Universidad de Valladolid

Escuela Técnica Superior
de Ingenieros de Telecomunicación

Proyecto Fin de Carrera
Ingeniero en Electrónica

Desarrollo de una Biblioteca de Funciones para Tratamiento de Imágenes en Tiempo Real

Autor:
Ivan Redondo Villahoz
Tutor:
Jesús M. Hernández Mangas

Valladolid, 27 de junio de 2005

Título:	**Desarrollo de una Biblioteca de Funciones para Tratamiento de Imágenes en Tiempo Real**
Autor:	**Ivan Redondo Villahoz**
Tutor:	**Jesús M. Hernández Mangas**
Departamento:	**Electricidad y Electrónica**

Miembros del tribunal

Presidente:	**Jesús Arias Álvarez**
Vocal:	**Jacinto San Pablo García**
Secretario:	**Jesús M. Hernández Mangas**
Fecha de Lectura:	
Calificación:	

Resumen del proyecto

En los últimos años, el procesamiento en tiempo real ha pasado a ser indispensable para cualquier tipo de aplicación y para cualquier ámbito de trabajo y, en particular, para el tratamiento de imágenes, bien sea para su captura, en nuestro caso a través de vídeo mediante una tarjeta capturadora, como para el posterior tratamiento de imágenes mediante el software desarrollado para una aplicación específica, así se pretende desarrollar una biblioteca de funciones para el tratamiento de imágenes escrita en lenguaje ensamblador de la arquitectura Intel 80x86 que emplea las últimas tecnologías como son las tecnologías MMX, SSE, SSE2 y SSE3, para que sea capaz de trabajar con vídeo en tiempo real.

Además estas rutinas deberán poder ser llamadas desde un programa C++ desde sistemas operativos como Windows, Linux, ...

Palabras clave

Biblioteca, Función, Tratamiento de imágenes, Tiempo real, Ensamblador, Últimas tecnologías.

Abstract

Last years, real time processing has became indispensable for any kind of application and for any compass of work and, particularly, for image processing, as for its capture, in our case though video with a capturated card, as for the later image processing with the software developed for an especific application, in that way we can develop a function library for written in assembler of architecture Intel 80x86 image processing which uses new technologies as MMX, SSE, SSE2 and SSE3 are, so that it will be able for working with video at real time.

Besides these routines could be called by a C++ programme, by operating systems such as Windows, Linux, ...

Key words

Library, Function, Image processing, Real time, Assembler, New technologies.

A mis padres, hermana y novia
y a todos los que han estado a mi lado
durante todos estos años.

Agradecimientos

En primer lugar quiero agradecer a mi tutor, Jesús M. Hernández Mangas, tanto su dedicación y orientaciones, como el haberme dado la oportunidad de realizar este interesante proyecto fin de carrera.

Quiero agradecer al Departamento de Electrónica de la Escuela Técnica Superior de Ingenieros de Telecomunicación de la Universidad de Valladolid haberme permitido utilizar el laboratorio 22 y todo el material necesario para llevar a cabo el proyecto.

También lanzo un agradecimiento para todos aquellos que durante estos años me han dado su apoyo y han hecho este camino un poco menos duro con su compañía.

Además de dedicar este proyecto a mis padres, les quiero agradecer todas las cosas que me han dado, oportunidades, enseñanzas, estudios, consejos, ... para poder ver cumplidos muchos de mis sueños y quiero agradecerles todos los sacrificios y esfuerzos que han realizado a lo largo de esta dura vida por mi. ¡¡Va por vosotros ***Luis*** *y* ***Leo****!!*

*Agradezco a mi queridísima "****bikuki****" sus interminables consejos, que aunque piensa que no la hago caso, algo siempre se me queda y agradecerla también esos momentos que pasamos juntos y nos reímos tanto de quienes tu ya sabes. Y agradecerla lo buena hermana que es aunque nunca se lo diga y que me atrevo a escribir aquí, pero que no te sirva de precedente. No creo que haya otra hermana igual en todo el mundo.*

Además agradezco a mis abuelos, a mi tío y a mi tía lo que hacen por mí. Creo abuelo que ya se acabó la vida del lobo y el picar para buscar comida bajo tierra...

También agradecer a mi ***amor*** *sus consejos y demás historias que pasamos juntos, aunque alguna vez la lie y que espero que duren para siempre.*

Por último agradecer a mi ***princesa*** *los buenos ratos que pasamos juntos y que nos duren muchos años, eso sí haber si no ladras tanto y dejas dormir mejor.*

Índice general

Índice de figuras

Listados

Parte I

Introducción

Capítulo 1

Objetivo y descripción del proyecto

En los últimos años, el procesamiento en tiempo real, ha pasado a ser indispensable para cualquier tipo de aplicación y para cualquier ámbito de trabajo y, en particular, para el tratamiento de imágenes, bien sea para su captura, en nuestro caso a través de vídeo mediante una tarjeta capturadora, como para su posterior tratamiento o manipulación de imágenes mediante el software desarrollado para una aplicación específica, siendo éstos dos últimos los aspectos más importantes abordados en la ejecución del proyecto.

El objetivo del proyecto ha sido el desarrollo de una biblioteca de funciones para el tratamiento de imágenes, escrita en ensamblador de la arquitectura Intel 80x86 empleando las últimas tecnologías, tales como las tecnologías MMX, SSE, SSE2 y SSE3, para ser capaz de trabajar con vídeo en tiempo real.

La biblioteca de funciones ha sido realizada a través del lenguaje de programación C++, dada su versatilidad para poder ser llamada desde sistemas operativos como Windows, Linux,...

A su vez cada función de la biblioteca generada ha sido escrita, en su parte más crítica, en lenguaje ensamblador de la arquitectura Intel 80x86, con el objetivo de conseguir la mayor rapidez en cuanto a su procesamiento para dichas funciones, es decir, las rutinas de tratamiento de imagen realizadas se han hecho rápidas y se han optimizado al máximo, de ahí que estén escritas en lenguaje ensamblador, debido a que nuestro objetivo final y más importante es el tratamiento de imagen en tiempo real y, por lo tanto, se premia la velocidad.

Una visión muy general del esquema seguido para la realización del proyecto se puede observar en la siguiente figura:

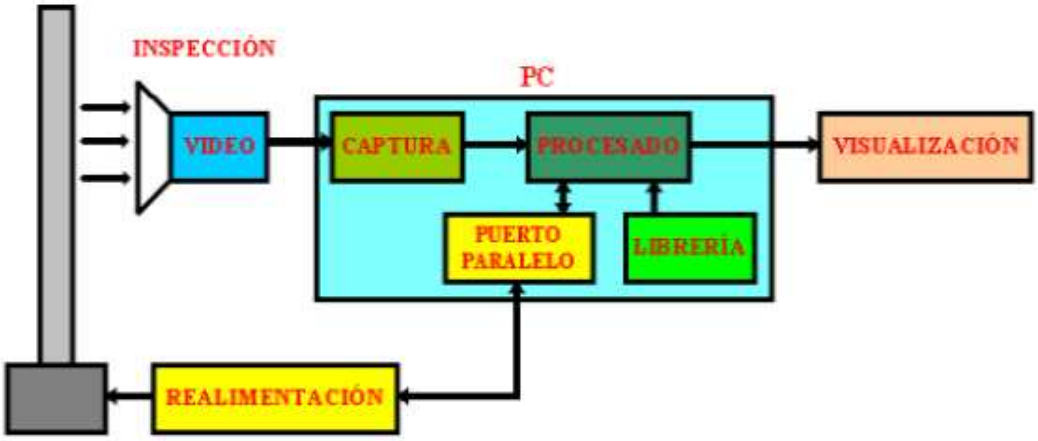

Figura 1.1: Esquema del proyecto.

Primeramente, se realiza la inspección del medio, con lo que obtenemos una imagen, que es transmitida en formato PAL por medio de una cámara de vídeo, esta imagen es recogida por la capturadora de vídeo, en nuestro caso esta basada en el chip BT878, y la imagen llega a nuestra etapa de procesado.

La etapa de procesado esta formada por un conjunto de funciones o rutinas (librería) para el tratamiento de la imagen en tiempo real que se pueden resumir en:

- Operaciones sobre una imagen: redimensionado, función NOT, función negativo, recorte.
- Operaciones con imagen sobre pantalla: captura y dibuja en pantalla.
- Transformaciones geométricas: escalado, traslación, rotación en 2D.
- Aplicación de convoluciones: filtros pasa baja, pasa alta, contornos, etc.
- Tratamiento del color. Ecualizados.
- Operaciones matemáticas entre imágenes: suma, resta, función OR, función XOR.
- Corrección de aberraciones.
- Transformada de Fourier e inversa de Fourier en 2D.

- Almacenamiento y carga de imágenes en formato gráfico BMP.

Una vez procesada la imagen, ésta se visualiza por pantalla, utilizando el modo gráfico VESA.

Además de incluir la librería generada las funciones o rutinas del procesado, también incluye rutinas para la captura de imágenes o de programación de la capturadora y rutinas de visualización de imágenes por pantalla.

Estas rutinas, como los términos que aparecen a lo largo de esta breve descripción, tales como formato PAL, chip BT878, modo VESA,... serán desarrollados a lo largo de los capítulos sucesivos.

Por último cabe destacar que para la realización de un buen diseño de captura y procesado de imágenes, se necesita una realimentación para poder trabajar en tiempo real, este hardware de realimentación implementado ha sido creado para una determinada aplicación práctica específica que veremos en el último capítulo y, que puede ser sustituido según la necesidad de la aplicación requerida.

Capítulo 2

Captura Vídeo

En este capítulo se abordarán todos los temas relacionados con la captura de vídeo, tanto para el sistema de vídeo empleado para la captura, como para la capturadora de vídeo que disponemos.

Primeramente, se realiza una introducción, para intentar comprender como se almacena, como se captura y como se muestra el vídeo, basándonos en una tecnología obsoleta como es la televisión por tubo de rayos catódicos.

Después veremos, que es el sistema PAL, como surgió, sus principales características como pueden ser su modulación ó su demodulación, los formatos que presenta y la distribución geográfica de dichos formatos.

A continuación, se muestra la tarjeta capturadora de vídeo Pinnacle PCTV Pro, con la que se ha trabajado, exponiendo sus principales características, de la cual la más destacada es que esta basada en el chip Bt878 de la compañia BrookTree.

Y por último, veremos las principales características de este chip, su diagrama de bloques, explicando sus principales bloques funcionales, se comentarán sus principales funciones como pueden ser el Ultra Lock, el Cropping, el escalado, el coring, los ajustes de vídeo,... y, finalmente se verá su funcionamiento y su forma de programación.

2.1. Introducción

Al intentar comprender cómo se almacena, cómo se captura y cómo se muestra el vídeo debemos retroceder en el tiempo y fijarnos en una tecnología muy obsoleta como es la televisión por tubo de rayos catódicos.

Así podemos definir, un tubo de un televisor como un gran trozo de metal que no tiene aire dentro. Dentro de este tenemos un cátodo que emite electrones cuando se calienta (por eso la imagen tarda un rato en aparecer cuando se enciende la televisión, debido a que el cátodo debe calentarse primero hasta la temperatura apropiada para emitir electrones). Hay también un fuerte campo electromagnético que acelera los electrones hacia la parte frontal del tubo, y que posiciona el haz de electrones (son muchos electrones los que se lanzan hacia la parte frontal del tubo). La parte frontal del tubo está cubierta de fósforo y cuando los electrones la golpean, emite luz hacia el otro lado (el lado en el que estaremos nosotros). A continuación, se puede observar un esquema de un CRT (Cathodic Ray Tube ó Tubo de rayos catódicos):

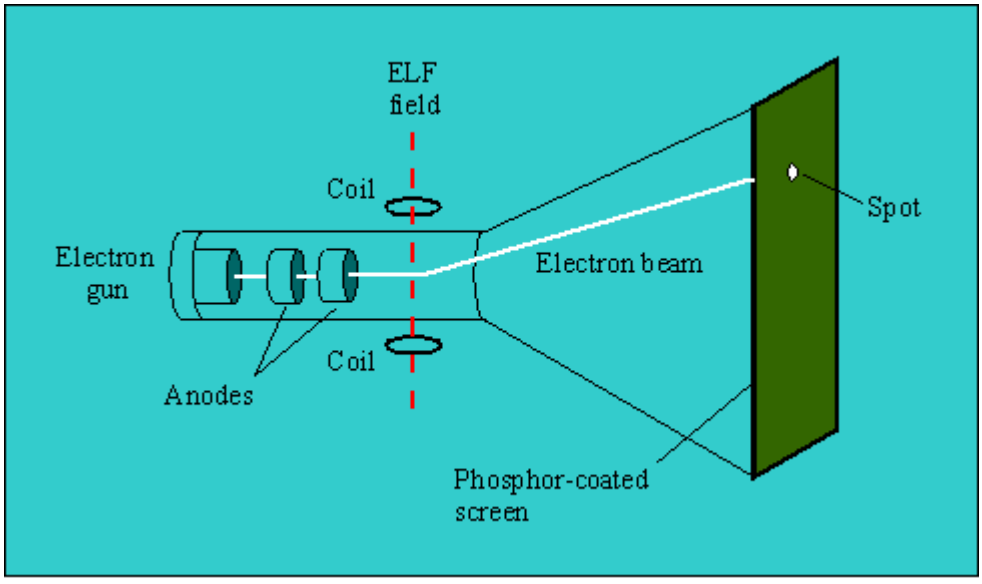

Figura 2.1: Esquema de un tubo de rayos catódicos (CRT).

En un principio las televisiones eran sólo en blanco y negro, con lo que era suficiente con un sólo haz de electrones. Ahora, para poder ver una película debes escribir esta por toda la pantalla, así que el haz de electrones debe barrer toda la pantalla. La frecuencia de barrido se conoce normalmente como tasa de refresco (refresh rate).

La tasa de refresco se escogió de acuerdo con los ciclos de los sistemas eléctricos que se usaban: Norteamérica y parte de Japón usan 60 Hz, Europa, Oriente Medio y partes de Asia usan 50 Hz. Esto dio lugar a dos sistemas de TV que compiten entre sí:

- NTSC: National Television Standard Committee. También conocido como "Never the same color" (nunca el mismo color) porque no hay dos imágenes NTSC que se vean igual. El sistema NTSC tiene 525 líneas horizontales de las cuales apenas 487 se ven en la pantalla y tiene una tasa de refresco de 60 Hz entrelazada.
- PAL: Phase Alternating Line. El sistema PAL tiene 625 líneas horizontales, de las cuales apenas 540 se ven en la pantalla y tiene una tasa de refresco de 50 Hz entrelazada.

Ahora, en la época en la que aparecieron en el mercado las primeras televisiones, la tecnología que permitía escribir 525 líneas 60 veces por segundo, o 625 líneas 50 veces por segundo tenía un precio prohibitivo que no era adecuado para el mercado de masas. Reducir la tasa de refresco habría requerido circuitos más complicados y no era una opción adecuada, además la mente humana tiene un límite inferior para lo que acepta como movimiento continuo. Pero los ingenieros de televisión tuvieron una idea: ¿Qué pasaría si escribiésemos únicamente una de cada dos líneas en cada barrido, y escribiésemos la otra mitad durante el siguiente barrido? De este modo sólo necesitaríamos 25/30 imágenes por segundo (lo que implica menos ancho de banda, lo que implica más cadenas de televisión en la misma banda de frecuencia), y el ojo humano seguiría aceptando esto como movimiento continuo. A esta idea de dividir la imagen en dos partes se la llamó entrelazado (interlacing), y a las imágenes divididas, campos (fields). Visto esto de modo gráfico, un campo es básicamente una imagen con una línea negra de cada dos (o blanca, lo que se prefiera). En la Figura 2.2 se muestra una imagen para que se pueda ver lo que está pasando.

Durante el primer barrido, el campo superior (upper field) se escribe en la pantalla. Como se puede observar, se escriben las líneas 1, 3, 5, etc.y después de escribir cada línea, el haz de electrones se mueve a la izquierda antes de escribir la siguiente línea. Como se puede notar, la imagen muestra un efecto de "peinado", parece que lo estamos viendo a través de un peine. Cuando la gente habla de artefacto de entrelazado (interlacing artifacts) o se dice que la imagen está entrelazada, se suelen referir a este efecto.

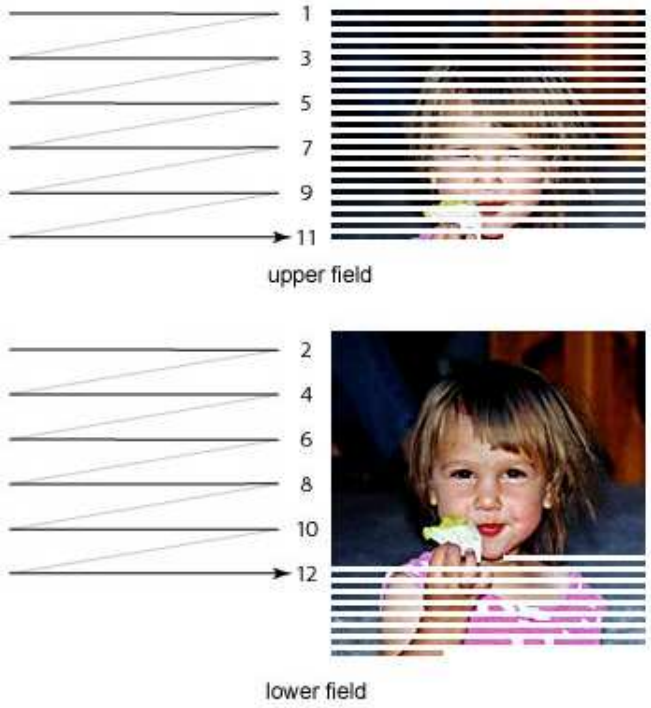

Figura 2.2: Ejemplo de entrelazado de imágenes.

Una vez que se han escrito todas las líneas impares, el haz de electrones regresa a la esquina superior izquierda de la pantalla y comienza a escribir las pares (lower field). Como el fósforo tarda un momento en dejar de emitir luz y dado que el cerebro humano es demasiado lento, en vez de dos campos distintos, lo que vemos es una combinación de los dos imágenes, es decir, en otras palabras, la imagen original.

Cuando finalmente llegó la televisión en color, la tecnología de entrelazado siguió siendo la misma, pero se necesitaba un tubo de rayos catódicos más sofisticado. En vez de emitir un único haz de electrones, se emiten tres haces de electrones, con los colores rojo, verde y azul. Cuando colocamos puntos de distintos colores lo suficientemente cerca unos de otros, el ojo humano no verá puntos individuales, sino un solo punto y unirá los colores para crear un nuevo color. En la Figura 2.3 se puede observar un ejemplo de un tubo de rayos catódicos en color.

Las televisiones usan un sistema de color aditivo para mostrar toda clase de colores. La televisión en color se consigue transmitiendo, además de la señal de brillo, o luminancia, necesaria para reproducir la imagen en blanco y negro, otra que recibe el nombre de señal de crominancia, encargada de transportar la información de color.

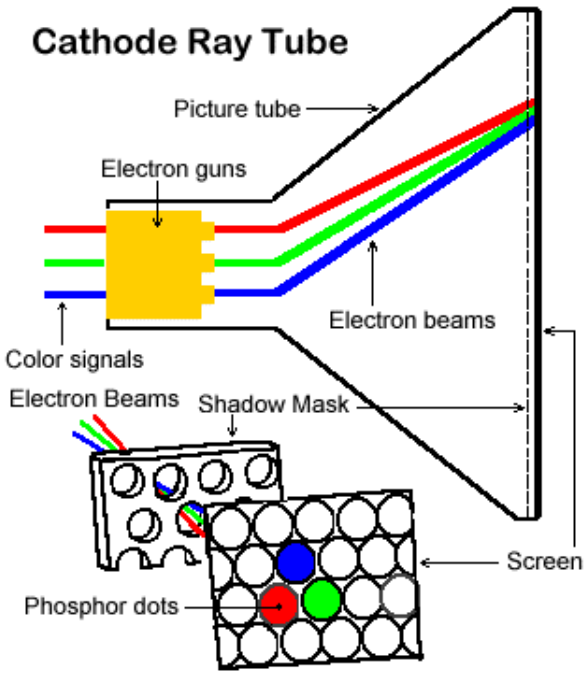

Figura 2.3: Esquema de un tubo de rayos catódicos (CRT) en color

Mientras que la señal de luminancia indica el brillo de los diferentes elementos de la imagen, la de crominancia especifica la tonalidad y saturación de esos mismos elementos. Ambas señales se obtienen mediante las correspondientes combinaciones de tres señales de vídeo, generadas por la cámara de vídeo, en nuestro caso, en color, y cada una corresponde a las variaciones de intensidad en la imagen vistas por separado a través de un filtro rojo, verde y azul. Las señales compuestas de luminancia y crominancia se transmiten de la misma forma que la primera en la televisión monocroma. Una vez en el receptor, las tres señales vídeo de color se obtienen a partir de las señales de luminancia y crominancia y dan lugar a los componentes rojo, azul y verde de la imagen, que vistos superpuestos reproducen la escena original en color. El sistema funciona de la siguiente manera.

La imagen de color pasa a través de la lente de la cámara e incide sobre un espejo dicroico que refleja un color y deja pasar todos los demás. El espejo refleja la luz roja y deja pasar la azul y la verde. Un segundo espejo dicroico refleja la luz azul y permite el paso de la verde. Las tres imágenes resultantes, una roja, otra azul y otra verde, se enfocan en la lente de tres tubos tomavistas (orticones de imagen o plumbicones). Delante de cada tubo hay unos filtros de color para asegurar que la respuesta en color de cada canal de la cámara coincide con los colores primarios (rojo, azul y verde) a reproducir. El haz de electrones en cada tubo barre el esquema de imagen y produce una señal de color

primario. Las muestras de estas tres señales de color pasan a un sumador electrónico que las combina para producir la señal de brillo, o blanco y negro. Las muestras de señal también entran en otra unidad que las codifica y las combina para generar una señal con la información de tonalidad y saturación. La señal de color se mezcla con la de brillo a fin de formar la señal completa de color que sale al aire.

El receptor de televisión en color lleva un tubo de imágenes tricolor con tres cañones de electrones, uno para cada color primario, que exploran y activan los puntos fosforescentes en la pantalla del televisor. Estos puntos minúsculos, que pueden sobrepasar el millón, están ordenados en grupos de tres, uno rojo, otro verde y otro azul. Entre los cañones de electrones y la pantalla hay una máscara con diminutas perforaciones dispuestas de forma que el haz de electrones de cada cañón sólo pueda incidir sobre su correspondiente punto fosforescente. El haz que pinta la información roja sólo chocará con las fosforescencias rojas, y lo mismo para los otros colores.

Cuando la señal de color entrante llega a un televisor de color, pasa por un separador que aísla el color del brillo. A continuación se descodifica la información de color. Al volverse a combinar con la información del brillo, se producen diferentes señales de color primario que se aplican al tubo tricolor, recreándose la imagen captada por la cámara de color. Si la señal de color llega a un televisor en blanco y negro, los circuitos del receptor ignoran los datos relativos a tonalidad y saturación y sólo tienen en cuenta la señal de brillo. La norma de televisión en color adoptada en Estados Unidos por el National Television System Committee (NTSC) y que es la usual en América Latina, no ha sido aceptada en otras partes del mundo. Quizá sobre todo por la ausencia de consenso acerca del equilibrio entre calidad y complejidad de la norma a utilizar. En muchas partes de Europa se rechaza la norma NTSC. En consecuencia, existen en el mundo varias normas, cada una de ellas con sus propias características. En el Reino Unido, la norma actual es PAL (Phase Alternate Line), mientras que Francia utiliza la norma Color Secuencial de Memoria (SECAM). A grandes rasgos ambas pueden coexistir, pero existe un cierto grado de incompatibilidad en los equipos receptores.

Una vez analizado como se realiza la captación del medio a través de una cámara, su posterior visualización por pantalla y, de haber introducido los diferentes formatos de transmisión de señales, nos vamos a centrar en lo que nos concierne como es la transmisión de las imágenes captadas por nuestra cámara de video en formato PAL y la obtención o adecuación de estas imágenes a través de la tarjeta capturadora basada en el chip BT878 de la que disponemos, para su posterior tratamiento.

2.2. Sistema PAL

2.2.1. Introducción

Cuando surgió en Europa el problema de introducir la televisión en color, el sistema americano NTSC tuvo una gran influencia en todos los aspectos. Los métodos posteriores, desarrollados en Europa, realmente han sido llamados variantes y mejoras del sistema NTSC como justo reconocimiento del trabajo avanzado hecho en este campo, por los Estados Unidos, desde 1950, cuando estuvo virtualmente completo el desarrollo de un sistema de televisión en color compatible y completamente electrónico. Este desarrollo incluía el tubo de imagen de color. En 1954 comenzaron en Estados Unidos transmisiones regulares de televisión en color después de haber sido aprobadas las normas NTSC por la Federal Communication Commission el 17 de Diciembre de 1953. Esta norma, propuesta por el National Television Standard Committee, era el resultado de una intensiva investigación de los fundamentos, especialmente por parte de la industria. Actualmente se está utilizando aún en su forma original. En 1960 se introdujo en Japón. La situación europea queda mejor ilustrada por el desarrollo de la televisión monocromática. El Reino Unido había introducido en 1937 la televisión pública con una norma de 405 líneas. Un año más tarde seguía Francia con una norma de 455 líneas, Alemania empezó en 1938 con 441 líneas. Después de la segunda guerra mundial, casi todos los países europeos adoptaron un sistema monocromático con 625 líneas y 50 imágenes por segundo, muy similar al sistema norteamericano con sus 525 líneas y 60 imágenes, El Reino Unido mantuvo su anticuado sistema de 405 líneas y Francia introdujo una nueva norma con un número de líneas diferente: 819. La situación británica y francesa se hizo aún más compleja cuando se inició un segundo programa en la banda de UHF con 625 líneas. Resumiendo, la televisión a través de las fronteras nacionales europeas no ha sido posible para blanco y negro, y no será posible para el color por razones muy similares.

El término PAL es la sigla de Phase Alternating Line (línea alternada en fase). Este es el nombre con que se designa al sistema de codificación empleado en la transmisión de señales de televisión en color en la mayor parte del mundo. De origen alemán, se utiliza en la mayoría de los países africanos, asiáticos y europeos (entre ellos España), además de Australia.

Como oposición, nos encontramos con otros sistemas en uso como son NTSC, en casi toda América, Japón y el Sudeste Asiático, y SECAM, en uso en Francia, en ciertos países del Este de Europa y África. El sistema PAL deriva directamente del NTSC con algunas correcciones técnicas.

El sistema PAL surgió en el año 1963, de manos del Dr. Walter Bruch en los laboratorios de Telefunken en su intento por mejorar la calidad y reducir los defectos en los tonos de color que presentaba el sistema NTSC. No obstante, los conceptos fundamentales de la transmisión de señales han sido adoptados del sistema NTSC.

2.2.2. Base común para todos los sistemas de televisión en color

Todos los sistemas europeos de televisión en color están basados en algunas características importantes de la televisión monocromática que serán mantenidas y, por tanto, se describen en primer lugar.

En el extremo transmisor, los valores de luminancia de una escena se convierten, línea por línea, en voltajes con ayuda de una fotocélula. El primer barrido vertical de la escena comprende $312+\frac{1}{2}$ líneas, el segundo empieza completando la línea 313 y termina con la línea 625, en una relación de entrelazado, obteniendo de esta forma dos cuadros de una imagen. La frecuencia de barrido horizontal es un múltiplo exacto de la mitad de la frecuencia de cuadro.

En el receptor, se reproduce la escena por medio del haz que explora las líneas a las mismas frecuencias. Para obtener este resultado se han de transmitir dos clases de señales: el brillo asociado con cada elemento del cuadro y la señal de sincronismo que asegura el sincronismo de la señal de barrido del tubo receptor con la de funcionamiento del transmisor. La señal de sincronismo está comprendida entre el 75 % y el 100 % de la amplitud de la señal compuesta de televisión, mientras que la señal de video varía entre el 10 % y el 75 %.

Cualquiera que fuera el sistema de televisión en color que se adoptase, estos datos monocromáticos habían de ser tomados en consideración para cumplir la característica de compatibilidad, que exige que cualquier receptor monocromático de televisión, con independencia de su diseño y antigüedad, sea capaz de recibir señales de televisión en color y reproducir un cuadro en blanco y negro a partir de ellas, y que cualquier receptor de color pudiera reproducir una señal monocromática de un transmisor de esta clase.

Esta exigencia había sido planteada como condición "sine qua non" en los años cuarenta por la industria americana. En Europa, el acuerdo sobre una norma común de televisión en color fué dificil de alcanzar a causa de las diferentes normas utilizadas en blanco y negro dentro de un área geográficamente limitada, y las soluciones básicamente diferentes encontradas por Francia y Alemania para mejorar el sistema NTSC.

No obstante, existen principios en el manejo del color que se han de incorporar a cualquier sistema cromático. Deben ser recordados antes de pasar a un determinado sistema europeo de televisión en color.

Principios coloriméfricos

Un importante principio para la televisión en color es que la casi totalidad de los colores naturales, incluído el blanco, pueden simularse a base de una mezcla aditiva de tres colores primarios.

Los colores primarios son aquellos que no se pueden generar por mezcla aditiva. Se puede demostrar experimentalmente que estos tres primarios son: rojo, verde y azul.

Este principio permite basar la televisión en color en la mezcla de estos tres primarios, tanto en la exploración como en la reproducción de la escena.

Toma de la escena

La toma de los elementos del cuadro y su conversión en señales eléctricas quedan explicados en relación con el analizador de diapositivas de color de la Figura 2.4.

Un barrido en blanco sobre la pantalla del tubo de imagen se proyecta sobre la diapositiva cuyos elementos de cuadro modulan el haz luminoso explorador. En funcionamiento monocromático este haz modulado se convierte directamente en un voltaje de video por medio de una fotocélula. Para televisión en color, los colores primarios se filtran, por ejemplo, con espejos dicroicos que son capaces de transmitir ciertas partes del espectro y de reflejar otras. El primer espejo dicroico refleja la luz roja hacia la parte superior de la Figura 2.4, el segundo refleja la luz azul hacia abajo, pero ambos espejos son transparentes para la luz verde de forma que se necesitan tres fotocélulas para las señales primarias de color.

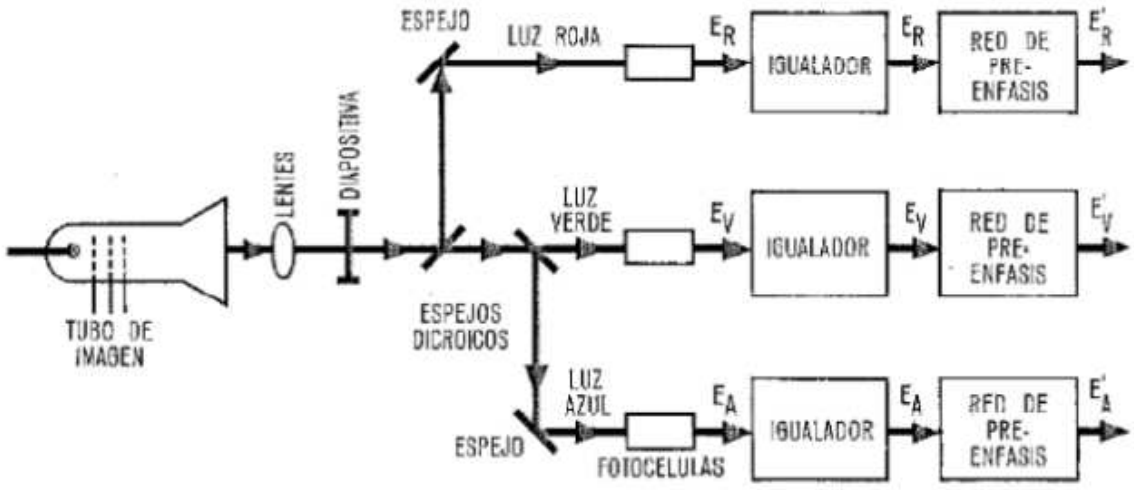

Figura 2.4: Esquema de la exploración de una diapositiva en color.

Luminancia y señales-diferencia de color

La señal de Luminancia

Para el proceso posterior de estas tres señales E'_R , E'_V y E'_A se han de combinar en una determinada proporción para formar la señal de luminancia, la única a la que el receptor monocromático puede responder. Supongamos que la escena no consiste más que en tres barras de colores primarios como muestra la Figura 2.5. El canal azul transmitirá el impulso E_A, de una duración correspondiente al ancho de la barra A. Impulsos similares de onda cuadrada aparecerán como E_V y E_R, en los canales verde y rojo respectivamente.

Si sumáramos sencillamente estas tres señales para obtener la señal de luminancia, el resultado sería un impulso de onda cuadrada de triple duración. Un receptor monocromático reproduciría las tres barras con igual brillo y sería imposible diferenciar los tres objetos.

En realidad, la percepción del brillo por el ojo humano depende del color tal como se indica en la Figura 2.5. Por tanto, para que el receptor monocromático reproduzca los colores con la proporción adecuada de brillo, los voltajes de señal de colores primarios se multiplican por los factores derivados de la curva de sensibilidad: 0.ll, 0.59 y 0.3 para el azul, verde y rojo respectivamente. La señal de luminancia E_Y, se obtiene así como suma de estos tres voltajes. La ecuación indicada es válida para cualquier sistema de televisión en color; es una primera condición para cumplir el requisito de compatibilidad.

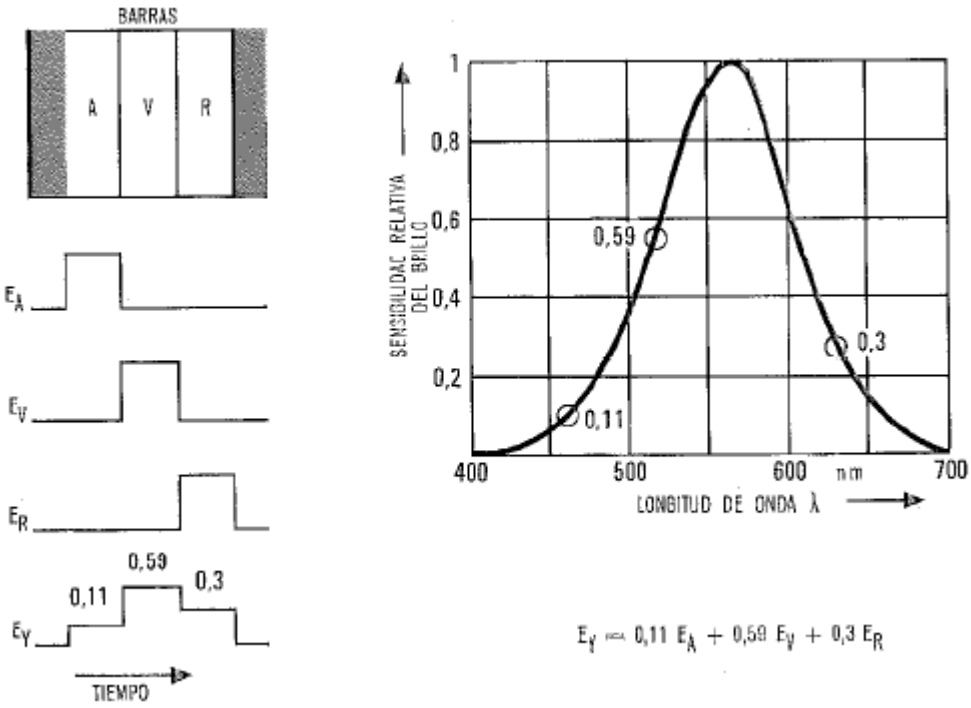

Figura 2.5: Generación de señal de luminancia E_Y.

Las señales-diferencia de color

El método de añadir la información de color a la señal de luminancia es también común a todos los sistemas de televisión en color: las llamadas señales-diferencia se forman por un procedimiento similar al de la estereofonía. Son:

- La señal-diferencia azul: E_{DA}
- La señal-diferencia roja: E_{DR}

Estas señales-diferencia tienen una propiedad importante para la televisión en color. Desaparecen donde el contenido del cuadro carece de color. Cada una de las señales de color primario es entonces igual a las otras dos; los voltajes de señal E_A, E_V y E_R, tienen las mismas amplitudes y el valor máximo normalizado para el blanco es 1. De esta cantidad 1, solamente se toman porcentajes para formar la señal de luminancia como se describió anteriormente, esto es, 11 % del voltaje de azul, 59 % del verde y 30 % del rojo. De acuerdo con la ecuación para E_Y, la suma es nuevamente 1.

Para formar la señal-diferencia azul, el valor 1 de la señal de luminancia se resta del valor 1 para el azul. El resultado es 0. Esto es igualmente cierto para la señal-diferencia roja. Otra característica común para todos los sistemas de televisión en color es la utilización de una subportadora para la transmisión de ambas señales-diferencia. La subportadora con su espectro armónico se adiciona a la banda de frecuencia ya ocupada por la señal de luminancia.

Modulación

La siguiente etapa es combinar las señales-diferencia que llevan la información de color con la señal de luminancia para obtener una señal compuesta. A causa de la compatibilidad exigida, todos los datos indispensables para la transmisión monocromática han de ser conservados. El ancho de banda nominal para blanco y negro no debe aumentar a pesar de tener que transmitir adicionalmente la señal de crominancia.

La diferencia más importante entre los distintos sistemas de televisión en color se reduce realmente al tipo de modulación de la subportadora. Básicamente, solamente hay dos: modulación de amplitud en cuadratura y modulación de frecuencia. La primera se utiliza en NTSC y PAL, la segunda en el sistema francés SECAM. El sistema NTSC es la base del sistema PAL; por tanto sus características más importantes se resumen en la siguiente sección.

2.2.3. NTSC como Base del sistema PAL

Las secciones anteriores mostraron cómo una señal de luminancia, se obtiene de tres señales cromáticas y cómo se obtienen las señales cromáticas diferencia. La utilización de una subportadora de color, en todos los sistemas de televisión en color, también ha sido señalada. Ahora se describirá la modulación de la subportadora con las dos señales cromáticas.

Para resolver este problema, se estipularon las siguientes condiciones al desarrollar el sistema NTSC:

1. La portadora debería ser capaz de ser modulada simultáneamente con ambas señales de crominancia E_{R-Y} y E_{A-Y}.

2. Para elementos del cuadro sin color, la amplitud de la subportadora debería ser 0.

Modulación NTSC

Para cumplir estas condiciones se escogió un tipo de modulación en el que tanto la fase como la amplitud de la subportadora se modulan simultáneamente. Esta modulación se obtiene descomponiendo la subportadora en dos componentes en cuadratura y modulando en amplitud estas componentes con E_{DR} y E_{DA}.

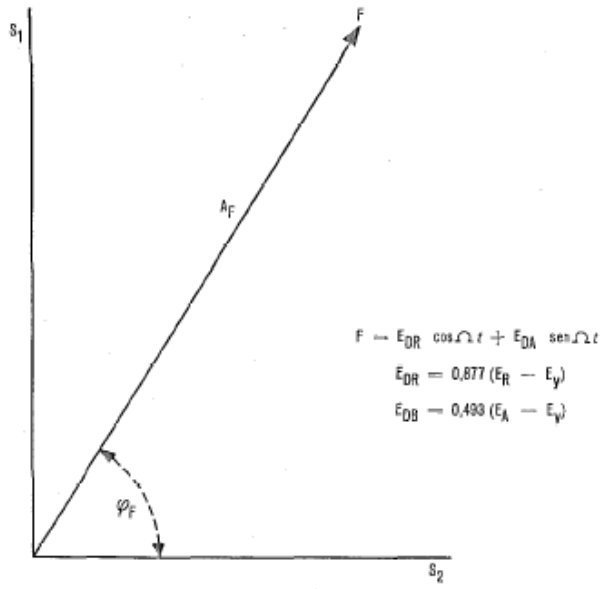

Figura 2.6: Diagrama vectorial de la subportadora modulada.

Cada una de las dos señales-diferencia E_{DR} y E_{DA}, pueden tomar independientemente cualquier valor positivo o negativo entre cero y un máximo. En otras palabras: el vector F resultante de la frecuencia de la subportadora (mostrado en la Figura 2.6) puede pertenecer a cualquiera de los cuatro cuadrantes.

Este tipo de modulación se entiende mejor con referencia a la Figura 2.7. Aquí A es la amplitud de la subportadora sin modular, M es la profundidad de modulación, Q la frecuencia de la portadora (en nuestro caso la frecuencia de la subportadora), ω la frecuencia moduladora que está en la banda de 0 a 1.5 MHz.

En la Figura 2.7, el vector de la portadora gira a la frecuencia de la subportadora mientras que los vectores de las bandas laterales giran, en sentido opuesto, alrededor del vector portadora. Cuando se suprime la portadora como en la Figura 2.8, los vectores de las bandas laterales permanecen con una resultante que desaparece cuando m = 0. Esta resultante tiene siempre la fase de la portadora original.

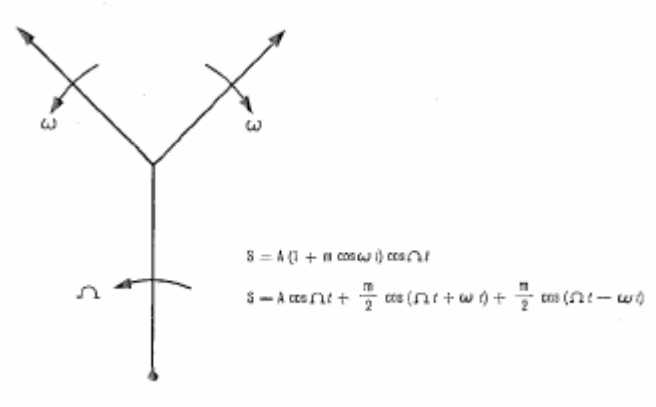

Figura 2.7: Modulación de amplitud sin supresión de portadora.

