertical(int pagina, int columna, int reng_ini, int reng_fin);
        // Pinta una linea horizontal en la posición indicada
        void LineaHorizont(int pagina, int renglon, int colum_ini, int columna_fin);
        // Retorna la relación de aspecto de la pantalla
        double Aspecto(void);
```

```
        // Pinta un circulo
        void PintaCirculo(int pagina,int x,int y,int radio);
        // Pinta una reja en toda la pantalla con la separación indicada
        void Reja(int separacion);
        // Pinta una rectángulo de un algún color en la pantalla
        void Superficie(int pagina,int Xini,int Yini,
                                int Xfin,int Yfin,int color);


};

#include "PANTALLA.CPP"
#endif
```

```
//*****************************************************************************
//*                                                                           *
//*                          Clase PANTALLA                                   *
//*                                                                           *
//*****************************PANTALLA.CPP************************************

// Oculta el cursor
        void PANTALLA::OcultaCursor()
                {
                SetCursorType(0x20,0x00);
                }

// Muestra el cursor
        void PANTALLA:: MuestraCursor()
                {
                int modo=GetModoVideo();
                if(modo==7) SetCursorType(11,12);
                else SetCursorType(6,7);
                }

// Limpia la pantalla de video
        void PANTALLA::LimpiaPantalla()
                {
                int modo=GetModoVideo();
                SetVideoMode(modo);
                }

// Guarda en memoria un segmento de la pantalla
        void PANTALLA::SalvaPantalla(int x1,int y1,int x2,int y2,short int pant[])
         {
         int y,x,pos=0;
         short int  caracter,atributo;
         for(y=y1;y<=y2;y++)
          {
               for (x=x1;x<=x2;x++)
                {
                SetCursorPosition(0,y,x);
                ReadCharAttribute(0, caracter,atributo);
                pant[pos++]=caracter;
                pant[pos++]=atributo;
                }
          }
         }
```

```
//Coloca en la pantalla la información que se guardó con SalvaPantalla()
        void PANTALLA::ColocaPantalla(int x1,int y1,int x2,int y2,short int pant[])
        {
         int x,y,num=0;
         short int caracter,atributo;
         for(y=y1;y<=y2;y++)
         {
              for (x=x1;x<=x2;x++)
               {
               SetCursorPosition(0,y,x);
               caracter=pant[num++];
               atributo=pant[num++];
               WriteCharAttribute(0,caracter,atributo,1);
               }
         }
        }

// Pinta una doble linea al rededor de la pantalla visible
        void PANTALLA::PintaMargen(int pagina)
               {
               int maxreng,maxcol;
               maxreng=25;
               maxcol=GetMaxCol();
               SetCursorPosition(pagina,0,0);
               WriteCharAttribute(pagina,201,0x07,1);
               SetCursorPosition(pagina,0,1);
               WriteCharAttribute(pagina,205,0x07,maxcol-2);
               SetCursorPosition(pagina,0,maxcol-1);
               WriteCharAttribute(pagina,187,0x07,1);
               for(int i=1;i<maxreng;i++)
                 {
                      SetCursorPosition(pagina,i,0);
                      WriteCharAttribute(pagina,186,0x07,1);
                      SetCursorPosition(pagina,i,1);
                      WriteCharAttribute(pagina,32,0x07,maxcol-2);
                      SetCursorPosition(pagina,i,maxcol-1);
                      WriteCharAttribute(pagina,186,0x07,1);
                      }
               SetCursorPosition(pagina,maxreng-1,0);
               WriteCharAttribute(pagina,200,0x07,1);
               SetCursorPosition(pagina,maxreng-1,1);
               WriteCharAttribute(pagina,205,0x07,78);
               SetCursorPosition(pagina,maxreng-1,maxcol-1);
               WriteCharAttribute(pagina,188,0x07,1);
```

```
//Despliega en la pantalla los carácteres ASCII en modo monocromático
void PANTALLA::PintaCaracteresM(void)
        {
        OcultaCursor();
        int maxcol=GetMaxCol();
        int k=46;
        PintaMargen(0);

        for(int j=1;j<24;j++)
                {
                k=46;
                for(int i=2;i<maxcol-2;i++)
                        {
                        if(i>11)k=51;
                        SetCursorPosition(0,j,i);
                        WriteChar(0,i+k,1);
                        }
                }
        }


//Despliega en la pantalla los carácteres ASCII en color
void PANTALLA::PintaCaracteresC(void)
        {
        OcultaCursor();
        int maxcol=GetMaxCol();
        int k=46;
        PintaMargen(0);
        for(int j=1;j<24;j++)
                {
                k=46;
                for(int i=2;i<maxcol-2;i++)
                        {
                        if(i>11)k=51;
                        SetCursorPosition(0,j,i);
                        WriteCharAttribute(0,i+k,j%16,1);
                        }
                }
        }
```

```
//Activa textos con los tipos de atributos disponibles en modo monocromático
void    PANTALLA::AtributosCaracter()
        {
        OcultaCursor();
        PintaMargen(0);
        int maxcol=GetMaxCol();
        int i=1;
        if(maxcol>40) i=maxcol/4;
        EscribeCaracteres(0,i,5,"Texto Azul",0x01);
        EscribeCaracteres(0,i,6,"Texto Verde que Parpadea",0x82);
        EscribeCaracteres(0,i,7,"Texto Rojo",0x04);
        EscribeCaracteres(0,i,8,"Texto Blanco que Parpadea",0x087);
        EscribeCaracteres(0,i,10,"Texto Azul Brillante",0x09);
        EscribeCaracteres(0,i,11,"Texto Verde Brillante que Parpadea",0x8A);
        EscribeCaracteres(0,i,12,"Texto Rojo Brillante",0x0C);
        EscribeCaracteres(0,i,13,"Texto Blanco Brillante que Parpade",0x8F);
        EscribeCaracteres(0,i,15,"Texto Verde fondo Rojo",0x4A);
        EscribeCaracteres(0,i,16,"Texto Rojo que Parpadea, fondo Blanco",0xF4);
        EscribeCaracteres(0,i,17,"Texto Amarillo fondo Azul",0x1E);
        EscribeCaracteres(0,i,18,"Texto Blanco que Parpadea, fondo Azul",0x9F);
        }


// Escribe una cadena de caracteres en la posición indicada
        void PANTALLA::EscribeCaracteres(int pagina,int columna,int renglon,
                                         char *mensaje,int atributo)
                {
                int i;
                for(i=columna; *mensaje;i++)
                        {

                        SetCursorPosition(pagina,renglon,i);
                        WriteCharAttribute(pagina,(short)*mensaje,atributo,1);
                        *mensaje++;
                        }
                }

// Pinta una linea vertical en la posición indicada
        void PANTALLA::LineaVertical(int pagina,int columna, int reng_ini, int reng_fin)
                {
                for(int i=reng_ini;i<reng_fin+1;i++)
                  {
                        SetCursorPosition(pagina,i,columna);
                        WriteChar(pagina,179,1);
                        }
                }
```

```
// Pinta una linea horizontal en la posición indicada
        void PANTALLA::LineaHorizont(int pagina,int renglon, int colum_ini, int colum_fin)
                {
                SetCursorPosition(pagina,renglon,colum_ini);
                WriteChar(pagina,196,colum_fin-colum_ini);
                }

// Retorna la relación de aspecto de la pantalla
        double PANTALLA::Aspecto(void)
                {
                if(GetMaxCol()==320 && GetMaxRow()==200)return 1.131;
                else if(GetMaxCol()==640 && GetMaxRow()==350)return 1.292;
                else if(GetMaxCol()==640 && GetMaxRow()==200)return 2.262;
                else if(GetMaxCol()==640 && GetMaxRow()==480)return 0.942;
                else return 1;
                }

// Pinta un círculo en un puntos
        void PANTALLA::PintaCirculo(int pagina,int Xcentro,int Ycentro,int radio)
                {
                int q1x,q1y,
                q2x,q2y,
                q3x,q3y,
                q4x,q4y;
                float punto,px,py,aspecto,PASO=.001;
                aspecto=Aspecto();
                for(punto=0;punto<3.26;punto+=PASO)
                        {
                        px=cos(punto)*radio;
                        py=sin(punto)*radio;
                        px*=aspecto;
                        q1x=Xcentro-px;
                        q1y=Ycentro-py;
                        q2x=Xcentro-px;
                        q2y=Ycentro+py;
                        q3x=Xcentro+px;
                        q3y=Ycentro-py;
                        q4x=Xcentro+px;
                        q4y=Ycentro+py;
                        WritePixel(0x0F,pagina,q1y,q1x);
                        WritePixel(0x0F,pagina,q2y,q2x);
                        WritePixel(0x0F,pagina,q3y,q3x);
                        WritePixel(0x0F,pagina,q4y,q4x);
                        }
```

```
// Pinta una reja en toda la pantalla con la separación indicada
        void PANTALLA::Reja(int separacion)
                {
                int Xmax=GetMaxCol();
                int Ymax=GetMaxRow();
                double aspecto=Aspecto();

                for(int j=0;j<Ymax;j+=separacion)
                        for(int i=0;i<Xmax;i++)
                                WritePixel(0x0F,0,j,i);

                for(int i=0;i<Xmax;i=i+(int)separacion*aspecto)
                        for(int j=0;j<Ymax;j++)
                                WritePixel(0x0F,0,j,i);

                for(j=0;j<Ymax;j++)
                        WritePixel(0x0F,0,j,Xmax-1);
                for(j=0;j<Xmax;j++)
                        WritePixel(0x0F,0,Ymax-1,j);
                }

// Pinta una rectángulo de un algún color en la pantalla
        void PANTALLA::Superficie(int pagina,int Xini,int Yini,int Xfin,int Yfin,int color)
                {
                int Xmax=GetMaxCol();
                int Ymax=GetMaxRow();
                if(Xini < 0) Xini = 0;
                if(Yini < 0) Yini = 0;
                if(Xfin > Xmax) Xfin=Xmax;
                if(Yfin > Ymax) Yfin=Ymax;
                for(int i=Xini;i<Xfin+1;i++)
                        for(int j=Yini;j<Yfin;j++)
                                WritePixel(color,pagina,j,i);
                }
```

```
//*****************************************************************************
//*                                                                           *
//*                          Función Presentación                             *
//*                                                                           *
//*****************************************************************************
#ifndef PRESENTA
#define PRESENTA 1

#include <conio.h>
#include "PANTALLA.H"