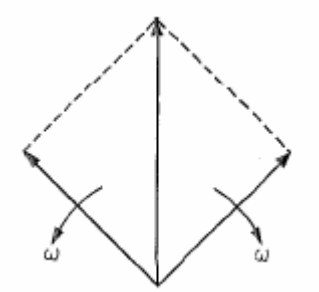

Figura 2.8: Modulación de amplitud con portadora suprimida.

Si este procedimiento se lleva a efecto en dos moduladores y las portadoras que los alimentan están en cuadratura, obtenemos dos resultantes S_1 y S_2 (Figura 2.6) o su suma, la subportadora F que contiene la información de color. Las relaciones entre el ángulo de fase φ_F y la amplitud A_Y de la subportadora modulada pueden obtenerse del diagrama vectorial de la Figura 2.9 que muestra los vectores asociados con los colores normalizados.

La Figura 2.10 es el diagrama bloque de un modulador basado en los principios del NTSC. Sus unidades son la matriz de color donde se obtiene la señal de luminancia E_Y y las de crominancia E_{DR}, E_{DA} a partir de las señales de color primarias; los dos moduladores alimentados en fase y en cuadratura por el generador de portadora de color; y el sumador donde las señales de crominancia y la señal de sincronismo se combinan para formar la señal compuesta de televisión de color.

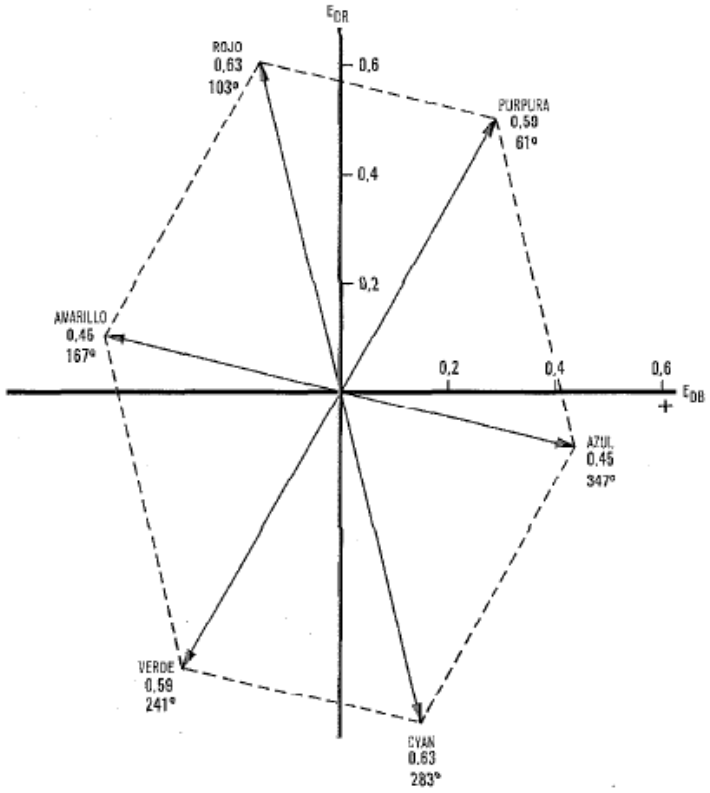

Figura 2.9: Relación de fase de las componentes de la subportadora.

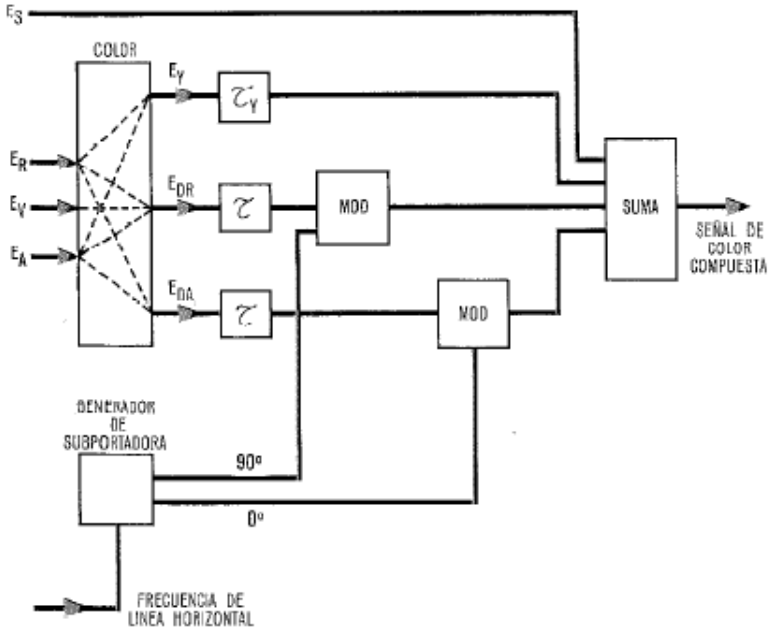

Figura 2.10: Modulador NTSC.

Demodulación NTSC

Las señales cromáticas se recuperan en el receptor por el decodificador Figura 2.11 que obtiene la señal compuesta con 5 MHz de ancho de banda, a partir de un demodulador de video del tipo usado en receptores monocromáticos. La señal de color modulada correspondiente a la ecuación:

$$F = E_{DF}cos\Omega t + E_{DA}sen\Omega t$$

puede hacerse cero ya que E_{DR} y E_{DA} son cero para áreas o elementos del cuadro sin color. La demodulación requiere la frecuencia de la portadora sin modular; ésta se genera en el generador de subportadora del receptor y se aplica en fase y en cuadratura a los demoduladores.

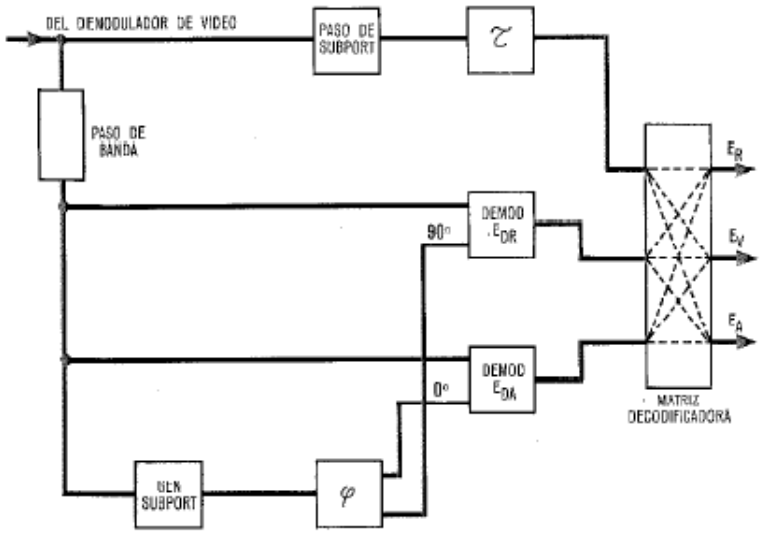

Figura 2.11: Demodulador NTSC.

En el demodulador E_{DR} la señal de color se multiplica por la oscilación coseno de la subportadora con lo que se obtiene la señal diferencia $E_{DR}/2$ y otros dos términos a frecuencia doble que pueden ser filtrados fácilmente. De forma similar se obtiene $E_{DA}/2$ en el segundo demodulador. La señal de luminancia y las dos señales-diferencia de color van a la matriz decodificadora que entrega las señales primarias da color E_R, E_V y E_A por medio de operaciones lineales de suma y resta similares a las de la matriz transmisora.

Desventajas del método NTSC

El método NTSC en sí podría calificarse de ideal. No obstante, los caminos de transmisión no son ideales y este hecho se refleja en el receptor NTSC. Ya que la información de crominancia importante se transmite por medio de una portadora modulada en fase, cualquier desplazamiento de fase indeseado de esta portadora, con respecto al impulso de sincronismo de color, producirá necesariamente una distorsión en el tono de color.

Este método de modulación fué el que indujo a los especialistas europeos de televisión a buscar una solución mejor, y, efectivamente, se encontraron soluciones.

2.2.4. El Sistema PAL

El sistema PAL se puede describir como el sistema NTSC al que se han añadido los circuitos adicionales que eliminan los errores de fase introducidos por el camino de transmisión.

La Figura 2.12 es la representación vectorial de una señal de crominancia F. En el sistema NTSC, las componentes en fase y en quadratura de la portadora están en la misma relación de fase (0 y 90 grados) en todo instante.

En el sistema PAL, una de las dos componentes (llamadas componentes U y V) se conmuta alternadamente entre 90º y 270º de línea a línea. En la Figura 2.12 se supone que se transmite un color primario constante. El vector resultante de la componente V cambiará por tanto su posición entre F_α y F_α^* de línea en línea.

Supongamos ahora que una distorsión de fase haga desplazarse al vector F_α el ángulo β en el sentido $F_{\alpha+\beta}$. Como la componente V del transmisor se invierte alternadamente en l80º, el vector resultante para la línea horizontal siguiente aparecerá como $F^*_{\alpha-\beta}$, es decir, también desplazado el ángulo β y en el mismo sentido de giro.

Como también en el receptor la componente V es conmutada alternadamente, $F^*_{\alpha-\beta}$ aparecerá, según una simetría respecto al eje U, en el primer cuadrante como $F^*_{\alpha-\beta}$.

Esta inversión de polaridad requiere que ambas componentes de portadora U y V sean accesibles individualmente en el receptor.

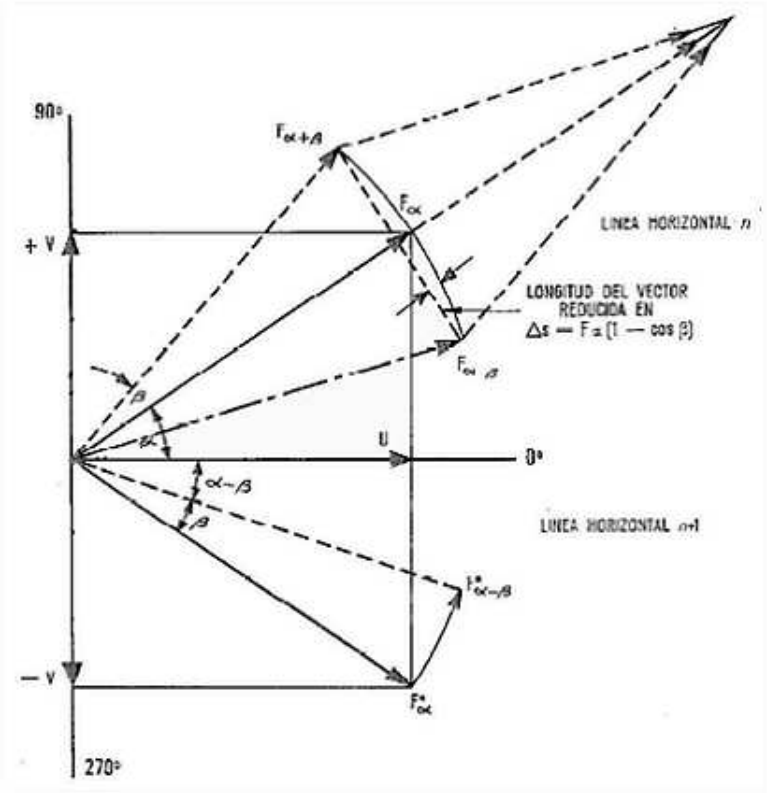

Figura 2.12: Principio de conmutación de línea alternada en PAL (diagrama vectorial)

El vector F_α, cambiará pues, de línea a línea entre $F^*_{\alpha+\beta}$ y $F^*_{\alpha-\beta}$, esto es, con el error de fase alternando como error positivo y negativo; no obstante, los vectores alternantes aparecerán en el primer cuadrante.

Para cancelar este error de fase se necesita solamente una condición: los vectores $F^*_{\alpha+\beta}$ y $F^*_{\alpha-\beta}$ deben aparecer al mismo tiempo. Ello se realiza con una línea de retardo. Entonces se pueden sumar los vectores para obtener su valor medio F_α, que tiene un ángulo α, esto es, que ya no contiene el error β, sino algo menor que el doble de la longitud F_α dependiendo de la magnitud del error eliminado. Este acortamiento del vector en Δs es todo lo que queda del original error de fase; tiene el efecto de reducir la saturación del color, pero en una cantidad tan pequeña que apenas se percibe.

Modulación PAL

El modulador PAL difiere del NTSC solamente por la función adicional de la inversión constante en 180º de una de las componentes de la subportadora. Las unidades adicionales están encerradas en línea discontinua en la Figura 2.13.

Las señales de los tres canales de color primarios se utilizan en la matriz para formar las señales-diferencia de color en tanto que la señal de luminancia E_Y, se forma como suma de las señales de color primario convenientemente pesadas. Un generador de subportadora suministra las tres componentes a 0, 90 y 270 grados. La componente 0º se mezcla con la señal-diferencia A-Y en el modulador U las otras dos alimentan el modulador V en una secuencia alternada marcada por el conmutador S. El voltaje de conmutación lo suministra un generador biestable sincronizado por la frecuencia horizontal (línea). Finalmente la señal compuesta de televisión en color se obtiene en un sumador de la misma forma que en el NTSC.

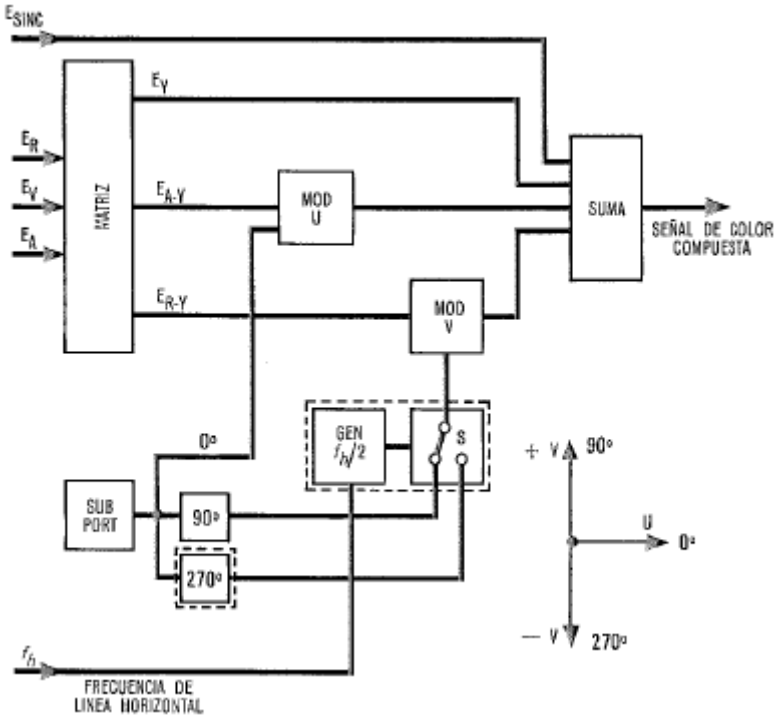

Figura 2.13: Modulador PAL.

Demodulación PAL

A partir del demodulador de vídeo del receptor, la señal de color se aplica - como en el NTSC - al punto A de la Figura 2.14, a través de un filtro pasobanda. Aparece en el sumador U y, a través de una línea de retardo que introduce un retardo igual al barrido de una línea, en ambos sumadores U y V. Finalmente, alimenta al sumador V después de un desplazamiento de fase de 180º.

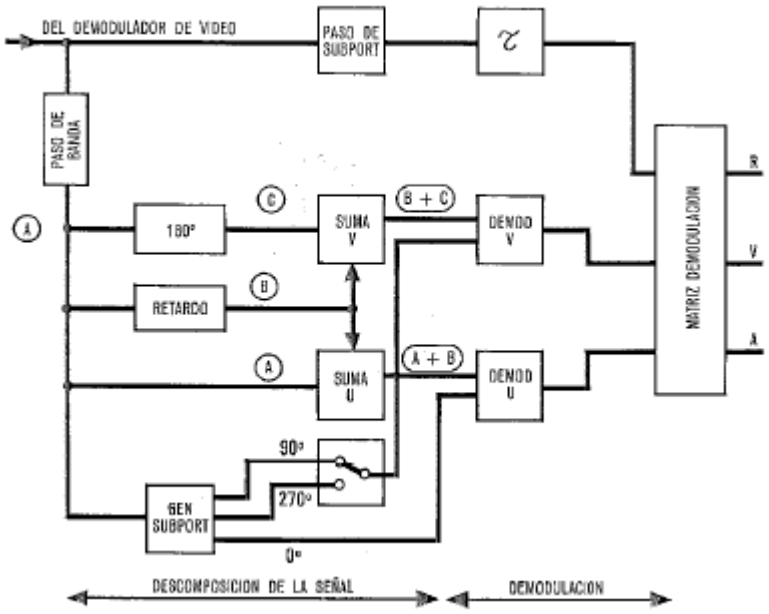

Figura 2.14: Demodulador PAL.

La Figura 2.15 muestra la secuencia de las señales que aparecen en los puntos A, B y C de la Figura 2.14 y detrás de los sumadores U y V (A+B y A+C) comprendiendo cuatro líneas horizontales. La fila superior A en la Figura 2.15 es la secuencia de la señal de entrada; la fila segunda es la A pero retrasada una línea mientras que la tercera representa también A con la polaridad invertida en 180º por el desplazador de fase.

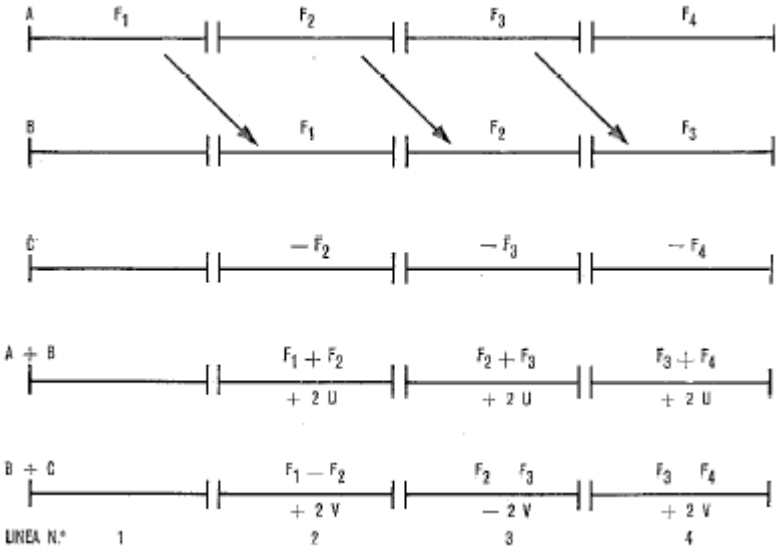

Figura 2.15: Secuencia de señal en el demodulador PAL.

Las sumas A+B y A+C así como A-B comprendiendo dos líneas seguidas (n^{os} 2 y 3) se ilustran en la Figura 2.16, supuesto que el color de la escena permanece inalterado durante estas dos líneas. La señal retardada F_1 de la línea nº. 1 y la señal F_2 alimentan simultáneamente al sumador U. La suma de F_1 y F_2 produce el vector 0º de magnitud 2U correspondiente a la señal-diferencia de color $EA - Y$.

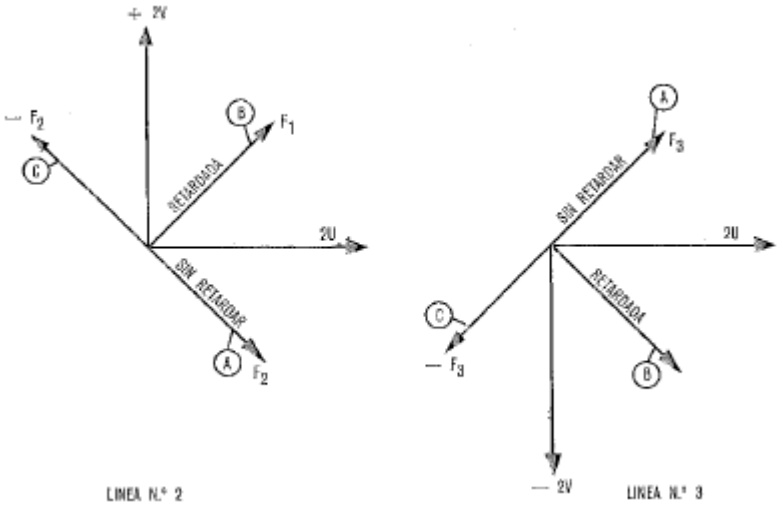

Figura 2.16: Descomposición de la señal PAL.

Al mismo tiempo, F_1 aparece en el terminal B y -F_2 en el terminal C del sumador V. Su suma produce una componente a 90° de la subportadora de color con magnitud +2V. Durante la línea nº. 3, F_3 en el terminal A está en el primer cuadrante (a causa de la inversión de la componente a 90º) y la señal F_2, como para la línea precedente, en el cuarto cuadrante. La suma de ambos es nuevamente 2U, esto es, componente a 0º. La suma de las señales en B y C, por el contrario, producen una componente -2V en el sumador V a causa de que F_2 y -F_3 están dirigidos hacia abajo. De esta forma ambas componentes de la subportadora se recuperan en posiciones 0º y 90º, pero la componente V cambia su polaridad de línea en línea.

Como en el sistema NTSC, cada componente se rectifica en un demodulador síncrono. En el demodulador U, la portadora reinsertada tiene su posición de fase a 0º. El demodulador del receptor PAL contiene las siguientes cuatro funciones adicionales, en comparación con el receptor NTSC:

1. retardo de la subportadora por la duración de una línea, de forma que la información de dos líneas está disponible constantemente;

2. descomposición de la señal de color modulada en amplitud y fase en las componentes originales U y V, lo que supone la inversión del proceso análogo en el transmisor;

3. conmutación alternada de la componente V en 180º de línea en línea;

4. derivación de impulsos de sincronismo de la subportadora para el ajuste de fase del mecanismo de conmutación en 3).

2.2.5. Detalles técnicos

El nombre "Phase Alternating Line" (línea alternada en fase) como hemos visto anteriormente describe el modo en que la información de crominancia (color) de la señal de vídeo es invertida en fase en cada línea, permitiendo la correción automática de los posibles errores en fase al cancelarse entre sí. En la transmisión de datos por radiofrecuencia, los errores en fase son comunes, y se deben a retardos de la señal en su llegada o procesado.

Aprovechando que habitualmente el contenido de color de una línea y la siguiente es similar, en el receptor se compensan automáticamente los errores de tono de color tomando para la muestra en pantalla el valor medio de una línea y la anterior, dado que el posible error de fase existente será contrario entre una línea y la siguiente. De esta forma dicho error, en lugar de un corrimiento del tono como ocurriría en NTSC, queda convertido en un ligero defecto de saturación de color que es mucho menos perceptible al ojo humano. Esta es la gran ventaja del sistema PAL frente al sistema NTSC.

Las líneas en que la fase está invertida respecto a cómo se transmitirían en NTSC se llaman a menudo líneas PAL, y las que coincidirían se denominan líneas NTSC.

El funcionamiento del sistema PAL implica que es constructivamente más complicado de realizar que el sistema NTSC. Esto es debido a que, si bien los primeros receptores PAL aprovechaban las imperfecciones del ojo humano para cancelar los errores de fase, sin la corrección electrónica explicada anteriormente, cuando nos referiamos a que toma del valor medio, esto daba lugar a un efecto muy visible de "peine" si el error excedía los 5º. La solución fue introducir una línea de retardo en el procesado de la señal de luminancia de aproximadamente 64 μs que sirve para almacenar la información de crominancia de cada línea recibida; la media de crominancia de una línea y la anterior es lo que se muestra por pantalla.

Los dispositivos que eran capaces de producir este retardo eran relativamente caros en la época en la que se introdujo el sistema PAL, pero en la actualidad se fabrican receptores a muy bajo coste.

Esta solución reduce la resolución vertical de color en comparación con NTSC, pero como la retina humana es mucho menos sensible a la información de color que a la de luminancia o brillo, este efecto no es muy visible. Los televisores NTSC incorporan un corrector de matiz de color (en inglés, tint control) para realizar esta correción manualmente.

Finalmente, en el sistema PAL es más probable que el aparato receptor malinterprete una señal de color como señal de luminancia, o viceversa, que en el sistema NTSC. En consecuencia, el sistema NTSC es técnicamente superior en aquellos casos en los que la señal es transmitida sin variaciones de fase (y, por tanto, sin los defectos de tono de color anteriormente descritos), por ejemplo en la televisión por cable, por satélite, en videojuegos, en reproductores de vídeo, y en general en todas las aplicaciones en banda base.

2.2.6. Formatos del sistema PAL

El sistema de color PAL se usa habitualmente con un formato de vídeo de 625 líneas por cuadro (un cuadro es una imagen completa, compuesta de dos campos entrelazados) y una tasa de refresco de pantalla de 25 cuadros por segundo, entrelazadas, como ocurre por ejemplo en las variantes PAL-B, G, H, I y N. Algunos países del Este de Europa que abandonaron el sistema SECAM ahora emplean PAL D o K, adaptaciones para mantener algunos aspectos técnicos de SECAM en PAL. En Brasil, se emplea una versión de PAL de 525 líneas y 29,97 cuadros por segundo, PAL M, muy próximo a NTSC en la frecuencia de subportadora de color. Casi todos los demás países que emplean el sistema M de color usan NTSC para la luminancia. En Argentina, Paraguay y Uruguay, se usa PAL con el sistema estándar de 625 lineas, aunque de nuevo con la frecuencia subportadora de color de NTSC. Estas variantes se llaman PAL-N y PAL-CN.

Los receptores de televisión PAL más recientes pueden mostrar todos estos sistemas, salvo en algunos casos PAL-M y PAL-N. La mayor parte también puede recibir señales SECAM del Este de Europa y de Oriente Medio, aunque normalmente no el SECAM francés, salvo en equipos de fabricantes franceses.

Muchos pueden incluso mostrar NTSC-M en banda base para señales de un reproductor de vídeo o consola de videojuegos, aunque generalmente no pueden recibir NTSC por radiofrecuencia.

Cuando el vídeo se transmite en banda base, la mayor parte de las diferencias entre variantes de PAL no son ya significativas, salvo por la resolución vertical y la tasa de refresco de cuadro. En este contexto, referirse a PAL implica sistemas de 625 líneas horizontales a 25 cuadros por segundo, entrelazadas, con el color en PAL.

2.2.7. PAL digital

Lo mencionado anteriormente se refiere al sistema PAL en dispositivos analógicos. En los dispositivos digitales, como televisión digital, consolas de videojuegos modernas, DVD, etc., ni siquiera importa la codificación de color empleada, y ya no hay diferencia entre sistemas, ni siquiera con SECAM, quedando el significado de PAL reducido a un número de líneas igual a 576 con una tasa de refresco de la imagen de 25 imágenes por segundo.

2.2.8. Distribución geográfica de los formatos PAL

La distribución geográfica de los formatos PAL dependiendo del tipo de sistema PAL utilizado se puede clasificar en:

1. Países y territorios que emplean PAL B/G o PAL D/K

   Europa

   Albania, Alemania, Isla Ascensión, Austria, Azores, Bélgica, Bosnia y Herzegovina, Cerdeña, Croacia, Dinamarca, Eslovenia, España, Estonia, República de Irlanda, Islas Feroe, Finlandia, Gibraltar, Grecia, Groenlandia, Islandia, Italia, Letonia, Liechtenstein, Lituania, Luxemburgo, Irlanda, Macedonia, Madeira, Malta, Paises Bajos, Noruega, Polonia, Portugal, Serbia y Montenegro, Suecia, Suiza, Tristan da Cunha, Turquía. Ciudad del Vaticano.

Asia

Afganistán, Bahrain, Bangladesh, Brunei, China, Chipre, Dubai, Gaza y Cisjordania, India, Indonesia, Israel, Jordania, Kuwait, Líbano, Malasia, Maldivas, Nepal, Omán, Pakistán, Qatar, Singapur, Sri Lanka, Siria, Tailandia, Turquía, Emiratos Árabes Unidos, Yemen.

América

Islas Malvinas.

África

Argelia, Angola, Botswana, Camerún, Cabo Verde, Eritrea, Etiopía, Gambia, Ghana, Guinea, Guinea-Bissau, Kenya, Lesotho, Liberia, Malawi, Mozambique, Namibia, Nigeria, Seychelles, Sierra Leona, Somalia, Sudáfrica, Sudán, Swazilandia, Tanzania, Uganda, Zambia, Zanzíbar, Zimbabwe.

Australia Oceanía

Australia, Isla Navidad, Islas Cook, Isla de Pascua, Nueva Zelanda, Isla Norfolk, Papúa Nueva Guinea, Islas Salomón, Tonga, Vanuatu.

2. Países y territorios que emplean PAL-I

   Hong Kong, Irlanda del Norte, Macao, Reino Unido.

3. Países y territorios que emplean PAL-M

   Brasil (NTSC & PAL-M), Laos (SECAM & PAL-M).

4. Países y territorios que emplean PAL-N o PAL-CN

   Argentina, Paraguay y Uruguay.

2.3. Tarjeta capturadora de vídeo

2.3.1. Introducción

La información de vídeo es provista en una serie de imágenes ó "cuadros" y el efecto del movimiento es llevado a cabo a través de cambios pequeños y continuos en los cuadros. Debido a que la velocidad de estas imágenes es de 30 cuadros por segundo, los cambios continuos entre cuadros darán la sensación al ojo humano de movimiento natural.

Las imágenes de video están compuestas de información en el dominio del espacio y el tiempo. La información en el dominio del espacio, es provista en cada cuadro, y la información en el dominio del tiempo, es provista por imágenes que cambian en el tiempo (por ejemplo, las diferencias entre cuadros). Puesto que los cambios entre cuadros colindantes son diminutos, los objetos aparentan moverse suavemente.

En los sistemas de vídeo digital, cada cuadro es muestreado en unidades de píxeles ó elementos de imagen.

El valor de luminancia de cada píxel es cuantificado con ocho bits por píxel para el caso de imágenes blanco y negro. En el caso de imágenes de color, cada píxel mantiene la información de color asociada; por lo tanto, los tres elementos de la información de luminancia designados como rojo, verde y azul, son cuantificados a ocho bits, por lo que tendremos un total de 24 bits.

La información de vídeo compuesta de esta manera posee una cantidad tremenda de información; por lo que, para transmisión o almacenamiento, se requiere de la compresión (o codificación) de la imagen.

La técnica de compresión de vídeo se realizan o consisten de tres pasos fundamentalmente:

- El primero el preprocesamiento de las diferentes fuentes de vídeo de entrada (señales de televisión de alta definición HDTV, señales de videograbadoras VHS, BETA, SVHS, etc.), paso en el cual se realiza el filtrado de la señal de entrada para remover componentes no útiles y el ruido que pudiera haber en esta.
- El segundo paso es la conversión de la señal a un formato intermedio común (CIF).
- El último paso es la compresión. Las imágenes comprimidas son transmitidas a través de la línea de transmisión digital y se hacen llegar al receptor donde son reconvertidas al formato común CIF y son desplegadas después de haber pasado por la etapa de post- procesamiento.

Mediante la compresión de la imagen se elimina información redundante, principalmente la información redundante en el dominio de espacio y del tiempo.

En general, las redundancias en el dominio del espacio son debidas a las pequeñas diferencias entre píxeles contiguos de un cuadro dado, y aquellas dadas en el dominio del tiempo son debidas a los pequeños cambios dados en cuadros contiguos causados por el movimiento de un objeto.

El método para eliminar las redundancias en el dominio del espacio es llamado codificación intracuadros, la cual puede ser dividida en:

- codificación por predicción
- codificación de la transformada
- codificación de la subbanda.

En el otro extremo, las redundancias en el dominio del tiempo pueden ser eliminadas mediante el método de codificación de intercuadros, que también incluye los métodos de compensación/estimación del movimiento, el cual compensa el movimiento a través de la estimación del mismo.

Por otro lado, existen circuitos integrados para la digitalización y compresión de vídeo. Actualmente en el mercado hay varios modelos de circuitos integrados que se encargan de realizar esta tárea. Entre los más conocidos se encuentran:

- El Bt848, Bt848A, Bt849A y otros circuitos integrados de BrookTree.
- Codificadores y decodificadores Philips. SAA711xA
- Compresores/Descompresores M-JPEG: ZR36060
- Familia TriMedia Philips: Tri-Media 1000, 1100, 1300

De entre los anteriormente citados circuitos integrados vamos a quedarnos con el que está basada nuestra capturadora de vídeo, que se corresponde con el chip Bt878.

2.3.2. Capturadora de vídeo utilizada

La capturadora de vídeo utilizada es la Pinnacle PCTV Pro cuyas características principales son:

- Procesador Booktree.
- Sintonizador TV Philips serie S1 12xx ó TEMIC.
- Estándar de Vídeo PAL/SECAM/NTSC.
- Entrada Antena Conexión CEI 75 ohms/Conexión F.
- Entrada Vídeo En tiempo real 720 x 576 (PAL/SECAM).
- Entradas de vídeo Conexión S-Vídeo / Conexión compuesta Calidad YUV 4:2:2.
- Audio estéreo y sintonizador de radio.
- Chip BT 878.
- Resolución captura:
  - NTSC: 320 x 240 a 60 frame/s.
  - PAL/SECAM: 384 x 288 a 50 frame/s.

Figura 2.17: Tarjeta capturadora Pinnacle PCTV Pro.

2.3.3. Chip BT878

Este modelo de circuito integrado pertenece a la compañía BrookTree, la cual tiene una amplia gama de circuitos integrados dedicados a la digitalización de la imagen. Este chip pertenece a la familia Bt8xx, y esta basado en una familia antecesora suya como es la familia Bt84X, que es una de las más conocidas, además el chip presenta algunas mejoras en referencia a sus antecesores.

Las principales características básicas que presenta este circuito integrado se pueden resumir en:

- Circuito integrado de bajo coste que proporciona una interfaz completa entre señales de vídeo analógicas (NTSC/PAL/SECAM) y el bus PCI.
- Incorpora en un único módulo.
  - Acondicionamiento de señales.
  - Subsistemas de muestreo y digitalización.
  - Funciones de control del bus PCI (132 Mbytes/s).
- Gestiona directamente el bus, puede realizar el control de la memoria de un PC mediante DMA.
- Puede conectarse directamente al bus (placa madre PC) o mediante tarjeta. Las tarjetas no requieren memoria interna ni procesador. La compresión del vídeo puede realizarse mediante software, tarjetas auxiliares, etc. Posibilidad de overlays con tarjetas gráficas u otros dispositivos.
- Es compatible con cualquier topología de bus PCI (Fully PCI Rev. 2.1 compliant).
- Soporta resoluciones de hasta 768x576 píxeles (PAL).
- Puede usar máscaras de cuadro/campo durante la adquisición. Puede adaptar los formatos de captura a las características de ancho de banda del sistema.
- Admite entradas de vídeo compuesto y S-Video.
- La salida puede ser en formato RGB o en formato YCrCb.
- El tamaño de la imagen de salida es escalable. Se utilizan filtros de interpolación y diezmado.

- Soporta distintos destinos para las imágenes en función de la configuración en base a cuadros o campos.
- dispone de un sistema de digitalización de audio que admite entradas analógicas procedentes del audio de TV, emisoras FM comercial o audio en banda base procedente de una entrada de línea o de micrófono.
- La controladora de DMA permite un manejo simultáneo de señales de audio y vídeo.
- Requiere un único cristal como oscilador, admite entradas de cámaras digitales a través del puerto GPIO y soporta la decodificación de la señal de teletexto.

Diagrama de bloques y funcionamiento

El Bt878 integra para la captura de vídeo un decodificador de NTSC/PAL/SECAM y de vídeo compuesto, un escalador, un controlador dma, y un bus PCI maestro en un solo dispositivo.

El Bt878 puede poner datos de vídeo directamente en la memoria principal para la captura de vídeo y en un buffer intermedio para otras aplicaciones de vídeo. Como inicialización del PCI, el Bt878 puede tomar el control del bus del PCI tan pronto como esté disponible, de tal modo que evita la necesidad de buffer intermedios. El Bt878 contiene en una memoria FIFO los datos del pixel para poder liberar el bus de alta velocidad del PCI de la toma continua de datos de vídeo. La entrada de los datos de vídeo puede ser escalada, modificada y transferida a una localización sobre un campo base. Esto permite la inspección simultánea de un campo y la captura de otro. Alternativamente, el Bt878 puede capturar ambos campos simultáneamente o ver ambos campos de antemano a la vez. Los campos se pueden entrelazar en memoria o enviar a los buffer intermedios separados del campo.

Existe un modo para la configuración y carga de los registros internos. Estos registros están situados en una memoria de 4 Kbytes a la que se accede directamente desde el bus PCI. Según la dirección y el tipo de instrucción se requiere la introducción de palabras adicionales en el bus PCI. El driver del Bt848 se encarga de gestionar a bajo nivel el proceso de configuración del hardware.

La decodificación de la señal de vídeo y el formato de digitalización de la señal depende del estado de los registros internos.

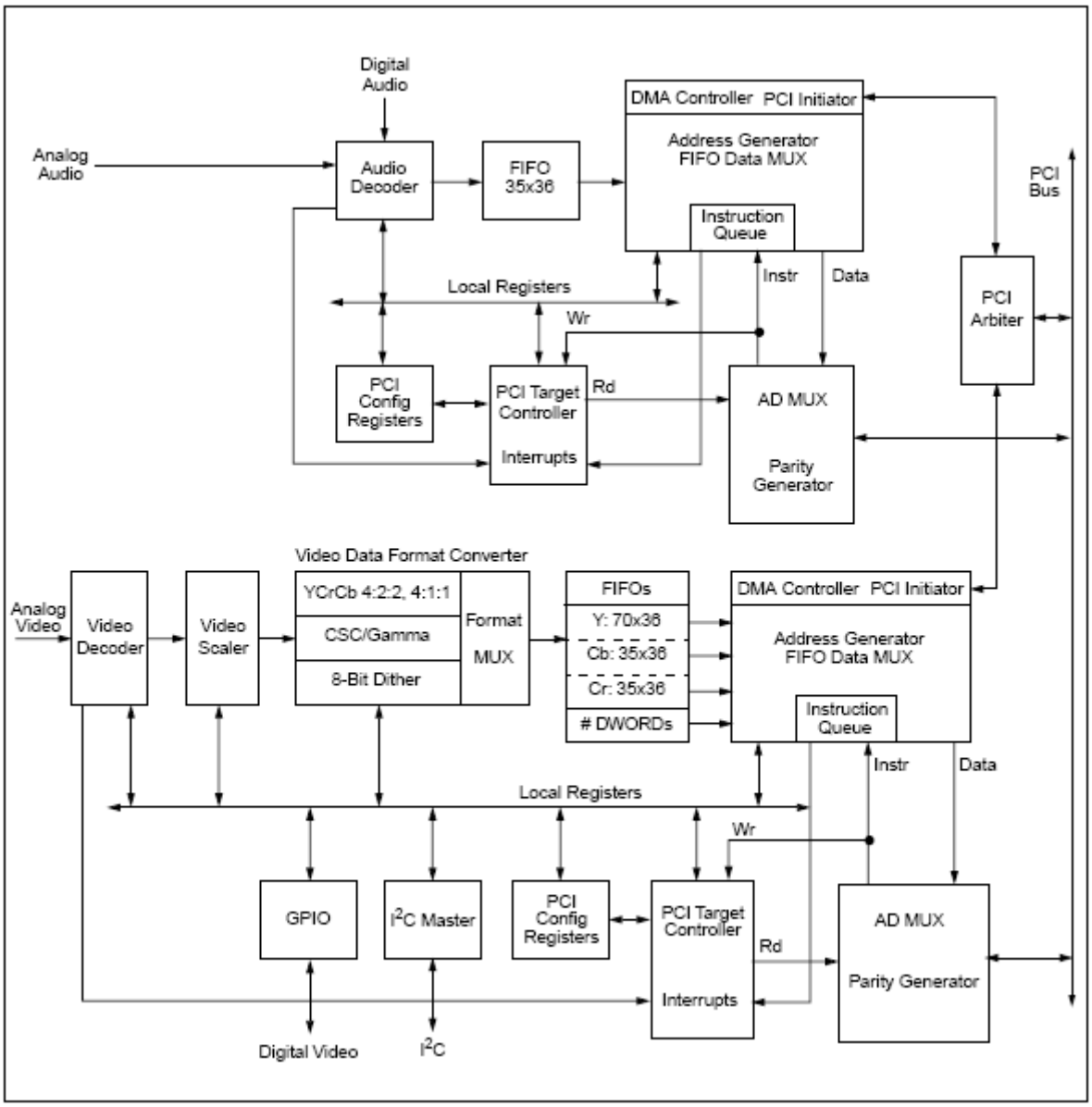

Figura 2.18: Diagrama de bloques del Bt878.

En modo de adquisición un controlador de DMA se encarga de proporcionar los datos de vídeo al bus PCI, extrayéndolos de una FIFO interna en la que se encuentran temporalmente almacenados. El controlador de DMA es una máquina RISC que recoge las instrucciones de una dirección de la memoria del host. El programa residente en el host es generado por el driver del Bt848 y proporciona su dirección de inicio en un registro que se carga durante la configuración del chip.

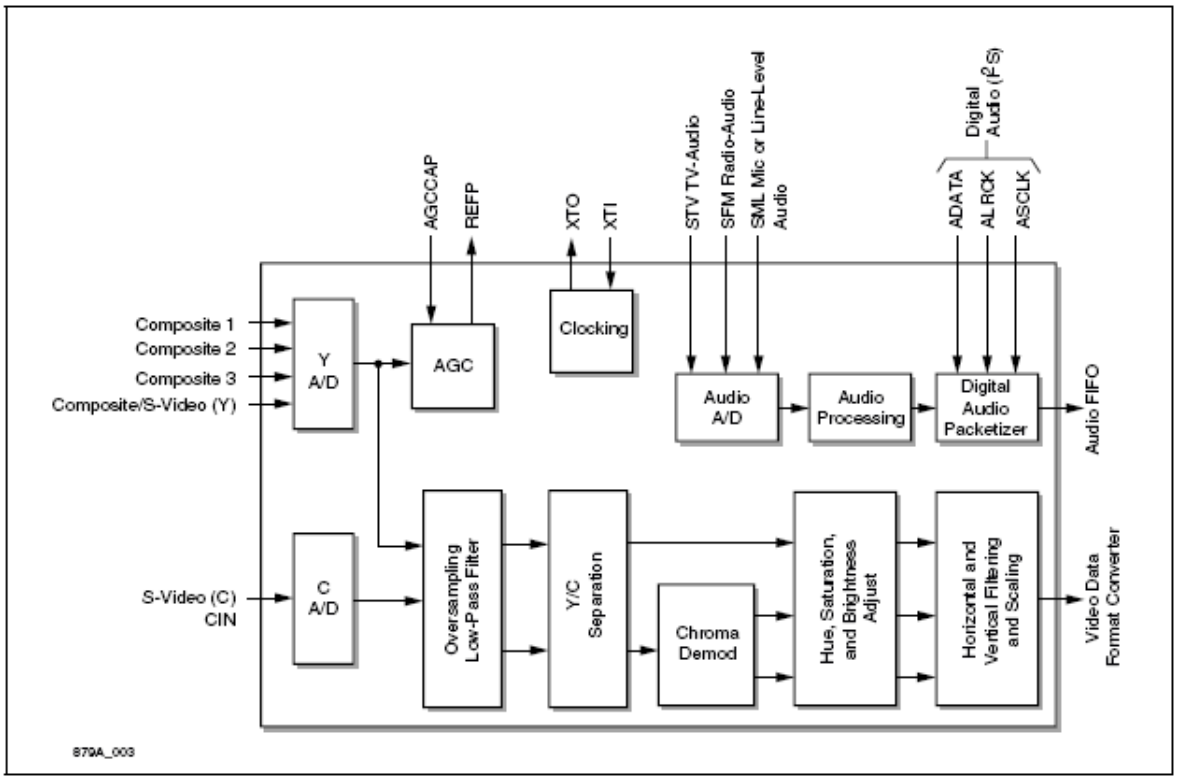

Figura 2.19: Diagrama del decodificador de vídeo del Bt878.

Patillaje

El chip bt878 presenta 128 pines (Figura 2.20) que se pueden clasificar en diversos grupos según sea la función de éstos, así tenemos:

- 50 pines para la interfaz con el bus PCI.
- 2 pines para la interfaz con el bus I^2C.
- 5 pines para estructura JTAG.
- 25 pines de próposito general entrada/salida.
- 3 pines para el testeo de señales digitales de entrada de vídeo.
- 2 pines para sincronización de señales de referencia.
- 7 pines para señales de vídeo de entrada.
- 10 pines para señales de audio TV/Radio de entrada.

- 14 pines para entrada/salida con niveles de alimentación.
- 6 pines para vídeo analógico.
- 4 pines para audio analógico.

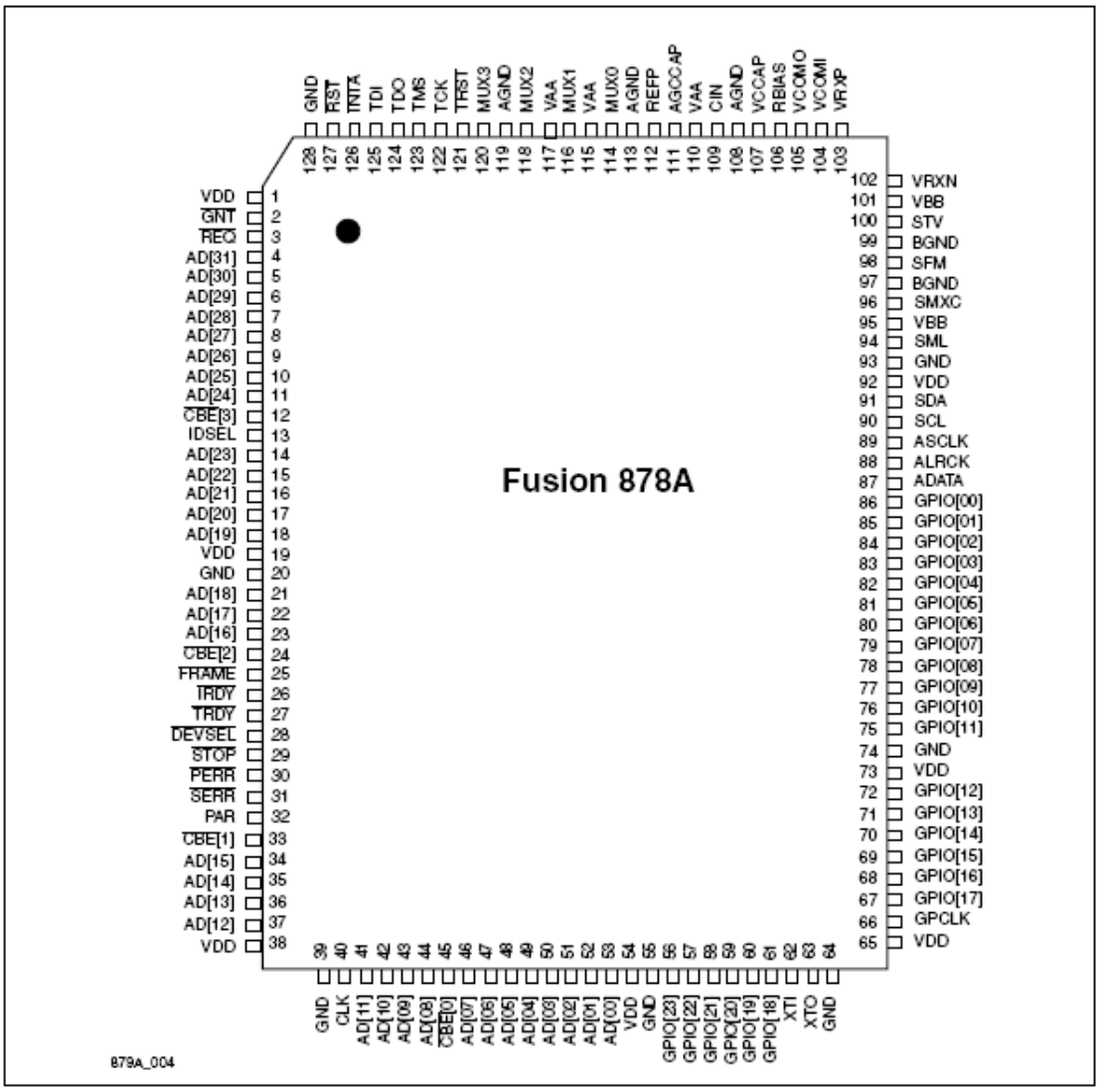

Figura 2.20: Diagrama de pines Bt878.

Características funcionales

El chip Bt878 presenta una serie de funciones adicionales para el tratamiento de las señales de vídeo, que son explicadas a continuación.

Ultra Lock^{TM}

La longitud de línea, definda como el intervalo entre los puntos medios de los bordes en los pulsos de la sincronización horizontal, de las fuentes de vídeo analógicas no es constante. Para fuentes estables tales como generadores de señal de fuente o de prueba de la calidad del estudio, esta variación es muy pequeña: ±2 ns. Sin embargo, para una fuente inestable tal como un VCR, un sintonizador de televisión, la variación de la longitud de la línea es tanto como algunos microsegundos. Los sistemas de display digital requieren un número fijo de pixeles por línea a pesar de estas variaciones. Así el chip Bt878 emplea una técnica conocida como UltraLock para implementar esta longitud de línea horizontal constante, generando el número requerido de pixeles por línea.

Para conseguir lo anteriormente mencionado se utiliza una frecuencia de muestreo de 4 Fsc. El número de muestras por línea (media) es de 910 para el NTSC y 1135 para el PAL/SECAM. El estándar establece que deberían tomarse 780 (NTSC) y 944 (PAL/SECAM. 944*(52/64)=768).

Los 944 se obtienen por interpolación a partir de las muestras reales.

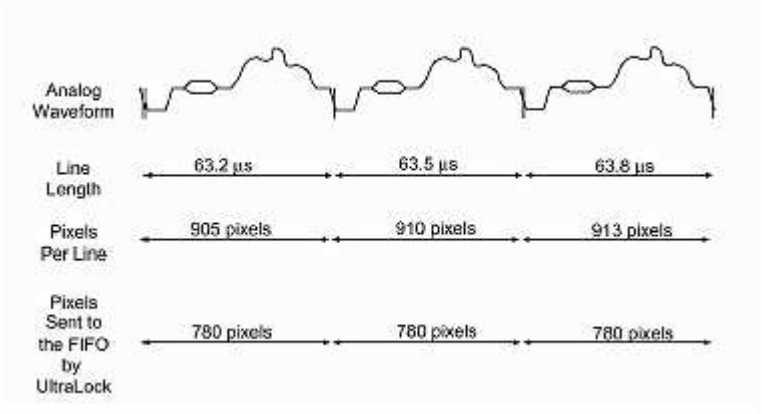

Figura 2.21: Comportamiento de la función UltraLock^{TM} para NTSC (salida de pixel cuadrado.)

Separación de luminancia y croma

Para entrada S-Vídeo (entrada de vídeo compuesto) se omite la separación de luma y croma. La componente de luminancia entra directamente en Y. La componente de croma entra al demodulador.

Para señales en B&W la señal se aplica directamente a Y y se ponen a cero los registros SAT_U y SAT_V.

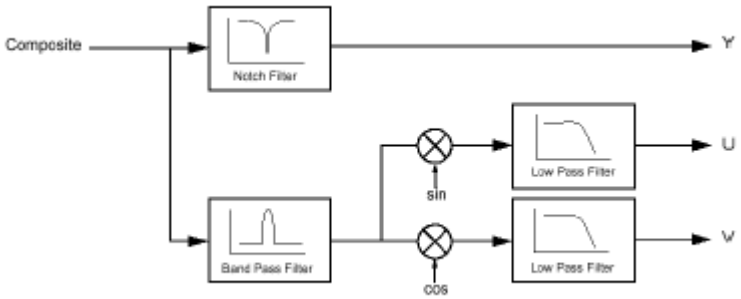

Figura 2.22: Separación Y/C y demodulación croma para vídeo compuesto.

Escalado de la señal de vídeo

Para poder operar con distintos formatos de salida (ITU-601, CIF, QCIF) es necesario aplicar filtros sobre la señal de vídeo que permitan reducir su ancho de banda y eviten la aparición de efectos de aliasing. El Bt878 dispone de la siguiente arquitectura para el filtrado de las señales en las direcciones horizontales y verticales.

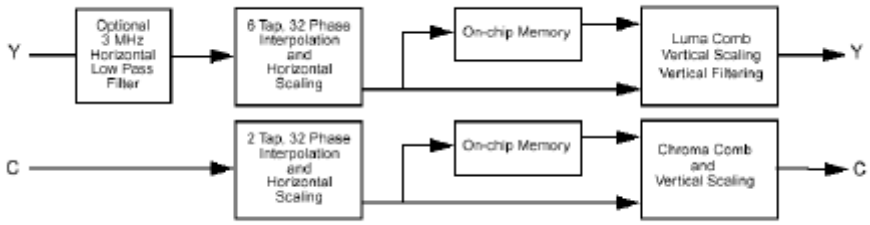

Figura 2.23: Arquitectura para el filtrado de la señal de vídeo.

A continuación se muestran las ecuaciones que rigen el escalado de la señal de vídeo, tanto para la disposición horizontal como vertical, siendo el máximo escalado permitido de razón 16:1:

- Escalado horizontal:
  - Luminancia: A + B $z^{-1} + Cz^{-2} + Dz^{-3} + Ez^{-4} + Fz^{-5}$.
  - Croma: G + H z^{-1}.
- Escalado vertical:
  - Luminancia: C + D z^{-1}.
  - Croma $\frac{1}{2}+\frac{1}{2}$ z^{-1}.
- Opciones filtrado vertical:
  - Luminancia=$\frac{1}{2}(1+z^{-1})$ = $\frac{1}{4}(1+2z^{-1}+z^{-2})$ = $\frac{1}{8}(1+3z^{-1}+3z^{-2}+z^{-3})$ = $\frac{1}{16}(1+4z^{-1}+6z^{-2}+4z^{-3}+z^{-4})$.

Los valores de los coeficientes se determinan a partir del factor de escala y del subsistema de UltraLock.

En la Figura 2.24 se puede observar diferentes respuestas en frecuencia para los distintos formatos de salida (ICON, QCIF, CIF).

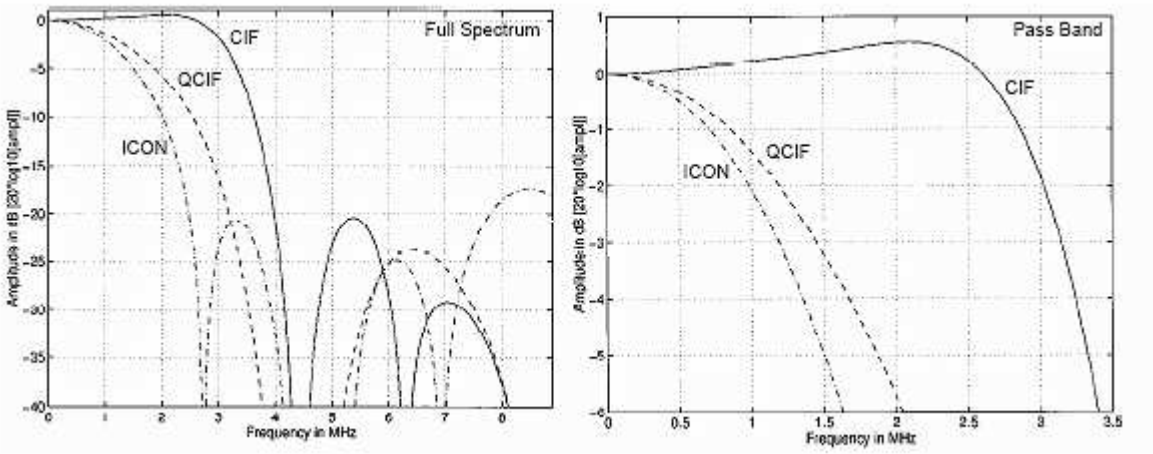

Figura 2.24: Ejemplo de respuestas en frecuencia.

Los coeficientes se determinan a partir de los valores existentes en los registros HSCALE y VSCALE. A continuación, se muestra un ejemplo de como se calculan los valores de los registros de escala para distintos formatos:

- HSCALE:
  - NTSC: HSCALE = [(754/HACTIVE)-1]*4096.
  - PAL/SECAM: HSCALE = [(922/HACTIVE)-1]*4096.
- VSCALE (NTSC/PAL/SECAM) = (0x10000-[scaling_ratio-1]*512)&0x1FFF siendo scaling_ratio = (4/1 para QCIF píxel cuadrado).

En la Figura 2.25 se pueden observar los valores que toman los registros de escala HSCALE y VSCALE para distintos formatos.

Scaling Ratio	Format	Total Resolution(1)	Output Resolution (Active Pixels)	HSCALE	VSCALE Register Values	
				Register Values	Use Both Fields	Single Field
Full Resolution 1:1	NTSC SQ Pixel	780x525	640x480	0x02AA	0x0000	N/A
	NTSC CCIR601	858x525	720x480	0x00F8	0x0000	N/A
	PAL CCIR601	864x625	720x576	0x0504	0x0000	N/A
	PAL SQ Pixel	944x625	768x576	0x033C	0x0000	N/A
CIF 2:1	NTSC SQ Pixel	390x262	320x240	0x1555	0x1E00	0x0000
	NTSC CCIR601	429x262	360x240	0x11F0	0x1E00	0x0000
	PAL CCIR601	432x312	360x288	0x1A09	0x1E00	0x0000
	PAL SQ Pixel	472x312	384x288	0x1679	0x1E00	0x0000
QCIF 4:1	NTSC SQ Pixel	195x131	160x120	0x3AAA	0x1A00	0x1E00
	NTSC CCIR601	214x131	180x120	0x3409	0x1A00	0x1E00
	PAL CCIR601	216x156	180x144	0x4412	0x1A00	0x1E00
	PAL SQ Pixel	236x156	192x144	0x3CF2	0x1A00	0x1E00
ICON 8:1	NTSC SQ Pixel	97x65	80x60	0x861A	0x1200	0x1A00
	NTSC CCIR601	107x65	90x60	0x7813	0x1200	0x1A00
	PAL CCIR601	108x78	90x72	0x9825	0x1200	0x1A00
	PAL SQ Pixel	118x78	96x72	0x89E5	0x1200	0x1A00
Notes: (1). Including sync and blanking interval.						

Figura 2.25: Cálculo de los registros de escala para distintos formatos.

Cropping

La función cropping permite definir la parte de la imagen que se enviará a la memoria del host. Se controla mediante los registros de 10 bits HDELAY, HACTIVE, VDELAY, VACTIVE. Los dos bits más significativos de cada registro se establecen en el registro CROP. Los 8 bits restantes se establecen en los registros HDELAY_LO, HACTIVE_LO, VDELAY_LO y VACTIVE_LO.

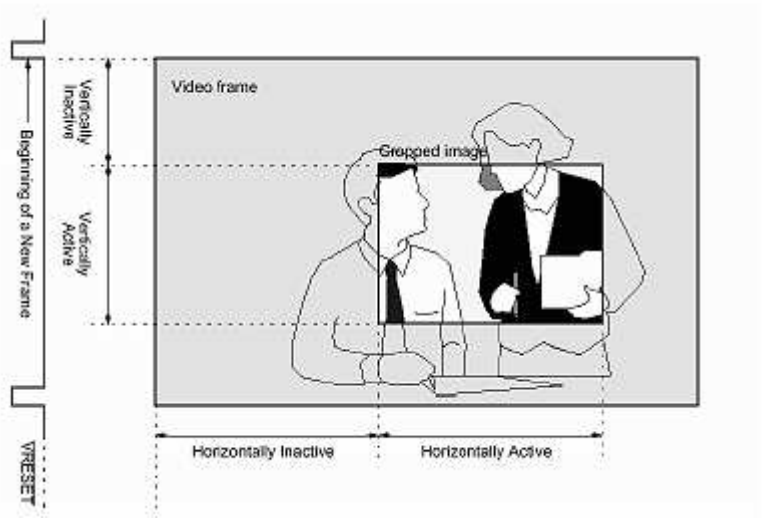

Figura 2.26: Ejemplo de cropping.

Cropping y escalado

El chip Bt878 permite la realización de las funciones de cropping y escaldo simultáneamente, es decir, el cropping puede usarse combinado con los factores de escala vertical y horizontal. Un ejemplo de esto se puede ver en la Figura 2.27.

Diezmado temporal

El Bt878 permite especificar un factor de compresión temporal para reducir el número de frames por segundo. El valor se especifica en el registro TDEC (Time Decimation). Con ello, puede controlarse el flujo de datos que es capaz de soportar el sistema (limitaciones de tiempo de acceso del disco duro, ancho de banda real del bus, etc)

TDEC se carga con un número comprendido entre 0-60 para NTSC y 0-50 para PAL. Este número se asigna a los 6 bits menos significativos.

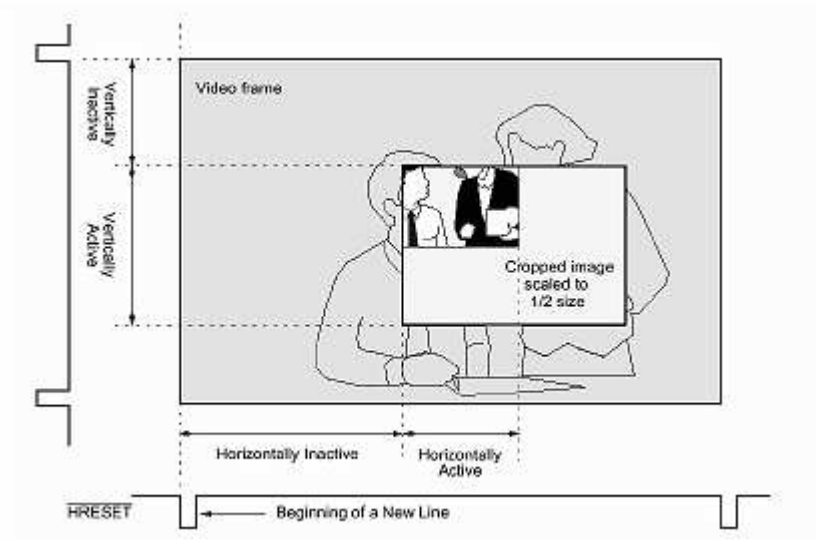

Figura 2.27: Ejemplo de cropping y escalado conjuntamente.

Los dos bits más significativos establecen si el diezmado se realiza en base a frames (00) o en base a fields (01). Cuando un campo o cuadro se considera inactivo se señaliza mediante un pin del Bt878 (ACTIVE).

Ajustes de vídeo

El chip Bt878, entre sus numerosas funciones, también nos permite realizar los siguientes ajustes de vídeo:

- Control de tono (HUE): Permite desplazar las fases de las componentes de croma respecto a la subportadora de color entre –90 y 90. Sólo está disponible cuando se opera en NTSC.
- Contraste (CONTRAST): La señal de luminancia se multiplica por el valor cargado en este registro. El contraste puede mejorar entre un 0 % y un 200 %.
- Saturación (SAT_U, SAT_V): Es la ganancia multiplicativa que se asigna a cada componente de color.
- Brillo (BRIGHT): Es un offset que se añade a la señal de luminancia. El registro puede tomar valores entre –128 y +127.

Control automático de ganancia de croma

El control automático de ganancia puede activarse o desactivarse. Se encarga de comparar el nivel del burst con el sincronismo y ajusta automáticamente la ganancia del decodificador de croma para realizar la compensación de color.

Detección y supresión de croma para niveles bajos de burst

La detección y supresión de croma para niveles bajos de burst puede activarse o desactivarse. Se encarga de omparar el nivel del burst con el sincronismo y si es inferior a un 35 % (25 % en NTSC) del valor nominal durante 127 líneas consecutivas se suprimen las componentes de color U y V (128). El decodificador de color vuelve a activarse cuando el nivel del burst supera el 60 % del valor nominal (43 % NTSC) durante 127 líneas consecutivas.

Coring

El coring consiste en forzar que todos los niveles de luminancia por debajo de un umbral sean reconvertidos a cero. Es útil debido a que el Sistema Visual Humano es más sensible a las variaciones que se producen en la proximidad del nivel negro. El chip Bt878 permite seleccionar un nivel de umbral de 0, 8, 16 o 32.

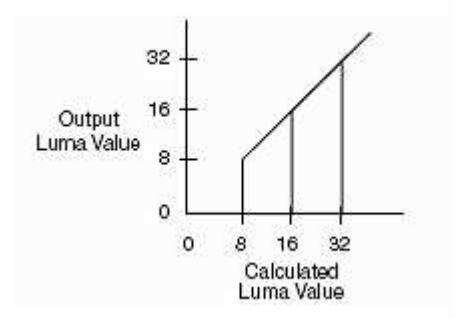

Figura 2.28: Ejemplo de coring.

Formato de datos VBI

El chip Bt878 permite la captura de los datos VBI de la señal de vídeo que se almacenan en la memoria del host para un tratamiento posterior (Normalmente decodificación de Teletexto).

Un frame del vídeo se compone de 525 líneas para NSTC y 625 para PAL/SECAM. La figura 2.29 ilustra un frame de vídeo para el formato PAL, en el cual nos encontramos con un número de diferentes regiones. Los datos de la imagen de vídeo estan contenidos en los campos impares y uniformes dentro de las líneas 24 a 310 y de 336 a 625, respectivamente. Cada campo de vídeo también contiene una región que presenta información sobre la sincronización vertical (líneas 1 a 6 y 311 a 318) así como una región que pueda contener los datos en blanco del vídeo (líneas 7 a 23 y 319 a 335). Estas regiones entre la región de la sincronización vertical y la región vídeo del cuadro se refieren a la porción de VBI de la señal vídeo.

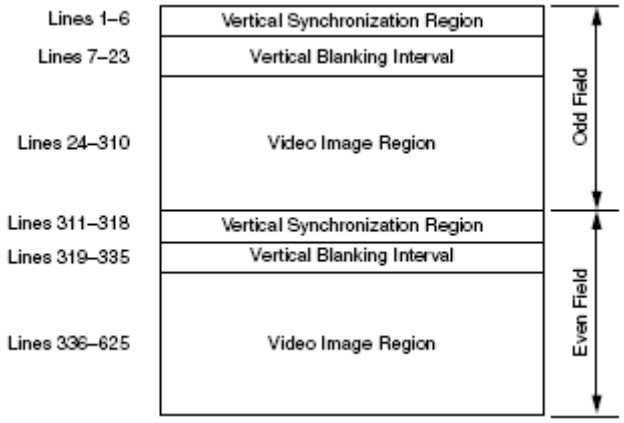

Figura 2.29: Ejemplo de regiones en formato vídeo PAL.

Unidad de conversión de formatos de vídeo

El bloque decodificador/escalador del vídeo genera una corriente de datos de vídeo en el formato YCrCb (packed 4:2:2). Los datos de vídeo son entonces convertidos y ajustados a un formato en un DWORD de 32 bit. En la Figura 2.30 se pueden ver los distintos formatos de vídeo que se pueden obtener.

La Figura 2.31 ilustra los pasos para convertir los datos de vídeo YCrCb al formato deseado.

Format	Dword	Pixel Data [31:0]			
		Byte Lane 3 [31:24]	Byte Lane 2 [23:16]	Byte Lane 1 [15:8]	Byte Lane 0 [7:0]
RGB32(1)	dw0	Alpha	R	G	B
RGB24	dw0	B1	R0	G0	B0
	dw1	G2	B2	R1	G1
	dw2	R3	G3	B3	R2
RGB16	dw0	{R1[7:3],G1[7:2],B1[7:3]}		{R0[7:3],G0[7:2],B0[7:3]}	
RGB15	dw0	{0,R1[7:3],G1[7:3],B1[7:3]}		{0,R0[7:3],G0[7:3],B0[7:3]}	
YUY2—YCrCb 4:2:2(2)	dw0	Cr0	Y1	Cb0	Y0
	dw1	Cr2	Y3	Cb2	Y2
BtYUV—YCrCb 4:1:1	dw0	Y1	Cr0	Y0	Cb0
	dw1	Y3	Cr4	Y2	Cb4
	dw2	Y7	Y6	Y5	Y4
Y8 (Gray Scale)	dw0	Y3	Y2	Y1	Y0
8-bit Dithered	dw0	B3	B2	B1	B0
VBI Data	dw0	D3	D2	D1	D0
YCrCb 4:2:2 Planar	dw0 FIFO1	Y3	Y2	Y1	Y0
	dw1 FIFO1	Y7	Y6	Y5	Y4
	dw0 FIFO2	Cb6	Cb4	Cb2	Cb0
	dw0 FIFO3	Cr6	Cr4	Cr2	Cr0
YUV12 Planar	Vertically sub-sampled to 4:2:2 by the DMA controller				
YCrCb 4:1:1 Planar	dw0 FIFO1	Y3	Y2	Y1	Y0
	dw1 FIFO1	Y7	Y6	Y5	Y4
	dw2 FIFO1	Y11	Y10	Y9	Y8
	dw3 FIFO1	Y15	Y14	Y13	Y12
	dw0 FIFO2	Cb12	Cb8	Cb4	Cb0
	dw0 FIFO3	Cr12	Cr8	Cr4	Cr0
YUV9 Planar	Vertically sub-sampled to 4:1:1 by the DMA controller				

Notes: (1). The alpha byte can be written as 0 data, or not written.
(2). UYVY can be achieved by byte swapping.
3. All planar modes require HACTIVE register to be multiple of 16 pixels.

Figura 2.30: Formatos de vídeo.

Como se puede observar, podemos conseguir diferentes formatos de vídeo, así tendremos los modos RGB, YUV, YcrCb,...

Para pasar a componentes de vídeo RGB es necesario convertir el formato 4:2:2 a un formato 4:4:4. Para ello, es necesario interpolar las componentes de croma. Las ecuaciones de interpolación son:

Para n=0,2,4, etc

$$Cb_n{=}Cb_n$$

$$Cr_n{=}Cr_n$$

$$Cb_{n+1}{=}(Cb_n{+}Cb_{n+2})/2$$

$$Cr_{n+1}{=}(Cr_n{+}Cr_{n+2})/2$$

Las ecuaciones que se aplican para la conversión a RGB son:

$$\mathrm{R} = 1.164\ (\mathrm{Y\text{-}16}) + 1.596\ (\mathrm{Cr\text{-}128})\mathrm{Y}\varepsilon[16{,}235]$$

$$\mathrm{G} = 1.164\ (\mathrm{Y\text{-}16}) - 0.813\ (\mathrm{Cr\text{-}128}) - 0.39\ (\mathrm{Cb\text{-}128})\ \mathrm{Cr/Cb}\varepsilon[16{,}240]$$

$$\mathrm{B} = 1.164\ (\mathrm{Y\text{-}16}) + 2.018\ (\mathrm{Cb\text{-}128})\ \mathrm{RGB}\ \varepsilon[0{,}255]$$

La conversión de 4:2:2 a 4:1:1 se realiza mediante las ecuaciones:

$$Cb_n{=}(Cb_n{+}Cb_{n+2})/2$$

$$Cr_n{=}(Cr_n{+}Cr_{n+2})/2$$

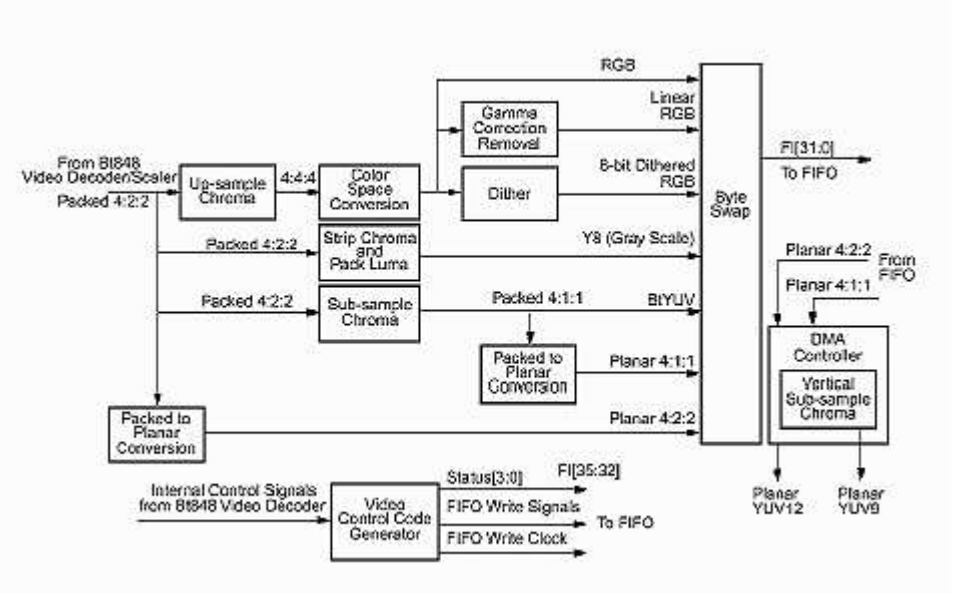

Figura 2.31: Estructura de la conversión de formato de datos de vídeo.

Extracción de la corrección gamma

Puede cancelarse la corrección gamma de la señal de vídeo para su representación en pantallas del tipo LCD. Sólo puede realizarse la extracción de la corrección gamma en modos RGB (No es posible en modos (YCbCr)). La compensación de la corrección gamma depende del sistema de vídeo:

$$\text{RGBout}=(\text{RGBin})2.2 \text{ NTSC}$$
$$\text{RGBout}=(\text{RGBin})2.8 \text{ PAL/SECAM}$$

Estructura de los datos en la FIFO

La memoria FIFO presenta un total de 630 bytes que se organizan lógicamente en tres segmentos:

1. FIFO1 que consta de 70 palabras de 36 bits cada una.
2. FIFO2 que consta de 35 palabras de 36 bits cada una.
3. FIFO3 que consta de 35 palabras de 36 bits cada una.

Cada una de las 140 palabras que forman la FIFO constan de una palabra (DWORD) con los datos del pixel y de cuatro bits de estado para el control del vídeo como se puede observar en la Figura 2.32. La FIFO es bastante eficiente para la transferencia de datos de 16 a 32 bits en el modo plano como en el modo empaquetado. Los bits de estado se utilizan para codificar la información sobre los píxeles y el estado de la unidad de tiempo del vídeo.

La información sobre el tiempo de vídeo y el control se pasan a través de la FIFO junto con los datos. La FIFO es la unidad que aísla los dominios del vídeo de entrada y la salida PCI.

El control de la entrada de datos en la FIFO sólo puede proceder de los registros del Bt878 y del decodificador de vídeo. El control de los datos de salida sólo se puede controlar a través del controlador de DMA y de la secuencia de instrucciones RISC que se encuentra almacenada en el HOST y que se ejecuta por el controlador de DMA.

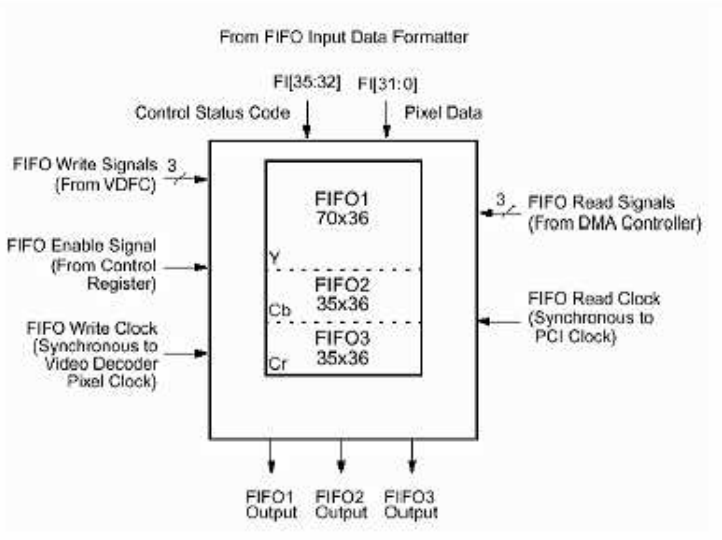

Figura 2.32: Estructura de la memoria FIFO.

Status[3:0]	Code	Description
0110	FM1	FIFO Mode: packed data to follow
1110	FM3	FIFO Mode: planar data to follow
0010	SOL	First active pixel/data DWORD of scan line
0001	EOL	Last active pixel/data DWORD of scan line, 4 valid bytes
1101	EOL	Last active pixel/data DWORD of scan line, 3 valid bytes
1001	EOL	Last active pixel/data DWORD of scan line, 2 valid bytes
0101	EOL	Last active pixel/data DWORD of scan line, 1 valid byte
0100	VRE	VRESET following an even field–falling edge of FIELD
1100	VRO	VRESET following an odd field–rising edge of FIELD
0000	PXV	Valid pixel/data DWORD

Figura 2.33: Bits de estado.

En la Figura 2.33 se representan los bits de estado que entran en juego en la escritura de datos en la FIFO:

El controlador de DMA es el único responsable de que en la FIFO no se produzcan sobreescrituras de los datos, pudiendo leer y descartar datos, que no se envían al bus PCI, cuando la FIFO está próxima a desbordar.

La lectura y escritura asíncrona de los datos en la FIFO se realiza mediante un contador que indica el nivel de datos en la FIFO está próximo al desbordamiento. En este

caso, el controlador de DMA debe descartar la lectura durante la próxima instrucción. Mientras, la escritura de datos es independiente de la lectura y ambas son asíncronas. En el peor caso (PAL) la frecuencia de píxel es de 17,73 MH. No obstante, la frecuencia real de escritura en cada una de las FIFOS es algo menor debido a que en una DWORD se comparten varios píxeles. La frecuencia de lectura de los datos en el bus PCI puede tener un máximo de 33 MHz. En los sistemas en los que la frecuencia del bus PCI sea inferior a los 33MHz es posible que no pueda trabajarse con todos los modos de vídeo que admite el Bt878.

Controlador DMA

El controlador de DMA tiene una arquitectura de máquina RISC que proporciona una gran flexibilidad para proporcionar los datos en la memoria.

El conjunto de instrucciones que se ejecuta se mantiene en la memoria del sistema y se genera a partir del driver software del Bt878. De esta forma, el controlador de DMA puede cambiar la dirección de memoria en función de las características de los datos de la FIFO. Así, es posible mantener 4 posibles direcciones de memoria: campo par, campo impar, linea 21 y línea 15 (NTSC).

El programa RISC pone su propia dirección de memoria en el bus PCI durante el proceso de configuración y carga de registros. Las instrucciones RISC que están disponibles son WRITE, SKIP, SYNC y JUMP. Los datos del vídeo compuesto codificados se almacenan en la FIFO. El controlador DMA después presenta los datos en la inicialización del PCI y solicita que los datos estén disponibles para su salida. El PCI hace que se vuelquen los datos de pixel sobre el bús del PCI. Es la responsabilidad del controlador DMA prevenir y manejar el desbordamiento de la memoria FIFO. Este proceso se ilustra en la Figura 2.34.

Para la programación del RISC, partimos de que hay dos regiones independientes de instrucciones del RISC en la memoria del host: uno para el campo impar y el otro para el campo uniforme. El primer campo comienza con una instrucción de sincronización que indica el tipo de datos (empaquetados o planos) de la FIFO (STATUS[3:0] = FM1 o FM3). El primer campo termina con una instrucción SYNC que indica el siguiente campo par o impar para continuar (STATUS[3:0] = VRE o VRO). El segundo campo comienza con una instrucción SYNC que indica el tamaño del campo, seguida por una instrucción de salto (JUMP) de nuevo al primer campo.

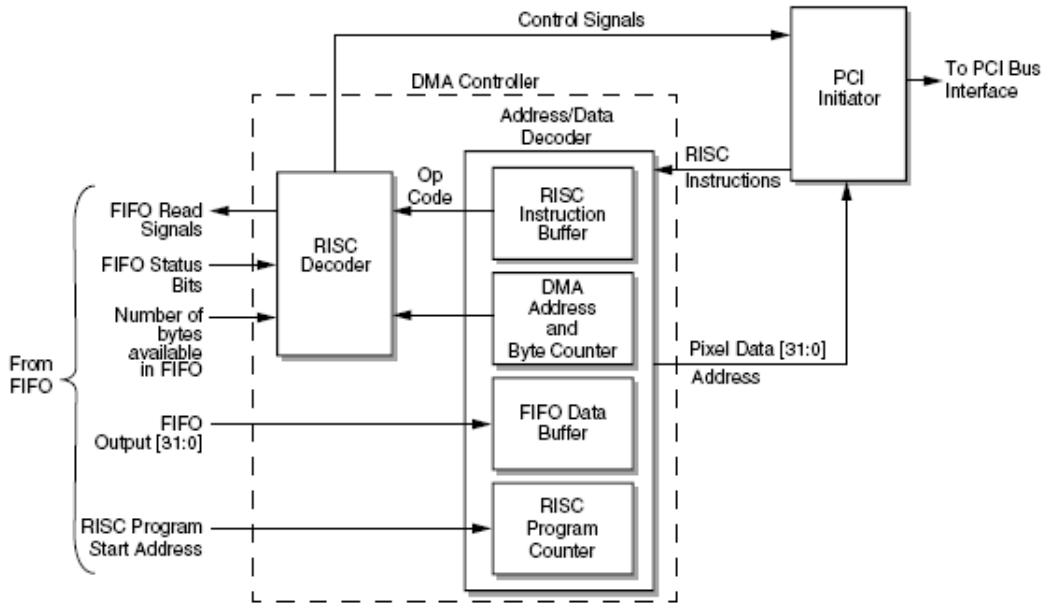

Figura 2.34: Diagrama de bloques RISC audio/vídeo.

Las instrucciones SYNC permiten la sincronización de la salida de la FIFO y de los puntos del programa comienzo/final del RISC.

El software instalará el flujo de datos de pixel creando una secuencia de instrucciónes del RISC en la memoria del host para el campo par y para el campo impar, independientemente. El controlador DMA ramifica normalmente la secuencia de instrucción del RISC a través de instrucciones de salto. La secuencia del programa del RISC será cambiada solamente cuando los parámetros del modo vídeo hayan sido cambiados, sino el controlador DMA ejecutará continuamente su programa.

Instrucciones RISC

Hay cinco tipos de instrucciones, para el modo RISC empaquetado (WRITE, WRITEC, SKIP, SYNC y JUMP), de control de los datos almacenados en la FIFO. Hay otras tres instrucciones adicionales para el modo plano, que substituyen a las instrucciones del modo empaquetado WRITE/SKIP. Los detalles de cada instrucción se pueden ver en el datasheet del Bt878.

Cada instrucción del RISC esta formada de 1 a 5 DWORDs. Los 32-bits contienen información del opcode, de la dirección base, del estado, de la sincronización, de los bytes activos/inactivos y del comienzo/final de la línea de código.

El bit SOL en las instrucciones de escritura (WRITE) y de salto (JUMP) indica que esta instrucción es la primera de la línea de exploración o de escaneado. El bit EOL en las instrucciones de escritura (WRITE) y SKIP indica que esta instrucción es la última de la línea de escaneado. Un flag EOL en la FIFO y los DWORDs pasados para la línea de escaneado, coinciden en terminar con la instrucción pasada de la línea de escaneado. Si ocurre la condición de la FIFO, la instrucción actual y todas las instrucciones que conducen a la que contiene el flag EOL serán ignoradas. Si solamente hay una instrucción para el procesado de la línea, los bit SOL y EOL serán fijados de antemano.

WRITE, WRITEC, y SKIP controlan el proceso de activación de los datos de pixel en la FIFO. Estas tres instrucciones de control de la secuencia de datos del modo empaquetado escriben en una dirección el contenido de la FIFO. La instrucción de WRITEC no provee una dirección base. En su lugar, confía en el DMA que presenta una dirección base en su contador. Este valor de dirección se va actualizando continuamente y guardandose durante los saltos. Sin embargo, WRITEC no se puede utilizar para comenzar una nueva línea; es decir, esta instrucción no puede tener el bit SOL fijado.