// Esta función pinta la pantalla de Presentación
short int Presenta()
        {
        short int maxcol,mod,pag;
        PANTALLA panta;
        panta.GetVideoModo(maxcol,mod,pag);
        panta.SetVideoModo(mod);
        panta.PintaMargen(0);
        pag=(maxcol-25)/2;
        panta.EscribeCaracteres(0,pag,5,"Facultad de Ingeniería",0x07);
        pag=(maxcol-25)/2;
        panta.EscribeCaracteres(0,pag,9,"Sistema de Pruebas de",0x0f);
        pag=(maxcol-19)/2;
        panta.EscribeCaracteres(0,pag,10,"Equipos de Video",0x0f);
        pag=(maxcol-30)/2;
        panta.EscribeCaracteres(0,pag,14,"Autor: Mateos Santillán Edgar",0x07);
        pag=(maxcol-29)/2;
        panta.EscribeCaracteres(0,pag,15,"Asesor: Jesús Ramírez Ortega",0x07);
        pag=(maxcol-34)/2;
        panta.EscribeCaracteres(0,pag,20,"Presione una tecla para continuar ",0x07);
        getch();
        return mod;
        }

#endif
```

```
//*************************************************************************
//*                                                                       *
//*                          PROGRAMA PRINCIPAL                           *
//*                                                                       *
//*********************************PRUEBAS.CPP*****************************
#ifndef PRUEBAS
#define PRUEBAS 1

#include "PRESENTA.CPP"
#include "CONTROL.H"

main()
{
        // Declaración de variables
        int estado=1;
        short int modo;
        MDA mda;
        CGA cga;
        EGA ega;
        VGAM vgam;
        VGA vga;
        CONTROL controlador;
        MENUVER mecu,mvele,mmdi,meyud ;
        MENUHOR mprin,mprue,mconfi,mmdi;
        // Leemos del puerto la prueba seleccionada
        controlador.LeePuerto();
        // Pinta la pantalla de Presentación
        modo=Presenta();
        while (estado==1)
        {
        // Active el menú principal
        estado=controlador.Principal(&mprin,&mprue,&mconfi,&mmdi,&mecu,&mvele,
                                     &mmdi,&meyud,&mdo,&cga,&ega,&vgam,&vga,modo);
        }
        // Limpia la pantalla antes de salir
        vga.LimpiaPantalla();

}
#endif
```

```
//*************************************************************************
//*                                                                       *
//*                              Clase VGA                                *
//*                                                                       *
//*************************************VGA.H*******************************

#ifndef VGA_H
#define VGA_H 1

#include "PANTALLA.H"

class VGA:public PANTALLA
{
public:
        VGA() {};
        // Pinta en pantalla algunos caracteres (40 x 25) en forma monocromática
        void CaracteresMono40();
        // Pinta en pantalla algunos caracteres (40 x 25) a colores
        void CaracteresColor40();
        // Pinta en pantalla caracteres (40 x 25) destacando sus atributos
        void Atributos40();
        // Pinta en pantalla algunos caracteres(80 x 25) en forma monocromática
        void CaracteresMono80();
        // Pinta en pantalla algunos caracteres (80 x 25) a colores
        void CaracteresColor80();
        // Pinta en pantalla caracteres (80 x 25) destacando sus atributos
        void Atributos80();
        // Pinta la pantalla de color rojo con resolución de 640 x 200
        void PuraRoja();
        // Pinta la pantalla de color azul con resolución de 640 x 200
        void PuraAzul();
        // Pinta la pantalla de color verde con resolución de 640 x 200
        void PuraVerde();
        // Pinta la pantalla de color blanco con resolución de 640 x 200
        void PuraBlanco();
        // Pinta la pantalla con los colores rojo,verde, azul y blanco
        // con resolución de 640 x 200
        void ColoresPrim();
        // Pinta en la pantalla una cuadrícula con puntos en el centro
        void Convergencia();
        // Pinta una cuadrícula y un círculo en el centro de la pantalla
        void Apariencia();

};
#include "VGA.CPP"
#endif
```

```
//*************************************************************************
//*                                                                       *
//*                          Clase VGA                                    *
//*                                                                       *
//*************************************VGA.CPP*****************************

// Pinta en pantalla algunos caracteres (40 x 25) en forma monocromática
        void VGA::CaracteresMono40()
                {
                int modo = GetModoVideo();
                if(modo != 1) SetVideoModo(1);
                PintaCaracteresM();
                }

// Pinta en pantalla algunos caracteres (40 x 25) a colores
        void VGA::CaracteresColor40()
                {
                int modo = GetModoVideo();
                if(modo != 1) SetVideoModo(1);
                PintaCaracteresC();
                }

// Pinta en pantalla caracteres (40 x 25) destacando sus atributos
        void VGA::Atributos40()
                {
                int modo = GetModoVideo();
                if(modo != 1) SetVideoModo(1);
                AtributosCaracter();
                }


// Pinta en pantalla algunos caracteres(80 x 25) en forma monocromática
        void VGA::CaracteresMono80()
                {
                int modo = GetModoVideo();
                if(modo != 3) SetVideoModo(3);
                PintaCaracteresM();
                }

// Pinta en pantalla algunos caracteres (80 x 25) a colores
        void VGA::CaracteresColor80()
                {
                int modo = GetModoVideo();
                if(modo != 3) SetVideoModo(3);
                PintaCaracteresC();
                }
```

```
//*************************************************************************
//*                                                                       *
//*                            Clase VGA                                  *
//*                                                                       *
//*****************************VGA.CPP*********************************

// Pinta en pantalla algunos carácteres (40 x 25) en forma monocromática
        void VGA::CaracteresMono40()
                {
                int modo = GetModoVideo();
                if(modo != 1) SetVideoModo(1);
                PintaCaracteresM();
                }

// Pinta en pantalla algunos carácteres (40 x 25) a colores
        void VGA::CaracteresColor40()
                {
                int modo = GetModoVideo();
                if(modo != 1) SetVideoModo(1);
                PintaCaracteresC();
                }

// Pinta en pantalla carácteres (40 x 25) destacando sus atributos
        void VGA::Atributos40()
                {
                int modo = GetModoVideo();
                if(modo != 1) SetVideoModo(1);
                AtributosCaracter();
                }


// Pinta en pantalla algunos carácteres(80 x 25) en forma monocromática
        void VGA::CaracteresMono80()
                {
                int modo = GetModoVideo();
                if(modo != 3) SetVideoModo(3);
                PintaCaracteresM();
                }

// Pinta en pantalla algunos carácteres (80 x 25) a colores
        void VGA::CaracteresColor80()
                {
                int modo = GetModoVideo();
                if(modo != 3) SetVideoModo(3);
                PintaCaracteresC();
                }
```

```
// Pinta en pantalla carácteres (80 x 25) destacando sus atributos
        void VGA::Atributos80()
                {
                int modo = GetModoVideo();
                if(modo != 3) SetVideoModo(3);
                AtributosCaracter();
                }

// Pinta la pantalla de color rojo con resolución de 640 x 200
        void VGA::PantallaRoja()
                {
                int modo = GetModoVideo();
                if(modo != 0x13) SetVideoModo(0x13);
                Superficie(0,0,0,320,200,0x04);
                }

// Pinta la pantalla de color azul con resolución de 640 x 200
        void VGA::PantallaAzul()
                {
                int modo = GetModoVideo();
                if(modo != 0x13) SetVideoModo(0x13);
                Superficie(0,0,0,320,200,0x01);
                }

// Pinta la pantalla de color verde con resolución de 640 x 200
        void VGA::PantallaVerde()
                {
                int modo = GetModoVideo();
                if(modo != 0x13) SetVideoModo(0x13);
                Superficie(0,0,0,320,200,0x02);
                }

// Pinta la pantalla de color blanco con resolución de 640 x 200
        void VGA::PantallaBlanca()
                {
                int modo = GetModoVideo();
                if(modo != 0x13) SetVideoModo(0x13);
                Superficie(0,0,0,320,200,0x0F);
                }

// Pinta la pantalla con los colores verde, rojo,azul y blanco
// con resolución de 640 x 200
        void VGA::ColoresPrim()
                {
                int modo = GetModoVideo();
```

```
        if(modo != 0x13) SetVideoModo(0x13);
        Superficie(0,0,0,160,50,0x04);
        Superficie(0,160,0,320,50,0x02);
        Superficie(0,80,50,240,150,0x0F);
        Superficie(0,0,150,320,200,0x01);
        Superficie(0,0,50,80,130,0x04);
        Superficie(0,240,50,320,130,0x02);
        Superficie(0,0,130,80,150,0x01);
        Superficie(0,240,130,320,150,0x01);
        }


// Pinta en la pantalla una cuadricula con puntos en el centro
    void VGA::Convergencia()
        {
        int modo = GetModoVideo();
        if(modo != 0x12) SetVideoModo(0x12);
        double aspecto=Aspecto();
        int i=(int)5*aspecto;
        int Xmax=GetMaxCol();
        int Ymax=GetMaxRow();
        LimpiaPantalla();
        Raja(10);
        for(i;i<Xmax;i=i+(int)10*aspecto)
            for(int j=5;j<Ymax;j+=10)
                WritePixel(0x0F,0,j,i);
        }

// Pinta una cuadricula y un circulo en el centro de la pantalla
    void VGA::Apariencia()
        {
        int modo = GetModoVideo();
        if(modo != 0x12) SetVideoModo(0x12);
        LimpiaPantalla();
        Raja(10);
        PintaCirculo(0,320,240,240);
        }
```

```
//*************************************************************************
//*                                                                       *
//*                            Clase VGAM                                 *
//*                                                                       *
//*****************************VGAM.H**************************************

#ifndef VGMA_H !
#define VGAM_H

#include "PANTALLA.H"

class VGAM:public PANTALLA
{
public:
        VGAM() {};
        void CaracteresModo80();
        // Pinta en pantalla caracteres (80 x 25) destacando sus atributos
        void Atributos80();
        // Pinta la pantalla de color blanco con resolución de 640 x 200
        void PuntosBlancos();
        // Pinta en la pantalla una cuadrícula con puntos en el centro
        void Convergencia();
        // Pinta una cuadrícula y un círculo en el centro de la pantalla
        void Apariencia();

};

#include "VGAM.CPP"
#endif
```

```
//*************************************************************************
//*                                                                       *
//*                          Clase VGAM                                   *
//*                                                                       *
//*********************************VGAM.CPP********************************

// Pinta en pantalla algunos carácteres (80 x 25) en forma monocromática
void VGAM::CaracteresMono80()
        {
        int modo = GetModoVideo();
        if(modo != 7) SetVideoModo(7);
        OcultaCursor();
        PintaMargen(0);
        for(int j=1;j<24;j++)
                {
                for(int i=2;i<78;i++)
                        {
                        SetCursorPosition(0,j,i);
                        WriteChar(0,i+46,1);
                        }
                }
        }