WRITE123, WRITE1S23, y SKIP123 controlan el proceso de activación de los datos de pixel en la FIFO. Estas tres instrucciones de control de la secuencia de datos del modo plano escriben en una dirección el contenido de la FIFO. La instrucción WRITE1S23 soporta la acción de diezmado adicional del chroma en una línea base. Para cada una de estas instrucciones, serán procesados el mismo número de bytes para la FIFO2 y para la FIFO3.

La instrucción de salto (JUMP) es útil para repetir el mismo programa para el campo par y para el campo impar, o para cambiar a un nuevo programa cuando la secuencia necesita ser cambiada sin interrumpir el flujo de datos.

Se utiliza la instrucción SYNC para sincronizar el programa del RISC y la secuencia de datos de pixel. El controlador DMA realiza la sincronización mediante los bits de estado en DWORD0 de la instrucción SYNC y emparejándolo a los cuatro bits de estado de la FIFO, junto con los datos de pixel. Una vez que el controlador DMA haya emparejado los bits de estado entre la FIFO y la instrucción del RISC, se procede a la salida de datos.

Antes de establecer la sincronización, el controlador DMA lee y desecha los datos de la FIFO.

Los códigos 0000 y 1111 se reservan para detectar errores en las instrucciones. Si se detectan estos códigos o otros códigos no disponibles, se tendrá una interrupción. El controlador DMA parará la ejecución del programa RISC hasta que se vuelve a habilitar su ejecución. Esto también es aplicado a la instrucción SYNC que presenta códigos reservados. La detección de errores en las instrucciones del RISC es útil para detectar errores del software en la programación, o asegurarse de que el controlador DMA está siguiendo una secuencia válida del RISC. Es decir, se asegura de que el contador de programa no esté señalando a una localización incorrecta. Todos los bits reservados de la instrucción DWORDs se deben fijar a 0.

Ejecución de instrucciones del RISC

La ejecución de instrucciones comienza una vez que el controlador DMA haya alcanzado la sincronización entre la FIFO y el programa del RISC, comienzando a ejecutarse las instrucciones del RISC. Los datos de la FIFO serán alineados con los bytes de datos esperados por las instrucciones del RISC. El controlador DMA lee instrucciones del RISC y realiza la escritura en la FIFO. El controlador DMA se puede programar esperando 4, 8, 16, o 32 DWORDs de la FIFO antes de ejecutar una instrucción de escritura (WRITE). Fijado este punto en el trigger de la FIFO, la eficacia del bus se optimiza no permitiendo que el controlador DMA tenga acceso al bus cada vez que un DWORD se introduce en la FIFO. Sin embargo, no se hace caso al triggerde la FIFO cuando el controlador DMA esta cerca del final de una instrucción y el número de DWORDs a la izquierda a transferir es menos que el número de DWORDS en la FIFO. Permitiendo que la instrucción termine, si en la FIFO el trigger está debajo de su punto de disparo, las instrucciones del RISC se pueden limpiar rápidamente para cada línea de escaneado. Si no, el controlador DMA puede tener que esperar muchas líneas de escaneado antes de que el número requerido de DWORDs esté presente en la FIFO, especialmente al capturar imágenes altamente reducidas. Puede haber varias lineas horizontales antes de que otro DWORD introduzca datos en la FIFO.

El controlador DMA no hace caso al trigger de la FIFO durante las instrucciones SKIP. En el modo plano, el punto de disparo del trigger para la FIFO se debe fijar al mismo nivel, aunque los datos del luma se esten almacenando en la Y de la FIFO por lo menos tan rápidamente que los datos del chroma se están almacenando dos veces en el cr y los Cbes de la FIFO. Esto se asegura de que la Y de la FIFO sea seleccionada primero para poner datos sobre el bus del PCI.

Cuando el inicializador se desconecta del bus del PCI en el modo plano, es esencial recuperar el control del bus cuanto antes y entregar cualquier DWORDs. El controlador DMA no hará caso del punto del disparador de la FIFO porque necesita vaciar la FIFO inmediatamente. Si no puede tener la ocasión de vaciar el resto de las FIFOs, abandonará el bus. Esto no es una preocupación en el modo empaquetado porque las tres FIFOs se tratan como una FIFO única. Cuando en el PCI se detecta un error de paridad mientras que el inicializador del PCI está leyendo los datos de la instrucción, se produce una interrupción.

Capítulo 3

Procesado

En este capítulo se abordarán todos los temas relacionados con el procesamiento de imágenes.

Primeramente veremos en que consiste el lenguaje de programación C++, comentando sus principales características, observando que estructura presenta un programa realizado con este tipo de lenguaje e introduciremos una serie de conceptos relacionados con la programación tales como la definición de clases, estructuras y objetos, la definición de constructores y destructores, en qué consiste la sobrecarga de funciones, como se asigna memoria dinámica y finalmente, que es una biblioteca de funciones, que archivos la componen y como se crea.

A continuación, se verá el DJGPP que se pude definir como un sistema de desarrollo en C/C++ de 32 bits para ordenadores 386 y compatibles que se ejecuta en MS-DOS, donde haremos una pequeña introducción histórica y daremos los detalles técnicos del mismo. Además se verá el entorno de desarrollo integrado RHIDE, comentando sus principales características de funcionamiento.

Y por último, trataremos el lenguaje ensamblador, viendo una pequeña historia de él, mencionando sus principales ventajas e inconvenientes y finalmente se tratará la arquitectura Intel 80x86, sus principales características, su evolución histórica hasta nuestros días, y las nuevas tecnologías desarrolladas.

3.1. Lenguaje C++

El lenguaje C es un lenguaje de alto nivel, basado en funciones, que permite desarrollos estructurados. Entre otras muchas características contempla la definición de estructuras de datos, recursividad o indirecciones a datos o código (punteros).

Por su parte, C ++ es un superconjunto de C, al que recubre con una capa de soporte a la programación orientada a objetos (POO) que permite por tanto la definición, creación y manipulación de objetos.

El lenguaje C++ se conoce como un lenguaje compilado. Existen dos tipos de lenguaje: interpretados y compilados. Los interpretados son aquellos que necesitan del código fuente para funcionar (P.ej: Basic). Los compilados convierten el código fuente en un fichero objeto y éste en un fichero ejecutable. Este último es el caso del lenguaje C++.

El lenguaje C++ es un lenguaje que apoya la programación orientada a objetos y que se construyó basado en el lenguaje C debido a que sus desarrolladores les interesaba crear un lenguaje que soportara la programación orientada a objetos y que al mismo tiempo tuviera las características de eficiencia en la generación de código, flexibilidad y uso de funciones de bajo nivel del lenguaje C; además existían muchos programadores de C que emigrarían fácilmente a C++ al ver las similitudes de ambos.

Cada vez que queremos ser capaces de resolver problemas de una manera más eficiente se han creado los distintos paradigmas de programación. De esta manera surge la programación estructurada, la programación funcional, la modular y así sucesivamente.

La programación orientada a objetos es un paradigma que surge debido a que los sistemas a desarrollar se fueron haciendo cada vez mas extensos y complejos y los paradigmas existentes no pudieron manejar convenientemente este incremento de complejidad y extensión, por lo que se hizo necesario implementar una nueva manera de resolver los problemas a la que se denominó programación orientada a objetos.

Con la programación orientada a objetos se pretende crear programas en los que sus partes (entidades u objetos), correspondan fielmente a los objetos que conforman el problema a resolver y que incorpora características como la abstracción de datos, la herencia y el polimorfismo, que permiten el desarrollo de programas en los que es posible la reutilización de código y en los que se protegen los datos.

3.1.1. Características

El lenguaje de programación C++ presenta las siguientes características, muchas de ellas heredadas de su antecesor, el lenguaje C:

- Podemos decir que es un lenguaje de nivel medio, ya que combina elementos de lenguaje de alto nivel con la funcionalidad del lenguaje ensamblador.
- Es un lenguaje estructurado, ya que permite crear procedimientos en bloques dentro de otros procedimientos. Hay que destacar que el C++ es un lenguaje estándar, ya que permite utilizar el mismo código en diferentes equipos y sistemas informáticos: el lenguaje es independiente de la arquitectura de cualquier máquina en particular.
- Se puede calificar como lenguaje relativamente pequeño; se puede describir en poco espacio y aprender rápidamente.
- C++ es fuertemente tipeado, es decir, que cada objeto debe pertenecer a un cierto tipo y que cada operación tal como la asignación o la comparación son solamente permitidas entre objetos del mismo tipo. Dado que las funciones requieren entradas de un cierto tipo no aceptarán entradas de otro tipo. Sin embargo, esto no es totalmente cierto en un sentido estricto ya que hay algunas reglas que permiten conversiones entre tipos (por ejemplo, un entero puede temporalmente ser cambiado temporalmente a un número real para realizara alguna operación determinada).
- Proporciona el concepto de clase, un tipo de registro que combina datos y las funciones que operan sobre ellos.
- Es posible sobrecargar operadores con clases definidas por el usuario. Por ejemplo una clase definida por el usuario podría ser una de números racionales que pudiera implementar la operación de adición ordinaria usando el operador +. Como consecuencia, estas clases pueden comportarse mas como tipos que incorporan tipos.
- Soporta tipos parametrizados o templates. Las funciones templates pueden trabajar sobre diferentes tipos de entradas. Por ejemplo, es posible escribir una simple función swap que trabaje sobre todos los tipos posibles. Sin embargo, las dos variables serán verificadas para asegurar la correspondencia entre sus tipos. Con los templates es posible definir que una clase de arreglos que trabaje en forma booleana, carácteres, enteros, reales, entre otros.

- Soporta herencia, un mecanismo que hace posible la construcción de nuevas clases (llamadas clases derivadas) sobre las clases existentes (llamadas las clases base) sin tener que repetir el código de la clase base para cada nueva clase. Herencia es un gran avance para la reutilización de código.
- Soporta polimorfismo aún cuando es fuertemente tipeado, una variable de apuntadores del tipo de la clase base puede dinámicamente asumir el tipo de la clase derivada. Junto con la herencia, esto vuelve a C++ un lenguaje orientado a objetos completamente maduro.
- Viene con dos librerías Estándar Library y Estándar Template Library (STL), cada una de estas librerías extiende las capacidades del lenguaje base:
  - La Standard Library proporciona todas las viejas librerías de C así como también las facilidades de entrada y salida.
  - La STL proporciona una librería de tipos de contenedores (tipos que mantienen o çontienençolecciones de objetos) así como también un conjunto de algoritmos de propósitos generales para estructuras de datos comunes que se denominan algoritmos tipo "attendant". Es decir, los suplementos de STL son tipos empotrados de C++ con vectores, listas ligadas, árboles balanceados y otros tipos útiles.
- Permite variables de referencia, lo que hace posible llamadas por referencia, el compilador mejora mucho el costo de la asignación de memoria, ya que permite a este accesos de solo lectura a úna área de almacenamiento particular.

3.1.2. Estructura de un programa en C++

Todo programa en C consta de una o más funciones, una de las cuales se llama main. El programa comienza en la función main, desde la cual es posible llamar a otras funciones. Cada función estará formada por:

- La cabecera de la función, compuesta por el nombre de la misma y la lista de argumentos (si los hubiese).
- La declaración de las variables a utilizar.
- Y por último, por la secuencia de sentencias a ejecutar.

A continuación vemos un ejemplo de como sería la estructura de un programa escrito en lenguaje C++:

```
declaraciones globales
    inclusión de librerías;
    declaración de constantes;
    declaración de variables globales;
    declaración de estructuras, objetos;
    .....

main()

    declaración variables locales;
    bloque principal;

funcion1()

    declaración de variables locales;
    bloque función;

funcion n()
....
```

3.1.3. Clases, estructuras y objetos

Los tipos definidos por el usuario o tipos abstractos de datos (TAD) empaquetan elementos dato con las operaciones que se realizan sobre esos datos. C++ soporta los TAD con el tipo clase que puede ser implementado con estructuras, uniones y clases.

Concepto de clase

Una clase es un tipo de dato que contiene uno o más elementos dato llamados miembros dato, y cero, una o más funciones que manipulan esos datos (llamadas funciones miembro). Una clase se puede definir con struct, union o class. La sintaxis de una clase es:

```
class nombre_clase
{
    miembro1;
    miembro2;
    ....
    funcion_miembro1();
    funcion_miembro2();
    ....
};
```

Una clase es sintácticamente igual a una estructura, con la única diferencia de que en el tipo class todos los miembros son por defecto privados mientras que en el tipo struct son por defecto públicos.

En C se utiliza el término variable estructura para referirse a una variable de tipo estructura. En C++ no se utiliza el término variable de clase, sino instancia de la clase.

A los miembros que se declaran en la región pública de una clase se puede acceder a través de cualquier objeto de la clase de igual modo que se accede a los miembros de una estructura en C.

```
class alfa
{
    int x; //miembros dato privados
    float y;
    char z;
    public:double k; //miembro dato público
    void fijar(int,float,char);//funciones miembro públicas
    void visualizar();
};

void main()
{
    alba obj; //declaración de un objeto
    obj.fijar(3,2.1,'a'); //invocar a una función miembro
    obj.visualizar(); //invocar a una función miembro
    obj.x=4; //error: no se puede acceder a datos privados
    obj.k=3.2; //válido: k está en la región pública
}
```

La definición de funciones miembro es muy similar a la definición ordinaria de función. Tienen una cabecera y un cuerpo y pueden tener tipos y argumentos. Sin embargo, tienen dos características especiales:

1. Cuando se define una función miembro se utiliza el operador de resolución de ámbito (::) para identificar la clase a la que pertenece la función.

2. Las funciones miembro (métodos) de las clases pueden acceder a las componentes privadas de la clase.

```
Opción 1
class ejemplo
{
    int x,y;
    public: void f()
    {
        printf(....);
    }
};

Opción 2
class ejemplo
{
    int x,y;
    public:void f();
};
void ejemplo::f()
{
    printf(....);
}
```

En la primera opción la función está en línea (inline). Por cada llamada a esta función, el compilador genera (vuelve a copiar) las diferentes instrucciones de la función. En la segunda opción (la más deseable) la función f se llamará con una llamada verdadera de función.

La declaración anterior significa que la función f es miembro de la clase ejemplo. El nombre de la clase a la cual está asociada la función miembro se añade como prefijo al nombre de la función. El operador :: separa el nombre de la clase del nombre de la función. Diferentes clases pueden tener funciones del mismo nombre y la sintaxis indica la clase asociada con la definición de la función.

El término objeto es muy importante y no es más que una variable, que a su vez no es más que una instancia de una clase. Por consiguiente una clase es:

```
class cliente
{
    char nom[20];
    char num;
};
```

y un objeto de esta clase se declara: cliente cli;

Una definición de una clase consta de dos partes: una declaración y una implementación. La declaración lista los miembros de la clase. La implementación o cuerpo define las funciones de la clase.

Una de las características fundamentales de una clase es ocultar tanta información como sea posible. Por consiguiente, es necesario imponer ciertas restricciones en el modo en que se puede manipular una clase y de cómo se pueden utilizar los datos y el código dentro de una clase.

Una clase puede contener partes públicas y partes privadas. Por defecto, todos los miembros definidos en la clase son privados. Para hacer las partes de una clase públicas (esto es, accesibles desde cualquier parte del programa) deben declararse después de la palabra reservada public. Todas las variables o funciones definidas después de public son accesibles a las restantes funciones del programa. Dado que una característica clave de la POO es la ocultación de datos, debe tenerse presente que aunque se pueden tener variables públicas, desde un punto de vista conceptual se debe tratar de limitar o eliminar su uso. En su lugar, deben hacerse todos los datos privados y controlar el acceso a ellos a través de funciones públicas.

```
class articulo
{
    private:float precio;
    char nombre[ ];
    public:void indicar();
};
```

Por defecto u omisión todo lo declarado dentro de una clase es privado y sólo se puede acceder a ello con las funciones miembro declaradas en el interior de la clase o con funciones amigas.

Los miembros que se declaran en la sección protegida de una clase sólo pueden ser accedidos por funciones miembro declaradas dentro de la clase, por funciones amigas o por funciones miembro de clases derivadas.

Concepto de estructura

Las estructuras son tipos de clase. El tipo struct permite agregar componentes de diferentes tipos y con un solo nombre. Las estructuras en C++ se declaran como en C:

```
struct datos
{
    int num;
    char nombre[20];
};
```

Concepto de objeto

En C++ un objeto es un elemento declarado de un tipo clase. Se conoce también como una instancia de una clase.

Los objetos se pueden tratar como cualquier variable C. La principal diferencia es que se puede llamar a cualquiera de las funciones que pertenecen a un objeto, esto es, se puede enviar un mensaje a ella.

```
class rectangulo
{
    int base,altura;
    public: void dimensiones(int,int);
    int area();
};

void rectangulo::dimensiones(int b,int h)
{
    base=b; altura=h;
}
int rectangulo::area()
{
    return base*altura;
}
void main()
{
    rectangulo r; //declarar el objeto
    r.dimensiones(3,5); //definir el tamaño
    printf("area=",r.area());
}
```

Acceso a los miembros de una clase

A los miembros de una clase se accede de igual forma que a los miembros de una estructura. Existen dos métodos para acceder a un miembro de una clase: el operador punto (.) y el operador flecha (->) que actúan de modo similar.

3.1.4. Constructores y destructores

Concepto de constructor

Un constructor es una función especial que sirve para construir o inicializar objetos. En C++ la inicialización de objetos no se puede realizar en el momento en que son declarados; sin embargo, tiene una característica muy importante y es disponer de una función llamada constructor que permite inicializar objetos en el momento en que se crean.

Un constructor es una función que sirve para construir un nuevo objeto y asignar valores a sus miembros dato. Se caracteriza por:

- Tener el mismo nombre de la clase que inicializa.
- Puede definirse inline o fuera de la declaración de la clase.
- No devuelve valores.
- Puede admitir parámetros como cualquier otra función.
- Puede existir más de un constructor, e incluso no existir.

Si no se define ningún constructor de una clase, el compilador generará un constructor por defecto. El constructor por defecto no tiene argumentos y simplemente sitúa ceros en cada byte de las variables instancia de un objeto. Si se definen constructores para una clase, el constructor por defecto no se genera.

Un constructor del objeto se llama cuando se crea el objeto implícitamente, nunca se llama explícitamente a las funciones constructoras. Esto significa que se llama cuando se ejecuta la declaración del objeto. También, para objetos locales, el constructor se llama cada vez que la declaración del objeto se encuentra. En objetos globales, el constructor se llama cuando se arranca el programa.

El constructor por defecto es un constructor que no acepta argumentos. Se llama cuando se define una instancia pero no se especifica un valor inicial. Se pueden declarar en una clase constructores múltiples, mientras tomen parte diferentes tipos o número de argumentos. El compilador es entonces capaz de determinar automáticamente a qué constructor llamar en cada caso, examinando los argumentos. Los argumentos por defecto se pueden especificar en la declaración del constructor. Los miembros dato se inicializarán a esos valores por defecto, si ningún otro se especifica.

C++ ofrece un mecanismo alternativo para pasar valores de parámetros a miembros dato. Este mecanismo consiste en inicializar miembros dato con parámetros.

```
class prueba
{
    tipo1 d1; tipo2 d2; tipo3 d3;
    public:
        prueba (tipo1 p1, tipo2 p2, tipo3 p3):d1(p1),d2(p2),d3(p3)
        { }
};
```

Un constructor que crea un nuevo objeto a partir de uno existente se llama constructor copiador o de copias. El constructor de copias tiene sólo un argumento, una referencia constante a un objeto de la misma clase. Un constructor copiador de una clase complejo es:

```
complejo::complejo(const complejo &fuente)
{
    real=fuente.real;
    imag=fuente.imag;
}
```

Si no se incluye un constructor de copia, el compilador creará un constructor de copia por defecto. Este sistema funciona de un modo perfectamente satisfactorio en la mayoría de los casos, aunque en ocasiones puede producir dificultades. El constructor de copia por defecto inicializa cada elemento de datos del objeto a la izquierda del operador = al valor del elmento dato equivalente del objeto de la derecha del operador =. Cuando no hay punteros implicados, eso funciona bien. Sin embargo, cuando se utilizan punteros, el constructor de copia por defecto inicializará el valor de un elemento puntero de la izquierda del operador = al del elemento equivalente de la derecha del operador; es decir, que los dos punteros apuntan en la misma dirección. Si ésta no es la situación que se desea, hay que escribir un constructor de copia.

Concepto de destructor

Un destructor es una función miembro con igual nombre que la clase, pero precedido por el carácter ~. Una clase sólo tiene una función destructor que, no tiene argumentos y no devuelve ningún tipo. Un destructor realiza la operación opuesta de un constructor, limpiando el almacenamiento asignado a los objetos cuando se crean. C++ permite sólo un destructor por clase. El compilador llama automáticamente a un destructor del objeto cuando el objeto sale fuera del ámbito. Si un destructor no se define en una clase, se creará por defecto un destructor que no hace nada. Normalmente los destructores se declaran public.

3.1.5. Sobrecarga de funciones

En C++ dos o más funciones pueden tener el mismo nombre representando cada una de ellas un código diferente. Esta característica se conoce como sobrecarga de funciones.

Una función sobrecargada es una función que tiene más de una definición.

Estas funciones se diferencian en el número y tipo de argumentos, pero también hay funciones sobrecargadas que devuelven tipos distintos.

Sobrecarga se refiere al uso de un mismo nombre para múltiples significados de un operador o una función.

Dado el énfasis del concepto de clase, el uso principal de la sobrecarga es con las funciones miembro de una clase. Cuando más de una función miembro con igual nombre se declara en una clase, se dice que el nombre de la función está sobrecargado en esa clase. El ámbito del nombre sobrecargado es el de la clase.

La sobrecarga de funciones amigas es similar a la de funciones miembro, con la única diferencia de que es necesaria la palabra reservada friend al igual que en la declaración de cualquier función amiga.

3.1.6. Asignación dinámica de memoria: new y delete

Cuando hablamos de asignación dinámica de memoria, nos referimos al hecho de crear variables anónimas, es decir, reservar espacio en memoria para estas variables en tiempo de ejecución, y también a liberar el espacio reservado para una variable anónima en tiempo de ejecución, en caso de que dicha variable ya no sea necesaria.

La zona de la memoria donde se reservan espacios para asignarlos a variables dinámicas se denomina "heap" o montón. Puesto que esta zona tiene un tamaño limitado, puede llegar a agotarse si únicamente asignamos memoria a variables anónimas y no liberamos memoria cuando ya no sea necesaria, de ahí la necesidad de un mecanismo para liberar memoria.

El núcleo del sistema de asignación dinámica de C++ está compuesto por la función new para la asignación de memoria, y la función delete para la liberación de memoria. Estas funciones trabajan juntas usando la región de memoria libre. Esto es, cada vez que se hace una petición de memoria con new, se asigna una parte de la memoria libre restante. Cada vez que se libera memoria con delete, se devuelve la memoria al sistema.

El subsistema de asignación dinámica de C++ se usa para dar soporte a una gran variedad de estructuras de programación, tales como listas enlazadas que se verán más adelante en este tema. Otra importante aplicación de la asignación dinámica es la asignación dinámica de arrays.

Función de reserva de memoria: new

La función new es simple de usar y será la función que siempre utilizaremos para la reserva de memoria. El prototipo de la función new es de la forma:

```
void *new <nombre_tipo>
```

donde nombre_tipo es el tipo para el cual se quiere reservar memoria.

La función new reservará la cantidad apropiada de memoria para almacenar un dato de dicho tipo. La función new devuelve un puntero de tipo void *, lo que significa que se puede asignar a cualquier tipo de puntero. Convierte automáticamente el puntero devuelto al tipo adecuado, por lo que el programador no tiene que preocuparse de hacer la conversión de forma explícita.

La función new también se encarga de averiguar cual es el tamaño en bytes del tipo de datos para el cual se desea reservar memoria (recordemos que la cardinalidad de un tipo nos permite saber la memoria necesaria para su almacenamiento). Observar en el ejemplo siguiente que se indica el que se quiere reservar memoria para un dato de tipo char y no cual es el tamaño en bytes que se quiere reservar.

```
char *p;
p = new char; // reserva espacio para un carácter
```

Tras una llamada con éxito, new devuelve un puntero al primer byte de la región de memoria dispuesta del montón. Si no hay suficiente memoria libre para satisfacer la petición, se produce un fallo de asignación y new devuelve un NULL.

A tener en cuenta, que como el montón no es infinito, siempre que se reserve memoria con new debe comprobarse, antes de usar el puntero, el valor devuelto para asegurarse que no es nulo, vemos esto en el siguiente ejemplo:

```
char *p;
p = new char; // reserva espacio para un carácter
if (p == NULL)
{
    // No se ha podido reservar la memoria deseada
    // Tratar error
}
```

Función de liberación de memoria dinámica: delete

La función delete es la complementaria de new. Una vez que la memoria ha sido liberada, puede ser reutilizada en una posterior llamada a new. El prototipo de la función delete es:

```
void delete <variable_puntero>
```

donde p es un puntero a memoria que ha sido previamente asignado con new. Vemos un ejemplo:

```
char *p;
p = new char; // reserva espacio para un carácter
delete p;//libera la memoria previamente reservada
```

A tener en cuenta, no llamar nunca a la función delete con un argumento no válido; se dañará el sistema de asignación y además que la función delete p no asigna el valor nulo al puntero p después de liberar la memoria a la que apuntaba.

3.1.7. Biblioteca de funciones

Una biblioteca de funciones no es más que un conjunto de archivos con una aplicación específica.

Archivos de cabecera

Normalmente, las definiciones de las clases están incluidas en ficheros cabecera, existiendo un archivo de cabecera por cada clase que definamos. En este archivo incluiremos la definición de la clase, cualquier constante, enumeración, etc..., que se pueda usar con un objeto de la clase y los métodos inline, ya que estos necesitan estar definidos antes de usarse.

Creación de bibliotecas

En cada uno de los archivos que necesiten hacer uso de una clase será necesario incluir los archivos cabecera en los cuales están definidas las clases que van a ser usadas. Sin embargo, normalmente estas bibliotecas son compiladas obteniendo ficheros.OBJ que serán enlazados junto a las aplicaciones.

En el caso de que tengamos varios ficheros se puede, o bien generar un código objeto por cada uno de los archivos fuente, o bien varios archivos. OBJ, o bien unirlos todos en una biblioteca .LIB.

En definitiva, el proceso de creación y utilización de una biblioteca de funciones podría simplificarse a:

1. Definición de las clases en los archivos cabeceras.
2. Codificación de los métodos en uno o varios archivos fuentes, incluyendo archivos de cabecera.

3. Compilación de cada uno de los archivos fuente, con la consiguiente generación del código objeto.

4. Unión de los códigos objetos en los distintos archivos en una sóla biblioteca con cada objeto.

5. Incluir los archivos de cabecera en la aplicación que use la biblioteca y enlazarla con ella.

Con un enlace seguro podemos dar la característica de sobrecarga a funciones implementadas en C que queremos usar en C++. Para ello, ponemos en la declaración del prototipo extern "C". Esto también se puede hacer con bloques de código y con archivos cabecera.

El uso de bibliotecas incrementa la transportabilidad de los programas. Las funciones se dividen en grupos, todas las funciones que pertenecen al mismo grupo estan declaradas en el archivo de cabecera (aquel que dice Nombre_Cabecera.h), la letra "h"significa header en inglés y es lo que hemos llamado cabecera y definidas en el archivo Nombre_Cabecera.cpp.

Para incluir alguna función perteneciente a estas cabeceras debemos escribir líneas de código como se muestra en el siguiente ejemplo:

```
#include <Nombre_Cabecera>
```

3.1.8. Ficheros en C++

Los ficheros son estructuras de datos almacenadas en memoria secundaria. Para utilizar la información en memoria principal se emplea fundamentalmente la instrucción de asignación; sin embargo, para guardar o recuperar información de un fichero es necesario realizar una serie de operaciones que se describirán en este apartado. El formato de declaración de un fichero es el siguiente:

```
FILE * nom_var_fich
```

En otros lenguajes la declaración del fichero determina el tipo de datos que se van a almacenar en él.

En C la filosofía es distinta, todos los ficheros almacenan bytes y es cuando se realiza la apertura y la escritura cuando se decide cómo y qué se almacena en el mismo; durante la declaración del fichero no se hace ninguna distinción sobre el tipo del mismo.

En la operación de apertura se puede decidir si el fichero va a ser de texto o binario, los primeros sirven para almacenar carácteres y los segundos para almacenar cualquier tipo de dato.

Si deseamos leer un fichero como el autoexec.bat utilizaremos un fichero de texto, si queremos leer y escribir registros (struct) usaremos un fichero binario.

Apertura y cierre de ficheros

Hasta ahora, para obtener y almacenar datos de una estructura de datos bastaba con realizar asignaciones a la misma. Para utilizar los ficheros el procedimiento es distinto.

Antes de usar un fichero es necesario realizar una operación de apertura del mismo; posteriormente, si se desea almacenar datos en él hay que realizar una operación de escritura y si se quiere obtener datos de él es necesario hacer una operación de lectura. Cuando ya no se quiera utilizar el fichero se realiza una operación de cierre del mismo para liberar parte de la memoria principal que pueda estar ocupando (aunque el fichero en sí está almacenado en memoria secundaria, mientras está abierto ocupa también memoria principal).

La instrucción más habitual para abrir un fichero es:

```
FILE * fichero;
fichero = fopen ( nombre-fichero, modo);
```

La función fopen devuelve un puntero a un fichero que se asigna a una variable de tipo fichero. Si existe algún tipo de error al realizar la operación, por ejemplo, porque se desee abrir para leerlo y éste no exista, devuelve el valor NULL.

El nombre-fichero será una cadena de carácteres que contenga el nombre (y en su caso la ruta de acceso) del fichero tal y como aparece para el sistema operativo.

El modo es una cadena de carácteres que indica el tipo del fichero (texto o binario) y el uso que se va ha hacer de él como la lectura, escritura, añadir datos al final, etc. Los modos disponibles son:

- r: abre un fichero para lectura. Si el fichero no existe devuelve error.
- w: abre un fichero para escritura. Si el fichero no existe se crea, si el fichero existe se destruye y se crea uno nuevo.
- a: abre un fichero para añadir datos al final del mismo. Si no existe se crea.
- +: símbolo utilizado para abrir el fichero para lectura y escritura.
- b: el fichero es de tipo binario.
- t: el fichero es de tipo texto. Si no se pone ni b ni t el fichero es de texto. Los modos anteriores se combinan para conseguir abrir el fichero en el modo adecuado.

Por ejemplo, para abrir un fichero binario ya existente para lectura y escritura el modo será "rb+"; si el fichero no existe, o aun existiendo se desea crear, el modo será "wb+". Si deseamos añadir datos al final de un fichero de texto bastará con poner "a", etc.

La forma habitual de utilizar la instrucción fopen es dentro de una sentencia condicional que permita conocer si se ha producido o no error en la apertura, por ejemplo:

```
FILE * fich;
if ((fich = fopen("nomfich.dat", "r")) == NULL)
{
    /* control del error de apertura * /
    printf ("Error en la apertura. Es posible que el fichero no
    exista");
}
```

El resultado de fopen se almacena en la variable fich y después se compara fich con NULL para saber si se ha producido algún error. Toda la operación se puede realizar en la misma instrucción, tal y como aparece en el ejemplo.

Cuando se termine el tratamiento del fichero hay que cerrarlo; si la apertura se hizo con fopen el cierre se hará con fclose (fich).

Para utilizar las instrucciones de manejo de ficheros que vemos en esta sección es necesario incluir la librería $< stdio.h >$ en la cabecera del programa.

Lectura y escritura en ficheros

Para almacenar datos en un fichero es necesario realizar una operación de escritura, de igual forma que para obtener datos hay que efectuar una operación de lectura. En C existen muchas y variadas operaciones para leer y escribir en un fichero; entre ellas tenemos: fread - fwrite, fgetc - fputc, fgets - fputs, fscanf - fprintf . Es aconsejable utilizarlas por parejas; es decir, si se escribe con fwrite se debe leer con fread. Nosotros solo nos basaremos en las instrucciones de lectura y escritura de bloques, que se pueden ver a continuación.

Lectura y escritura de bloques (fread – fwrite)

Para leer y escribir en ficheros que no sean de texto las operaciones que se deben utilizar son fread y fwrite. El formato de escritura en bloque es el siguiente:

```
fwrite (direcc_dato, tamaño_dato, numero_datos, punt_fichero);
```

Escribe tantos datos como indique numero de datos en el fichero, tomando los datos a partir de la dirección del dato.

Los datos tienen que tener tantos bytes como especifique tamaño. La función fwrite devuelve el número de elementos escritos, este valor debe coincidir con numero de datos.

Para calcular el tamaño en bytes de un dato o un tipo de dato se suele utilizar la función sizeof (dato) o sizeof (tipo-de-dato);

Por ejemplo:

```
int i, v[3];
    -> sizeof (i) daría lo mismo que sizeof (int)
    -> sizeof (v) daría 3 veces el resultado de sizeof (V[1])
```

La sentencia de lectura de bloque es la siguiente:

```
fread (direcc_dato, tamaño_dato,numero_datos,punt_fichero);
```

Lee tantos datos como indique numero de datos del fichero, colocando los datos leídos a partir de la dirección del dato. Los datos tienen que tener tantos bytes como especifique tamaño del dato. La función fread devuelve el número de elementos leídos, y el valor devuelto debe coincidir con numero de datos. Vemos una serie de ejemplos:

```
f = fopen ("datos.dat", "rb");
elem-escritos = fread(&v[2], sizeof(int), 3, f);
fread (v, sizeof(int), 1, f);
fread (&V[0], sizeof(int), 1, f);
fread (&num, sizeof(int), 1, f);
fread (&num, sizeof(num), 1, f);
```

3.1.9. Archivos BMP

El formato BMP (Windows BitMaP) es probablemente el formato de fichero para imágenes más simple que existe. Aunque teóricamente permite compresión, en la práctica nunca se usa, y consiste simplemente en una cabecera y a continuación los valores de cada pixel, comenzando por la línea de más abajo y terminando por la superior, pixel a pixel de izquierda a derecha. Parece ser el formato preferido por Bill Gates.

Se caracteriza por su enorme sencillez, pero por contra tiene la gran desventaja del tamaño de los ficheros, que es enorme.

El formato BMP, cuya lectura y escritura se realiza en binario, consta de los siguientes elementos:

- Cabecera del archivo: que nos muestra información acerca de si se trata de un mapa de bits, el tamaño del archivo, y en qué punto del archivo comienza la imagen en sí.
- Cabecera de información: que nos dice cosas como las dimensiones horizontales y verticales de la imagen, su profundidad de color, etc.
- Paleta: la paleta de colores, que en nuestro caso no nos afecta ya que empleamos un formato de 24 bits por pixel.
- Mapa de bits: que es la imagen en sí, por así decirlo.

Cabecera del archivo

La cabecera de un archivo BMP consta de los siguientes campos:

- Tipo de formato (2 bytes): debe ser "BM", por lo que en la práctica se lee como dos bytes por separado.
- Tamaño del archivo (4 bytes): indica el tamaño en bytes del archivo, dividiendo entre 1024 se obtiene el mismo en KB.
- Reservado (2 bytes): debe ser cero.
- Reservado (2 bytes): debe ser cero.
- Offset (4 bytes): indica la posición en bytes en la que empiezan los datos en sí.

Aunque muchos bytes contengan información que no nos interesa, debemos leerlos de todos modos para poder seguir avanzando en el archivo y seguir leyendo lo que venga después.

Los primeros dos bytes del archivo siempre serán "BM", para leerlo siempre leeremos un byte y después otro, y los transformaremos por separado a ascii. Solo los usaremos para comprobar que se trate realmente de un archivo BMP, aunque en realidad podemos prescindir de esta comprobación.

El Tamaño del archivo es el tamaño en bytes que el archivo ocupará en nuestro disco.

Los siguientes 4 bytes son dos valores de 2 bytes cada uno que deben tener el valor 0, es algo propio del formato que NUNCA cambia.

Y los siguientes 4 bytes nos indican el Offset hasta que comienzan los datos, es decir, cuántos bytes hay desde el inicio del archivo hasta la información sobre los pixels, colores, etc. Puede sernos útil según de qué modo trabajemos, pero nosotros no lo necesitaremos, ya que el offset siempre será 54.

Cabecera de información

La cabecera de información de un archivo BMP consta de los siguientes campos:

- Tamaño cabecera (4 bytes): indica el tamaño en bytes de la cabecera de información. Siempre 40.
- Ancho (4 bytes): indica el ancho en pixels de la imagen.
- Alto (4 bytes): indica la altura en pixels de la imagen.
- Planos (2 bytes): indica los planos del dispositivo de salida. Obviamente no nos importa.
- Bits por pixel: indica los bits por pixel, para nuestro caso siempre 24.
- Compresión (4 bytes): indica el tipo de compresión, en la práctica nunca se encuentran BMP's comprimidos. Por tanto vale cero.
- Tamaño de imagen (4 bytes): indica el tamaño de la estructura de datos.
- Pixeles por metro h (4 bytes): indica el número de pixeles por metro para dispositivos de salida, medida horizontal. No nos importa.
- Pixeles por metro v (4 bytes): indica el número de pixeles por metro para dispositivos de salida, medida vertical. No nos importa.
- Colores usados (4 bytes): indica la cantidad de colores usados, puede valer cero para que lo calcule el programa.
- Colores importantes (4 bytes) indica la cantidad de colores "importantes" ó cero si todos lo son.

Tanto para la cabecera del archivo como para la cabecera de información, para leer la información que contengan ambas, tendremos que definir una estructura.

Paleta de colores

A continuación vendría la paleta de colores, en caso de que la haya, en nuestro caso al trabajar con color verdadero, es decir, pixeles de 24 bits, no existirá dicha paleta.

Mapa de bits

El mapa de bits consiste en una secuencia de bytes, que almacenarán los datos de la imagen.

El orden en que están almacenados los pixeles es: desde abajo hacia arriba, y desde la izquierda hacia la derecha.

Un detalle importante es que la cantidad de bytes que ocupa cada línea en sentido horizontal es múltiplo de cuatro (o sea: 32 bits). Por ejemplo si en un bitmap de color verdadero (24 bits) el ancho del gráfico es de 169 pixeles, en realidad habrá 508 bytes, es decir, el último byte va de relleno, y debe ser descartado.

3.2. DJGPP

3.2.1. Breve reseña histórica

Para entender la situación que originó toda esta corriente debemos situarnos en la realidad que se vivía hace unos cuantos años atrás allá por los ‘70. En esos años las computadoras venían acompañadas de su software y el negocio de los fabricantes era vender computadoras y no sus sistemas operativos. En esa época no era difícil encontrar software gratis acompañado de sus fuentes. Esta política generó un ambiente de cooperación entre los usuarios de computadoras de la época que podían entonces intercambiar información y rutinas libremente. Se debe tener en cuenta que los usuarios eran universades, centros de desarrollo, grandes empresas y entidades gubernamentales.

Hacia los años ‘80 la situación fue cambiando bastante debido a que los fabricantes comenzaron a ver en el software un gran negocio, de hecho hoy día es más redituable que el hardware. La situación originada fue la siguiente: los departamentos de las universidades y centros de desarrollo utilizaban casi en forma exclusiva workstations y computadoras aún más grandes corriendo sistemas operativos UNIX. Cada vez que se decidía un cambio de modelo de computadora se debía no solo adquirir la computadora, sino también el sistema operativo y las herramientas de desarrollo para el mismo. Los precios eran muy altos debido al impacto del costo del sofware. A todo esto debe agregarse que los fabricantes ya no proveían los fuentes y que el intercambio de información que se vió en los ‘70 quedó bloqueado.

En 1983 Richard Stallman, que desarrollaba tareas en el departamento de Inteligencia Artificial del MIT, desidió iniciar un titánico proyecto con el objetivo de restaurar aquel ambiente de los años ‘70. Como toda computadora necesita un sistema operativo para funcionar el proyecto comenzó por allí.

Vale aclarar que Stallman tiene una forma de pensar un tanto particular, según él el software no debe venderse como tal, lo que debe venderse es el servicio que lo acompaña. De esta manera las compañías deberían “liberar” el software y cobrar por los servicios asociados, por ejemplo: asistencia al usuario, medio de transporte del mismo (discos, CDs, etc.), manuales, modificaciones particulares para un usuario, etc.

El concepto de “liberar” el software va incluso un poco más allá. Para Stallman el sofware no sólo debe ser gratuito sino también “libre”, es decir, que el que adquiere un software debe poseer la libertad de modificarlo para acomodarlo a sus necesidades, esto implica en forma directa que el software debe ir acompañado de todo su código fuente. Esto es aún más “raro” si se lo compara con la norma actual, las empresas podrían estar de acuerdo en volver gratis su software y solo cobrar por los servicios pero dificilmente aceptarían entregar sus fuentes. Nace así la llamada FSF (Free Software Foundation).

Con esta filosofía Stallman inicia la creación de un nuevo sistema operativo llamado GNU, que significa “GNU isn’t Unix” y como se puede ver es una definición circular. Este sistema operativo no se ha popularizado y dificilmente lo haga debido a razones que se exponen más adelante. A pesar de ello existe actualmente un kernel llamado Hurd que es el kernel de GNU.

GNU sería un sistema operativo con un kernel (núcleo) diferente al Unix, pero con herramientas de trabajo compatibles y a su vez compatible con los estándares POSIX. Esto facilitaría su uso y la migración desde Unix a GNU. Debido a que en el entorno Unix el lenguaje de programación más usado es el C, y que el mismo es adecuado para la programación de bajo nivel, tal es el caso de un sistema operativo, es que el comienzo del proyecto GNU se da con la creación de un compilador de C llamado GCC (Gnu C Compiler).

Consecuencias y frutos

Como ya se ha mencionado anteriormente GNU, aunque no es usado como sistema operativo, el proyecto dio sus frutos. Gracias al esfuerzo de decenas, sino cientos, de voluntarios que trabajaron, y continuan trabajando en el proyecto, es que hoy en día todas las herramientas que deben acompañar a un sistema operativo del estilo de Unix han sido desarrolladas por GNU/FSF. Esto se logró ya a principios de los '90.

A modo de ejemplo se pueden nombrar algunos de los paquetes de software. Hay que destacar que los mismos cumplen con las consignas de la FSF y por lo tanto se pueden obtener en forma gratuita y todos sus fuentes están disponibles: gcc (Compilador de C/C++), gas (GNU Assembler), ld,ar,... (Utilidades que complementan al compilador), flex (Analizador lexicográfico), bison (Parser o Generador de compiladores), ls,df,... (Utilidades de manejo de archivos), cat (Utilidades de manejo de texto), make (Accesorio para automatizar el compilado de proyectos), sed (Utilidad para modificación de textos usando scripts), g77 (Compilador Fortran),...

El rincón olvidado: DOS

El sistema operativo DOS es el más usado en las PC's y aún cuando la misma corra Windows 95/98 el mismo se encuentra sobre un DOS y brinda un alto grado de compatibilidad con su versión anterior. Las herramientas de desarrollo para DOS cuestan muy caras, es decir, que alguien que desee programar en C/C++ deberá desembolsar una alta cifra para lograrlo o caer en la ilegalidad.

En el año 1986 DJ Delorie y un grupo de colaboradores decidieron adaptar las herramientas de GNU a DOS, el resultado se conoce como DJGPP (DJ's Gnu Programming Platform). Su objetivo es llevar al mundo de DOS las ventajas del "free software".

3.2.2. Detalles técnicos del DJGPP

La tarea de adaptar las herramientas GNU a DOS no fue algo trivial. En primer lugar todas estas herramientas han sido escritas para correr sobre sistema operativo tipo Unix y por lo tanto esperan que muchas de las rutinas de librería de C reaccionen como lo harían bajo Unix.

A esto debe sumarse que la idea de DJ Delorie no es totalmente coincidente con la de la FSF, es decir que DJ Delorie tuvo en mente crear un paquete de desarrollo que no estuviera atado a todas las reglas de juego de la FSF. La más importante de ellas es que si uno crea un programa que utiliza las librerías de rutinas creadas por la FSF, que estan protegidas por la licencia LGPL, dicho programa debe cumplir los requisitos de los programas de la FSF, es decir, que el autor debe hacer disponibles los fuentes del programa a sus clientes. Este último, es un detalle técnico referente a DOS, las librerías de Linux pueden utilizarse sin este requisito ya que son dinámicamente enlazadas.

Por estas dos razones, técnicas y de propiedad intelectual, es que DJ Delorie se vio forzado a crear una librería de rutinas de C que cumpliera con los siguientes requisitos:

1. Fuera altamente compatible con la libería standard de Unix.
2. No estuviera cubierta por copyrights que limitarán el uso de la misma.

Esta tarea no fue nada fácil ya que hubo que crear más de 650 rutinas, que por supuesto no fueron creadas por una sola persona y parte de ellas provienen de la librería de la Universidad de Berkeley que no poseen restricciones, aunque actualmente casi no queda ni una de ellas ya que fueron gradualmente reemplazadas.

Otro punto de suma importancia es que el GCC es un compilador diseñado para microprocesadores de 32 bits y al momento en que DJ Delorie comenzó su trabajo el GCC podía generar programas para microprocesadores 386 o superiores de Intel. El problema es que DOS es un sistema operativo de 16 bits, debido a esto es que el djgpp necesitó un "DOS Extender", esto es que necesitó un programa que permitiera correr programas de 32 bits bajo DOS. Hoy en día el djgpp hace uso de un standard denominado DPMI (DOS Protected Mode Interface) que permite ejecutar programas de 32 bits bajo DOS. Debido a que DOS no posee servicios de DPMI (Win3.1, Win95 y OS/2 si) es que también se tuvo que desarrollar un servidor de DPMI (CWSDPMI por Charles Sandsman).

En la actualidad el djgpp es un entorno completo de programación el cual posee casi todas las herramientas creadas para GNU.

Como resumen, se puede decir que el DJGPP es un puerto de las herramientas del GNU al DOS hecho por DJ Delorie (y amigos), que incluye un compilador de gran alcance del GCC y diseñado para microprocesadores de 32 bits.

3.2.3. RHIDE

Rhide es un entorno de desarrollo integrado (IDE) creado por Robert Höhne. Muy al estilo de los antiguos entornos de Turbo C, Turbo Pascal o Borland C 3.1. No es un compilador, sino que hace uso del compilador al tiempo que ofrece al desarrollador un entorno amigable de trabajo. Algunas de sus características más importantes son:

- Permite compilar proyectos sin tener que usar la línea de comando.
- "Syntax Highlight": muy útil para descubrir errores de sintaxis.
- Lleva un debugger integrado en el entorno Rhide.
- Es configurable en colores, modos de pantalla, compiladores, opciones de línea de comando, ...

3.3. Ensamblador

3.3.1. Introducción

Todo procesador, grande o pequeño, desde el de una calculadora hasta el de un supercomputador, ya sea de propósito general o específico, posee un lenguaje único que es capaz de reconocer y ejecutar. Por razones que resultan obvias, este lenguaje ha sido denominado Lenguaje de Máquina y más que ser propio de un computador pertenece a su microprocesador. El lenguaje de máquina está compuesto por una serie de instrucciones, que son las únicas que pueden ser reconocidas y ejecutadas por el microprocesador. Este lenguaje es un conjunto de números que representan las operaciones que realiza el microprocesador a través de su circuitería interna. Estas instrucciones, por decirlo así, están grabadas o "alambradas" en el hardware y no pueden ser cambiadas. El nivel más bajo al que podemos aspirar a llegar en el control de un microprocesador es precisamente el del lenguaje de máquina.

Ahora bien, siendo el lenguaje de máquina un conjunto de números, ¿cómo es capaz el microprocesador de saber cuándo un número representa una instrucción y cuándo un dato? El secreto de esto reside en la dirección de inicio de un programa y en el estado del microprocesador. La dirección de inicio nos indica en qué localidad de memoria comienza un programa, y en consecuencia que datos deberemos considerar como instrucciones.

El estado del microprocesador nos permite saber cuándo éste espera una instrucción y cuándo éste espera un dato.

Obviamente, el lenguaje de máquina de un microprocesador no puede ser ejecutado por otro microprocesador de arquitectura distinta, a menos que haya cierto tipo de compatibilidad prevista. Por ejemplo, un 80486 es capaz de ejecutar lenguaje de máquina propio y soporta el código generado para microprocesadores anteriores de la misma serie (desde un 8086 hasta un 80386). Por otra parte, un PowerPC es capaz de ejecutar instrucciones de los microprocesadores Motorola 68xxx y de los Intel 80xx/80x86. En ambos casos, el diseño de los microprocesadores se hizo tratando de mantener cierto nivel de compatibilidad con los desarrollados anteriormente. En el segundo caso, este nivel de compatibilidad se extendió a los de otra marca. Sin embargo, un 8088 no puede ejecutar código de un 80186 o superiores, ya que los procesadores más avanzados poseen juegos de instrucciones y registros nuevos no contenidos por un 8088. Un caso similar es la serie 68xxx, pero de ninguna manera podemos esperar que un Intel ejecute código de un Motorola y viceversa. Y esto no tiene nada que ver con la compañía, ya que Intel desarrolla otros tipos de microprocesadores como el 80860 y el iWARP, los cuales no pueden compartir código ni entre ellos ni entre los 80xx/80xxx.

Ahora bien, mientras que con el lenguaje de máquina, nosotros obtenemos un control total del microprocesador, la programación en este lenguaje resulta muy difícil y fácil para cometer errores. No tanto por el hecho de que las instrucciones son sólo números, sino porque se debe calcular y trabajar con las direcciones de memoria de los datos, los saltos y las direcciones de llamadas a subrutinas, además de que para poder hacer ejecutable un programa, se deben enlazar las rutinas de run-time y servicios del sistema operativo. Este proceso es al que se le denomina ensamblado de código. Para facilitar la elaboración de programas a este nivel, se desarrollaron los Ensambladores y el Lenguaje Ensamblador.

Existe una correspondencia 1 a 1 entre las instrucciones del lenguaje de máquina y las del lenguaje ensamblador. Cada uno de los valores numéricos del lenguaje de máquina tiene una representación simbólica de 3 a 5 letras como instrucción del lenguaje ensamblador. Adicionalmente, este lenguaje proporciona un conjunto de pseudo-operaciones (también conocidas como directivas del ensamblador) que sirven para definir datos, rutinas y todo tipo de información para que el programa ejecutable sea creado de determinada forma y en determinado lugar.

El lenguaje ensamblador fue diseñado para hacer más fácil la programación de bajo nivel, ésta resulta todavía complicada y muy laboriosa. Por tal motivo se desarrollaron los lenguajes de alto nivel, para facilitar la programación de los computadores, minimizando la cantidad de instrucciones a especificar. Sin embargo, esto no quiere decir que el microprocesador ejecute dichos lenguajes. Cada una de las instrucciones de un lenguaje de alto nivel o de un nivel intermedio, equivalen a varias de lenguaje máquina o lenguaje ensamblador.

De esta manera tenemos nuestro compilador, C en nuestro caso y ya mencionado en el apartado 3.1 que se encargan de unir las rutinas para formar el código objeto que, después de enlazar las rutinas de run-time y llamadas a otros programas y servicios del sistema operativo, se transformará en el programa ejecutable.

Para crear un programa ejecutable a partir de un código objeto se requiere que se resuelvan las llamadas a otros programas y a los servicios del sistema operativo, y agregar las rutinas o información de run-time para que el programa pueda ser cargado a memoria y ejecutado. Este proceso es lo que se conoce como Link o proceso de liga, y se realiza a través de un ligador o Linker que toma de entrada el código objeto y produce de salida el código ejecutable.

Las rutinas de run-time son necesarias, puesto que el sistema operativo requiere tener control sobre el programa en cualquier momento, además de que la asignación de recursos y su acceso deben hacerse solamente a través del sistema operativo. Para los computadores personales, esto no es tan complejo como para otros computadores y sistemas operativos, pero es requerido.

3.3.2. Ventajas e inconvenientes

El lenguaje ensamblador como cualquier lenguaje de programación presenta una serie de ventajas y una serie de desventajas que se presentan a continuación:

Ventajas

- Velocidad: como trabaja directamente con el microprocesador al ejecutar un programa, pues como este lenguaje es el más cercano a la máquina la computadora lo procesa mas rápido.

- Eficiencia de tamaño: un programa en ensamblador no ocupa mucho espacio en memoria porque no tiene que cargar librerías y demás como son los lenguajes de alto nivel.

- Flexibilidad: es flexible porque todo lo que puede hacerse con una máquina, puede hacerse en el lenguaje ensamblador de esta máquina; los lenguajes de alto nivel tienen en una u otra forma limitaciones para explotar al máximo los recursos de la máquina. O sea que en lenguaje ensamblador se pueden hacer tareas especificas que en un lenguaje de alto nivel no se pueden llevar acabo porque tienen ciertas limitantes que no se lo permiten.

Desventajas

- Tiempo de programación: como es un lenguaje de bajo nivel requiere más instrucciones para realizar el mismo proceso, en comparación con un lenguaje de alto nivel. Por otro lado, requiere de más cuidado por parte del programador, pues es propenso a que los errores de lógica se reflejen más fuertemente en la ejecución.

- Programas fuente grandes: por las mismas razones que aumenta el tiempo, crecen los programas fuentes; simplemente requerimos más instrucciones primitivas para describir procesos equivalentes. Esto es una desventaja porque dificulta el mantenimiento de los programas, y nuevamente reduce la productividad de los programadores.

- Peligro de afectar recursos inesperadamente: todo error que podamos cometer, o todo riesgo que podamos tener, podemos afectar los recursos de la máquina, programar en este lenguaje lo más común que pueda pasar es que la máquina se bloquee o se reinicialize. Porque con este lenguaje es perfectamente posible (y sencillo) realizar secuencias de instrucciones inválidas, que normalmente no aparecen al usar un lenguaje de alto nivel.

- Falta de portabilidad: para cada máquina existe un lenguaje ensamblador; por ello, evidentemente no es una selección apropiada de lenguaje cuando deseamos codificar en una máquina y luego llevar los programas a otros sistemas operativos o modelos de computadoras.

3.3.3. Arquitectura Intel 80x86

Descripción y Características

El procesador 80x86 fue el primer microprocesador de 16 bits que la compañia Intel fabricó a principios del año 1978. Los objetivos de la arquitectura de dicho procesador fueron los de ampliar la capacidad de versiones anteriores, como el intel 80x80 de forma simétrica, añadiendo una potencia de proceso no disponible en los micros de 8 bits. Algunas de estas características introducidas son: aritmética en 16 bits, multiplicación y división con o sin signo, manipulación de cadena de carácteres y operación sobre bits. También se realizó un mecanismo de software para la construcción de códigos reentrante y reubicable. Su estructura interna consta de dos unidades claramente diferenciadas denominadas EU (Unidad de Ejecución) y BIU (interfaces del Bus).

La EU ejecuta las operaciones requeridas por la instrucciones sobre una unidad aritmético-lógica (ALU) de 16 bits. No tiene conexión con el exterior y solamente se comunica con la BIU que es la parte que realiza todas las operaciones en el bus solicitadas por la EU. Un mecanismo, tal vez único dentro de los microprocesadores aunque muy empleado dentro de los mínimos y grandes ordenadores, es el denominado de búsqueda anticipada de instrucciones (prefetch). En el procesador de Intel 8086 existe una estructura FIFO en RAM de 6 octetos de capacidad que es llenada por la BIU con los contenidos de las intrusiones siguientes a la que la EU esta ejecutando en ese momento.

Los registros que presenta esta procesador, en su primera versión, se enumeran a continuación:

- Cuatro registros de 16 bits, denominados AX,BX,CX y DX, que pueden ser direccionados de 8 registros de 8 bits, denominados AH,AL,....DL. los siete últimos son equivalentes a los registros A, H, L, B, C, D y E, del microprocesador Intel 80x86. El registro AX sirve fundamentalmente como acumulador y como registro de trasferencia en las intrusiones E/S. El registro BX puede usarse como acumulador y como registro base para calcular la direcciones de los datos de memoria. El registro CX puede usarse como acumulador y se utiliza como contador para las intrusiones interactivas. El registro DX puede usarse como acumulador y se emplea como puntero de datos en ciertas intrusiones especificas de E/S.

- Cuatro registros de puntero de segmento denominado CS, DS, SS y ES. Dicho puntero definen cuatro segmentos de 64 K bytes cada uno. Cualquier dirección de memoria se forma, sumando al puntero del segmento una dirección efectiva calculada por diversos procedimientos. El registro CS, (CODE SEGMENT) se usa junto con el PC para calcular las direcciones de las intrucciones del programa; el registro SS (STACK SEGMENT) se emplea junto con el SP (STACK POINTER) para calcular la dirección de las intrucciones que manejan la pila tales como PUSH, POP, CALL y RETURN; por su parte, el registro DS (DATA SEGMENT) se usa en instrucciones que manejan datos de memoria y el registro ES (EXTRA SEGMENT) se utiliza en instrucciones que manejan cadena de carácteres.

- Cuatro registros que contiene direcciones de desplazamiento dentro de los segmentos denominadas SP, BP, SI, DI. El registro SP puntero de la pila los registros SI (INDEX SEGMENT) y DI (Índice Destino) contienen desplazamientos de los punteros de segmento DS y ES en las intrusiones que manejan cadena de caracteres. El registro BP (BASE POINTER) es el puntero base.

- Un registro contador de programas, PC.

- Un registro de estado S, de 16 bits con la siguiente asignación: Bit b_o(C) es el acarreo, Bit b_2(P) es el de paridad, Bit b_4(A) es el de acarreo auxiliar, Bit b_6(Z) el de cero, Bit b_7(S) el de signo, Bit b_8 (T) el de Trap, Bit b_9(I) que sirve para controlar el bloqueo de las intrusiones, Bit b_{10}(D) que determinan si se han de autoincrementar o autodecrementar los punteros SI y DI en las intrusiones que manejan cadenas de carácteres y un Bit b_{11}(O) que especifica el desbordamiento (Overflow).

El 8086 representa la arquitectura base para todos los microprocesadores de 16 bits de Intel: 8088, 8086, 80188, 80186 y 80286. Aunque han aparecido nuevas características a medida que estos microprocesadores han ido evolucionando, como veremos posteriormente, todos los procesadores Intel, usados en la actualidad en los PC's y compatibles son miembros de la familia 8086. El conjunto de instrucciones, registros y otras características son similares, a excepción de algunos detalles, todo la familia 80x86 en adelante poseen dos características en común como son:

- Arquitectura segmentada: esto significa que la memoria es divida en segmentos con un tamaño máximo de 64k (información importante para el direccionamiento de la memoria en la futura programación segmentada en el lenguaje ensamblador).

- Compatibilidad de Las intrucciones y registros de las anteriores versiones son soportados por las nuevas versiones, y estas versiones son soportadas por versiones anteriores.

Evolución histórica

Vamos a ver como ha ido evolucionando la arquitectura 80x86 desde el primer procesador que introdujo Intel en el mercado hasta nuestros días.

8086

Este procesador aparece en 1978. Esta CPU tenía una arquitectura de 16 bits (aunque podía trabajar con datos de 8 bits gracias a sus registros de datos "partidos") y podía direccionar hasta 1 MB de memoria gracias a sus 20 bits en el bus de direcciones.

Una versióon "económica" del 8086, el 8088 (su bus de datos teía 8 líneas en lugar de 16), fue la elegida por IBM para lanzar el PC (Personal Computer), un ordenador destinado a su uso en hogares. El PC fue un gran éxito de ventas, sobre todo debido a que IBM publicó su arquitectura, permitiendo a otros fabricantes desarrollar interfaces, programas e incluso ordenadores compatibles, también llamados "clónicos". A partir de este momento el éxito de la arquitectura IA-32 queda ligado al PC.

El 8086 con sus registros de 16 bits y el bus de direcciones de 20 bits introdujo el concepto de acceso a memoria segmentado. Cada segmento podía direccionar hasta 64 KB de memoria, como existían 4 registros de segmento podían llegar a direccionarse 256 KB sin modificar el contenido de los registros de segmento.

80286

Después de una versión mejorada del 8086, conocida como 801862, y que no resultó muy popular, se llegó en 1982 al 80286. Esta CPU supuso una revolución, seguía siendo una arquitectura de 16 bits, pero su bus de direcciones tenía 24 líneas, lo que le permitía direccionar hasta 16 MB. Con ella Intel pretendía cumplir un doble objetivo, por un lado proporcionar el soporte de protección necesario para los sistemas operativos multitarea y por otro lado mantener la compatibilidad con el 8086, de forma que fuera posible ejecutar la enorme cantidad de programas existentes para esta CPU.

Esto dio lugar a que la CPU se comportará como 2 CPU's, cada una con un modo de funcionamiento diferenciado. Así, al nuevo modo de funcionamiento en el que la CPU aportaba protección al sistema operativo recibió el nombre de "modo protegido". En cambio, a la forma de funcionamiento en la que se comportaba como un 8086 se la denominó "modo real".

Entre las mejoras en modo protegido que introdujo el procesador 80286, estaba el uso de los registros de segmento como selectores o punteros a una tabla de descriptores de segmento. Estos descriptores proporcionaban una dirección base de 24 bits que permitía la gestión de la memoria virtual en una estructura de segmentos intercambiables. Otros mecanismos de protección que implementaba el 80286 incluían el control del límite de acceso a segmentos, opciones de segmentos de sólo lectura o sólo ejecución y hasta cuatro niveles de privilegio para la ejecución de instrucciones. A pesar se las mejoras aportadas, el modo protegido del 80286 no "cuajó" y en cambio el 80286 sí fue un éxito de ventas como un 8086 mucho más rápido (su frecuencia de reloj llegada hasta los 12.5MHz).

80386

Más tarde, en 1985, llegó el procesador 80386, este procesador es el primer procesador de 32 bits de la arquitectura IA-32. Esto quiere decir, que tanto sus registros como las unidades de ejecución y los buses de direcciones y datos tenían 32 bits. Con 32 bits en el bus de direcciones es capaz de direccionar 2^{32}, ó lo que es lo mismo 4 GB.

Con el 80386 Intel mantiene la política ya iniciada con el 80286 de establecer la compatibilidad de sus modelos. De esta forma, el 80386 también presenta dos modos de funcionamiento, "modo real" en el que funciona como un 8086 mejorado y "modo protegido" en el que se comporta como una nueva CPU.

En modo real, esta CPU era totalmente compatible con el 8086, pero además presentaba la mejora de utilizar los registros de 32 bits. Esta nueva posibilidad fue inmediatamente aprovechada por los programadores. En cambio, las diferencias más importantes las presentaba en modo protegido. En primer lugar, el modo protegido del 80386 era diferente del 80286, lo que los hacia por tanto incompatibles. Este hecho motivo la rápida desaparición del 80286, para el cual de todas formas apenas se habían escrito programas en modo protegido.

La nueva arquitectura de 32 bits en modo protegido establecía un espacio de direcciones lógico para cada proceso, utilizando para ello un modelo de memoria segmentado (la memoria se compartimentaba en trozos denominados segmentos y referenciados mediante un descriptor de segmento). La segmentación de memoria del modo protegido podía utilizarse como tal, o bien de forma simplificada en lo que se llamó "modelo plano de memoria". En este caso todos los registros de segmento se inicializan con el mismo valor para acceder a los 4 GB del espacio de direcciones.

El procesador 80386 también introdujo la paginación en la arquitectura IA-32, con un tamaño de página fijo de 4 KB. El mecanismo de paginación aportó un método de manejo de la memoria virtual significativamente superior al uso de segmentos. La paginación resulta mucho más eficiente para la gestión de memoria en los sistemas operativos y además es completamente transparente a las aplicaciones, y sin necesidad de sacrificar velocidad de ejecución. Estas nuevas características hicieron que el 80386 fuera adecuado para soportar los sistemas operativos multitarea modernos como Windows y Linux. Pero, el modo protegido a un presentaba otra novedad, el 80386 funcionando en modo protegido permitía la posibilidad de ejecutar programas como si fuera un 8086, es lo que Intel denominó el modo "8086 virtual", aunque para utilizarlo es necesario disponer de un sistema operativo adecuado.

El 80386 supuso un revulsivo en el desarrollo de los computadores, no sólo por el establecimiento del nuevo modo protegido que daba soporte a los sistemas operativos multitarea, sino que además introdujo mejoras tecnológicas que aumentaron en gran medida el rendimiento del procesador. En este sentido, el 80386 fue el primer procesador de la arquitectura IA-32 en introducir varias etapas de procesamiento de las instrucciones trabajando en paralelo, en concreto seis etapas. La misión de cada una de estas etapas era:

- La unidad de interfaz al bus, para llevar a cabo el acceso a memoria y a los dispositivos de E/S.
- La unidad de prebúsqueda de código, encargada de recibir el código máquina de la unidad de interfaz al bus y colocarlo en una cola de 16 bytes.
- La unidad de decodificación de instrucciones, que decodificaba el código máquina de la unidad de prebúsqueda a microcódigo.
- La unidad de ejecución, encargada de ejecutar el microcódigo de las instrucciones.

- La unidad de segmento, cuya misión era traducir las direcciones lógicas a direcciones lineales y llevar a cabo las comprobaciones de protección.
- La unidad de paginación, que traduce las direcciones lineales a direcciones físicas y lleva a cabo las comprobaciones de protección de acceso a páginas de memoria.

Resumiendo, se puede considerar al 80386 como la base de funcionamiento de los PC's modernos tal como los conocemos. El 80386 definió el modo protegido que soporta a los sistemas operativos multitarea actuales, Windows y Linux. A partir del 80386 los procesadores posteriores aportan fundamentalmente innovaciones tecnológicas orientadas a mejorar el rendimiento y extender su funcionalidad, incorporando para ello nuevas instrucciones orientadas a tareas específicas.

80846

El procesador 80486, aparecido en 1989, aportó una mayor capacidad de ejecución en paralelo. En el 80486 las unidades de decodificación y de ejecución se dividieron en cinco etapas segmentadas cada una de ellas, de forma que podían estar ejecutándose simultáneamente hasta 5 instrucciones.

Hasta este momento, todos los procesadores vistos eran capaces de manejar números enteros en la ALU, pero no tenían hardware capaz de operar con números reales. Las operaciones con reales se lograban por software mediante largos y complicados programas. Para agilizar estos cálculos se podía añadir la CPU un compañero llamado "coprocesador matemático", que es como una ALU especializada en operar con números reales. Este coprocesador recibía el mismo nombre que la CPU pero terminado en 7. Así por ejemplo, el coprocesador para el 80386 era el 80387. Sin embargo, el 80486 fue el primer procesador en el cual el coprocesador matemático se integraba dentro de la CPU.

El 80486, además, fue el primer procesador en incorporar una memoria ultrarápida en el propio chip de la CPU. Esta memoria recibió el nombre de "cache" de primer nivel, tenía una capacidad de 8 KB y su misión era mantener los datos más frecuentemente usados.

Las mejoras tecnolóogicas en el proceso de fabricación permitieron también ir incrementando paulatinamente la frecuencia de reloj, llegando a alcanzar al final de la serie los 100 MHz.

También al final de la serie, se incorporaron al procesador opciones para el manejo de energía, para usarse con las primeras versiones de ordenadores operados con baterías.

Pentium

En 1993 aparece el Pentium, rompiendo con la nomenclatura que Intel había seguido hasta entonces con sus procesadores. Esta CPU incorpora una gran cantidad de mejoras internas destinadas a lograr que las instrucciones se ejecuten muchísimo más rápido, sin necesidad de incrementar mucho la velocidad de reloj, inicialmente de 66 MHz, sino logrando que se ejecuten más instrucciones en menos ciclos. Esto se consiguió a costa de triplicar la complejidad interna de la CPU.

Así, el Pentium estaba dotado de dos "cauces de ejecución", o lo que es lo mismo, podía ejecutar dos instrucciones a la vez replicando todas las unidades de ejecución (dos ALUs, dos unidades de control, etc). También se duplicó el tamaño de la memoria cache de primer nivel dividiéndola en 8 KB para datos y 8 KB para código. En esta memoria cache se permitió por primera vez la técnica de "escritura diferida" (write-back) aparte de la de la existente de "escritura a través" (write-through). Se le incorpora además una unidad encargada de realizar la predicción del destino de los saltos, el gran problema de la segmentación, utilizando para ello una tabla que recordaba los saltos más recientes, e integrada en el propio chip. Aunque los registros se mantienen en 32 bits, los buses internos se amplían llegando a 128 y 256 bits, de forma que se aceleran las transferencias internas, mientras que el bus de datos externo crece hasta 64 bits.

A medida que fue avanzando la serie, se fue incrementando la frecuencia de reloj llegando a alcanzar los 166 MHz. Se le incorporó también al Pentium un controlador de interrupciones avanzado (APIC Advanced Programmable Interrupt Controller) para soportar sistemas con varios procesadores, y finalmente la incorporación de la tecnología MMX (Matrix Math Extensions). Las instrucciones MMX, basadas en registros de 64 bits, permiten que una única instrucción se lleve a cabo sobre un conjunto de datos, en lo que se conoce como modelo de ejecución SIMD (Simple instrucción múltiples datos) y están orientadas a la computación paralela. Estas instrucciones aceleran el procesamiento de imágenes, sonido y aplicaciones de compresión.

La familia P6

En 1995 Intel introdujo una nueva familia de procesadores conocida con el nombre en clave de P6 y de los cuales el primer representante fue el Pentium Pro, y al que siguieron el Pentium II, Pentium II Xeon, Pentium Celeron, Pentium III y Pentium III Xeon.

El objetivo de esta nueva familia era utilizar una nueva micro arquitectura que permitiera incrementar el rendimiento del procesador Pentium manteniendo la misma tecnología de fabricación (0.6 micras) y por tanto sin aumentar la frecuencia de trabajo.

De esta forma, el procesador Pentium Pro es un procesador superescalar de 3 vías (tiene tres cauces de ejecución) con 12 etapas en cada una de ellas. En este nuevo procesador se introduce el concepto de ejecución dinámica (análisis del flujo de micro-datos, ejecución fuera de orden, mejora de la predicción de salto y ejecución especulativa) en la implementación superescalar.

Las tres unidades de decodificación de instrucciones trabajan en paralelo para decodificar el código máquina en operaciones más pequeñas llamadas micro-operaciones (micro-op's códigos de operación de la micro-arquitectura). Las micro-operaciones se llevan a un almacén (pool) de instrucciones, y cuando las interdependencias lo permiten, pueden ser ejecutadas fuera de orden por las cinco unidades de ejecución paralelas (2 de enteros, 2 de coma flotante y una de interfaz con la memoria). La unidad de retirada se encarga de retirar las micro operaciones completadas en el orden original del programa teniendo en cuenta cualquier salto.

La familia P6 también incorporó mejoras en el sistema de caches, introduciendo un segundo nivel de caches de 256 KB en el propio chip del procesador y aumentando la velocidad de acceso al primer nivel. Externamente, el bus de direcciones se incrementó hasta 36 bits, lo que le permitía direccionar hasta 64 GB.

En 1997 aparece el Pentium II, el cual además de seguir incrementandola velocidad de reloj (que llega ya a los 300 MHz) y la complejidad interna, añade instrucciones nuevas a la familia P6. Las nuevas instrucciones añadidas son las MMX (Matrix Math Extensions), que permiten a esta CPU trabajar con datos enteros de 64 bits segmentándolos en varios trozos con los que puede operar simultáneamente gracias a sus múltiples ALUs. Este tipo de operaciones resultan útiles cuando los datos a procesar representan imágenes o sonido, por lo que la CPU está pensada para mejorar la velocidad de los programas "multimedia".

Paralelamente, se incrementa el tamaño de las caches de primer nivel, pasando a 16 KB para datos y 16 KB para instrucciones y la cache de segundo nivel puede soportar 256 KB, 512 KB ó 1 MB, a la vez que incrementa su velocidad.

Posteriormente van apareciendo nuevas versiones derivadas de la familia P6 que aportan incrementos de rendimiento a la par que la frecuencia de reloj, gracias a las mejoras en la tecnología de fabricación. El procesador Pentium II Xeon, orientado a servidores, mantiene las características de sus predecesores pero además está pensado para trabajar en entornos de multiprocesamiento, colocando 4, 8 ó más de estos procesadores. El tamaño de la cache de segundo nivel llega hasta los 2 MB.

Por el contrario, el procesador Intel Celeron es una versión barata de procesador orientado a PC domésticos. Este procesador reduce el coste al implementar una cache de segundo nivel de sólo 128 KB y utilizar un encapsulado para el chip de plástico.

En 1999 aparece el procesador Pentium III que eleva la frecuencia de reloj, introduce las extensiones al procesamiento de instrucciones SIMD (SSE). Las instrucciones SSE extendían el modelo de ejecución SIMD ya introducido con la tecnología MMX con un nuevo conjunto de registros de 128 bits y la capacidad de llevar a cabo operaciones SIMD sobre datos flotantes en simple precisión y no sólo sobre enteros.

Por último, el procesador Pentium III Xeon incrementa los niveles de prestaciones por la ganancia en velocidad al utilizar una tecnología de fabricación de 0,18 micras.

Pentium IV

El procesador Intel Pentium 4 es el primer procesador basado en una nueva micro arquitectura denominada NetBurst que permite al procesador funcionar a velocidades de reloj significativamente más altas que sus predecesores y por tanto presentar valores de rendimiento muy superiores.

De forma muy breve, las mejoras que introduce esta nueva micro arquitectura son las siguientes:

- Mejora la velocidad del motor de ejecución al conseguir que las unidades aritmético lógicas funcionen al doble de velocidad que el procesador. De esta forma, las operaciones básicas con enteros se ejecutan en medio ciclo de reloj.
- El cauce de ejecución de instrucciones alcanza las 20 etapas de segmentación.

- Se mejora la unidad de ejecución dinámica, permitiendo que la unidad de ejecución especulativa tenga en cuenta hasta 128 instrucciones a ejecutar. Se mejora también la capacidad de predicción de saltos.
- Se mejora el subsistema de memoria cache. En el primer nivel se introduce un nuevo concepto "cache de micro operaciones" para las instrucciones. También mejora la velocidad de acceso a la cache de segundo nivel.

La última innovación de Intel introducida en el Pentium 4 es lo que denomina tecnología "Hyper-Threading". Esta tecnología está orientada a la ejecución simultánea de porciones o instancias de un programa denominadas threads o hilos. Con esta nueva tecnología, el procesador es capaz de ejecutar en paralelo dos threads o hilos, aprovechando la replicación de unidades funcionales. Sin embargo, para conseguirlo ha sido necesario duplicar el estado del procesador, o lo que es lo mismo la mayor parte de sus registros. Estas mejoras están orientadas a programas que trabajan con muchas instancias réplicas, como son los servidores de páginas web y las aplicaciones de ebusiness en general.

Estas y otras mejoras en la micro arquitectura, así como la mejora en el proceso tecnológico de fabricación han permitido alcanzar las velocidades de procesamiento tan elevadas de los PCs actuales.

A modo de resumen en la Figura 3.1 muestra la evolución que ha seguido la arquitectura IA-32 en sus poco más de 25 años de vida. Se puede observar como se ha incrementado tanto la complejidad como la capacidad de los PCs con unas prestaciones cada vez más elevadas.

El futuro: las arquitecturas de 64 bits

El siguiente paso en el incremento de las prestaciones de los PCs es incrementar el ancho de sus registros y buses hasta alcanzar los 64 bits, como hace tiempo que han hecho los procesadores de los grandes computadores.

El principal impacto de incrementar el bus de direcciones (el datos hace tiempo que se utiliza con 64 bits) sería incrementar el espacio de memoria direccionable que pasaría a ser de 264, si bien por motivos tecnológicos aún no se utilizan todos los bits disponibles.

Nombre	Fecha	Frecuencia	Número de Transistores	Espacio de direcciones	Caches L1 y L2
8086	1978	8 MHz	29 K	1 MB	Ninguna
80286	1982	12.5 MHz	134 K	16 MB	Ninguna
80386	1985	20 MHz	275 K	4 GB	Ninguna
80486	1989	25 MHz	1.2 M	4 GB	L1: 8KB
Pentium	1993	60 MHz	3.1 M	4 GB	L1: 16KB
Pentium Pro	1995	200 MHz	5.5 M	64 GB	L1: 16KB L2: 256KB ó 512KB
Pentium II	1997	266 MHz	7 M	64 GB	L1: 32KB L2: 256KB ó 512KB
Pentium III	1999	500 MHz	8.2 M	64 GB	L1: 32KB L2: 512KB
Pentium 4	2000	1.50 GHz	42 M	64 GB	Cache μop: 12K L1: 8KB L2: 256KB

Figura 3.1: Características principales de los procesadores de la arquitectura IA-32.

En la actualidad nos encontramos con varias arquitecturas de 64 bits para los PCs más modernos:

- La arquitectura IA-64 actual, representada por el Itanium. Este procesador fue el fruto de una aventura tecnológica entre Intel y Hewllet Packard para crear un procesador de 64 bits. Las principales características de este procesador son sus registros de 64 bits, la incorporación de un tercer nivel de cache, el uso de un bus de direcciones de 44 bits, lo que le permite direccionar 16 TB (TeraBytes). Este procesador está pensado para la integración en sistemas multiprocesadores densos. Plantea el inconveniente que rompe con la compatibilidad de la arquitectura IA-32, aunque si puede ejecutar todas las aplicaciones escritas para la arquitectura IA-32, pero mediante emulación, lo que reduce sensiblemente el rendimiento.

- La arquitectura AMD64. AMD es el gran rival de Intel en la fabricación de procesadores y que hasta ahora siempre había ido a la zaga de Intel. Sin embargo con la arquitectura AMD64, implementada en los procesadores Athlon 64 y Opterom, incorpora registros de 64 bits y un bus de direcciones de 40 bits, lo que le per-

mite direccionar hasta 1 TB. A diferencia de la arquitectura IA-64, la arquitectura AMD64 mantiene la compatibilidad con la arquitectura IA-32, con lo que no existen problemas para ejecutar las aplicaciones escritas para esta arquitectura.

3.3.4. Nuevas tecnologías

Como se ha comentado en el apartado anterior, las nuevas tecnologías aparecieron a raiz de la ejecución de instrucciones SIMD diseñado por Intel. SIMD son las siglas de "Single Input Multiple Data" que explican por sí solas la naturaleza de esta técnica: procesamos en paralelo varios datos, aplicándoles la misma operación.

Las operaciones SIMD se introdujeron en el repertorio IA-32 con la tecnología MMX. Esta tecnología permitía que se realizasen operaciones sobre 64 bits empaquetados en bytes, palabras o dobles palabras. El Pentium III extendió el modelo con la introducción de las Streaming SIMD Extenseions (SSE). Los registros pasaron a ser de 128 bits, y se podían realizar operaciones sobre operandos que contenían cuatro elementos en punto-flotante de precisión simple.

El Pentium 4 ha aumentado aún más la funcionalidad de las operaciones SIMD con las llamadas SSE2. Estas operaciones son capaces de trabajar con elementos en punto flotante y doble precisón y con enteros empaquetados en 128 bits. Dispone de 144 nuevas instrucciones que pueden operar sobre dos datos empaquetados en doble precisión, o sobre enteros de 16 bytes, 8 palabras, 4 doble palabras y 2 quadwords.

El repertorio SIMD mejora cuantiosamente el rendimiento en aplicaciones multimedia, como el procesamiento de gráficos 3D, reconocimiento del habla.... y cualquier otra aplicación que tenga un gran paralelismo inherente, que se traduce en patrones de acceso a memoria muy regulares, realizando las mismas operaciones sobre los datos accedidos.

Representación de datos

Los datos que se pueden representar en la arquitectura IA-32 pueden tener un tamaño de byte, de palabra (word), de doble palabra (double word), de cuádruple palabra (quad-word) y de doble cuádruple palabra (double quadword).

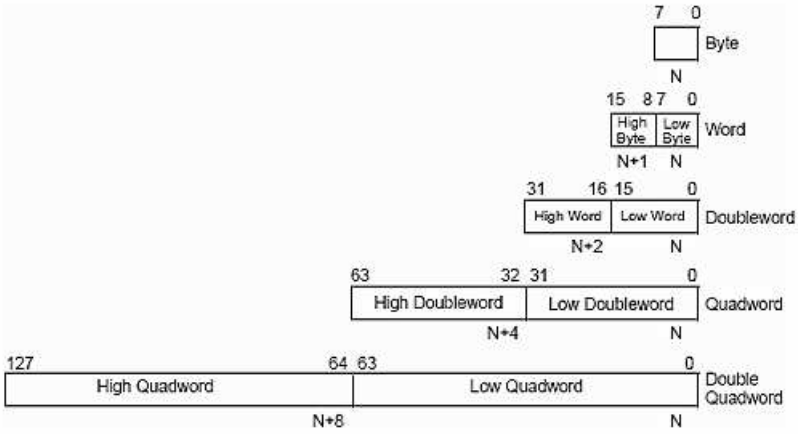

Figura 3.2: Tamaño de datos para la arquitectura IA-32.

Los datos se pueden representar como enteros, con y sin signo de tamaño byte, palabra, doble palabra, cuádruple palabra y doble cuádruple palabra y se pueden representar en punto flotante de simple y doble precisión, como se puede observar en la Figura 3.3.

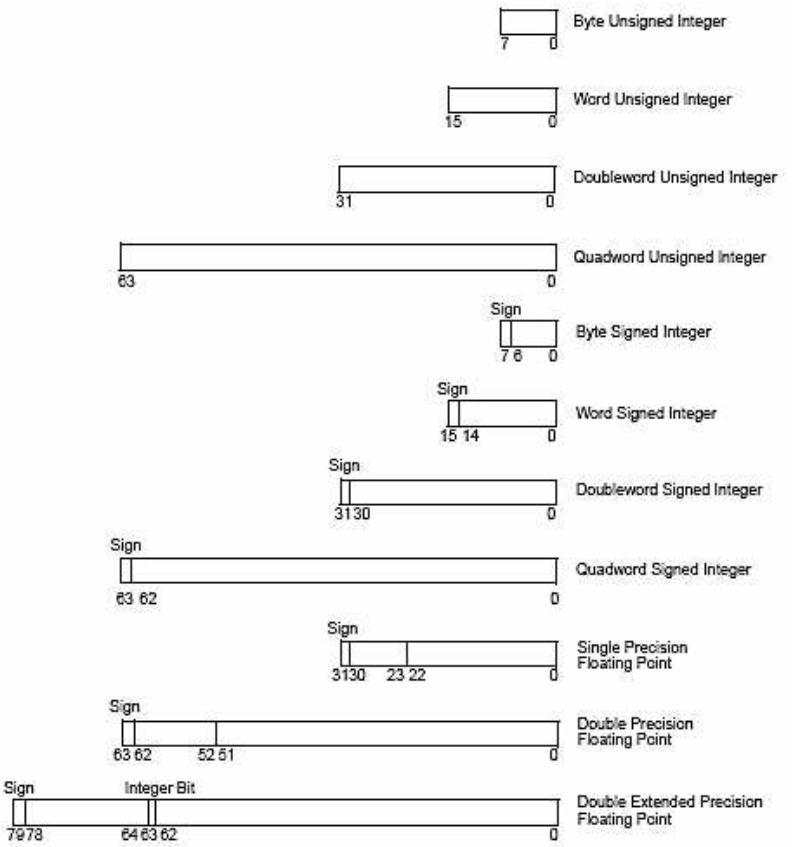

Figura 3.3: Tipo de datos para la arquitectura IA-32.