// Pinta en pantalla carácteres (80 x 25) destacando sus atributos
void VGAM::Atributos80()
        {
        int modo = GetModoVideo();
        if(modo != 7) SetVideoModo(7);
        PintaMargen(0);
        int i=22;
        EscribeCaracteres(0,i,5,"Texto Normal",0x02);
        EscribeCaracteres(0,i,6,"Texto Normal que Parpadea",0x82);
        EscribeCaracteres(0,i,7,"Texto Normal Subrrayado",0x01);
        EscribeCaracteres(0,i,8,"Texto Subrayado que Parpadea",0x081);
        EscribeCaracteres(0,i,10,"Texto Brillante",0x0A);
        EscribeCaracteres(0,i,11,"Texto Brillante que Parpadea",0x8A);
        EscribeCaracteres(0,i,12,"Texto Brillante y Subrrayado",0x09);
        EscribeCaracteres(0,i,13,"Texto Brillante que Parpadea y Subrrayado",0x89);
        EscribeCaracteres(0,i,15,"Texto Inverso",0x10);
        EscribeCaracteres(0,i,17,"Texto Inverso que Parpadea",0x90);
        }

// Pinta la pantalla de color blanco con resolución de 640 x 200
void VGAM::PurezaBlanco()
        {
        int modo = GetModoVideo();
```

```
        if(modo != 0x13) SetVideoModo(0x13);
        Superficie(0,0,0,320,200,0x0F);
        }

// Pinta en la pantalla una cuadrícula con puntos en el centro
        void VGAM::Convergencia()
                {
                int modo = GetModoVideo();
                if(modo != 0x11) SetVideoModo(0x11);
                double aspecto=Aspecto();
                int i=(int)5*aspecto;
                int Xmax=GetMaxCol();
                int Ymax=GetMaxRow();
                LimpiaPantalla();
                Reja(10);
                for(i;j<Xmax;j=i+(int)10*aspecto)
                        for(int j=5;j<Ymax;j+=10)
                                WritePixel(0x0F,0,j,i);
                }

// Pinta una cuadrícula y un círculo en el centro de la pantalla
        void VGAM::Apariencia()
                {
                int modo = GetModoVideo();
                if(modo != 0x11) SetVideoModo(0x11);
                LimpiaPantalla();
                Reja(10);
                PintaCirculo(0,320,240,240);
                }
```

```
//*****************************************************************************
//*                                                                           *
//*                              Clase VIDEOBIOS                              *
//*                                                                           *
//* ********************************VIDEOBIOS.H*********************************

//Clase de video utilizando la interrupción 10h

#ifndef VIDEOBIO_H
#define VIDEOBIO_H 1

#include <dos.h>

#define VIDEO 0x10


class  VIDEOBIOS
{
private:
        int MAXR, MAXC, ModoV;
public:

        VIDEOBIOS() {};

/*  Control del interfaz de la CRT  */

        // Fija el modo de video
        void SetVideoMode(int mode);
        // Fija el tipo de cursor
        void SetCursorType(short int starting, short int ending);
        // Fija la posición del cursor
        void SetCursorPosition(short int page, short int row, short int column);
        // Obtiene la posición del cursor
        void GetCursorPosition(short int page, short int &row,
                                        short int &column);
        // Lee la posición del lápiz óptico
        short int GetLightPenPosition(short int &row, short int &column,
                                        short int &pixelrow, short int &pixelcolumn);
        // Fija la página activa
        void SetDisplayPage(short int page);
        // Desplaza la página hacia arriba
        void ScrollUp (short int lines,short int atribute,short int Yupper,
                        short int Xleft, short int Ylower, short int Xright);
        // Desplaza la página hacia abajo
        void ScrollDown(short int lines,short int atribute,short int Yupper,
                        short int Xleft, short int Ylower, short int Xright);
```

```
/*  Gestión de caracteres  */

        // Lee el carácter y atributo de la posición del cursor
        int  ReadCharAttribute(short int page, short int &character,
                                              short int &attribute);
        // Escribe un carácter y atributo en la posición del cursor
        void WriteCharAttribute(short int page, short int character,
                                              short int attribute, int count);
        // Escribe un carácter en la posición del cursor
        void WriteChar(short int page, short int character, short int count);

/*  Interfaz gráfico  */

        // Selecciona un color para el color del fondo
        void SetPalette(short int function, short int color);
        //  Pinta un pixel en la pantalla
        virtual void WritePixel(short int color, short int page, int row, int column);
        // Lee el color de un pixel en una página específica
        short int ReadPixel(short int page, int row, int column);
        // Escribe un carácter en la posición del cursor en modo Teletipo
        void WriteTtyChar(short int character, short int page, short int color);

/*  Salida de terminal ASCII  */

        // Lee el actual modo de video
        void GetVideoMode(short int &columns,short int &mode,short int &page);
        // Lee o escribe en pantalla combinaciones de código
        void GetDisplayCode(short int &primary,    short int &secundary);
        // Lee el número de renglones de la pantalla
        int  GetMaxRow(void) {return MAXR;};
        // Lee el número de columnas de la pantalla
        int  GetMaxCol(void) {return MAXC;};
        // Lee el modo de video de la pantalla
        int  GetModeVideo(void);

};
#include "VIDEOBIO.CPP"

#endif
```

```
//*****************************************************************************
//*                                                                           *
//*                              Clase VIDEOBIOS                              *
//*                                                                           *
//*****************************VIDEOBIOS.CPP*******************************
/*  Control del interfaz de la CRT  */

// Fija el modo de video
void VIDEOBIOS::SetVideoMode(int mode)
        {
        union REGS regs;
                regs.h.ah = 0x00;
                regs.h.al = mode;
                int86 (VIDEO, &regs, &regs);
                switch(mode)
                        {
                        case 0x00 : ModeV= mode; MAXC = 40; MAXR = 25; break;
                        case 0x01 : ModeV= mode; MAXC = 40; MAXR = 25; break;
                        case 0x02 : ModeV= mode; MAXC = 80; MAXR = 25; break;
                        case 0x03 : ModeV= mode; MAXC = 80; MAXR = 25; break;
                        case 0x04 : ModeV= mode; MAXC =320; MAXR =200; break;
                        case 0x05 : ModeV= mode; MAXC =320; MAXR =200; break;
                        case 0x06 : ModeV= mode; MAXC =640; MAXR =200; break;
                        case 0x07 : ModeV= mode; MAXC = 80; MAXR = 25; break;
                        case 0x0D : ModeV= mode; MAXC =320; MAXR =200; break;
                        case 0x0E : ModeV= mode; MAXC =640; MAXR =200; break;
                        case 0x0F : ModeV= mode; MAXC =640; MAXR =350; break;
                        case 0x10 : ModeV= mode; MAXC =640; MAXR =350; break;
                        case 0x11 : ModeV= mode; MAXC =640; MAXR =480; break;
                        case 0x12 : ModeV= mode; MAXC =640; MAXR =480; break;
                        case 0x13 : ModeV= mode; MAXC =320; MAXR =200; break;
                        }
        }

// Fija el tipo de cursor
void VIDEOBIOS::SetCursorType(short int starting, short int ending)
        {
        union REGS regs;
                regs.h.ah = 0x01;
                regs.h.ch = starting;
          //    regs.h.ch &= 0x0F;
                regs.h.cl = ending;
                regs.h.cl &= 0x0F;
                int86 (VIDEO, &regs, &regs);
        }
```

```
// Fija la posición del cursor
void VIDEOBIOS::SetCursorPosition(short int page, short int row, short int column)
        {
        union REGS regs;
                regs.h.ah = 0x02;
                regs.h.bh = page;
                regs.h.dh = row;
                regs.h.dl = column;
                int86 (VIDEO, &regs, &regs);
        }

// Obtiene la posición del cursor
void VIDEOBIOS::GetCursorPosition(short int page, short int &row, short int &column)
        {
        union REGS regs;
                regs.h.ah = 0x03;
                regs.h.bh = page;
                int86 (VIDEO, &regs, &regs);
                row = regs.h.dh;
                column = regs.h.dl;
        }

// Lee la posición del lápiz óptico
short int VIDEOBIOS::GetLightPenPosition(short int &row, short int &column,
                                         short int &pixelrow, short int &pixelcolumn)
        {
        union REGS regs;
                regs.h.ah = 0x04;
                int86(VIDEO, &regs, &regs);
                row = regs.h.dh;
                column = regs.h.dl;
                pixelrow = regs.h.ch;
                pixelcolumn = regs.h.cl;
                return (regs.h.ah);  /* AH = 0 ( no lápiz óptico) */
        }

// Fija la página activa
void VIDEOBIOS::SetDisplayPage(short int page)
        {
        union REGS regs;
                regs.h.ah = 0x05;
                regs.h.al = page;
                int86(VIDEO, &regs, &regs);
        }
```

```
// Desplaza la página hacia arriba
void VIDEOBIOS::ScrollUp (short int lines,short int attribute,short int Yupper,
                          short int Xleft, short int Ylower, short int Xright)
        {
        union REGS regs;
                regs.h.ah = 0x06;
                regs.h.al = lines;
                regs.h.bh = attribute;
                regs.h.ch = Yupper;
                regs.h.cl = Xleft;
                regs.h.dh = Ylower;
                regs.h.dl = Xright;
                int86(VIDEO, &regs, &regs);
        }

// Desplaza la página hacia abajo
void VIDEOBIOS::ScrollDown(short int lines,short int attribute,short int Yupper,
                           short int Xleft, short int Ylower, short int Xright)
        {
        union REGS regs;
                regs.h.ah = 0x07;
                regs.h.al = lines;
                regs.h.bh = attribute;
                regs.h.ch = Yupper;
                regs.h.cl = Xleft;
                regs.h.dh = Ylower;
                regs.h.dl = Xright;
                int86(VIDEO, &regs, &regs);
        }