Modo de trabajo

La arquitectura IA-32 trabaja con operadores de 64 y 128 bits para la ejecución de instrucciones SIMD. El tipo de datos empleados son los definidos en el apartado anterior y que pueden estar empaquetados, es decir, se puede trabajar con paquetes de datos en una misma operación.

Las tecnología MMX trabaja con enteros empaquetados de 64 bits, que pueden ser 8 enteros de tamaño byte (con signo o sin signo), 4 enteros de tamaño palabra (con signo o sin signo) ó 2 enteros de tamaño doblepalabra (con signo o sin signo).

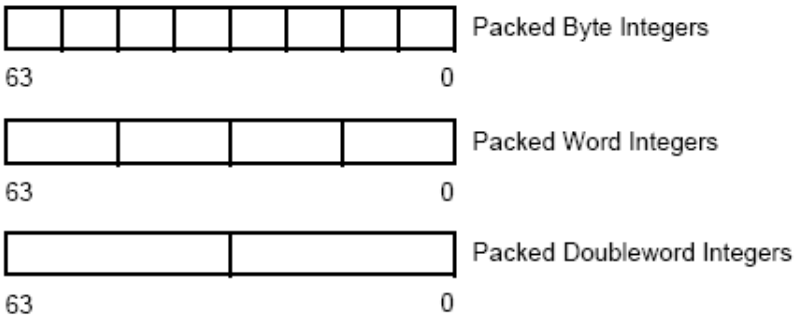

Figura 3.4: Tipos de paquetes de datos SIMD de 64 bits.

Las tecnologías SSE, SSE2 y SSE3 trabajan con enteros empaquetados de 128 bits, que pueden ser 16 enteros de tamaño byte (con signo o sin signo), 8 enteros de tamaño palabra (con signo o sin signo), 4 enteros de tamaño doble palabra (con signo o sin signo) ó 2 enteros tamaño cuádruple palabra (con signo o sin signo).

Además la tecnología SSE trabaja con paquetes de datos en punto flotante de simple precisión (32 bits), la tecnología SSE2 con paquetes de datos en punto flotante de doble precisión (64 bits) y por último la tecnología SSE3 engloba el trabajo de las tecnologías anteriormente mencionadas como se puede observar en la Figura 3.5.

Para poder operar con estas nuevas tecnologías la arquietectura IA-32 dispone de una serie de registros, así tendremos los registros:

- Los registros MMX, es un conjunto de 8 registros (MM0..MM7) que sólo se pueden utilizar para realizar cálculos con tipos de datos MMX, no permite el direccionamiento de la memoria con éllos, ya que sólo contienen datos. Además, están mapeados en los registros (R0..R7) de la pila de la unidad de punto flotante (FPU).

Figura 3.5: Tipos de paquetes de datos SIMD de 128 bits.

- Los registros XMM son de 128 bits, son un total de 8 y que son usados por las recnologías SSE, SSE2 y SS3. Los operandos de las instrucciones SSE pueden ser obtenidos de la memoria, de los registros de propósito general, de los registros MMX o de los registros XMM.

Juego de instrucciones para las nuevas tecnologías

Se van a ver el conjunto de instrucciones para cada tecnología de las desarrolladas anteriormente.

1. **Instrucciones MMX**
   Trabajan, utilizando la técnica SIMD (Múltiples Datos en una Única Instrucción), con datos enteros localizados en los registros MMX y XMM ó en memoria, es decir, pueden trabajar sobre 64 y sobre 128 bits simultáneamente. Todas las instrucciones comienzan por P.

   a) Transferencia de datos:
   MOVD / MOVQ Copia de 32 y 64 bits, respectivamente.

b) Aritméticas:
PADDB / PADDW / PADDD Suma empaquetada con enrollamiento de tamaño bytes (B), palabra (W) ó de doble palabra (D).
PADDSB / PADDSW Suma empaquetada con saturación (S) y con signo.
PADDUSB / PADDUSW Suma empaquetada con saturación y sin signo (U).
PSUBB / PSUBW /PSUBD Resta empaquetada con enrollamiento.
PSUBSB / PSUBSW Resta empaquetada con saturación y con signo.
PSUBUSB / PSUBUSW Resta empaquetada con saturación y sin signo.
PMULLW / PMULHW Hace 4 productos de 16 bits con resultados de 32 bits de los que se elige la parte baja o la parte alta, respectivamente.

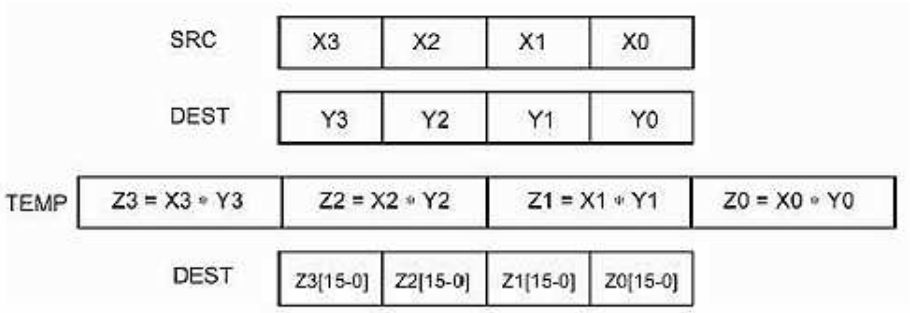

Figura 3.6: Ejemplo de instrucción PMULLW.

PMADDWD Multiplicación y suma en una única operación.

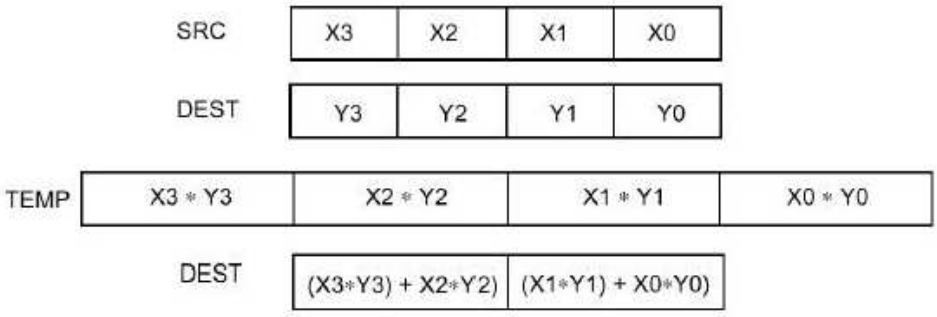

Figura 3.7: Ejemplo de instrucción PMADDWD.

c) De comparación:
PCMPEQB / PCMPEQW / PCMPEQD Hace las comparaciones elemento a elemento de tamaño byte, word y dword y almacena todo ceros si no eran iguales y todo unos si eran iguales.

PCMPGTB / PCMPGTW / PCMPGTD Hace las comparaciones elemento a elemento y almacena todo ceros si no era mayor y todo unos si era mayor.

d) De conversión:
PACKSSWB / PACKSSDW Empaqueta con saturacón y con signo, pasando de word a byte, y de dword a word, respectivamente.
PACKUSWB Empaqueta con saturación y sin signo.
PUNPCKLBW / PUNPCKLWD / PUNPCKLDQ Desempaqueta y entremezcla la parte baja.

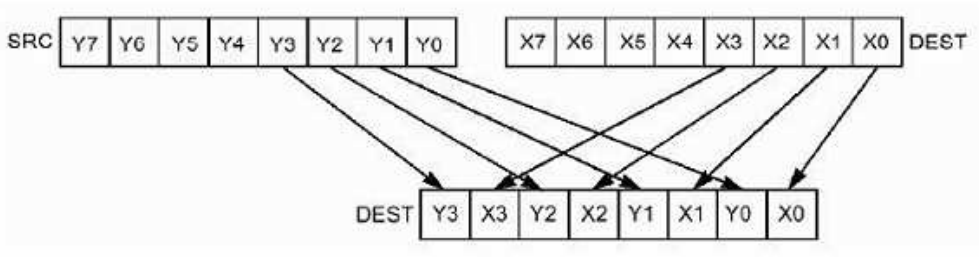

Figura 3.8: Ejemplo de instrucción PUNPCKLBW.

PUNPCKHBW / PUNPCKHWD / PUNPCKHDQ Desempaqueta y entremezcla la parte alta.

e) Lógicas:
PAND AND lógico.
PANDN NOT destino AND fuente.
POR OR lógico.
PXOR XOR lógico.

f) De desplazamiento:
PSLLW / PSLLD / PSLLQ Desplazamiento lógico a la izquierda.
PSRLW / PSRLD / PSRLQ Desplazamiento lógico a la derecha.
PSRAW / PSRAD Desplazamiento aritmético a la derecha.

2. **Instrucciones SSE**
Este conjunto de instrucciones operan con paquetes de datos en punto flotante de simple precisión localizados en registros XMM o en memoria.

a) Transferencia de datos:
MOVAPS / MOVUPS Copia de 128 bits, alineados (A) o no alineados (U) en memoria.

MOVHPS / MOVLPS Copia de 64 bits de la parte alta o de la parte baja, respectivamente.
MOVHLPS / MOVLHPS Copia de 64 bits entre las partes alta y baja, y viceversa.
MOVMSKPS Extrae el signo de 4 paquetes en memoria (32 bits).
MOVSS Copia de 32 bits.

b) Aritméticas:
ADDPS / SUBPS Suma/Resta de cuatro paquetes de datos.
ADDSS / SUBSS Suma/Resta de un paquete de datos.
MULPS / DIVPS Multiplicación/División de cuatro paquetes de datos.
MULSS / DIVSS Multiplicación/División de un paquete de datos.
RCPPS Realiza la operación 1/Fuente para cuatro paquetes de datos. El error relativo cometido va a ser menor o igual que $1{,}5 \cdot 2^{-12}$.
RCPSS Igual que el anterior pero para un único paquete de datos.
SQRTPS / SQRTSS Raíz cuadrada de cuatro ó de un paquete de datos respectivamente.
RSQRTPS / RSQRTSS Inversa de raíz cuadrada de cuatro ó de un paquete de datos, respectivamente.
MAXPS / MAXSS Cálculo del máximo entre registros para cuatro ó para un paquete de datos, respectivamente.
MINPS / MINSS Cálculo del mínimo entre registros para cuatro ó para un paquete de datos, respectivamente.

c) De comparación:
CMPPS / CMPSS Comparación para cuatro ó para un paquete de datos a través de un entero de 8 bits.
COMISS / UCOMISS Comparaíón ordenada/desordenada modificando los flags.

d) De entremezclado y empaquetamiento:
SHUFPS Entremezclado de paquetes de datos a través de un entero de 8 bits.

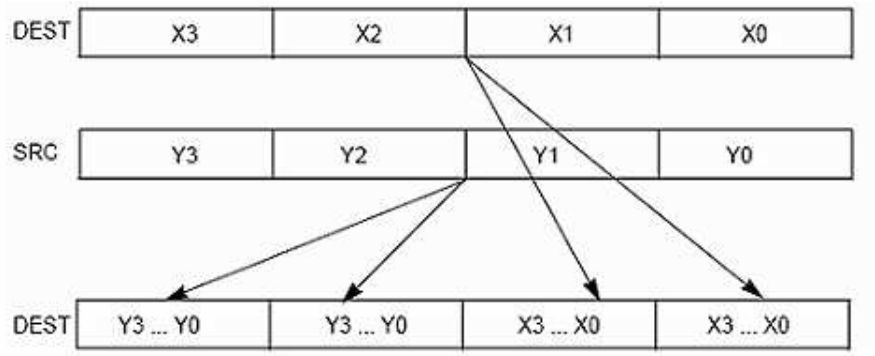

Figura 3.9: Ejemplo de instrucción SHUFPS.

UNPCKHPS / UNPCKLPS Desempaqueta y entremezcla partes altas y partes bajas, respectivamente.

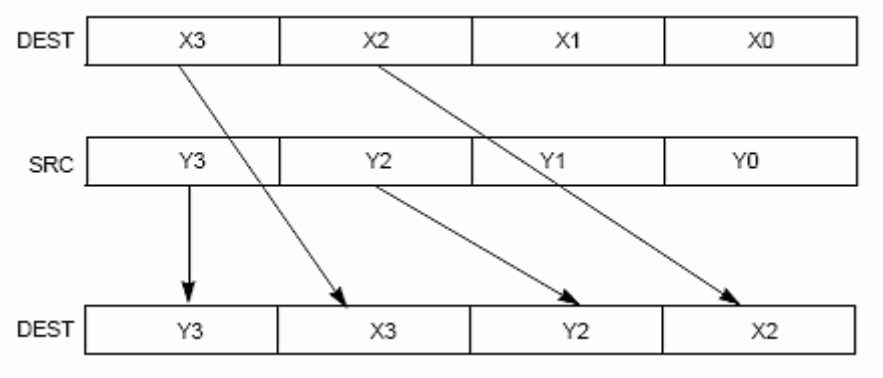

Figura 3.10: Ejemplo de instrucción UNPCKHPS.

e) Lógicas:
ANDPS AND lógico.
ANDNPS NOT destino AND fuente.
ORPS OR lógico.
XORPS XOR lógico.

f) De conversión:
CVTPI2PS / CVTPS2PI Convierte un paquete de dos dword a punto flotante de simple precisión y viceversa.
CVTSI2SS / CVTSS2SI Convierte un dword a punto flotante de simple precisión y viceversa.
CVTTPS2PI Convierte un paquete de dos datos expresados en punto flotante de simple precisión en enteros de tamaño dword usando truncamiento.
CVTTSS2SI Convierte un dato expresado en punto flotante de simple precisión en entero de tamaño dword usando truncamiento.

g) De administración del registro de estado (MXCSR):
LDMXCSR Carga el registro MXCSR desde memoria.
STMXCSR Salva el registro MXCSR en memoria.

h) Instrucciones con enteros:
PAVGB / PAVGW Cálculo de la media de datos enteros de tamaño byte (B) y word (W), respectivamente.
PEXTRW Extración de una palabra de un registro a través de un número inmediato de 8 bits.
PINSRW Inserción de una palabra a través de un número inmediato de 8 bits.
PMAXUB Devuelve el máximo, comparando enteros sin signo de tamaño byte.
PMAXSW Devuelve el máximo, comparando enteros con signo de tamaño palabra.
PMINUB Devuelve el mínimo, comparando enteros sin signo de tamaño byte.
PMINSW Devuelve el máximo, comparando enteros con signo de tamaño palabra.
PMOVMSKB Copia de una máscara de byte.
PMULHUW Multiplica paquetes de datos enteros de tamaño palabra sin signo y almacena la parte alta de la multiplicación (Figura 3.11).
PSADBW Suma de las diferencias absolutas (Figura 3.12).
PSHUFW Desempaquetado y entremezclado a través de un número inmediato de 8 bits (Figura 3.13).

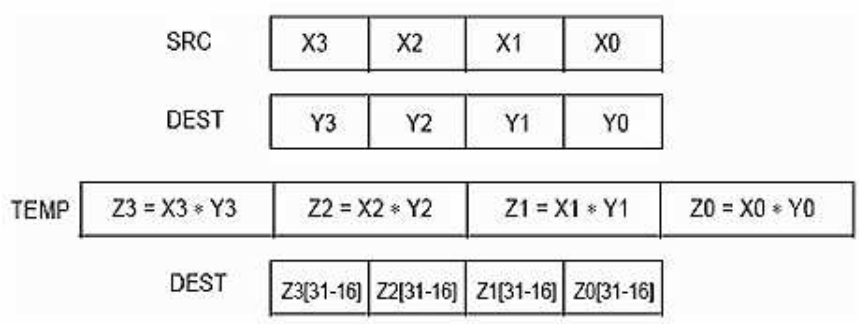

Figura 3.11: Ejemplo de instrucción PMULHUW.

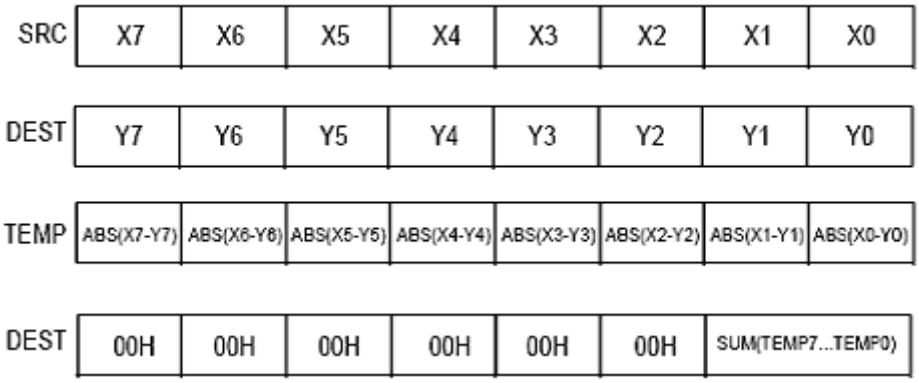

Figura 3.12: Ejemplo de instrucción PSADBW.

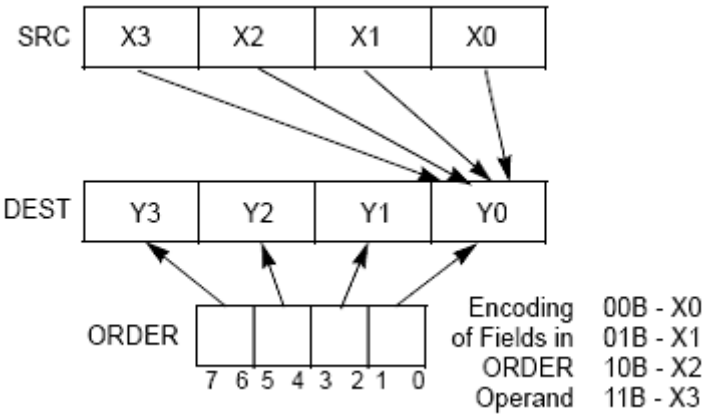

Figura 3.13: Ejemplo de instruccion PSHUFW

i) Otras instrucciones:
MASKMOVQ Escribe los bytes seleccionados en memoria a través de una máscara.
MOVNTQ Mueve una qword desde un registro a memoria directamente.
MOVNTPS Mueve cuatro paquetes de datos en punto flotante de simple precisión desde un registro a memoria directamente.
PREFETCHh Movimiento de datos entre cachés usando T0(datos temporales), T1 (datos temporales respecto al primer nivelde caché), T2 (datos temporales respecto al segundo nivel de caché) ó NTA (datos no temporales respecto a todos niveles de caché).
SFENCE Serializa operaciones de memoria.

3. **Instrucciones SSE2**
Este conjunto de instrucciones operan con paquetes de datos en punto flotante de doble precisión localizados en registros XMM o en memoria.

a) Transferencia de datos:
MOVAPD / MOVUPD Copia de 128 bits, alineados (A) o no alineados (U) en memoria.
MOVHPD / MOVLPD Copia de 64 bits de la parte alta o de la parte baja, respectivamente.
MOVHLPD / MOVLHPD Copia de 64 bits entre las partes alta y baja, y viceversa.
MOVMSKPD Extrae el signo de 4 paquetes en memoria (32 bits).
MOVSD Copia de 64 bits.

b) Aritméticas:
ADDPD / SUBPD Suma/Resta de dos paquetes de datos.
ADDSD / SUBSD Suma/Resta de un paquete de datos.
MULPD / DIVPD Multiplicación/División de dos paquetes de datos.
MULSD / DIVSD Multiplicación/División de un paquete de datos.
SQRTPD / SQRTSD Raíz cuadrada de dos ó de un paquete de datos respectivamente.
MAXPD / MAXSD Cálculo del máximo entre registros para dos ó para un paquete de datos, respectivamente.
MINPD / MINSD Cálculo del mínimo entre registros para dos ó para un paquete de datos, respectivamente.

c) De comparación:
CMPPD / CMPSD Comparación para cuatro ó para un paquete de datos a través de un entero de 8 bits.
COMISD / UCOMISD Comparaíón ordenada/desordenada modificando los flags.

d) De entremezclado y empaquetamiento:
SHUFPD Entremezclado de paquetes de datos a través de un entero de 8 bits.

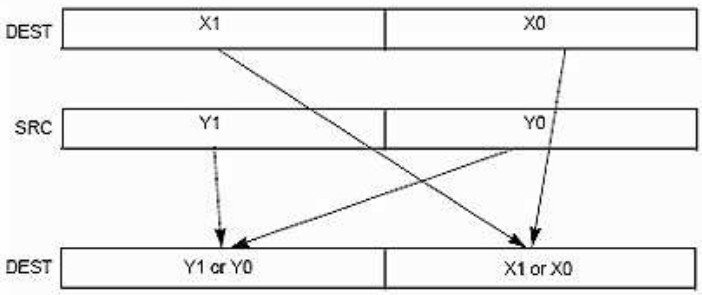

Figura 3.14: Ejemplo de instrucción SHUFPD.

UNPCKHPD / UNPCKLPD Desempaqueta y entremezcla partes altas y partes bajas, respectivamente.

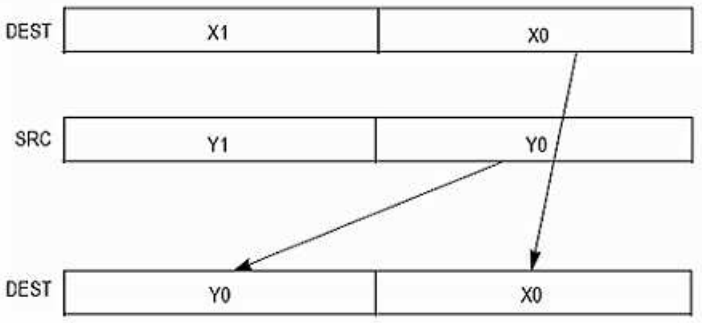

Figura 3.15: Ejemplo de instrucción UNPCKLPD.

e) Lógicas:
ANDPD AND lógico.
ANDNPD NOT destino AND fuente.
ORPD OR lógico.
XORPD XOR lógico.

f) De conversión:
CVTPD2PI / CVTPI2PD Convierte un paquete de dos datos en expresados en punto flotante de doble precisión en enteros de tamaño dword y viceversa.
CVTTPD2PI Convierte un paquete de dos datos en expresados en punto flotante de doble precisión en enteros de tamaño dword con truncamiento.
CVTPD2DQ / CVTDQ2PD Convierte un paquete de dos datos en expresados en punto flotante de doble precisión en enteros de tamaño qword y viceversa.
CVTTPD2DQ Convierte un paquete de dos datos en expresados en punto flotante de doble precisión en enteros de tamaño qword con truncamiento.
CVTPD2PS / CVTPS2PD Convierte un paquete de dos datos expresados en punto flotante de doble precisión en dos datos expresados en punto flotante de simple precisión y viceversa.
CVTSD2SS / CVTSS2SD Convierte un dato en expresado en punto flotante de doble precisión en un dato expresado en punto flotante de simple precisión y viceversa.
CVTSD2SI / CVTSI2SD Convierte un dato expresado en punto flotante de doble precisión en un entero de tamaño dword y viceversa.
CVTTSD2SI Convierte un dato expresado en punto flotante de doble precisión en un entero de tamaño dword con truncamiento.

g) Instrucciones con paquetes de datos en punto flotante de simple precisión:
CVTPS2DQ / CVTDQ2PS Convierte un paquete de cuatro datos expresados en punto flotante de simple precisión en enteros de tamaño dword y viceversa.
CVTTPS2DQ Convierte un paquete de cuatro datos expresados en punto flotante de simple precisión en enteros de tamaño dword con truncamiento.

h) Instrucciones con enteros:
MOVDQA / MOVDQU Copia de 128 bits, alineados (A) o no alineados (U) en memoria.
MOVQ2DQ / MOVDQ2Q Copia de 64 bits entre registros MMX y XMM y viceversa.
PMULUDQ Multiplica datos enteros sin signo de tamaño dword.
PADDQ Suma de paquete de datos de enteros de tamaño qword con y sin signo.
PSUBQ Resta de paquete de datos de enteros de tamañp qword con y sin signo.

PSHUFLW / PSHUFHW Desempaquetado y entremezclado de partes bajas y de las partes altas respectivamente, de enteros de tamaño palabra, a través de un número inmediato de 8 bits.
PSHUFD Desempaquetado y entremezclado de enteros de tamaño dword a través de un número inmediato de 8 bits.
PSLLDQ Desplazamiento lógico en bytes hacia la derecha.
PSRLDQ Desplazamiento lógico en bytes hacia la izquierda.
PUNPCKHQDQ / PUNPCKLQDQ Desempaquetado y entremezclado de las partes altas y bajas respectivamente, de enteros de tamaño qword.

4. **Instrucciones SSE3**
   Este conjunto de instrucciones operan con paquetes de datos enteros, con paquetes de datos en punto flotante de simple precisión o con paquetes de datos en doble precisión localizados en registros XMM, que incluyen además el procesamiento en paralelo de instrucciones.

   - De carga ó de movimiento ó de duplicado:
     MOVSHDUP: Copia ó mueve 128 bits, duplicando la segunda y la cuarta doble palabra del registro fuente. Trabaja con datos de 32 bits, tanto enteros de tamaño dword como datos en punto flotante de simple precisión.
     MOVSLDUP: Copia ó mueve 128 bits, duplicando la primera y la tercera doble palabra del registro fuente. Trabaja con datos de 32 bits, tanto enteros de tamaño dword como datos en punto flotante de simple precisión.
     MOVDDUP: Copia ó mueve 128 bits, duplicando los 64 bits de la fuente. Trabaja con datos de 64 bits, tanto enteros de tamaño qword como datos en punto flotante de doble precisión.

   - De suma ó resta:
     ADDSUBPD Suma/resta de 64 bits para paquetes de datos expresados enn punto flotante de doble precisión (Figura 3.16).
     ADDSUBPS Suma de la segunda y de la cuarta doble palabra y resta de la primera y de la tercera doble palabra para paquetes de datos expresados en punto flotante de simple precisión.

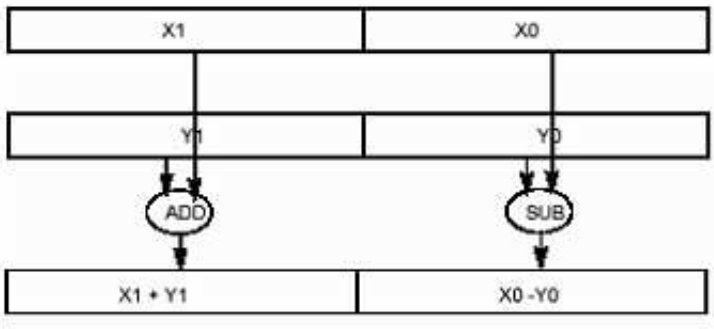

Figura 3.16: Ejemplo de instrucción ADDSUBPD.

- De suma ó resta en procesamiento paralelo:
  HADDPD Suma de datos, expresados en punto flotante de doble precisión, en paralelo.

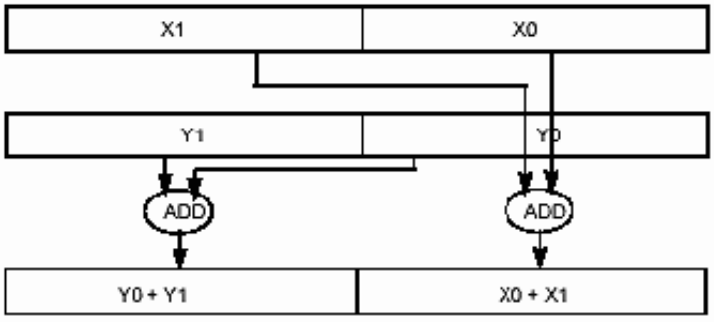

Figura 3.17: Ejemplo de instrucción HADDPD.

  HADDPS Suma de datos, expresados en punto flotante de simple precisión, en paralelo.
  HSUBPD Resta de datos, expresados en punto flotante de doble precisión, en paralelo.
  HSUBPS Resta de datos, expresados en punto flotante de simple precisión, en paralelo.

Capítulo 4

Visualización

En este capítulo se abordarán todos los temas relacionados con la visualización por pantalla de cualquier imagen que dispongamos.

Se tratará el estándar VBE (VESA Bios Extension) de la asociación VESA (Video Electronics Standards Association), donde veremos una pequeña introducción, que tratará de la historia del nacimiento de dicho estándar.

Seguidamente veremos como se define, donde aparecerán sus principales características de funcionamiento como son, la introducción de los conceptos de bus local (novedad en su aparición) y de LFB (Linear Frame Buffer).

Más tarde, se expondrán las principales características que presenta este estándar VESA.

A continuación, veremos como se programa con el estándar VESA, como se accede a sus diferentes funciones, tanto en modo real como en modo protegido (DPMI), como se obtienen los modos VESA de los que disponemos, como se selecciona un modo y, cuales son las principales características de un modo VESA seleccionado.

Y por último, veremos los dos modos fundamentales que tenemos para trabajar con la pantalla, como son el modo texto y el modo gráfico.

4.1. Breve historia

Al principio todo era frío, sin vida, eran tiempos del CGA, del EGA con sus flamantes 16 colores, del Hercules (HGC, muy famoso, snif), y otras muchas otras tarjetas no estandares que todavía se pueden ver en las viejas BYTE, pero eran pensadas para aplicaciones profesionales como AutoCAD.

IBM tenía que dar el paso, trás el desatre comercial de la PS/2 sobrevivió algo bueno, el VGA, que con sus primeros 64k podía ofrecer 320x200x256c y 640x480x16c. Se popularizó, se difundió, la nueva regla surgía, quien hizo una importante contribución fue Michael Abrash, que nos regaló los Modos X, principalmente permitían usar todos los 256k, hacer scroll por hardware, y alcanzar nuevas resoluciones.

El modo X tiene una propiedad característica, que dependiendo el tipo de programa que uno realice, puede acceder a la memoria dividida por planos. A partir de aquí, los fabricantes empezaron a escabiar y surgieron las primeras tarjetas SuperVGA, descendientes de aquellas viejas tarjetas no estandarizadas, solo que ahora soportaban el VGA, pero además le añadían características extras, como más resolucion y más colores. El problema con ellas es que no estan diseñadas bajo un estandar, cada una de ellas se programaba de diferentes formas, si uno quiere usar una resolución SuperVGA, tiene que hacer varias versiones de sus rutinas para cada uno de los chipsets mas famosos, así que una agrupación llamada Video Electronics Standars Association (VESA) creó un estandar para todas las placas SVGA llamado VESA BIOS Extension (VBE).

4.2. Definición

Las siglas VESA corresponden a Video Electronics Standards Association, que es una asociación encargada de realizar estándares relacionados con las tarjetas gráficas para ordenadores. Esta asociación realizó una arquitectura de placas base de 32 bits (Vesa Local Bus o VLB) y definió una serie de modos de video estándar para diversas tarjetas gráficas SVGA.

El VLB fue introducido a raíz de la introducción de los 386 con el fin de sacar partido a las posibilidades de las nuevas generaciones de procesadores, en especial a sus buses internos de 32 bits.

La solución consistió en conectar directamente con el procesador dos o tres de los dispositivos externos, que necesitaban de una conexión rápida, mediante un bus de 32 bits dotado de unos zócalos especiales.

Fue el primer desarrollo que utilizaba el concepto de bus local. La idea consiste en acercar el bus lo más posible a la CPU, lo que supone un esfuerzo técnico considerable. El objetivo es que el bus funcione a la velocidad de la CPU o, al menos, a una fracción no demasiado pequeña de ésta (uno o dos tercios). Así, VLB soporta velocidades de hasta 50 Mhz de 32 bits, pero por razones técnicas, con tres tarjetas sólo llega a 33 Mhz, con dos a 40 Mhz, y con 1 a 50MHz.

Hasta hoy en día nos encontramos con tres versiones, la versión VESA v1.2, la versión VESA v2.0 y la más actual, la versión VESA v3.0.

A partir de la versión VESA 2.0 aparece una característica fundamental para los programadores, como es el Linear Frame Buffer. Cuando el micro está en Protected Mode (modo protegido), se aprovechan plenamente los registros de 32 bits y desaparece la segmentación de la memoria del modo real. A partir de este momento, toda la memoria es lineal desde el primer byte de nuestra RAM hasta el último. Esto permite trabajar en SVGA sin segmentos y sin bancos, con buffers de memoria totalmente lineales.

4.3. Características VESA

Las principales características que ofrece VBE (VESA BIOS Extension) se enumeran a continuación:

- Ofrece el interfaz estándar para el uso de los controladores gráficos (dispositivos de SVGA).
- Interfaz opcional del modo protegido para sistemas operativos como DOS, Windows y UNIX.
- Método estándar para los modos soportados.
- Método estándar de identificación de productos y de fabricantes.
- Disposición para las extensiones del OEM con Subfunción 14h.
- Interfaz extensible con especificaciones suplementarias.

4.4. Programando con VBE

Cualquier programa realizado debe soportar las tres versiones anteriomente mencionadas.

El VBE es una extensión d e la interrupción 10h, el código puede estar en la ROM de la placa ó en la RAM, esta última manera se logra cargando un TSR como el UniVBE. Cuando queremos acceder a VBE se debe poner en el registro AH el valor de 4Fh (para distinguir del viejo Video BIOS) y en AL el número de función que se desee, a continuación se llama a la interrupción 10h.

El retorno en AX tiene siempre el mismo significado para todas las funciones:

- AL==4Fh La función esta soportada.
- AL!=4Fh La función no esta soportada.
- AH==00h La función se realizó con éxito.
- AH==01h La función falló.
- AH==02h El software soporta esta función, pero el hardware no.
- AH==03h La llamada a la función es inválida en el modo de vídeo actual.

Hay que destacar que hay que controlar estos valores, principalmente en las llamadas a las funciones 00h==Return SVGA Info, 01h==Return SVGA Mode Info y 02h==Set SVGA Mode.

Una vez mencionado como se llaman a las funciones y sus parámetros de retorno, lo siguiente es dectar si VESA esta presente, pidiendo que nos de la información acerca de la SVGA instalada. Esto es fácil si se trabaja en modo Real, pero se complica si es con DPMI (modo protegido).

A continuación, podemos ver como se llamaría a la función 0h (Return Super VGA Information), trabajando en modo real o trabajando con DPMI:

- Modo Real:
  - Alojamos un buffer de 256 bytes.
  - Ponemos en AX 4F00h, (Elejimos función).
  - Ponemos ES:DI apuntando al buffer que había alojado.
  - Llamada a la interrupción INT 10h.
  - Si AX!=4Fh entonces VESA no está presente.
  - Utilizamos la información del buffer si fuese necesario.
  - Liberamos el buffer.
- DPMI:
  - AX=100h (Funcion de DPMI "Allocate Buffer in Real Memory").
  - BX=16 (16*16=256 bytes que le pidimos).
  - INT 31h (DPMI int).
  - Si el flag C está seteado, entonces "Error: no hay suficiente memoria".
  - En AX nos devuelve el segmento real para pasarle a la función VESA.
  - En DX me devuelve el selector, para acceder desde mi programa.
  - Ahora tenemos que usar la siguiente estructura RMI (Real Mode Interrupt), primero la ponemos toda a 0 y, luego la pasamos al DPMI, el DPMI llenará los registros con el contenido de esta estrucura y con esos valores llamaremos a la int 10h:

```
DWORD EDI
DWORD ESI
DWORD EBP
DWORD Reservado
DWORD EBX
DWORD EDX
DWORD ECX
DWORD EAX = 4f00h (Función VESA 00h)
WORD flags;
WORD ES = El segmento retornado anteriormente por el
          DPMI host
WORD DS,FS,GS,IP,CS,SP,SS
```

  - AX=300h (función DPMI "Simulate Real Mode Interrupt").
  - BX=10h (le decimos al DPMI que interrupción queremos que llame).

- CX=0.
- Pongemos ES:EDI apuntando a RMI (ES=DS por supuesto).
- INT 31h (DPMI int).
- Lo que nos devuelve la int 10h se encuentra en RMI, es decir, si RMI.EAX!= 4Fh entonces VESA no esta presente.
- Utilizamos la información del buffer si fuese necesario.
- AX=101h (función DPMI "Deallocate Real Mem Buffer").
- Ponemos DX igual al selector que nos había retornado anteriormente.
- INT 31h (DPMI int).
- Si el flag C esta seteado entonces no pudo desalojar el buffer de memoria.

Hemos visto cómo se llama, pero lo que nos interesa principalmente es la información que nos ha dado la función, lo que se ha alojado en el buffer, este último presenta la siguiente estructura y que esta construida a partir del lenguaje de programación C++:

```
struct VBEInfoBlock
{
    BYTE VESAsignature[4]; // Debe contener VESA
    WORD VESAversion; // Versión de VESA
    DWORD OEM_ptr; // Puntero a una string de OEM
    BYTE Capacidad[4]; // Capacidades del vídeo actual
    DWORD Vmode_ptr; // Puntero a los modos SVGA soportados
    BYTE MemTotal; // Número de bloques (64k) de memoria en el video
    // Lo que se enumera a continuación aparece a partir de la
    // versión VESA 2.0 en adelante
    WORD OEM_Software_Rev_ptr; // Revisión de la implementación por
                               // software
    DWORD OEM_Vendor_Name_ptr; // Nombre del Vendedor
    DWORD OEM_Product_Name_ptr;// Nombre del Producto
    DWORD OEM_Product_Rev_ptr; // Revisión del Producto
    BYTE Reservado[222] ; // Reservado
};
```

Hay que tener en cuenta que en la versión VESA 2.0 el tamaño de esta estructura es de 512 bytes, o sea que si la queremos usar como tal (2.0), debemos en los pasos anteriores alojar un buffer de 512 bytes en vez de 256, y poner en VESAsignature, antes de llamar a la funíón 00h, 'VBE2', con esto le indicamos que es un buffer de 512.

De esta estructura, sólo vamos a encontrar utilidad en los siguientes campos que se enumeran a continuación:

- VESAversion: obtenemos con que versión estamos trabajando. El byte alto de esta palabra (word) es el número mayor de la version (1 en V1.2, 2 en V2.0), en el byte bajo esta el numero menor (2 en V1.2, 0 en V2.0).
- Vmode_ptr: este puntero apunta a los modos de vídeo soportados por la placa, los modos estan descriptos por una WORD que más adelente se explicarán y la lista termina con un -1 (0FFFFh).
- MemTotal: cuando nuestro programa esta preparado para un modo de vídeo especifico, ya sabemos de antemano cuanta memoria tiene que tener como mínimo la placa para que nuestro programa funcione, con esto podemos evitar el mensaje "Error: No hay suficiente memoria en la placa de vídeo para modo....". Este byte contiene el número de bloques de 64k que tiene la placa.

Ahora el siguiente paso depende del diseño de nuestro programa, si lo hemos realizado para que acepte varias resoluciones y/o colores, entonces nos tendremos que fijar en la lista de modos disponibles, y elegir el que más nos guste o decirle al usuario que elija uno, pero también pudimos hacerlo para una sola resolución, entonces ya sabemos que número corresponde a nuestro modo de video...

Para pedir la información de un modo seleccionado, se utiliza la función VESA 01h (Return VBE mode Information), de forma similar a la que utilizamos anteriormente, vemos las dos posibilidades que tenemos dependiendo del modo de trabajo:

- Modo Real:
  - Alojamos un buffer de 256 bytes.
  - Ponemos en AX 4F01h, (Elejimos función).
  - Ponemos en CX 101h (Número de Modo de Vídeo elegido).
  - Ponemos ES:DI apuntando al buffer que había alojado.
  - Llamada a la interrupción INT 10h.
  - Si AL!=4Fh entonces algo falló.
  - Si AH!=00h entonces esta placa no soporta el modo de vídeo seleccionado.

- Utilizamos la información del buffer si fuese necesario.
- Liberamos el buffer.

- DPMI:
  - Alojamos el buffer (ya hecho anteriormente).
  - Ahora tenemos que usar la siguiente estructura RMI (Real Mode Interrupt), primero la ponemos a 0 y luego, la pasamos al DPMI, el DPMI llenará los registros con el contenido de esta structa y con esos valores llamaremos a la int 10h:

```
DWORD EDI
DWORD ESI
DWORD EBP
DWORD Reservado
DWORD EBX
DWORD EDX
DWORD ECX = 101h (Modo de Vídeo deseado)
DWORD EAX = 4f01h (Función VESA 01h)
WORD flags;
WORD ES = El segmento retornado anteriormente por el
          DPMI host
WORD DS,FS,GS,IP,CS,SP,SS
```

  - AX=300h (función DPMI "Simulate Real Mode Interrupt").
  - BX=10h (le decimos al DPMI que interrupción queremos que llame).
  - CX=0.
  - Ponemos ES:EDI apuntando a RMI (ES=DS por supuesto).
  - INT 31h (DPMI int).
  - Lo que nos devuelve la int 10h se encuentra en RMI, es decir, si RMI.EAX & 0x000000FF != 4Fh entonces algo falló y si RMI.EAX & 0x0000FF00 != 00h entonces esta placa no soporta este modo de vídeo.
  - Utilizamos la información del buffer si fuese necesario.
  - Desalojamos el buffer (ya hecho anteriormente).

Hemos visto cómo se llama, pero lo que nos interesa principalmente es la información que nos ha dado la función, lo que se ha alojado en el buffer, este último presenta la siguiente estructura y que será construida a partir del lenguaje de programación C++:

```
struct VesaModeInfoBlock
{
    /* Obligatorios en toda VESA */
    WORD ModeAttributes ; Atributos del Modo
    BYTE WinAAttributes ; Atributos de la Win A
    BYTE WinBAttributes ; Atributos de la Win B
    WORD WinGranularity ; Granularidad de la Win
    WORD WinSize ; Tamaño de la Win
    WORD WinASegment ; Segmento de la Win A (Casi siempre A000)
    WORD WinBSegment ; Segmento de la Win B (Casi siempre A000)
    DWORD WinFuncPtr ; Puntero a la funcion de cambio de Win
    WORD BytesPerScanLine ; Bytes por Scanline
    /* Obligatorios desde VESA 1.2 */
    WORD XResolution ; Resolución horizontal en pixeles
    WORD YResolution ; Resolución vertical en pixeles
    BYTE XCharSize ; Ancho de la celda de carácter en pixeles
    BYTE YCharSize ; Alto de la celda de carácter en pixeles
    BYTE NumberOfPlanes ; Número de planos
    BYTE BitsPerPixel ; Bits por pixel
    BYTE NumberOfBanks ; Número de bankos
    BYTE MemoryModel ; Tipo de memoria
    BYTE BankSize ; Tamaño del banco en Kb
    BYTE NumberOfImagePages ; Número de páginas
    BYTE ReservedP ; Reservado
    /* Campos de Direct Color (no se van a utilizar) */
    BYTE RedMaskSize
    BYTE RedFieldPosition
    BYTE GreenMaskSize
    BYTE GreenFieldPosition
    BYTE BlueMaskSize
    BYTE BlueFieldPosition
    BYTE RsvdMaskSize
    BYTE RsvdFieldPosition
    BYTE DirectColorModeInfo
    /* Obligatorios desde VESA 2.0 */
    DWORD PhysBasePtr ; Dirección Física del LFB
    DWORD OffScreenMemOffset ; Puntero a la segunda página del LFB
    WORD OffScreenMemSize ; Tamaño de lo que queda de memoria
    BYTE Reserved[206] ; Reservado
};
```

De esta estructura, sólo vamos a encontrar utilidad en los siguientes campos que se enumeran a continuación:

- ModeAttributes: atributos del modo de vídeo, los bits más importantes son:
  - Bit 0 = Modo soportado por el hard (0==NO, 1==SI).
  - Bit 6 = Modo de Window's o Bank soportado (0==SI, 1==NO).
  - Bit 7 = Linear Frame Buffer disponible (0==NO, 1==SI).
- XResolution & YResolution: resolución horizontal y vertical en pixeles del modo elegido.
- BitsPerPixel: el número de bits por pixel, una buena resolución se consigue con 24 bits por pixel (modelo RGB).
- PhysBasePtr: direccion física del LFB, es la dirección física de memoria, luego tendremos que mapearla en nuestro espacio de memoria para usarla.

4.5. Modos de pantalla

Nos encontramos con dos posibilidades a la hora de trabajar en pantalla, que ésta este en modo texto ó en modo gráfico.

Pantalla en modo texto

Se trabaja en un entorno de texto (no gráfico), el programa en ejecución controla la información representada en la totalidad de la pantalla; el control de esta se realiza en término de filas y columnas (generalmente 25 filas x 80 ó 40 columnas suele ser el estándar) y un surtido muy limitado de 256 carácteres (código ASCII), como se puede observar la unidad de medida de este mod es el carácter.

Los únicos atributos que pueden tener los carácteres suelen ser: color de tinta y de papel (trazo y fondo); subrayado y parpadeo. Este atributo viene definido por un byte que permite definir el color de fondo de los carácteres (0-7) con los bits 4-6, el de la tinta (0-15) con los bits 0-3 y el parpadeo con el bit 7.

Pantalla en modo gráfico

Se trabaja en un entorno gráfico; la representación se realiza en pixeles, y se dispone de un amplísimo surtido de herramientas y parámetros de representación; no solo un amplio juego carácteres todos sus atributos, también un pincel, una pluma, así como iconos e imágenes preconstruidas de todo tipo.

El modo gráfico permite una mayor definición que varia de acuerdo al tipo de controlador gráfico que se este utilizando y su unidad de medida es el pixel. En la mayor parte de las pantallas de los ordenadores actuales, se autodetecta el modo gráfico.

Ni que decir tiene que la programación de un entorno gráfico es muchísimo más compleja que para un entorno de texto, aunque afortunadamente los entornos de desarrollo actuales ofrecen multitud de soluciones preconstruidas (clases) y librerías que facilitan la labor de un programador.

Capítulo 5

Realimentación

En este capítulo se abordarán todos los temas relacionados con el puerto paralelo del PC.

Primeramente se verá una breve introducción sobre los modos de transmisión de datos que podemos realizar con un PC a través de sus puertos.

A continuación, nos centraremos en el puerto paralelo, viendo sus principales características como son la conexión con el puerto, los pines que presenta para dicha conexión, los registros de datos, control y estado para trabajar con él y sus características de E/S.

Por último, se mencionarán las instrucciones en el lenguaje de programación C++ que nos permiten la lectura y escritura en el puerto.

5.1. Introducción

Los dispositivos o puertos de entrada/salida permiten realizar transferencias de información entre el exterior y el microprocesador. Existen dos modos de transferencia:

- Paralelo: el puerto utiliza un conjunto de líneas, tantas como bits a transmitir simultáneamente queramos, por las que en cada una pasa un bit en un intervalo de tiempo.
- Serie: el puerto utiliza una única línea por la que, en intervalos de tiempo diferentes, se transmiten, uno a uno, todos los bits del dato.

En la siguiente figura podemos ver un ejemplo de como son estas transmisiones en paralelo y en serie.

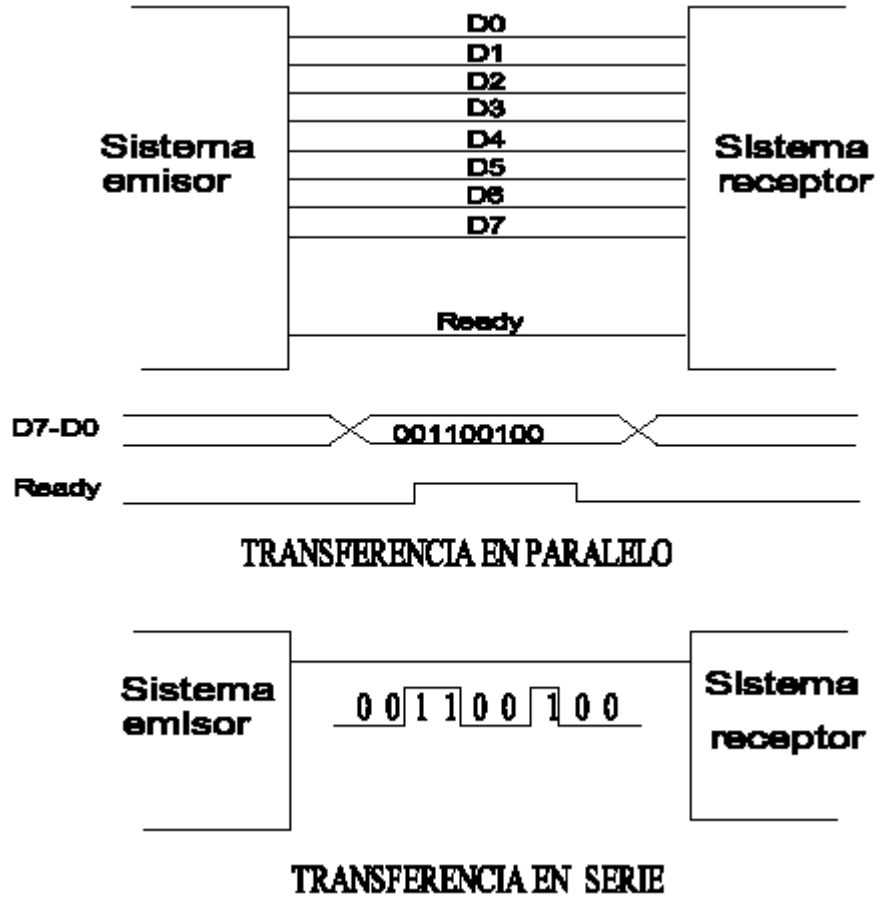

Figura 5.1: Ejemplo de transferencia de datos en serie y en paralelo

5.2. Conexión del puerto paralelo

Las comunicaciones en paralelo se realizan mediante la transferencia simultánea de todos los bits que constituyen el dato (byte o palabra). Presentan la ventaja de que la transmisión puede ser más rápida. Sin embargo, las comunicaciones en paralelo no pueden ser implementadas para grandes distancias debido a que no es viable la conexión física de todas las líneas necesarias.

Las comunicaciones en paralelo propiamente dichas no han sido normalizadas, lo que sí se reconoce es la norma Centronic para la conexión del PC a la impresora, mediante el envío simultáneo de 8 bits de datos (un byte), además de un conjunto de líneas de protocolo (handshake o intercambio). La operación más frecuente en la que interviene el puerto paralelo del PC es en el envío de datos a la impresora.

Los antiguos circuitos integrados que se incluían en las tarjetas de interface del puerto paralelo no permitían la recepción de datos, sólo estaban diseñados para el envío de información al exterior.

Las versiones recientes de estas tarjetas de interface de puertos paralelo sí permiten la recepción de datos y dan la posibilidad, por ejemplo, de intercambiar información entre PC a través del puerto paralelo, siempre que se utilice el software adecuado.

La norma Centronics hace referencia a las características de la conexión entre un interface de puerto paralelo y una impresora. Las líneas son latcheadas, esto es, mantienen siempre el último valor establecido en ellas mientras no se cambien expresamente y los niveles de tensión y de corriente coinciden con los niveles de la lógica TTL, cuyos valores típicos son:

- Tensión de nivel alto: 5 V.
- Tensión de nivel bajo: 0 v.
- Intensidad de salida máxima: 2.6 mA.
- Intensidad de entrada máxima: 24 mA.

La norma Centronics establece el nombre y las características de 36 líneas eléctricas para la conexión entre el PC y la impresora.

En realidad, para la transferencia de las señales de datos y de control a través de la tarjeta de interface paralelo sólo se requieren 18 líneas, las restantes son líneas de masa que se enrollan alrededor de los cables de señal para proporcionarles apantallamiento y protección contra interferencias. Por esto, las citadas tarjetas suelen incorporar un conector hembra DB-25, mientras que prácticamente todas las impresoras incorporan un conector hembra tipo Centronics macho de 36 pines.

Los cables comerciales para la conexión paralelo entre el PC y la impresora tienen una longitud de 2 metros, aunque no es recomendable que tengan una longitud superior a 5 metros si se desea una conexión fiable y sin interferencias.

En la Figura 5.2 se describen todas las líneas del estándar Centronics, con indicación de su denominación y el número de pin que le corresponde, tanto en el conector tipo Centronics de 36 pines como en el conector DB-25. En dicha figura se indica que las 8 líneas correspondientes a los bits de datos (D0 a D7) son líneas de salida, pues así lo establece el estándar Centronics, sin embargo y sobre todo en las implementaciones más recientes, la circuitería asociada al interface del puerto paralelo puede ser tal que las líneas de datos pueden ser leídas desde el PC y, por tanto, ser consideradas como líneas bidireccionales, aunque sea en determinadas condiciones y con el software adecuado.

El puerto paralelo en un PC

Todos los ordenadores tipo PC están equipados, al menos, con una tarjeta de interface paralelo, frecuentemente junto a un interface serie.

Como sistema operativo, el DOS puede gestionar hasta cuatro interfaces de puertos paralelo, LPT1, LPT2, LPT3 y LPT4, además, reserva las siglas PRN como sinónimo del LPT1, de modo que puede ser tratado como un archivo genérico. En el byte 0040:0011 del BIOS almacena el número de interfaces de puertos paralelo que se hayan instalado en el equipo.

N.° de pin		Señal	Sentido	Descripción
DB - 25	DB - 36		PC - PRN	
1	1	/STR	->	STROBE. Validación del dato (activa a nivel bajo). Un nivel L indica a la impresora que el dato es válido.
2	2	D0	->	Bit 0 de datos.
3	3	D1	->	Bit 1 de datos.
4	4	D2	->	Bit 2 de datos.
5	5	D3	->	Bit 3 de datos.
6	6	D4	->	Bit 4 de datos.
7	7	D5	->	Bit 5 de datos.
8	8	D6	->	Bit 6 de datos.
9	9	D7	->	Bit 7 de datos.
10	10	/ACK	<-	ACKNOWLEDGE. (Activa a nivel bajo). Un nivel L indica que la impresora está en disposición de recibir un nuevo dato.
11	11	BSY	<-	BUSY. Ocupada. Un nivel H indica que la impresora está ocupada y no puede recibir datos.
12	12	PAP	<-	PAPER END. Sin papel. Un nivel H indica que la impresora se ha quedado sin papel.
13	13	OFON	<-	ON LINE. Conectada. Un nivel H indica que la impresora está conectada y en línea.
14	14	/ALF	->	AUTO LINE FEED. Cambio de línea automático (Activa a nivel bajo). Un nivel L indica a la impresora que cuando reciba un retorno de carro debe hacer también un cambio de línea.
15	32	/ERR	<-	ERROR. (Activa a nivel bajo). Un nivel L indica que se ha producido un error en la impresora (buffer lleno, impresora fuera de línea, etc).
16	31	/INT	->	INITIALIZE PRINTER. Inicialización (Activa a nivel bajo). Un nivel L inicializa o provoca un reset en la impresora (si la impresora lo admite).
17	36	/DSL	->	SELECT. (Activa a nivel bajo). Un nivel L selecciona o pone on line la impresora (si la impresora lo admite).
18-25	19-30,33	Masa		Referencia de tensión para las señales.
--	16	0 v		--
--	17	Chasis		Conexión al chasis del equipo.
--	18	+ Vcc		Tensión de +5 v.
--	34,35	--		No utilizada.

Figura 5.2: Descripción de los pines del puerto paralelo.

La dirección de entrada/salida de cada uno de los puertos paralelo y el número de puertos instalados en un PC se muestra en la pantalla inicial de arranque del equipo es frecuente, casi estándar que las direcciones de los dos primeros puertos paralelo sean las siguientes:

- LPT1 = 0x378 Hexadecimal.
- LPT2 = 0x278 Hexadecimal.

Las tarjetas del puerto paralelo tiene una estructura muy simple; consta de tres registros: de control, de estado y de datos. Todas las señales que intervienen en el puerto tienen asociado un bit en uno de esos registros, de acuerdo con las funciones asignadas a cada línea en particular.

El registro de datos

Es de tipo latch de 8 bits, que puede ser leído y escrito desde el procesador. Es el registro donde el procesador, en operaciones de salida (OUT), pone el dato que se quiere enviar a la impresora y su dirección coincide con la dirección base del puerto paralelo (0x378 en LPT1). En la Figura 5.3 se muestra la distribución de los bits de este registro y los pines asociados a cada uno de ellos en el conector DB-25.

	REGISTRO DE DATOS (D)							
	7	6	5	4	3	2	1	0
	D7	D6	D5	D4	D3	D2	D1	D0
Pines →	9	8	7	6	5	4	3	2

Figura 5.3: Registro de datos del puerto paralelo en el PC y los pines correspondientes en el conector DB-25.

El registro de estado

El registro de estado indica la situación actual de la impresora conectada al puerto, de acuerdo con los niveles de tensión que tengan las líneas ACK, BSY, PAP y OF/ON, lo que permite controlar el comportamiento de la impresora.

Se trata de un registro de entrada (Lectura) de información, su dirección se obtiene sumando 1 a la dirección base del puerto (0x379 en LPT1).

	REGISTRO DE ESTADO (S)							
	7	6	5	4	3	2	1	0
	/BSY	/ACK	PAP	OFON	ERR	X	X	X
	(•)							
Pines →	11	10	12	13	15	4	3	2

Figura 5.4: Registro de estado del puerto paralelo en el PC y los pines correspondientes en el conector DB-25.

Los bits de este registro se designan según se muestran en la Figura 5.4, en la que el símbolo "/" delante del nombre del bit indica que es activo a nivel bajo. Pero el bit 7 además (/BSY) del registro de estado (bit 7) es invertido por el hardware y, por tanto, la línea tiene un nivel complementado al que aparece en ese bit. El significado que tienen los bits de este registro es el siguiente:

- Si el bit 7 (/BSY → Busy) está a 0, significa que la impresora está ocupada (buffer de impresión lleno, procesando información, pendiente de inicializar, etc.).
- El bit 6 (/ACK → Acknoledge) indica que se ha producido una transferencia correcta: cuando del puerto paralelo se transfiere un byte a la impresora, la impresora activa la línea ACK de reconocimiento del carácter y, como consecuencia, el bit ACK del registro de estado pasa a nivel bajo; cuando el bit ACK está a nivel alto, significa que la impresora está ocupada y no se pueden realizar envíos.
- El bit 5 (PAP → Paper) si está a 1, señala que la impresora no dispone de papel.
- El bit 4 (OF/ON → Line Off) indica cuando está a 1, que la impresora no está en línea.
- El bit 3 (ERR) si está a 0, indica que se ha producido un error de impresora (mal funcionamiento, falta de papel, impresora fuera de línea ...).
- Los bits 0,1 y 2 no se utilizan.

El registro de control

El registro de control permite controlar las transferencias de información con la impresora, y puede ser escrito y leído desde el microprocesador.

Es un registro de entrada/salida cuya dirección se obtiene sumando 2 a la dirección base del puerto (0x37A en LPT 1). Los bits de este registro se designan en la Figura 32.3, donde el símbolo «/» delante del nombre del bit indica que es activo a nivel bajo.

Figura 5.5: Registro de control del puerto paralelo en el PC y los pines correspondientes en el conector DB-25.

El símbolo (*) indica que los bits STR, ALF y OSL del registro de control son invertidos por el hardware con relación a las líneas correspondientes al cable de conexión, por lo que el nivel de los bits 0,1 y 3 del registro es complementado con relación a las líneas correspondientes.

El significado que tienen los bits de este registro es el siguiente:

- El bit 4 (IRQ): es el que permite controlar la generación de interrupciones de tipo hardware desde el puerto paralelo. Si este bit está a 1, el interface paralelo puede generar la petición de interrupción IRQ7 (en LPT1), que se corresponden con las interrupción 0x0Fh respectivamente del procesador 80X86. Esta petición de interrupción se produce cuando se da una transición H→L en la línea ACK.
- El bit 3 (DSL): La mayoría de las impresoras paralelo IBM-compabiles, no utilizan esta línea y son activadas con un pulsador de on-line.
- El bit 2 (INI): produce una inicialización de la impresora (es poco utilizado).
- El bit 1 (ALF): si está a nivel alto, la impresora produce automáticamente un cambio de línea (LF) cada vez que recibe un retorno de carro (CR).

- El bit 0 (STR): controla la línea que permite validar el dato existente en el registro de datos. La puesta a 1 del bit STR genera un impulso corto que indica a la impresora que el carácter del registro de datos es válido y debe ser aceptado. Así pues, cada vez que se precise enviar un carácter, no basta con ponerlo en el registro de datos, sino que hay que hacer un reset en el bit STR del registro de control y validar el dato volviendo a poner un 1 en ese bit.
- Los bits 5, 6 y 7 no se utilizan.

Entradas y salidas por el puerto paralelo

Al hablar de operaciones de entrada y salida por el puerto paralelo no debe olvidarse que, inicialmente, este elemento se desarrolló de acuerdo con el están dar Centronics con el fin, casi exclusivo, de que el PC pudiese enviar datos en paralelo a la impresora conectada, no se pensó en la posibilidad inversa: que el PC pudiese recibir datos a través de ese puerto.

Las operaciones de entrada y salida de información a través del puerto paralelo en el PC las realizaremos gestionando el puerto paralelo en el nivel de registros, es decir, programando directamente los circuitos integrados o chips que constituyen la tarjeta de interface, lo cual permitirá aprovechar al máximo todas las posibilidades que ofrezca realmente el hardware de la tarjeta de interface.

Características E/S

Cuando usamos el puerto paralelo para otro cometido distinto al original, solo podemos hablar de dos aspectos principalemente:

- 12 líneas de salida de información desde el ordenador:
  - pines del 2 al 9 → tregistro de datos.
  - pines 1, 14, 16 y 17 → registro de control.
- 15 líneas de entrada al mismo:
  - pines del 2 al 9 → registro de datos.
  - pines 10, 11, 12, 13 y 15 → registro de estado.

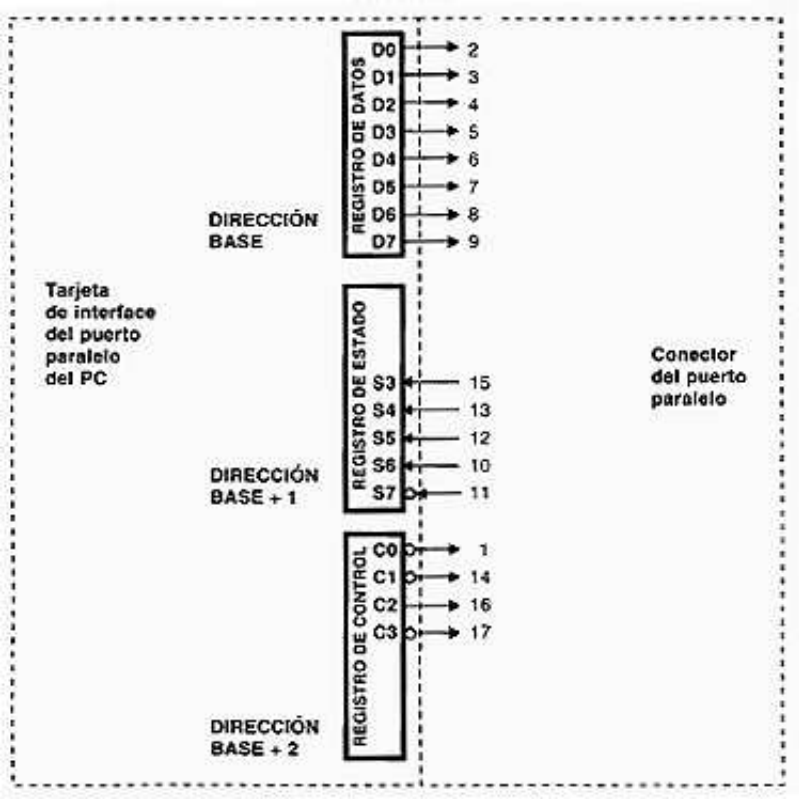

Figura 5.6: Líneas del conector DB-25 del puerto paralelo.

Esto hace del puerto paralelo un interface de comunicación con el exterior bastante flexible.

El registro de estado es de sólo lectura. Cuando se lee este registro, lo que se recibe es el estado lógico de los pines 10, 11, 12, 13 y 15 del conector DB-25 (el bit S7 contiene el complemento del estado de la línea). Los tres bits de menor peso (SO-S2) no se utilizan y, habitualmente, se encuentran a nivel alto.

El registro de control es parcialmente bidireccional. Cuando se escribe en los cuatro bits de menor peso (C0-C3) lo que se hace es establecer el nivel lógico de los pines C2 de forma directa y C0, C1 y C3 de forma complementada. Los tres bits de mayor peso (C5-C7) no se utilizan.

De forma experimental, se ha podido comprobar que, sólo en algunas tarjetas de interface paralelo, el bit C6 del registro de control influye en la configuración, de modo que si C6=0 las I líneas de datos se configuran como ENTRADAS y si C6=1 las líneas de datos se configuran como SALIDAS. Otras tarjetas, sobre todo si son bidireccionales, no permiten el cambio de nivel de ese bit.

El registro de datos es de tipo latch de lectura y de escritura, de modo que cuando se realiza una operación de escritura (OUT) el dato se carga en los bits correspondientes y las líneas asociadas del conector tienden a alcanzar la tensión correspondiente a ese estado.

En algunas ocasiones las líneas de datos de la tarjeta de interface paralelo (Centronics) son bidireccionales, pero la etapa de salida se ha construido mediante buffers con transistores en colector abierto. En este caso, el hecho de que las operaciones de entrada y salida se hagan por las mismas líneas, condiciona notablemente el proceso de lectura, ya que con esa configuración electrónica de las líneas de datos (D0-D7), los valores lógicos leídos dependerán del nivel lógico presente en el registro y del valor de tensión en la línea (que no tienen por qué coincidir) de acuerdo con lo mostrado en la siguiente tabla.

Nivel lógico de los bits en el registro de datos	Nivel de tensión en los pines del conector	Nivel lógico leído en una operación de lectura del puerto paralelo
L	L	L
L	H	L
H	L	L
H	H	H

A la vista de la tabla anterior, lo que se deduce es que, si se va a realizar una operación de lectura sobre el puerto paralelo, lo que se va a leer realmente es la operación AND lógica entre el nivel lógico del registro y el nivel lógico de la línea, lo que implica que, si se desea realizar una lectura real del estado de las líneas, deberá escribirse antes el dato 0xFF en el registro de datos del puerto paralelo.

5.3. Programación con el puerto paralelo

Las instrucciones en C++ que dan acceso a los puertos, tanto para lectura como para escritura, están en la librería conio.h y son:

- para escribir:

```
outp (puerto,dato)
```

- para leer:

```
inp (puerto)
```

Vemos unos ejemplos del empleo de las instrucciones de lectura y escritura en el puerto paralelo:

```
outp (0x3780,1); // Escribiría un "1" en la direccion
                 // 0x378 = puerto paralelo

int valor;
valor = inp (0x3780) // Leería el puerto paralelo y metería
                     // el resultado en la variable valor
```

Parte II

Software desarrollado

Capítulo 6

Vídeo

En este capítulo veremos el software desarrollado en el campo que tiene que ver con la captura de vídeo.

Se desarrollarán los archivos de la biblioteca creada que gestionan la captura de vídeo.

A continuación, se explicará el contenido de los archivos anteriormente mencionados, que constarán de la definición de la clase empleada para tal efecto, con su conjunto de variables y funciones realizadas.

Y por último, se comentarán las funciones realizadas que nos permiten trabajar con la capturadora de vídeo, así tendremos funciones para la inicialización, capturay parada del vídeo.

6.1. Archivos

Se han incluido a la biblioteca de funciones generada una serie de archivos para la captura de vídeo. Para la creación de estos archivos, se parte de un conjunto de archivos ya creados para trabajar con la capturadora de vídeo que disponemos y se modifican algunos de ellos para adaptarlos a nuestras necesidades. Estos archivos ya existentes se describen a continuación:

- djgpp.cpp: archivo del compilador para trabajar con registros.
- compiler.h: define la clase VIRTUAL_MEM que será utilizada por otros archivos.
- devlist.cpp: para exploración y búsqueda de los dispositivos conectados al bus PCI. Trabaja con la clase DEVICE_LIST definida en el archivo devlist.h.
- dma.cpp: para gestión del buffer del DMA y que ha sido modificado para albergar el buffer de dicho DMA a partir de la dirección física 63M y 64M en memoria. Define la clase DMA_BUFFER en el archivo dma.h.
- gpio.cpp y gpio_848.cpp: para gestión del puerto de próposito general de entrada/salida que presenta el chip. Trabaja con las clases GPIO y GPIO_848 definidas en los archivos gpio.h y gpio_848.h, respectivamente.
- i2c.cpp e i2c_878.cpp: para la interfaz I^2C. Trabaja con las clases I2C e I2C_878 definidas en los archivos i2c.h e i2c_878.h, respectivamente.
- pci.cpp: para implementación del bus del PCI. Trabaja con la clase PCI definida en el archivo pci.h.
- pcidecod.cpp y pcivideo.cpp: para configuración de los registros asociados al bus PCI. Trabaja con las clases PCI_DECODER y PCI_VIDEO definidas en los archivos pcidecod.h y pcivideo.h, respectivamente.
- vidfield.cpp: para la configuración del campo de vídeo. Define la clase VIDEO_FIELD en el archivo vidfield.h.
- vef_848.cpp y vof_848.cpp: para trabajar con el campo par e impar del vídeo. Trabaja con las clases VIDEO_EVEN_FIELD_848 y VIDEO_ODD_FIELD_848 definidas en los archivos vef_848.h y vof_848.h, respectivamente.

- vidconst.h: contiene la definición de constantes para el vídeo, cuando queremos ejecutar instrucciones RISC.
- scaler.cpp y scaler.h: para la obtención y escalado de la imagen. Este es el otro archivo que ha sido modificado.

Además de los archivos vistos anteriormente, se han creado dos nuevos, que lógicamente han sido incluidos en la biblioteca, estos archivos son:

- video.h: contiene la definición de la clase tVideo creada para tal efecto.
- video.cpp: contiene la implementación de las funciones definidas en la clase tVideo.

6.2. Clase tVideo

Se define la clase tVideo para la captura de vídeo, la cual esta formada por un conjunto de variables para la caracterización del vídeo con el que se esta trabajando, y por una serie de funciones que arrancan y paran el vídeo, por un lado y, por otro lado, por funciones que capturan el vídeo, bien sea en la pantalla ó bien sea en un bloque de datos ó imagen, estas últimas es un claro ejemplo de sobrecarga de funciones. Las variables que se definen se enumeran a continuación:

- Direccion: dirección del vídeo.
- Ancho: resolución horizontal del vídeo en pixeles.
- Alto: resolución vertical del vídeo en pixeles.

6.3. Funciones

6.3.1. Inicializa vídeo

Descripción

Inicializa todos los parámetros del chip BT878 para la captura de vídeo.

Sintaxis

```
Video.InicializaVideo()
```

donde Video es de clase tVideo.

Parámetros de entrada

No precisa ningún parámetro de llamada.

Errores de salida

No presenta ningún tipo de error de salida.

Ejemplo

```
Video.InicializaVideo(); // Inicializaría el vídeo
```

6.3.2. Captura de vídeo en pantalla

Descripción

Captura de vídeo en la pantalla a partir de la posición (x,y) con una determinada Anchura y Altura, estas dos últimas están limitadas por la resolución máxima de la capturadora de vídeo utilizada. Se permiten dos modos de captura, modo continuo (ModoCaptura=1) y modo de captura de una sola imagen (ModoCaptura=0).

Sintaxis

```
Video.Captura( tPantalla Pantalla, int x, int y, int Anchura, int
               Altura, int ModoCaptura )
```

donde:

- Video: es de clase tVideo.
- Pantalla: es de clase tPantalla.
- x, y, Anchura, Altura y ModoCaptura: son variables enteras.

Parámetros de entrada

Necesitamos seis parámetros de entrada como son, Pantalla que indica el tipo de pantalla con el que trabajamos, las variables enteras x e y que indican a partir de la posición en pantalla donde queremos dibujar y las variables enteras Anchura y Altura que delimitan el tamaño de la imagen a capturar. Y por último la variable entera ModoCaptura que indica el modo de captura que queremos realizar.

Errores de salida

Nos encontramos con dos posibilidades de error de introducción de parámetros, cuando el ModoCaptura es distinto de 0 ó 1 y, cuando x e y son mayores que la anchura y la altura, respectivamente, del tamaño de pantalla con el que estemos trabajando ó sean menores de 0.

Ejemplo

```
V1.Captura(P,100,50,300,200,1); // Capturaría de modo continuo el
// vídeo con un tamaño de 300 por 200 y lo mostraría en la pantalla
// P a partir de la posición (100,50).
```

6.3.3. Captura de vídeo en imagen

Descripción

Captura de vídeo en la imagen I con una determinado Ancho y Alto, definidos en la imagen I, estas dos variables estan limitadas por la resolución máxima de la capturadora de vídeo utilizada. Se permiten dos modos de captura, uno continuo (ModoCaptura=1) y otro de captura de una sola imagen (ModoCaptura=0).

Sintaxis

```
Video.Captura( tImagen Imagen, int ModoCaptura, tPantalla Pantalla )
```

donde:

- Video: es de clase tVideo.
- Imagen: es de clase tImagen.
- Pantalla: es de clase tPantalla.
- ModoCaptura: es una variable entera.

Parámetros de entrada

Necesitamos tres parámetros de llamada como son, Imagen que albergará la imagen que queremos capturar, ModoCaptura que indica el modo de captura que queremos realizar y por último Pantalla para poder realizar la captura.

Errores de salida

Nos encontramos con tres posibilidades de error de introducción de parámetros, cuando el ModoCaptura es distinto de 0 ó 1 y, cuando el Ancho y el Alto de la Imagen a capturar supera la resolución máxima de la capturadora de vídeo con la que estamos trabajando. Y por último, cuando no hay memoria suficiente de pantalla para albergar la imagen que se quiere capturar, como se explica a continuación.

Nota

Se pasa a la funcion el parametro pantalla P, porque se captura el vídeo en la memoria de pantalla no visible, siempre que haya suficiente espacio, y de ahí se copia en nuestra imagen.

Ejemplo

```
tImagen I1(300,200); // Definición de imagen I1
V1.Captura(I1,1,P);  // Capturaría de modo continuo el vídeo con un
                     // tamaño de 300 por 200 (indicado por las
                     // variables Ancho y Alto en I1) en la imagen
                     // I1
```

6.3.4. Para vídeo

Descripción

Para la captura de vídeo.

Sintaxis

```
Video.Paralo()
```

donde Video es de clase tVideo.

Parámetros de entrada

No precisa ningún parámetro de llamada.

Errores de salida

No presenta ningún tipo de error de salida.

Ejemplo

```
Video.Paralo(); // Pararía la captura de vídeo
```

Capítulo 7

Imagen

En este capítulo veremos el software desarrollado en el campo que tiene que ver con el tratamiento de imágenes.

Se desarrollarán los archivos de la biblioteca creada que gestionan el tratamiento de imágenes.

A continuación, se explicará el contenido de los archivos anteriormente mencionados, que constarán de la definición de la clase empleada para tal efecto, con su conjunto de variables y funciones realizadas.

Y por último, se comentarán las funciones realizadas que nos permiten trabajar con una o varias imágenes, así tendremos funciones que trabajan con la pantalla, como dibuja ó captura de pantalla, funciones que trabajan con una sola imagen, como funciones que realizan operaciones lógicas, convoluciones, transformaciones afines, transformada de Fourier,.... y, finalmente funciones que trabajan con varias imágenes como suma ó resta de imágenes.

7.1. Archivos

Se han incluido a la biblioteca de funciones generada dos archivos para el tratamiento de las imágenes, estos archivos son:

- imagen.h: contiene la definición de la clase tImagen creada para tal efecto.
- imagen.cpp: contiene la implementación de las funciones definidas en la clase tImagen.

7.2. Clase tImagen

Se define la clase tImagen para el tratamiento de imágenes, esta formada por un conjunto de variables para la caracterización de la imagen con la que se esta trabajando, por el constructor y destructor correspondiente, y por una serie de funciones que realizarán todo tipos de operaciones, para una sóla imagen ó para un par de imágenes, como veremos en la sección 7.3. Las variables que se definen se describen a continuación:

- *M: puntero que apunta al incio de la imagen o bloque de datos.
- Direccion: dirección de comienzo de la imagen es el selector DS.
- Ancho: ancho de la imagen en pixeles.
- Alto: alto de la imagen en pixeles.
- TotalBytes: total de bytes que presenta una imagen y será igual a Ancho x Alto x 4, siendo este último el número de bytes por pixel.
- DirecciónFísica: contendrá la dirección física de comienzo de la pantalla para cuando estemos trabajando con el vídeo.

7.3. Funciones

Se van a describir todas las funciones desarrolladas en la biblioteca generada, se verá su sintaxis, sus parámetros de entrada, sus errores de salida, su desarrollo teórico si corresponde y se pondrá un ejemplo de llamada y gráfico cuando corresponda.

Las imágenes originales, que aparecerán utilizadas a lo largo de los ejemplos que presentarán las funciones, y que presenta un tamaño ambas de 200 x 300 pixeles de resolución, son:

Figura 7.1: Ejemplo de imagen 1.

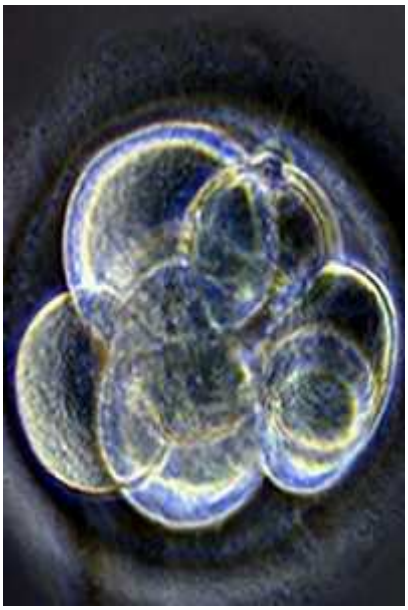

Figura 7.2: Ejemplo de imagen 2.

7.3.1. Carga BMP

Descripción

La función realiza la carga de un archivo BMP en un bloque de memoria o Imagen.

Sintaxis

```
Imagen.CargaBMP( char *Nombre_Fichero )
```

donde:

- Imagen: es de clase tImagen.
- *Nombre_Fichero: es un puntero que apunta a la dirección donde esta contenido el nombre del fichero.

Parámetros de entrada

Necesitamos un único parámetro de entrada, como es Nombre_Fichero que indica el nombre del fichero que queremos leer o cargar.

Errores de salida

Nos encontramos dos posibilidades de error, cuando se produce un error de apertura del fichero para su lectura y cuando el archivo que se quiere cargar no es un archivo BMP.

Ejemplo

```
I1.CargaBMP("Dibujo.BMP"); // Cargaría en la Imagen I1 el contenido
                           // del archivo Dibujo.BMP
```

7.3.2. Salva BMP

Descripción

La función guarda una Imagen o un bloque de datos en un archivo BMP.

Sintaxis

```
Imagen.SalvaBMP( char *Nombre_Fichero )
```

donde:

- Imagen: es de clase tImagen.
- *Nombre_Fichero: es un puntero que apunta a la dirección donde esta contenido el nombre del fichero.

Parámetros de entrada

Necesitamos un único parámetro de entrada, como es Nombre_Fichero que indica el nombre del fichero que le queremos dar al archivo que contendrá la imagen guardada.

Errores de salida

Nos encontramos con una única posiblidad de error de apertura ó de creación del fichero que queremos crear para escritura.

Ejemplo

```
I1.SalvaBMP("Dibujo.BMP"); // Guardaría la Imagen I1 en un archivo
                           // BMP llamado Dibujo.BMP
```

7.3.3. Dibuja en pantalla

Descripción

Dada una Imagen la dibuja en un tipo de Pantalla a partir de la posición (x, y) indicada y pasada como parámetros.

Sintaxis

```
Imagen.Dibujalo ( int Coordenada\_X, int Coordenada\_Y, tPantalla
                  Pantalla )
```

donde:

- Imagen: es de clase tImagen.
- Coordenada_X y Coordenada_Y: son variables enteras.
- Pantalla: de clase tPantalla.

Parámetros de entrada

Necesitamos tres parámetros de llamada como son, las variables enteras X e Y, que indicarán a partir de la posición en pantalla donde queremos comenzar a pintar la imagen y la variable Pantalla de clase tPantalla que indicará el tipo de pantalla que tenemos y donde queremos pintar.

Errores de salida

Nos encontramos con varias posibilidades de errores de introducción de parámetros, cuando las coordenadas x e y son menores de 0 o cuando éstas más el tamaño de la imagen a dibujar superan el tamaño del tipo de pantalla con el que estamos trabajando.

Ejemplo

```
I1.Dibujalo (100, 200, P1); // Dibujaría en la pantalla P1,
// suponiendo que esta sea de un tamaño de 800 por 600 pixeles de
// resolución la imagen I1 a partir de la posición en pantalla
// (100,200)
```

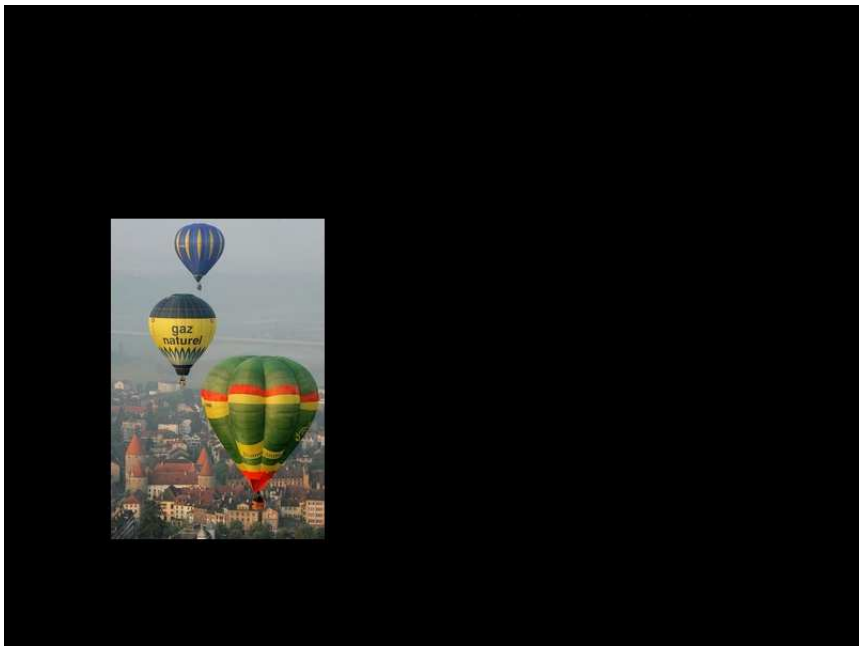

Figura 7.3: Ejemplo de imagen 1 pintada en pantalla.

7.3.4. Captura de pantalla

Descripción

Captura de un tipo de Pantalla una Imagen a partir de la posición (x, y) con una determinada Altura y Anchura.

Sintaxis

```
Imagen.Capturalo( int X, int Y, int Ancho, int Alto, tPantalla
                  Pantalla )
```

donde:

- Imagen: es de clase tImagen.
- X, Y, Ancho y Alto: son variables enteras.
- Pantalla: es de clase tPantalla.

Parámetros de entrada

Necesitamos cinco parámetros de llamada como son, el tipo de Pantalla con el que estamos trabajando, las variables enteras x e y que indicarán la posición en pantalla a partir de la que queremos capturar y, por último, las variables enteras Ancho y Alto que indicarán el tamaño de la imagen resultante de la captura.

Errores de salida

Nos encontramos con varias posibilidades de errores de introducción de parámetros, cuando las coordenadas x e y son menores de 0 o cuando éstas más el tamaño de la imagen a capturar excede del tamaño del tipo de pantalla con el que estamos trabajando.

Ejemplo

```
I1.Capturalo(50,150,200,200,P); // Capturaría de la pantalla P,
// suponiendo que ésta presenta la forma vista en la Figura 7.3,
// una imagen de tamaño 200 por 200 pixeles a partir de la posición
// (50,150)
```

Figura 7.4: Ejemplo de imagen capturada de pantalla.