/*  Gestión de caracteres  */

// Lee el carácter y atributo de la posición del cursor
int  VIDEOBIOS::ReadCharAttribute(short int page, short int &character,
                                  short int &attribute)
        {
        union REGS regs;
        int i;
                regs.h.ah = 0x08;
                regs.h.bh = page;
                i=int86(VIDEO, &regs, &regs);
                character = regs.h.al;
                attribute = regs.h.ah;
                return(i);
        }
```

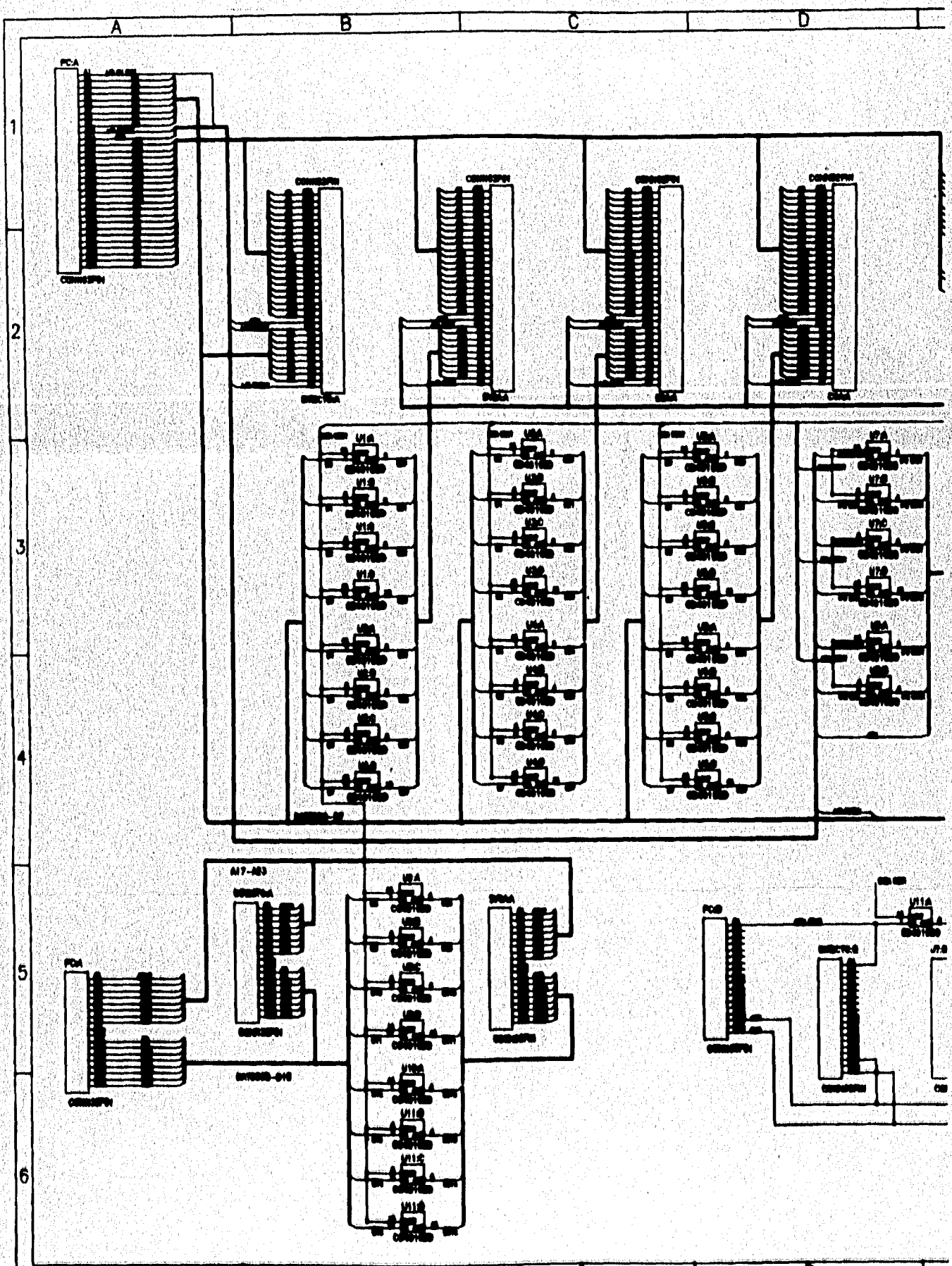
```
// Escribe un carácter y atributo en la posición del cursor
void VIDEOBIOS::WriteCharAttribute(short int page, short int character,
                                   short int attribute, int count)
        {
        union REGS regs;
                regs.h.ah = 0x09;
                regs.h.al = character;
                regs.h.bh = page;
                regs.h.bl = attribute;
                regs.x.cx = count;
                int86(VIDEO, &regs, &regs);
        }

// Escribe un carácter en la posición del cursor
void VIDEOBIOS::WriteChar(short int page, short int character, short int count)
        {
        union REGS regs;
                regs.h.ah = 0x0A;
                regs.h.al = character;
                regs.h.bh = page;
                regs.x.cx = count;
                int86(VIDEO, &regs, &regs);
        }

/*  Interfaz gráfico  */

// Selecciona un color para el color del fondo
void VIDEOBIOS::SetPalette(short int function, short int color)
        {
        union REGS regs;
                regs.h.ah = 0x0B;
                regs.h.bh = function;
                regs.h.bl = color;
                int86(VIDEO, &regs, &regs);
        }

//  Pinta un pixel en la pantalla
void VIDEOBIOS::WritePixel(short int color, short int page, int row, int column)
        {
        char far *ApVideo;
        int addr,i,modo=GetModoVideo();
        unsigned int nbit;
        int maxcol=GetMaxCol();
        switch(modo)
        {
          case 0x12 :
```

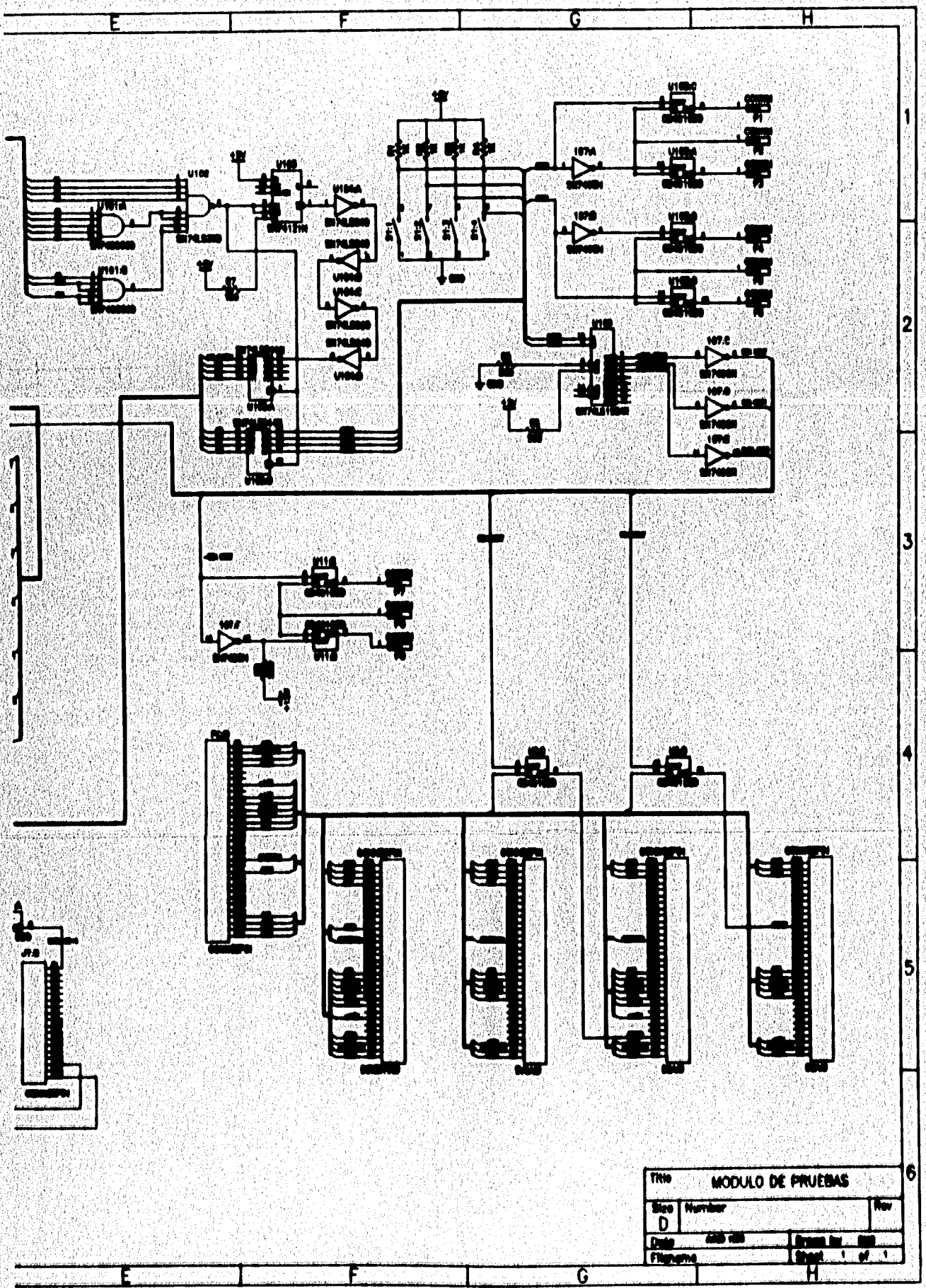
```
                    ApVideo = (char far *) 0xA0000000L;
                    addr=(row*80)+column/8;
                    nbit=0x80;
                    nbit=nbit>>(column%8);
                    outportb(0x3ce,5);
                    outportb(0x3cf,0x00);
                    outportb(0x3ce,3);
                    outportb(0x3cf,0x00);
                    outportb(0x3ce,1);
                    outportb(0x3cf,0x0f);
                    outportb(0x3ce,8);
                    outportb(0x3cf,nbit);
                    outportb(0x3ce,0);
                    outportb(0x3cf,color);
                    *(ApVideo+addr) |=nbit;
                    outportb(0x3ce,8);    /* restablece condiciones de la tarjeta de video */
                    outportb(0x3cf,0xff);
                    outportb(0x3ce,1);
                    outportb(0x3cf,0x00);
                    outportb(0x3ce,0);
                    outportb(0x3cf,0x00);
                    break;
        case 0x13 :
                    ApVideo = (char far *) 0xA0000000L;
                    addr=(row*maxcol)+column;
                    *(ApVideo+addr) = color;
                    break;

        default:
                    union REGS regs;
                    regs.h.ah = 0x0C;
                    regs.h.al = color;
                    regs.h.bh = page;
                    regs.x.cx = column;
                    regs.x.dx = row;
                    int86(VIDEO, &regs, &regs);
                    break;
            }
    }

// Lee el color de un pixel en una página específica
short int VIDEOBIOS::ReadPixel(short int page, int row, int column)
    {
    union REGS regs;
            regs.h.ah = 0x0D;
            regs.h.bh = page;
```

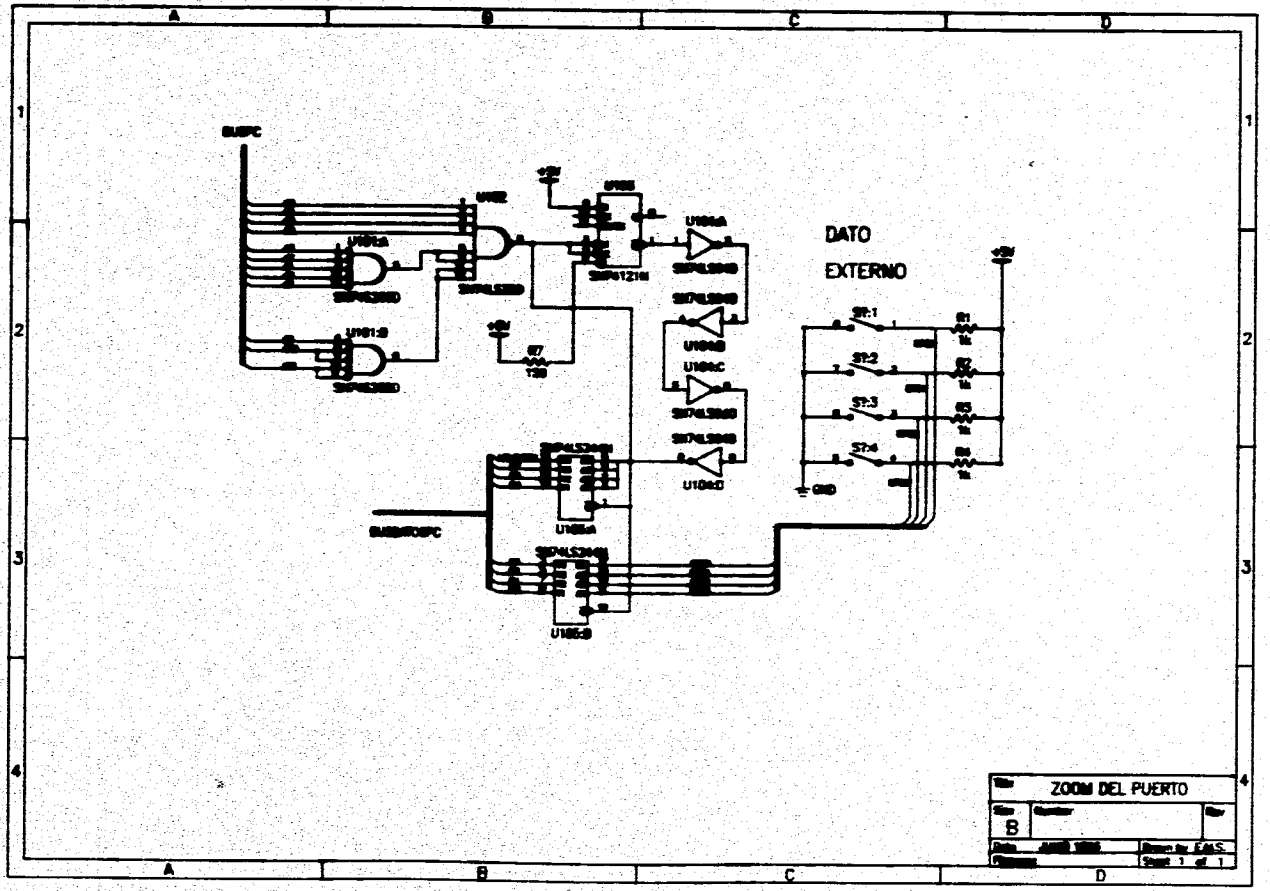
```
            regs.x.cx = column;
            regs.x.dx = row;
            int86(VIDEO, &regs, &regs);
            return(regs.h.al);
        }

// Escribe un carácter en la posición del cursor en modo Teletipo
void VIDEOBIOS::WriteTtyChar(short int character, short int page, short int color)
        {
        union REGS regs;
            regs.h.ah = 0x0E;
            regs.h.al = character;
            regs.h.bh = page;
            regs.h.bl = color;
            int86(VIDEO,&regs, &regs);
        }

/*  Salida de terminal ASCII  */

// Lee el actual modo de video
void VIDEOBIOS::GetVideoMode(short int &columns,short int &mode,short int &page)
        {
        union REGS regs;
            regs.h.ah = 0x0F;
            int86(VIDEO, &regs, &regs);
            columns = regs.h.ah;
            mode = regs.h.al;
            page = regs.h.bh;
            ModoV = mode;
        }

// Lee o escribe en pantalla combinaciones de código
void VIDEOBIOS::GetDisplayCode(short int &primary, short int &secundary)
        {
        union REGS regs;
            regs.h.ah = 0x1A;
            regs.h.al = 0x00;
            int86(VIDEO, &regs, &regs);
            if (regs.h.al == 0x1A)   //AL = 0x1A si la función es soportada
              {
              primary = regs.h.bl;
              secundary = regs.h.bh;
              }
            else
              {
              primary = 0xFF;
```

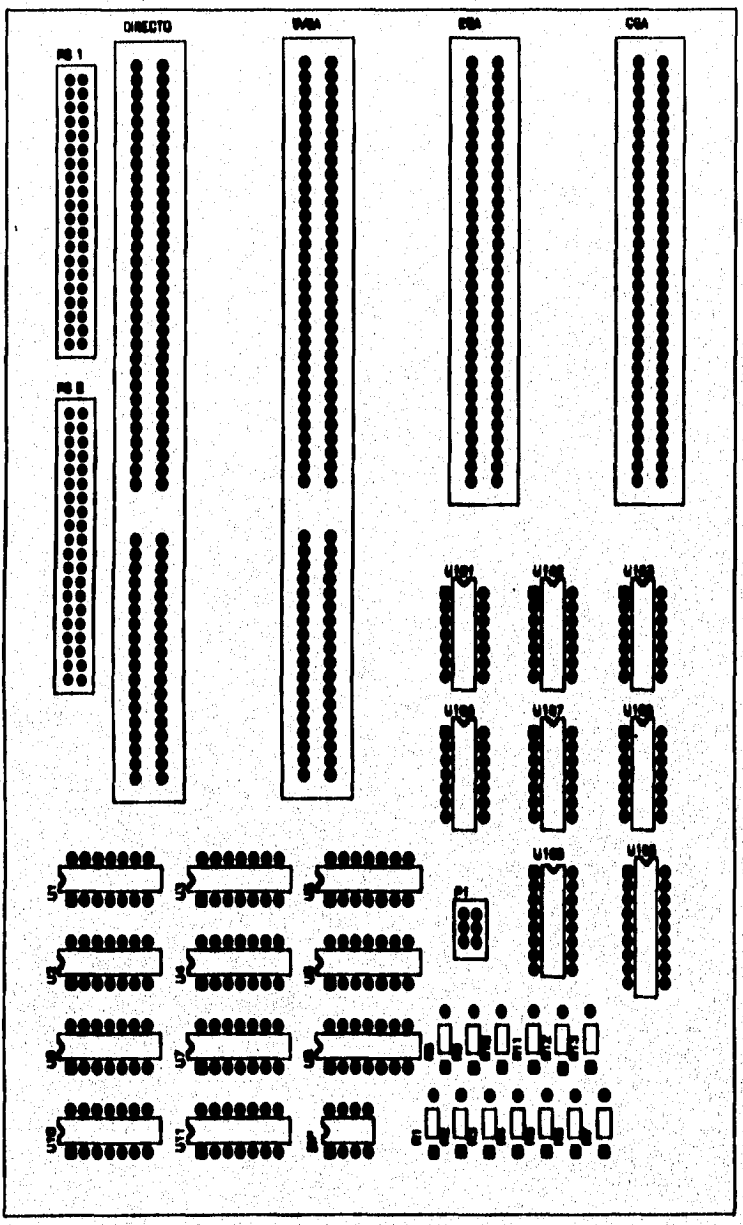
```
                secundary = 0xFF;
                }
        }

// Lee el modo de video de la pantalla
int  VIDEOBIOS::GetModoVideo(void)
        {
        short int maxcol,mod,pag;
        GetVideoMode(maxcol,mod,pag);
        return ModoV;
        }
```

APÉNDICE III


A
B
C
D
1
2
3
4
5
6


E
F
G
H
1
2
3
4
5
6
MODULO DE PRUEBAS
Size
D
Number
Rev
Date
File


DATO
EXTERNO
ZOOM DEL PUERTO
B
Sheet 1 of 1



UBICACION DE LOS COMPONENTES

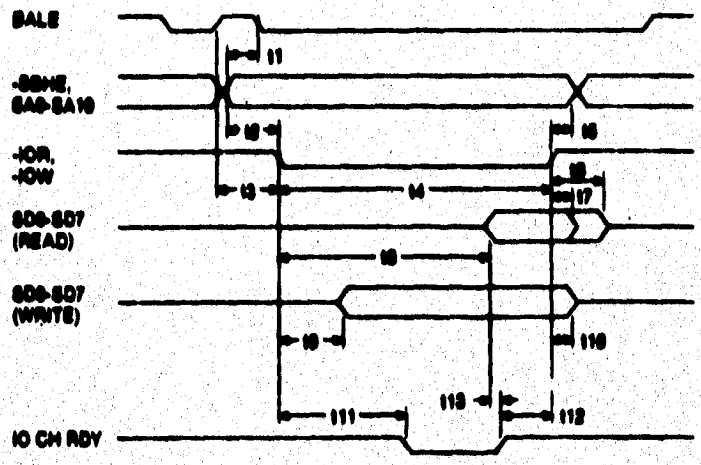
A.III.4

APÉNDICE IV

Signal Timings

The following diagrams show the signal timings for I/O and memory operations.

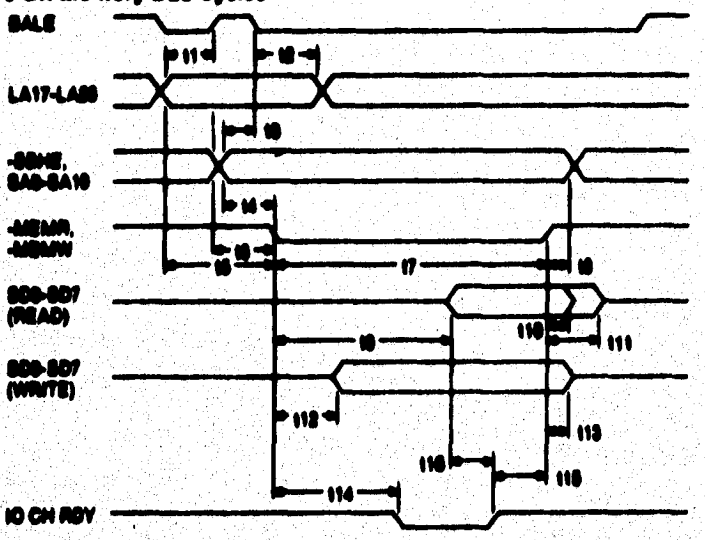
8-Bit I/O Bus Cycles



Symbol	Description	Min (ns)	Max (ns)
t1	-SBHE, SA0 – SA19 valid to BALE inactive	20	
t2	IOR/W active from SA0 – SA19 valid	75	
t3	IOR/W active from BALE active	65	
t4	IOR/W pulse width	640	
t5	SA0 – SA19 hold from IOR/W inactive	40	
t6	Read Data valid from IOR active		595
t7	Read Data hold from IOR inactive	0	
t8	Read Data bus tri-state from IOR inactive		60
t9	Write Data valid from IOW active		55
t10	Write Data hold from IOW inactive	50	
t11	I/O CH RDY inactive from IOR/W active		305
t12	IOR/W inactive from I/O CH RDY active	125	
t13	Read Data valid to I/O CH RDY active	0	

Figure 2. 8-Bit I/O Bus Cycles

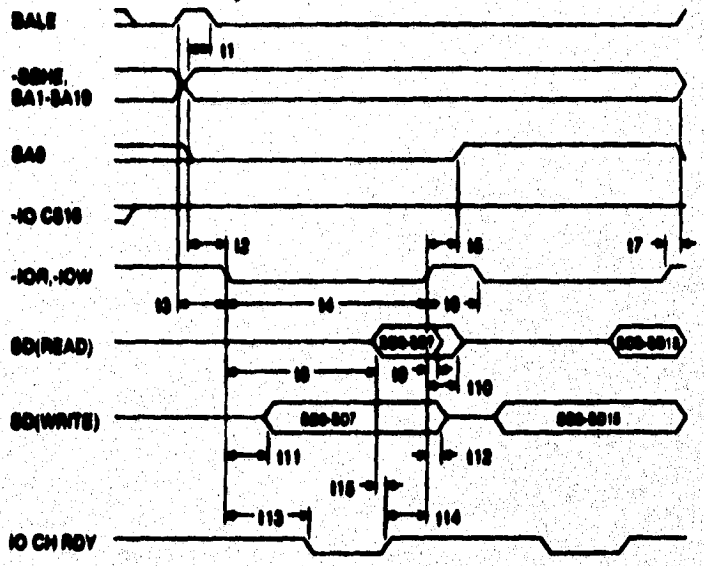
8-Bit Memory Bus Cycles



Symbol	Description	Min (ns)	Max (ns)
I1	LA17 – LA00 valid to BALE active	60	
I2	LA17 – LA00 hold from BALE inactive	30	
I3	-SBHE, SA0 – SA19 valid to BALE inactive	30	
I4	MEMR/W active from SA0 – SA19 valid	70	
I5	MEMR/W active from LA17 – LA00 valid	110	
I6	MEMR/W active from BALE active	95	
I7	MEMR/W pulse width	540	
I8	SA0 – SA19 hold from MEMR/W inactive	40	
I9	Read Data valid from MEMR active		500
I10	Read Data hold from MEMR inactive	0	
I11	Read Data bus tri-state from MEMR inactive		60
I12	Write Data valid from MEMW active		60
I13	Write Data hold from MEMW inactive	60	
I14	I/O CH RDY inactive from MEMR/W active		300
I15	MEMR/W inactive from I/O CH RDY active	125	
I16	Read Data valid to I/O CH RDY active	0	

Figure 3. 8-Bit Memory Bus Cycles

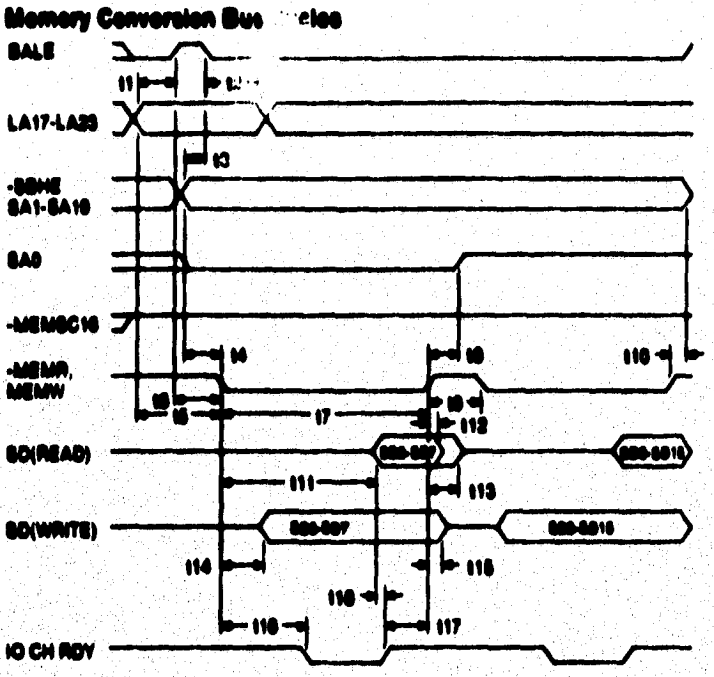
I/O Conversion Bus Cycle



Symbol	Description	Min (ns)	Max (ns)
t1	-SBHE/SA1 – SA19 valid to BALE inactive	20	
t2	IOR/W active from SA1 – SA19 valid	80	
t3	IOR/W active from BALE active	65	
t4	IOR/W pulse width	540	
t5	SA0 valid from IOR/W inactive	65	105
t6	IOR/W active from IOR/W inactive	165	210
t7	SA0 – SA19 hold from IOR/W inactive	40	
t8	Read Data valid from IOR active		500
t9	Read Data hold from IOR inactive	0	
t10	Read Data bus tri-state from IOR inactive		60
t11	Write Data valid from IOW active		65
t12	Write Data hold from IOW inactive	50	
t13	I/O CH RDY inactive from IOR/W active		305
t14	IOR/W inactive from I/O CH RDY active	125	
t15	Read Data valid to I/O CH RDY active	0	

Figure 4. I/O Conversion Bus Cycle

Note: These timings are provided for operations involving 16-bit transfers to and from 8-bit slaves.

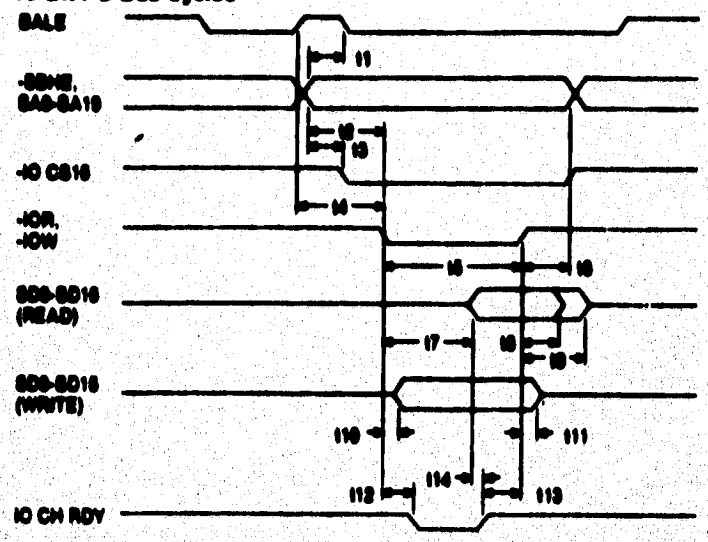


Symbol	Description	Min (ns)	Max (ns)
t1	LA17 – LA23 valid to SALE active	20	
t2	LA17 – LA23 hold from SALE inactive	20	
t3	-SBHE/SA1 – SA19 valid to SALE inactive	20	
t4	MEMR/W active from SA1 – SA19 valid	20	
t5	MEMR/W active from LA17 – LA23 valid	110	
t6	MEMR/W active from SALE active	20	
t7	MEMR/W pulse width	240	
t8	SA0 valid from MEMR/W inactive	20	100
t9	MEMR/W active from MEMR/W inactive	100	210
t10	SA1 – SA19 hold from MEMR/W inactive	40	
t11	Read Data valid from MEMR active		200
t12	Read Data valid from MEMR inactive	0	
t13	Read Data bus tri-state from MEMR inactive		20
t14	Write Data valid from MEMW active		20
t15	Write Data valid from MEMW inactive	20	
t16	I/O CH RDY inactive from MEMR/W active		200
t17	MEMR/W inactive from I/O CH RDY active	125	
t18	Read Data valid to I/O CH RDY active	0	

Figure 5. Memory Conversion Bus Cycles

Note: These timings are provided for operations involving 16-bit transfers to and from 8-bit slaves.

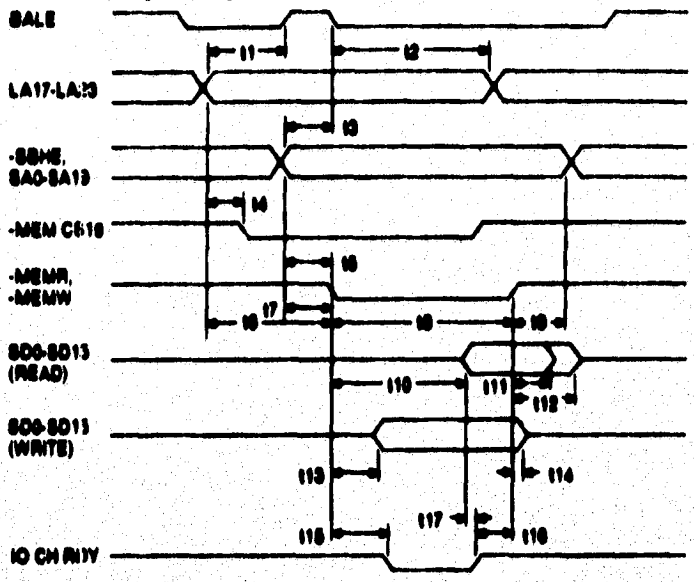
16-Bit I/O Bus Cycles



Symbol	Description	Min (ns)	Max (ns)
I1	SA0 – SA19 valid to BALE inactive	20	
I2	IOR/W active from SA0 – SA19 valid	75	
I3	I/O CS16 active from SA0 – SA19 valid		90
I4	IOR/W active from BALE active	80	
I5	IOR/W pulse width	165	
I6	SA0 – SA19 hold from IOR/W inactive	45	
I7	Read Data valid from IOR active		120
I8	Read Data hold from IOR inactive	0	
I9	Read Data bus tri-state from IOR inactive		60
I10	Write Data valid from IOW active		65
I11	Write Data hold from IOW inactive	60	
I12	I/O CH RDY inactive from IOR/W active		90
I13	IOR/W inactive from I/O CH RDY active	135	
I14	Read Data valid to I/O CH RDY active	0	

Figure 6. 16-Bit I/O Bus Cycles

16-Bit Memory Bus Cycles

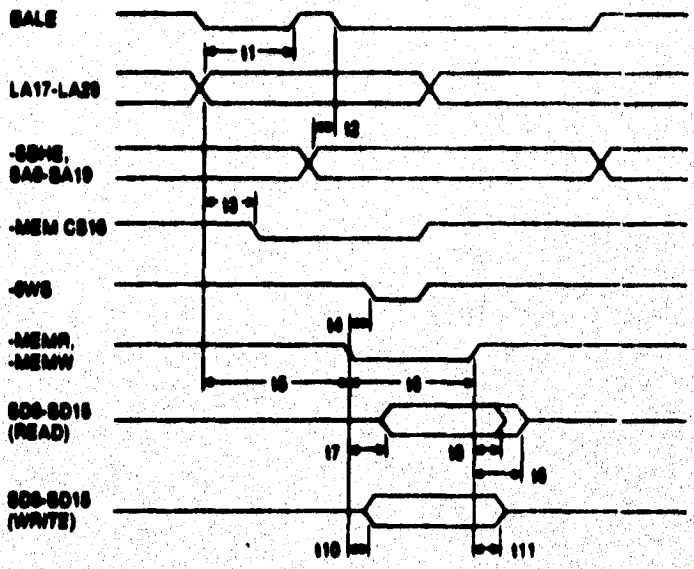


Symbol	Description	Min (ns)	Max (ns)
t1	LA17 – LA23 valid to SALE active	60	
t2	LA17 – LA23 hold from SALE inactive	35	
t3	SA0 – SA19 valid to SALE inactive	20	
t4	MEM CS16 active from LA17 – LA23 valid		80
t5 *	MEMR/W active from SA0 – SA19 valid	15	
t6	MEMR/W active from LA17 – LA23 valid	110	
t7	MEMR/W active from SALE active	25	
t8	MEMR/W pulse width	230	
t9	SA0 – SA19 hold from MEMR/W inactive	45	
t10	Read Data valid MEMR active		185
t11	Read Data hold from MEMR inactive	0	
t12	Read Data bus tri-state from MEMR inactive		60
t13	Write Data valid from MEMW active		35
t14	Write Data hold from MEMW inactive	60	
t15	I/O CH RDY inactive from MEMR/W active		85
t16	MEMR/W inactive from I/O CH RDY active	125	
t17	Read Data valid to I/O CH RDY active	0	

* This value is the minimum timing with respect to SA0.
The minimum timing with respect to SA1 – SA19 is 30 ns.

Figure 7. 16-Bit Memory Bus Cycles (1 Wait State)

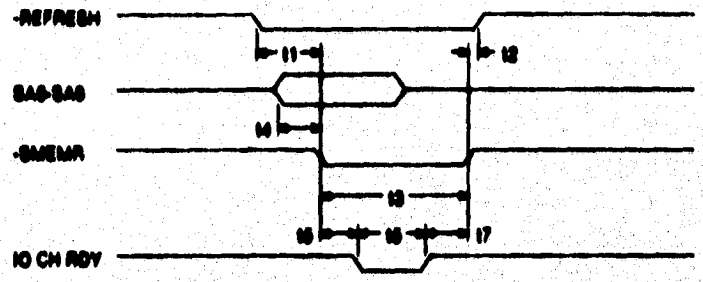
Fast 16-Bit Memory Bus Cycles



Symbol	Description	Min (ns)	Max (ns)
t1	LA17 – LA23 valid to BALE active	60	
t2	-SBHE, SA0 – SA19 valid to BALE inactive	20	
t3	MEM CS16 active from LA17 – LA23 valid		60
t4	-IOCS16 active from MEMR/W active		30
t5	MEMR/W active from LA17 – LA23 valid	115	
t6	MEMR/W pulse width		190
t7	Read Data valid from MEMR active		70
t8	Read Data hold MEMR inactive	0	
t9	Read Data bus tri-state from MEMR inactive		60
t10	Write Data valid from MEMW active		35
t11	Write Data hold from MEMW inactive	60	

Figure 8. 16-Bit Memory Bus Cycles (0 Wait State)

Memory Refresh Cycles



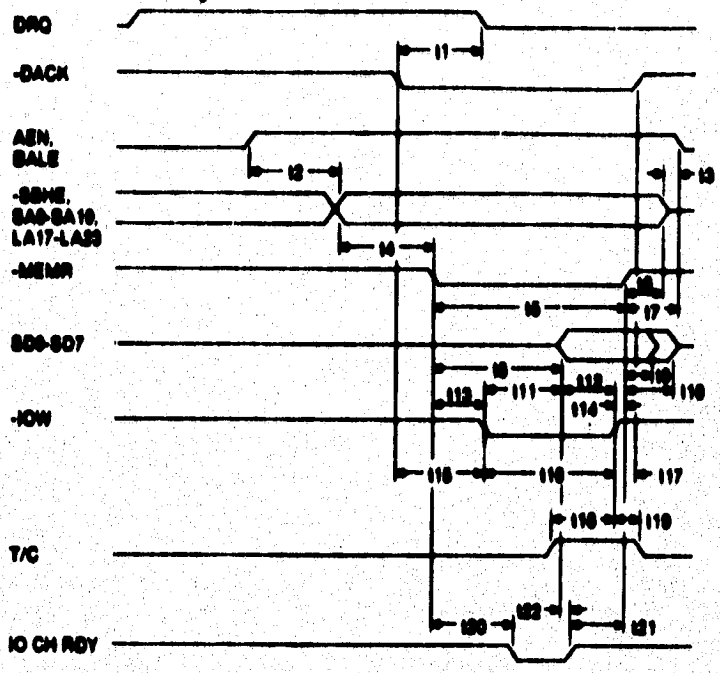
Symbol	Description	Min (ns)	Max (ns)
t1	-REFRESH set up to -MEMR active	166	
t2	-REFRESH hold from -MEMR inactive	10	
t3	-MEMR pulse width	833	
t4	-MEMR active from SA0 – SA9 valid	76	
t5	I/O CH RDY inactive from MEMR active		66
t6	I/O CH RDY pulse width		666
t7	-MEMR inactive from I/O CH RDY active	0	

Figure 9. Memory Refresh Cycles

Notes:

1. REFRESH may also be initiated by a bus master. In this case, the timing parameters for the REFRESH signal must be observed by the initiating bus master.
2. SA0 – SA9 and -SMEMR are generated by the system logic regardless of the origin of the REFRESH signal.
3. A memory refresh cycle must occur every 15 microseconds in order to maintain system memory integrity.

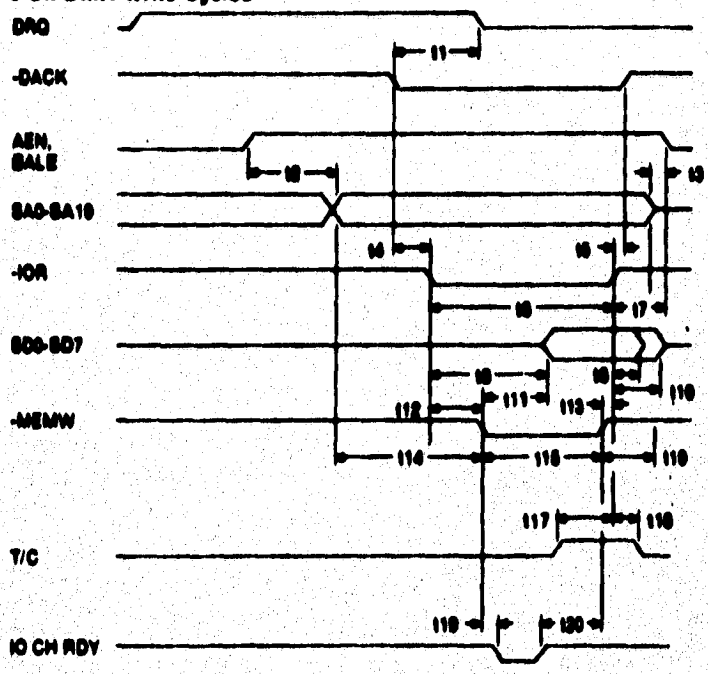
8-Bit DMA Read Cycles



Symbol	Description	Min (ns)	Max (ns)
t1	DRQ inactive from DACK active	0	
t2	SA0 – SA19 valid from AEN active	215	
t3	AEN inactive from SA0 – SA19 invalid	0	
t4	MEMR active from SA0 – SA19 valid	0	
t5	MEMR pulse width	470	
t6	SA0 – SA19 hold from MEMR inactive	20	
t7	AEN inactive from MEMR inactive	0	
t8	Data valid from MEMR active		360
t9	Data hold from MEMR inactive	0	
t10	Data bus tri-state from MEMR inactive		60
t11	Data valid from IOW active		120
t12	Data valid setup to IOW inactive	130	
t13	IOW active from MEMR active		360
t14	IOW inactive to MEMR inactive	0	
t15	IOW active from DACK active	200	
t16	IOW pulse width	250	
t17	DACK inactive from IOW inactive	0	
t18	T/C active setup to IOW inactive	300	
t19	T/C inactive from IOW inactive	0	
t20	I/O CH RDY inactive from MEMR active		200
t21	MEMR inactive from I/O CH RDY active	200	
t22	Data valid to I/O CH RDY active	0	

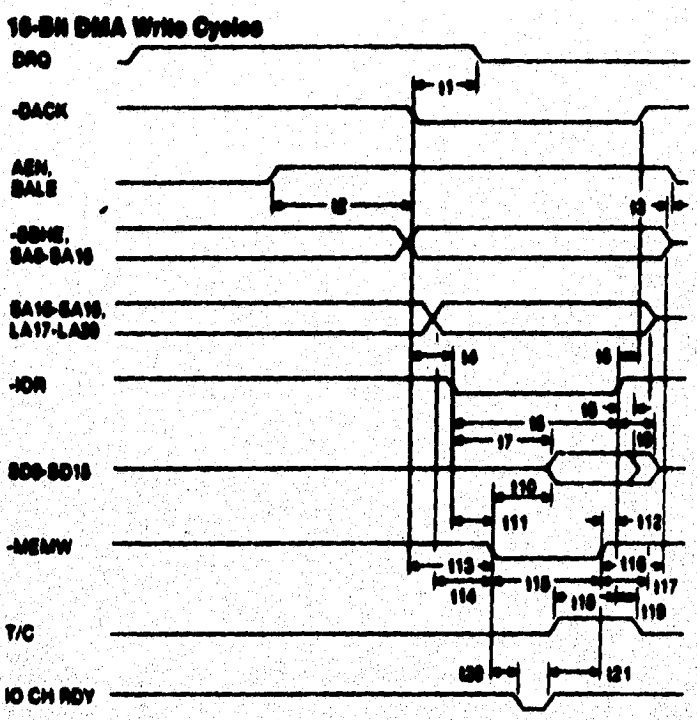
Figure 11. 8-Bit DMA Read Cycles

8-Bit DMA Write Cycles



Symbol	Description	Min (ns)	Max (ns)
t1	DRQ inactive from DACK active	0	
t2	SA0 – SA19 valid from AEN inactive	215	
t3	AEN inactive from SA0 – SA19 invalid	0	
t4	IOR active from DACK active	0	
t5	DACK inactive from IOR inactive	0	
t6	IOR pulse width	470	
t7	AEN inactive from IOR inactive	0	
t8	Data valid from IOR active		175
t9	Data hold from IOR inactive	0	
t10	Data bus tri-state from IOR inactive		80
t11	Data valid from MEMW active		120
t12	MEMW active from IOR active	85	
t13	MEMW inactive to IOR inactive	0	
t14	MEMW active from SA0 – SA19 valid	140	
t15	MEMW pulse width	360	
t16	SA0 – SA19 hold from MEMW inactive	45	
t17	T/C active setup to IOR inactive	290	
t18	T/C inactive from IOR inactive	0	
t19	I/O CH RDY inactive from MEMW active		30
t20	MEMW inactive from I/O CH RDY active	200	

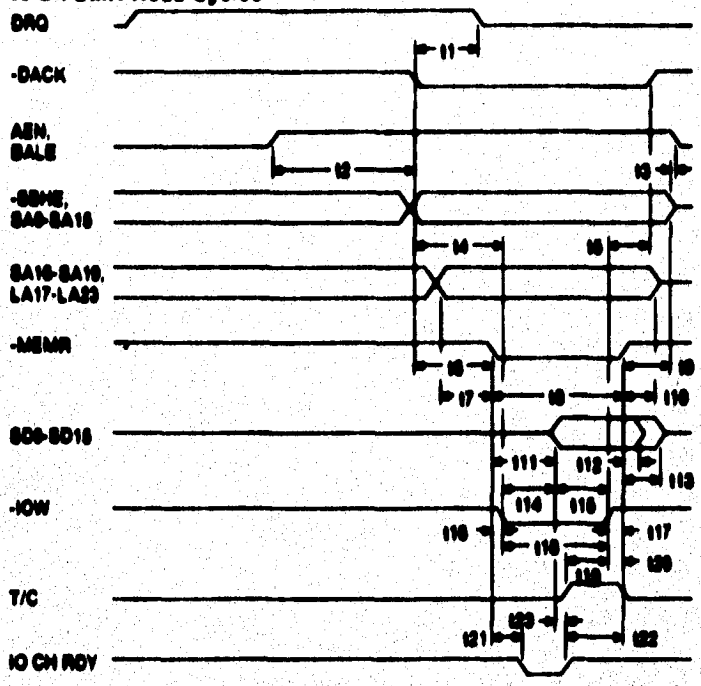
Figure 10. 8-Bit DMA Write Cycles



Symbol	Description	Min (ns)	Max (ns)
t1	DRQ inactive from DACK active	0	
t2	SA0 – SA19 valid from AEN active	675	
t3	AEN inactive from SA0 – SA19 invalid	0	
t4	IOR active from DACK active	0	
t5	DACK inactive from IOR inactive	0	
t6	IOR pulse width	885	
t7	Data valid from IOR active		165
t8	Data hold from IOR inactive	0	
t9	Data bus tri-state from IOR inactive		60
t10	Data valid from MEMW active		70
t11	MEMW active from IOR active	90	
t12	MEMW inactive to IOR inactive	0	
t13	MEMW active from SA0 – SA19 valid	130	
t14	MEMW active from LA17 – LA23, SA16 – SA23 valid	130	
t15	MEMW pulse width	330	
t16	SA0 – SA19 hold from MEMW inactive	135	
t17	LA17 – LA23, SA16 – SA19 hold from MEMW inactive	40	
t18	T/C active setup to IOR inactive	350	
t19	T/C inactive from IOR inactive	0	
t20	I/O CH RDY inactive from MEMW active		75
t21	MEMW inactive from I/O CH RDY active	300	

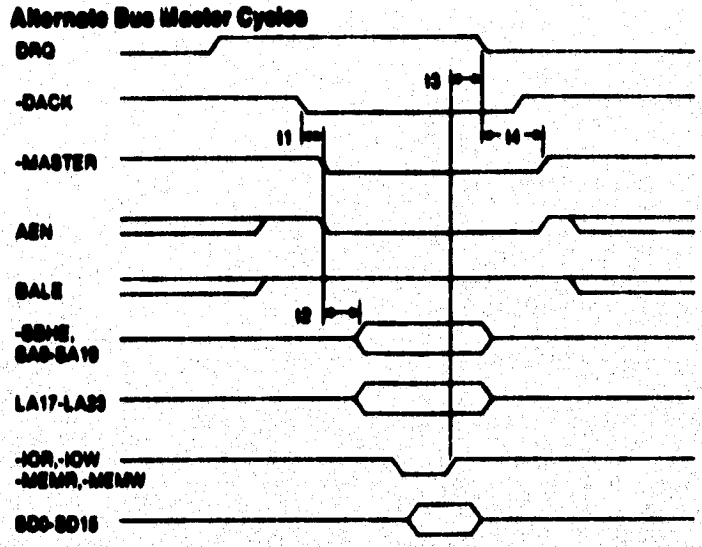
Figure 12. 16-Bit DMA Write Cycles

16-Bit DMA Read Cycles



Symbol	Description	8Mhz (ns)	8Mhz (ns)
t1	DRQ inactive from DACK active	0	
t2	SA0 – SA19 valid from AEN active	375	
t3	AEN inactive from SA0 – SA19 invalid	0	
t4	IOW active from DACK active	250	
t5	DACK inactive from IOW inactive	0	
t6	MEMR active from SA0 – SA19 valid	130	
t7	MEMR active from LA17 – LA23, SA16 – SA19 valid	130	
t8	MEMR pulse width	500	
t9	SA0 – SA19 hold from MEMR inactive	40	
t10	LA17 – LA23, SA16 – SA19 hold MEMR inactive	0	
t11	Data valid from MEMR active		190
t12	Data hold from MEMR inactive	0	
t13	Data bus tri-state from MEMR inactive		60
t14	Data valid from IOW active		0
t15	Data valid setup to IOW inactive	200	
t16	IOW active from MEMR active	0	
t17	IOW inactive to MEMR inactive	0	
t18	IOW pulse width	330	
t19	T/C active setup to IOW inactive	300	
t20	T/C inactive from IOW inactive	0	
t21	I/O CH RDY inactive from MEMR active		340
t22	MEMR inactive from I/O CH RDY active	200	
t23	Data valid to I/O CH RDY active	0	

Figure 13. 16-Bit DMA Read Cycles

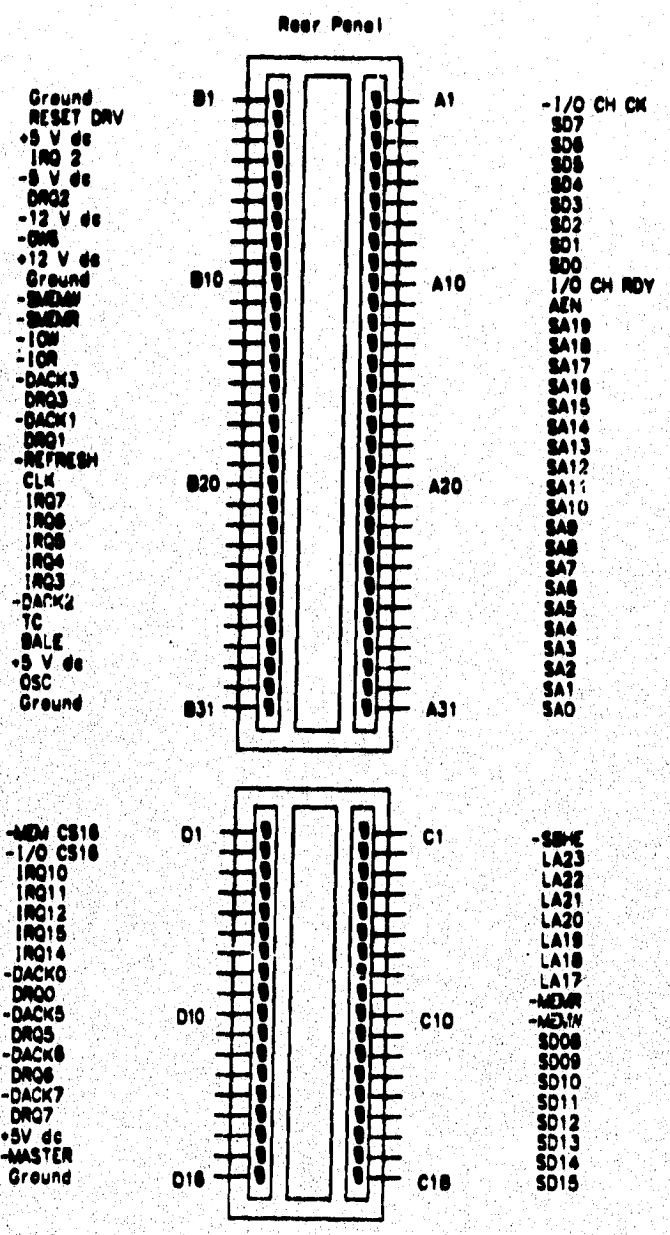


Symbol	Description	Min (ns)	Max (ns)
t1	MASTER active from BACK active	0	
t2	SA0 – SA19, LA17 – LA23 valid from MASTER active	60	
t3	DRQ inactive from IOR/W, MEMR/W inactive	0	
t4	MASTER inactive from DRQ inactive	0	125

Figure 14. Alternate Bus Master Cycles

Notes:

1. AEN and BALE are driven by the system logic and not by the alternate bus master.
2. After MASTER is active, the alternate bus master must wait 100 nanoseconds before driving the Data lines and 125 nanoseconds before driving the Read and Write lines.
3. The alternate bus master must tri-state (disable) the address, data, and control lines when MASTER is released.
4. Alternate bus masters must maintain an awareness of the system memory refresh requirements and may initiate a memory refresh cycle by driving -REFRESH active as defined in "Memory Refresh Cycles."


Rear Panel
Ground
RESET DRV
+5 V dc
IRQ 2
-5 V dc
DRQ2
-12 V dc
-SW6
+12 V dc
Ground
-SMEMW
-SMEMR
-IOW
-IOR
-DACK3
DRQ3
-DACK1
DRQ1
-REFRESH
CLK
IRQ7
IRQ6
IRQ5
IRQ4
IRQ3
-DACK2
TC
BALE
+5 V dc
OSC
Ground
B1
B10
B20
B31
A1
A10
A20
A31
-I/O CH CK
SD7
SD6
SD5
SD4
SD3
SD2
SD1
SD0
I/O CH RDY
AEN
SA19
SA18
SA17
SA16
SA15
SA14
SA13
SA12
SA11
SA10
SA9
SA8
SA7
SA6
SA5
SA4
SA3
SA2
SA1
SA0
-MEM CS16
-I/O CS16
IRQ10
IRQ11
IRQ12
IRQ15
IRQ14
-DACK0
DRQ0
-DACK5
DRQ5
-DACK6
DRQ6
-DACK7
DRQ7
+5V dc
-MASTER
Ground
D1
D10
D18
C1
C10
C18
-SBHE
LA23
LA22
LA21
LA20
LA19
LA18
LA17
-MEMR
-MEMW
SD08
SD09
SD10
SD11
SD12
SD13
SD14
SD15