7.3.5. Copia

Descripción

Dada una Imagen Origen y pasada como parámetro, la copia en otra Imagen Destino.

Sintaxis

```
Imagen.Copia ( tImagen Origen )
```

siendo Imagen y Origen de tipo tImagen.

Parámetros de entrada

Necesitamos un único parámetro de llamada Origen de tipo tImagen, que será la imagen origen que se copie en nuestra imagen destino.

Errores de salida

No presenta ningún tipo de error.

Ejemplo

```
I1.Copia(I2); // Copiaría la imagen I2 en la imagen I1
```

7.3.6. Suma

Descripción

La función realiza la suma de dos imágenes de igual tamaño.

Sintaxis

```
Imagen.Suma ( tImagen Imagen2 )
```

siendo Imagen e Imagen2 de tipo tImagen.

Parámetros de entrada

Necesitamos un único parámetro de llamada Imagen2 de tipo tImagen, que será la imagen que se sume con la imagen original.

Errores de salida

Nos encontramos con una única posibilidad de error de introducción de parámetros que se dará cuando el tamaño de las dos imágenes a sumar sea diferente, tanto para el ancho como para el alto de cada una.

Ejemplo

```
I1.Suma(I2); // Sumaría I1 e I2 en I1, siempre y cuando I1 e I2
             // sean del mismo tamaño, como es nuestro caso
```

Figura 7.5: Ejemplo de suma de las imágenes del ejemplo.

7.3.7. Resta

Descripción

La función realiza la resta de dos imágenes de igual tamaño.

Sintaxis

```
Imagen.Resta ( tImagen Imagen2 )
```

siendo Imagen e Imagen2 de tipo tImagen.

Parámetros de entrada

Necesitamos un único parámetro de llamada Imagen2 de tipo tImagen, que será la imagen que se reste con nuestra imagen original.

Errores de salida

Nos encontramos con una única posibilidad de error de introducción de parámetros que se dará cuando el tamaño de las dos imágenes a restar sea diferente, tanto para el ancho como para el alto de cada una.

Ejemplo

```
I1.Resta(I2); // Restaría I2 de I1 en I1, siempre y cuando I1 e I2
              // sean del mismo tamaño, como es nuestro caso
```

Figura 7.6: Ejemplo de resta de las imágenes del ejemplo.

7.3.8. Convolución entera

Descripción

Se realiza la convolución de una Imagen con una matriz de convolución de tamaño N x N, siendo N un número entero natural igual o mayor que tres, es decir, n=3,5,7,... Con la convolución con lo que se consigue es un filtrado de la imagen a través de la matriz de convolución considerada, así se podrá hacer un suavizado de ruido (filtro paso bajo), un scroll a la derecha, un filtro pasa alta (halla contornos), un filtrado Gaussiano tipo

(filtro paso bajo), un filtro Laplaciano de tipo kernel (filtro paso alto),... dependiendo de los valores de los coeficientes que tome dicha matriz de convolución.

Sintaxis

```
Imagen.Convolución ( int *M, int N )
```

donde:

- Imagen: es de clase tImagen.
- *M: es un puntero a M que apunta a la matriz de convolución de coeficientes enteros.
- N: es una variable entera.

Parámetros de entrada

Necesitamos dos parámetros de entrada, *M que apunta a la matriz de convolución deseada y N que indica el tamaño de la matriz de convolución cuyos coeficientes son números enteros.

Errores de salida

Nos encontramos con una única posibilidad de error de introducción de parámetros, cuando el tamaño de la matriz de convolución N es menor que tres.

Desarrollo teórico

La matriz de convolución tiene el formato de N x N de tamaño doble palabra (tamaño DWORD).

$$\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix}$$

Cada coeficiente a, b, c, d, e, f, g, h e i es una doble palabra (DWORD) con signo y cumple que en valor absoluto es menor que 256. Se accederá a cada pixel (i, j) de la Imagen, exceptuando los contornos (columnas/filas primera y última), evaluando para cada componente de color la siguiente operación matemática:

$$CompR(i,j) = (a*CompR(i-1,j-1)+b*CompR(i,j-1)+c*CompR(i+1,j-1)+d*$$

$$*CompR(i-1,j)+e*CompR(i,j)+f*CompR(i+1,j)+g*CompR(i-1,j+1)+h*$$

$$*CompR(i,j+1)+i*CompR(i+1,j+1)$$

Si la nueva componente es menor que cero, hay que saturar a cero, ya que una intensidad de color negativa no es real. Y lo mismo si se pasa por arriba, saturando a 255. Idem con CompG y CompB.

Ejemplo

```
int Matriz [3][3]={1,1,1,1,-7,1,1,1,1}; // Definición de la matriz
                                        // de convolución de valores enteros
I1.Convolucion(*Matriz, 3); // Realizará la convolución de la Imagen
// I1 con la matriz de convolución Matriz, con ello se consigue el
// realzado de los bordes que presente la imagen
```

Figura 7.7: Ejemplo de convolución entera de la imagen 1 (Tamaño 198 x 298 pixeles).

7.3.9. Convolución real

Descripción

Se realiza la convolución de una Imagen con una matriz de convolución de tamaño N x N, siendo N un número real. Con la convolución con lo que se consigue es un filtrado de la imagen a través de la matriz de convolución considerada, así se podrá hacer un suavizado de ruido (filtro paso bajo), un scroll a la derecha, un filtro pasa alta (halla contornos), un filtrado Gaussiano tipo (filtro paso bajo), un filtro Laplaciano de tipo kernel (filtro paso alto),... dependiendo de los valores de los coeficientes que tome dicha matriz de convolución.

Sintaxis

```
Imagen.Convolución ( double *M, int N )
```

donde:

- Imagen: es de clase tImagen.
- *M: es un puntero a M que apunta a la matriz de convolución de coeficientes reales.
- N: es una variable entera.

Parámetros de entrada

Necesitamos dos parámetros de entrada, *M que apunta a la matriz de convolución deseada y N que indica el tamaño de la matriz de convolución cuyos coeficientes son números reales.

Errores de salida

Nos encontramos con una única posibilidad de error de introducción de parámetros, cuando el tamaño de la matriz de convolución N es menor que tres.

Desarrollo teórico

Mismo desarrollo que en el apartado 7.3.8 pero teniendo en cuenta que los coeficientes de la matriz de convolución son números reales, en vez de enteros.

Ejemplo

```
double Matriz [3][3]={1.0,1.0,1.0,1.0,-7.0,1.0,1.0,1.0,1.0};
// Definición de la matriz de convolución de valores reales

I1.Convolucion(*Matriz, 3); // Realizará la convolución de la Imagen
// I1 con la matriz de convolución Matriz. El resultado de la imagen
// 1 convolucionada se puede ver en la Figura 7.7
```

Se observaría que se obtiene el mismo resultado aplicando, tanto la convolución real como la entera, a la Imagen 7.1, debido a que las matrices de convolución presentan los mismo valores.

7.3.10. Rotación

Descripción

Se realiza la rotación de una Imagen con un àngulo de rotaciòn, pasado como paràmetro de entrada, cuyo valor esta comprendido entre los 0 y 360 grados.

Sintaxis

```
Imagen.Rotacion( double Angulo )
```

donde:

- Imagen: es de clase tImagen.
- Angulo: es una variable real.

Parámetros de entrada

Necesitamos un único parámetro de llamada Angulo, que es un número real comprendido entre 0 y 360, que hace referencia al ángulo con el que queremos rotar nuestra imagen.

Errores de salida

Nos encontramos con una única posibilidad de error cuando el Angulo pasado es menor de 0 o mayor o igual de 360.

Ejemplo

```
I1.Rotacion(223.64); // Realizaría la rotación de la Imagen I1
                     // con un ángulo de 223.64º
```

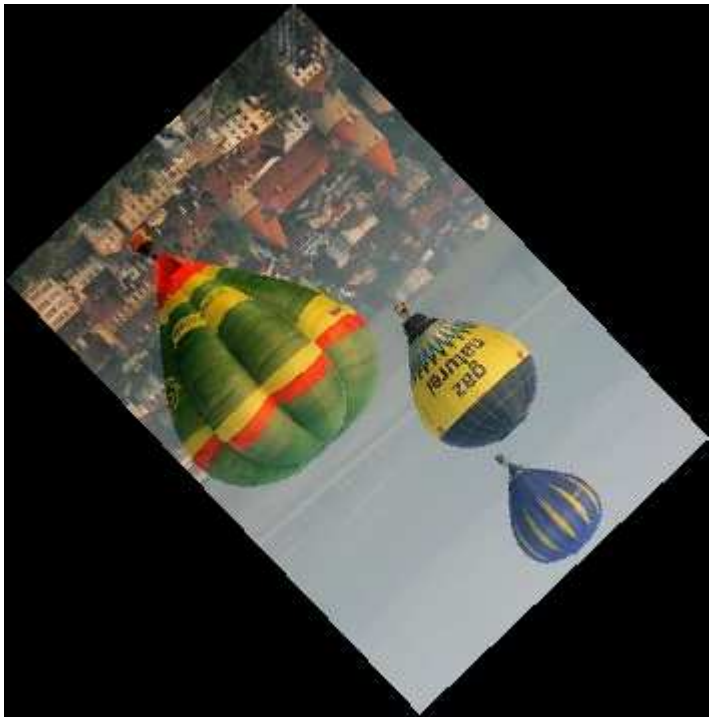

Figura 7.8: Ejemplo de imagen 1 rotada con un ángulo de 223.64º (Tamaño 352 x 356 pixeles).

7.3.11. Traslación

Descripción

Se realiza la traslación de una Imagen de tamaño Ancho x Alto a un nuevo punto (x,y), obteniéndose una nueva Imagen de tamaño (Ancho+X) x (Alto+Y).

Sintaxis

```
Imagen.Traslada( int X, int Y )
```

donde:

- Imagen: es de clase tImagen.
- X e Y: son variables enteras.

Parámetros de entrada

Necesitamos dos parámetros de llamada x e y, que nos indican las nuevas coordenadas donde queremos trasladar nuestra imagen original.

Errores de salida

Nos encontramos con dos posibles errores de introducción de parámetros cuando el valor de las coordenadas x e y es menor de 0.

Ejemplo

```
I1.Traslada(50,70); // Trasladaría la imagen I1 a una nueva
// posición indicada por las coordenadas, esta nueva posición
// sería la (50,70)
```

Figura 7.9: Ejemplo de imagen 1 trasladada (Tamaño 250 x 370 pixeles).

7.3.12. Escalado

Descripción

La función realiza un escalado de una Imagen a través de un factor de escala pasado como parámetro, asi si el factor de esclado es menor de la unidad haremos la imagen más pequeña, y en caso contrario haremos la imagen más grande. Es decir, dada una Imagen origen de tamaño Ancho x Alto, obtendremos una Imagen destino de tamaño (Ancho*Factor) x (Alto*Factor).

Sintaxis

```
Imagen.Escala ( double Factor )
```

donde:

- Imagen: es de clase tImagen.
- Factor: es una variable real.

Parámetros de entrada

Necesitamos un único parámetro de entrada Factor, que será un número real y por el que se escale nuestra imagen.

Errores de salida

Nos encontramos con una única posibilidad de error de introducción de parámetros cuando el factor de escala es igual o menor que 0.

Ejemplo

```
I1.Escala(0.68); // Reduciría la imagen I1 a un tamaño de
                 // 136 por 204 pixeles.
```

Figura 7.10: Ejemplo de imagen 1 escalada con un factor igual a 0.68 (Tamaño 136x204 pixeles).

7.3.13. Transformación Afín

Descripción

La función realiza la Transformacion Afín de una Imagen, donde se pasa una Matriz de tamaño 3 x 3 que presenta la siguiente forma:

$$\begin{pmatrix} S_x * cos & -sen & X_o \\ sen & S_y * cos & Y_o \\ 0 & 0 & 1 \end{pmatrix}$$

donde:

- X_o e Y_o indican el desplazamiento respecto del origen.
- S_x y S_y los factores de escala en los ejes x e y respectivamente.

Sintaxis

```
Imagen.TransfAfin ( double Matriz [3][3] )
```

donde:

- Imagen: es de clase tImagen.
- Matriz: es una matriz de tamaño 3 x 3 de variables reales.

Parámetros de entrada

Necesitamos un único parámetro de entrada Matriz, que será una matriz de tamaño 3 x 3 de números reales de la forma expuesta anteriormente.

Errores de salida

Nos encontramos con varias posibilidades de error de introducción de parámetros que se enumeran a continuación:

- Cuando el elemento de la matriz (0,1) que contiene el seno, no es igual al elemento (1,0) que contiene el negativo del seno.
- Cuando el elemento de la matriz (0,0) es nulo, ya que nos indicaría que el factor de escala correspondiente al eje x (S_x) sería nulo.

- Cuando el elemento de la matriz (1,1) es nulo, ya que nos indicaría que el factor de escala correspondiente al eje y (S_y) sería nulo.
- Cuando el elemento de la matriz (2,0) no es 0.
- Cuando el elemento de la matriz (2,1) no es 0.
- Cuando el elemento de la matriz (2,2) no es 1.

Ejemplo

```
double Matriz [3][3]={0.5*0.866,0.5,0,-0.5,1.5*0.866,0,0,0,1};
                        // Definición de la Matriz
I1.TransfAfin(Matriz); // Realizaría la transformación afín de la
// imagen I1 a través de Matriz, el resultado sería una nueva imagen
// de tamaño la mitad respecto al eje x y el 50% mayor respecto del
// eje y, rotada 30°
```

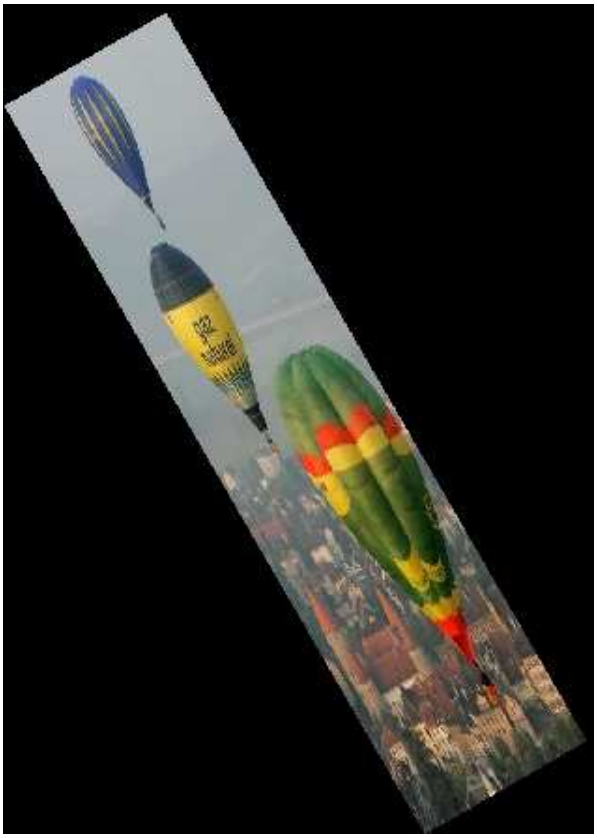

Figura 7.11: Ejemplo de la aplicación de una transformación afín a la imagen 1 (Tamaño 316 x 444 pixeles).

7.3.14. Función lógica OR

Descripción

La función realiza la operación lógica OR entre dos imágenes.

Sintaxis

```
Imagen.OR( tImagen Imagen2 )
```

siendo Imagen e Imagen2 de clase tImagen.

Parámetros de entrada

Necesitamos un único parámetro de llamada Imagen2 de tipo tImagen, que será la imagen con la que se realiza la función lógica OR.

Errores de salida

Nos encontramos con una única posibilidad de error de introducción de parámetros que se dará cuando el tamaño de las dos imágenes que intervienen en la operación sea diferente, tanto para el ancho como para el alto de cada una.

Ejemplo

```
I1.OR(I2); // Realizaría la función lógica OR entre las dos
           // imágenes, es decir, I1 = I1 OR I2
```

Figura 7.12: Ejemplo de función lógica OR entre las imágenes 1 y 2.

7.3.15. Función lógica XOR

Descripción

La función realiza la operación lógica XOR entre dos imágenes.

Sintaxis

```
Imagen.XOR( tImagen Imagen2 )
```

siendo Imagen e Imagen2 de clase tImagen.

Parámetros de entrada

Necesitamos un único parámetro de llamada Imagen2 de tipo tImagen, que será la imagen con la que se realiza la función lógica XOR.

Errores de salida

Nos encontramos con una única posibilidad de error de introducción de parámetros que se dará cuando el tamaño de las dos imágenes que intervienen en la operación sea diferente, tanto para el ancho como para el alto de cada una.

Ejemplo

```
I1.XOR(I2); // Realizaría la función lógica XOR entre las dos
            // imágenes, es decir, I1 = I1 XOR I2
```

Figura 7.13: Ejemplo de función lógica XOR entre las imágenes 1 y 2.

7.3.16. Función lógica NOT ò Negativo

Descripción

La función realiza la operación lógica NOT o el negativo de una Imagen dada.

Sintaxis

```
Imagen.NOT()
Imagen.Negativo()
```

siendo Imagen de clase tImagen, es un claro ejemplo de sobrecarga de funciones.

Parámetros de entrada

No precisa ningún parámetro de llamada.

Errores de salida

No presenta ningún tipo de error de salida.

Ejemplo

```
I1.NOT() ó I1.Negativo(); // Se realizaría la función lógica
                          // NOT ó el negativo de la imagen I1.
```

Figura 7.14: Ejemplo de negativo (NOT) de la imagen 1.

7.3.17. Redimensionado

Descripción

Dada una Imagen de tamaño Ancho por Alto, la función dará como resultado una nueva Imagen de tamaño Ancho' por Alto', que contiene a la anterior, es decir, la imagen destino será el resultado de un redimensionamiento de la imagen original.

Sintaxis

```
Imagen.Redimensiona ( int Anchura, int Altura )
```

donde:

- Imagen: es de clase tImagen.
- Anchura y Altura: son variables enteras.

Parámetros de entrada

Necesitamos dos parámetros de llamada enteros como son, la Anchura que indica el ancho deseado para la nueva imagen y la Altura que indica el alto deseado para la nueva imagen.

Errores de salida

Nos encontramos con dos posibles errores de introducción de parámetros, cuando la Anchura o la Altura deseada es menor del Ancho o del Alto, respectivamente de la imagen original.

Ejemplo

```
I1.Redimensiona (280,320); // Redimensionaría la imagen I1, cuyo
// tamaño de partida es de 320 pixeles de altura por 200 pixeles
// de ancho a un nuevo tamaño de 280 por 320 pixeles. Vemos un
// ejemplo en la Figura 7.15
```

Figura 7.15: Ejemplo de imagen 1 redimensionada (Tamaño 280 x 320 pixeles).

7.3.18. Recortar

Descripción

Dada una Imagen de tamaño Ancho x Alto, la función se encarga de recortar dicha imagen a partir de una posicion (x,y) con una determinada Anchura y Altura, es decir, se generará una nueva imagen de tamaño Anchura x Altura que es un recorte de la imagen original.

Sintaxis

```
Imagen.Recortar( int X, int Y, int Anchura, int Altura )
```

donde:

- Imagen: es de clase tImagen.
- X, Y, Anchura y Altura: son variables enteras,

Parámetros de entrada

Necesitamos cuatro parámetros de llamada como son, las variables enteras X e Y que indican las coordenadas de la imagen original a partir de las cuales queremos realizar el recorte y las variables enteras Anchura y Altura que indicarán el tamaño del recorte que queremos realizar.

Errores de salida

Nos encontramos con varias posibilidades de errores de introducción de parámetros, cuando las coordenadas x e y son menores de 0 o cuando éstas más el tamaño del recorte excede del tamaño de la imagen de partida.

Ejemplo

```
I1.Recortar(33,50,167,250); // Recortaría la imagen I1 a partir de
                            // la posición (33,50) con un tamaño de
                            // 167 x 250 pixeles
```

Figura 7.16: Ejemplo de recorte de la imagen 1 (Tamaño 167 x 250 pixeles).

7.3.19. Escribe texto en pantalla

Descripción

La función escribe Texto en modo gráfico a través de un BMP, creado anteriormente, a partir de la posición (x,y) de pantalla permitiendo un ColorTransparente, que implica que los puntos del carácter que coincidan con ese color (combinacion RGB) no sea pintado.

Sintaxis

```
Imagen.Escribe( char* Texto, int X, int Y, int ColorTransparente,
                tPantalla Pantalla )
```

donde:

- Imagen: es de clase tImagen.
- *Texto: es un puntero que apunta a la dirección donde esta el texto a escribir.
- X, Y e ColorTransparente: son variables enteras.
- Pantalla: es de clase tPantalla.

Parámetros de entrada

Necesitamos cinco parámetros de entrada como son, el tipo de Pantalla con el que estamos trabajando, Texto que es un puntero que apunta a la dirección de inicio del texto que queremos escribir, las variables enteras X e Y que indican la posición en pantalla a partir de la que queremos escribir.

Errores de salida

Nos encontramos con varias posibilidades de error de introducción de parámetros, cuando las coordenadas x e y son menores de cero, o cuando alguna de éstas excede del tamaño de la pantalla con el que estemos trabajando.

Nota

Antes de llamar a esta función debemos cargar en la Imagen el BMP que contiene los carácteres de escritura. Este BMP debe presentar la siguiente estructura:

0	1	2	3	4	5	6	7	8	9			
A	B	C	D	E	F	G	H	I	J	K	L	M
N	O	P	Q	R	S	T	U	V	W	X	Y	Z
a	b	c	d	e	f	g	h	i	j	k	l	m
n	o	p	q	r	s	t	u	v	w	x	y	z

El tamaño del BMP tiene que ser respecto a su anchura múltiplo de 13 y respecto a su altura múltiplo de 5 para el correcto funcionamiento de la función de escritura.

Ejemplo

```
I1.CargaBMP("letras.bmp"); // Cargaría en la imagen I1 el BMP
                           // que contiene los carácteres
```

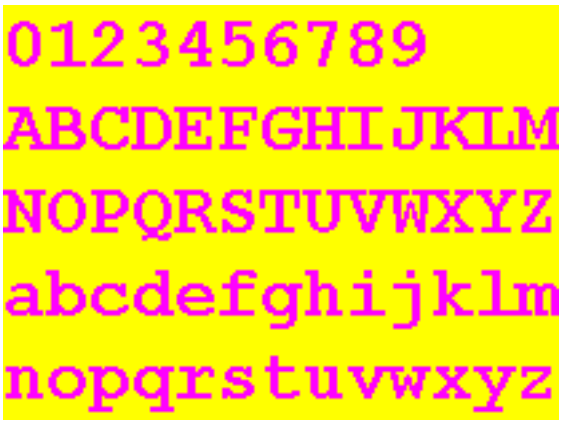

Figura 7.17: Ejemplo de fichero BMP que contiene las letras.

```
I1.Escribe ("Desarrollo de una biblioteca de funciones",208,300,0,P);
// Escribiría en la pantalla P a partir de la posición (208,300) el
// texto especificado
```

Figura 7.18: Ejemplo de texto escrito en modo gráfico a partir de un BMP.

7.3.20. Cálculo de Curvas de Bézier

Descripción

Dados cuatro puntos geométricos de la forma (x,y) se calcula la curva de Beizer que pasa por los cuatro puntos en el rango xε[0,255] y cuyos valores son almacenados en un vector de 256 posiciones, limitando el valor de f(x) en el rango [0,255], este vector será utilizado posteriormente para la ecualización de una imagen, por medio de la función ecualizado.

Sintaxis

```
Imagen.GeneraMatrizBezier( int MatrizEcualizado[255], int P0, int Q0,
int P1, int Q1, int P2, int Q2, int P3, int Q3 )
```

donde:

- Imagen: es de clase tImagen.
- MatrizEcualizado: es un vector de enteros.
- P_o,Q_o,....,P_3,Q_3: son variables enteras.

Parámetros de entrada

Necesitamos nueve parámetros de entrada, ocho de ellos P_o,..., P_3,Q_o,...,Q_3 se corresponden con las coordenadas de los puntos por los que queremos que pase la curva de Beizer y el otro parámetro MatrizEcualizado es un vector que contendrá el valor de los puntos de dicha curva en el rango [0,255].

Errores de salida

No presenta ningún error de salida.

Desarrollo teórico

Se denomina curva Bézier asociada a n+1 puntos P_o,P_1,...,P_n de $\Re^2$, a la curva parametrizada, definida para tε[0,1], cuyos puntos vienen dados mediante la siguiente expresión:

$$(x(t), y(t)) = \sum_{i=0}^{n} B_{i,n}(t) P_i \tag{7.1}$$

en la que los $B_{i,n}$(t) son los polinomios de Bernstein de grado n.

Los puntos P_o,P_1,...,P_n que determinan una curva de Bézier se denominan puntos de control, y la poligonal que los une es el polígono Bézier o B-polígono. La definición anterior, en la cabe la posibilidad de haber puntos repetidos, es la correspondiente a una curva Bézier en el plano. Sobre estas curvas se hablará en todo lo que sigue aunque, de forma análoga, se pueden definir las curvas Bézier en el espacio afín tridimensional $\Re^3$, como también se pueden extender los resultados que se expondrán posteriormente. Conviene observar que toda curva Bézier no sólo queda determinada por sus puntos de control sino que es fundamental el orden en el que éstos se dan.

Los polinomios de Bernstein de grado n, que denotamos por $B_{o,n}$(t),$B_{1,n}$(t),...,$B_{n,n}$(t) son:

$$B_{i,n}(t) = \left(\begin{array}{c} n \\ i \end{array} \right) (1-t)^{n-i} t^i para i = 0, ..., n donde \left(\begin{array}{c} n \\ i \end{array} \right) = \frac{n!}{i!(n-i)!}$$

Según la definición los polinomios de Bernstein de grado 3 (en nuestro caso el número de puntos es cuatro) son:

$$B_{o,3}(t) = (1-t)^3 \tag{7.2}$$
$$B_{1,3}(t) = 3(1-t)^2 t \tag{7.3}$$
$$B_{2,3}(t) = 3(1-t)t^2 \tag{7.4}$$
$$B_{3,3}(t) = t^3 \tag{7.5}$$

Así, dados los puntos $A_o = (P_o, Q_o)$,...,$A_3 = (P_3, Q_3)$ la curva de Beizer asociada tiene las siguientes ecuaciones paramétricas:

$$x(t) = B_{o,3}(t)P_o + B_{1,3}(t)P_1 + B_{2,3}(t)P_2 + B_{3,3}(t)P_3 \tag{7.6}$$
$$y(t) = B_{o,3}(t)Q_o + B_{1,3}(t)Q_1 + B_{2,3}(t)Q_2 + B_{3,3}(t)Q_3 \tag{7.7}$$

Por otro lado, sabemos que en el ecualizado xε[0,255] con lo que buscamos el valor de t para cada x, para ello se utiliza el método de la secante que es un método iterativo para la búsqueda de raíces de una ecuación.

Una vez buscado cada valor de t para cada x se calcula Y(t) la cual debe estar comprendida en el rango [0,255].

Ejemplo

```
int MatrizEcualizado[255]; // Definición del vector

I1.GeneraMatrizBezier(MatrizEcualizado,0,0,100,255,200,255,255,0);
// Se generaría en la variable MatrizEcualizado la curva de Bezier
// correspondiente que pasa por los puntos (0,0),(100,255),(200,255)
// y (255,0). La función de transferencia que se esta implementando
// se puede ver en la Figura 7.19
```

7.3.21. Cálculo de Splines cúbicos

Descripción

Dados cuatro puntos geométricos de la forma (x,y) se calcula los Splines cúbicos que pasan por los cuatro puntos en el rango xε[0,255] y cuyos valores son almacenados en un vector de 256 posiciones, limitando el valor de f(x) en el rango [0,255], este vector será utilizado posteriormente para la ecualización de una imagen, por medio de la función ecualizado.

Sintaxis

```
Imagen.GeneraMatrizSplines( int MatrizEcualizado[255], int P0, int Q0,
int P1, int Q1, int P2, int Q2, int P3, int Q3 )
```

donde:

- Imagen: es de clase tImagen.
- MatrizEcualizado: es un vector de enteros.
- P_o,Q_o,....,P_3,Q_3: son variables enteras.

Parámetros de entrada

Necesitamos nueve parámetros de entrada, ocho de ellos P_o,..., P_3,Q_o,...,Q_3 se corresponden con las coordenadas de los puntos para el calculo de splines y el otro parámetro MatrizEcualizado es un vector que contendrá el valor de los puntos de dicha curva en el rango [0,255].

Errores de salida

No presenta ningún error de salida.

Desarrollo teórico

El spline cúbico (k=3) es el spline más empleado, debido a que proporciona un excelente ajuste a los puntos tabulados y su cálculo no es excesivamente complejo. Sobre cada intervalo $[t_o, t_1], [t_1, t_2], ..., [t_{n-1}, t_n]$, S está definido por un polinomio cúbico diferente. Sea S_i el polinomio cúbico que representa a S en el intervalo $[t_i, t_{i+1}]$.

Los polinomios S_{i-1} y S_i interpolan el mismo valor en el punto t_i, es decir, se cumple:

$$S_{i-1}(t_i) = y_i = S_i(t_i) \tag{7.8}$$

por lo que se garantiza que S es continuo en todo el intervalo. Además, se supone que S´ y S´´ son continuas, condición que se emplea en la deducción de una expresión para la función del spline cúbico.

Aplicando las condiciones de continuidad del spline S y de las derivadas primera S´ y segunda S´´, es posible encontrar la expresión analítica del spline. La expresión resultante es:

$$S_i(x) = \frac{z_i}{6h_i}(t_{i+1}-x)^3 + \frac{z_{i+1}}{6h_i}(x-t_i)^3 + (\frac{y_{i+1}}{h_i} + \frac{z_{i+1}h_i}{6})(x-t_i) + (\frac{y_i}{h_i} - \frac{z_i h_i}{6})(t_{i+1}-x) \tag{7.9}$$

En la expresión anterior, $h_i = t_{i+1} - t_i$ y $z_o, z_1, ..., z_n$ son incógnitas. Para determinar sus valores, utilizamos las condiciones de continuidad que deben cumplir estas funciones. El resultado sería:

$$h_{i-1}z_{i-1} + 2(h_i + h_{i-1})z_i + h_i z_{i+1} = \frac{6}{h_{i-1}}(y_{i+1} - y_i) - \frac{6}{h_{i-1}}(y_i - y_{i-1}) \tag{7.10}$$

La ecuación anterior, con i=1,2,..,n-1 genera un sistema de n-1 ecuaciones lineales con n+1 incógnitas . Podemos elegir z_o y z_1 de forma arbitraria y resolver el sistema de ecuaciones resultante para obtener los valores de $z_1, z_2, ..., z_{n-1}$. Una elección especialmente adecuada es hacer $z_o = z_1 = 0$. La función spline resultante se denomina spline cúbico natural y el sistema de ecuaciones lineal expresado en forma matricial es:

$$\begin{pmatrix} u_1 & h_1 & & & & \\ h_1 & u_2 & h_2 & & & \\ & h_2 & u_3 & h_3 & & \\ & & \cdots & \cdots & \cdots & \\ & & & h_{n-3} & u_{n-2} & h_{n-2} \\ & & & & h_{n-2} & u_{n-1} \end{pmatrix} \begin{pmatrix} z_1 \\ z-2 \\ z-3 \\ \cdots \\ z_{n-2} \\ z_{n-3} \end{pmatrix} = \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ \cdots \\ v_{n-2} \\ v_{n-1} \end{pmatrix}$$

en donde:

$$h_i = t_{i+1} - t_i \tag{7.11}$$

$$u_i = 2(h_i + h_{i-1}) - \frac{h_{i-1}^2}{u_{i-1}} \tag{7.12}$$

$$b_i = \frac{6}{h_i}(y_{i+1} - y_i) \tag{7.13}$$

$$v_i = b_i - b_{i-1} - \frac{h_{i-1}v_{i-1}}{u_{i-1}} \tag{7.14}$$

$$\tag{7.15}$$

Este sistema de ecuaciones, que es tridiagonal, se puede resolver mediante elim inación gaussiana sin pivoteo. Así, el valor del spline S en un punto x cualquiera interpolado se puede calcular de forma eficiente empleando la siguiente expresión:

$$S_i(x) = y_i + (x - t_i)[C_i + (x - t_i)[B_i + (x - t_i)A_i]] \tag{7.16}$$

en donde:

$$A_i = \frac{1}{6h_i}(z_{i+1} - z_i) \tag{7.17}$$

$$B_i = \frac{z_i}{2} \tag{7.18}$$

$$C_i = -\frac{h_i}{6}z_{i+1} - \frac{h_i}{3}z_i + \frac{1}{h_i}(i_{i+1} - y_i) \tag{7.19}$$

Ejemplo

```
int MatrizEcualizado[255]; // Definición del vector

I1.GeneraMatrizSplines(MatrizEcualizado,0,0,100,255,200,255,255,0);
// Se generaría en la variable MatrizEcualizado las curvas de
// Splines correspondientes que pasan por los puntos (0,0),
// (100,255), (200,255) y (255,0)
```

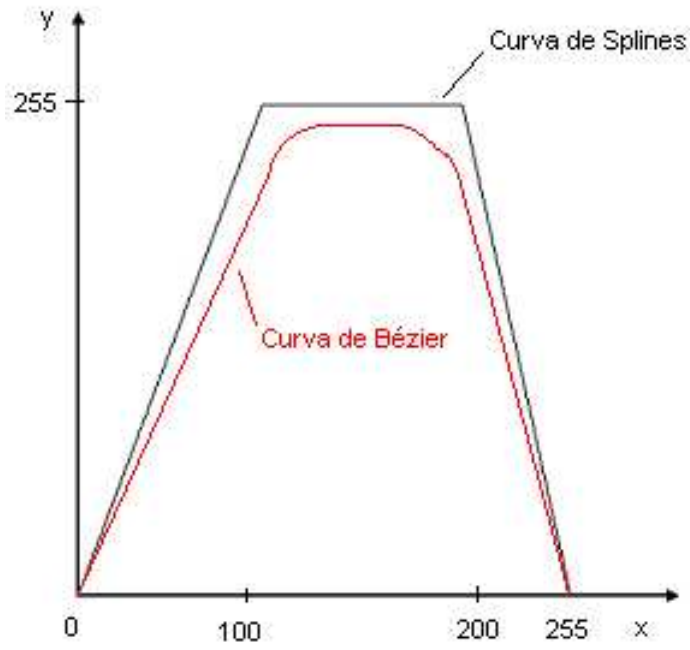

Figura 7.19: Ejemplo de funciones de transferencia para Beizer y Splines.

7.3.22. Ecualizado

Descripción

Se ecualiza una Imagen a través de un vector que contiene los valores o la función de ecualizado.

Sintaxis

```
Imagen.Ecualiza(int *MatrizEcualizado, int TipoEcualizado)
```

donde:

- Imagen: es de clase tImagen.
- *MatrizEcucalizado: es un puntero que apunta a la dirección de inicio de un vector de números enteros.
- TipoEcualizado: es una variable entera.

Parámetros de entrada

Necesitamos dos parámetros de entrada como son, un puntero *MatrizEcualizado, que apunta al vector que contiene los valores de la función de ecualizado y una variable entera TipoEcualizado que indica el tipo de ecualizado que queremos realizar sobre la imagen, así, según el valor de esta última variable, tendremos los siguientes ecualizados:

- (“000”)=“0” No ecualiza nada.
- (“001”)=“1” Solo ecualiza Componente Azul.
- (“010”)=“2” Solo ecualiza Componente Verde.
- (“011”)=“3” Ecualiza Componentes Verde y Azul.
- (“100”)=“4” Solo ecualiza Componente Roja.
- (“101”)=“5 Ecualiza Componentes Roja y Azul.
- (“110”)=“6” Ecualiza Componentes Roja y Verde.
- (“111”)=“7” Ecualiza las tres Componentes.

Errores de salida

Nos encontramos con una única posibilidad de error, cuando la variable TipoEcualizado sea menor de 0 o mayor de 7.

Ejemplo

```
int MatrizEcualizadoBeizer [255], MatrizEcualizadoSplines [255];
// Definición de vectores

I1.GeneraMatrizBeizer(MatrizEcualizadoBeizer,0,0,100,255,200,255,
255,0); // Generación de los valores de la curva de Bézier

I2.GeneraMatrizSplines(MatrizEcualizadoSplines,0,0,100,255,200,255,
255,0); // Generación de los valores de las curvas de Splines

I1.Ecualiza(MatrizEcualizadoBeizer,7); // Ecualizaría todas las
// componentes (roja, verde y azul) de la imagen I1 atendiendo a
// la MatrizEcualizadoBeizer

I2.Ecualiza(MatrizEcualizadoSplines,7); // Ecualizaría todas las
// componentes (roja, verde y azul) de la imagen I2 atendiendo a
// la MatrizEcualizadoSplines
```

Figura 7.20: Ecualizado de la imagen 1 con función de Bézier.

Figura 7.21: Ecualizado de imagen 1 con función de Splines.

Se puede observar la diferencia existente entre ambas imágenes, para la generación de valores a partir de unos mismos puntos, esto es debido a que la función de transferencia para ambos métodos es diferente, como se pudo ver en la Figura 7.19.

7.3.23. Corrección de aberraciones

Descripción

Se realiza la corrección de cualquier aberración, tanto positiva como negativa, que presente una imagen dada.

Sintaxis

```
Imagen.CorrigeAberracion( double Factor_Distorsion )
```

donde:

- Imagen: es de clase tImagen.
- Factor_Distorsión: es una variable real.

Parámetros de entrada

Necesitamos un único parámetro de llamada como es Factor_Distorsion que es indica el factor de distorsión que presenta la imagen que queremos corregir.

Errores de salida

No presenta ningún error de salida.

Desarrollo teórico

Una aberración se define como una deformación geométrica de una imagen, que se puede presentar de dos formas diferentes:

- Aberración positiva: cuando se produce un aumento de la imagen con respecto a la distancia al eje focal y hablaríamos de una imagen de tipo cojín ó corsé.
- Aberración negativa: cuando se produce una disminución de la imagen con respecto a la distancia al eje focal y hablaríamos de una imagen tipo barril.

La deformación geométrica de una imagen o aberración puede expresarse matemáticamente como:

$$x_{real} = (1 + g_x d^2) x_{teorica} \tag{7.20}$$

$$y_{real} = (1 + g_y d^2) y_{teorica} \tag{7.21}$$

con:

$$d = \sqrt{x_{teorica}^2 + y_{teorica}^2} \tag{7.22}$$

siendo g_x y g_y los coeficientes de distorsión que suelen tomarse iguales.

A continuación se observa como que forma presentan las imágenes cuando se produce una deformación geométrica en ellas:

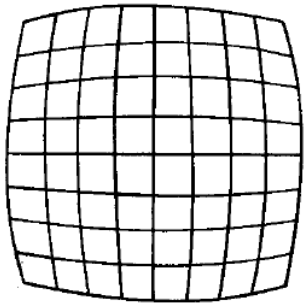

Figura 7.22: Imagen tipo barril.

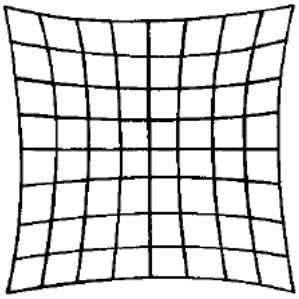

Figura 7.23: Imagen tipo cojín.

Ejemplo

```
I1.CorrigeAberracion(0.005); // Realizaría la correción de una
// deformación geométrica de la Imagen I1 con un factor de
// distorsión de 0.005
```

Figura 7.24: Ejemplo de imagen aberrada de tipo cojín.

Figura 7.25: Ejemplo de imagen corregida la aberracón.

7.3.24. Transformada de Fourier 2D

Descripción

Dada una Imagen, la función se encarga de rellenar tres tablas, una para cada componente de color (RGB) con los valores de la transformada de Fourier en dos dimensiones.

Sintaxis

```
Imagen.FFT2D( char *Comp_Rojo, char *Comp_Verde, char *Comp_Azul )
```

donde:

- Imagen: es de clase tImagen.
- *Comp_Rojo, *Comp_Verde y *Comp_Azul: son punteros que apuntan a la dirección donde comienza cada tabla.

Parámetros de entrada

Necesitamos tres parámetros de llamada como son *Comp_Rojo, *Comp_Verde y *Comp_Azul que apuntan a la dirección de inicio de cada tabla para cada componente de color.

Errores de salida

No presenta ningún error de salida.

Desarrollo teórico

La transformada de Fourier bidimensional se realiza para dos variables, el muestreo se hace en el plano xy donde los valores están igualmente espaciados sobre las rectas paralelas al eje x y las rectas paralelas al eje y.

Se define a $M_{N\times M}(C)$ como el conjunto de matrices de tamaño NxM cuyas entradas son números complejos. Sea f(x,y):$\Re^2 \rightarrow$ C. La transformada discreta de Fourier bidimensional de f(x,y) para $0 \leq x \leq N-1$, $0 \leq y \leq M-1$, se define como el operador:

$$2D - DFT : M_{NxM}(C) \rightarrow M_{NxM}(C)$$

$$(f(x,y))_{NxM} \rightarrow (F(u,v))_{NxM'}$$

donde:

$$F(u,v) = \frac{1}{N}\sum_{x=0}^{N-1}\sum_{y=0}^{M-1} f(x,y)e^{-i2\pi(\frac{ux}{N}+\frac{vy}{M})} \tag{7.23}$$

para u=0,1,...,N-1 y v=0,1,...,M-1.

La ecuación 7.23 puede expresarse como:

$$F(u,v) = \frac{1}{N}\sum_{x=0}^{N-1}\sum_{y=0}^{M-1} f(x,y)e^{-i2\pi\frac{ux}{N}}e^{-i2\pi\frac{vy}{M}} = \frac{1}{N}\sum_{x=0}^{N-1} F(x,v)e^{-i2\pi\frac{ux}{N}} \tag{7.24}$$

donde:

$$F(x,v) = M(\frac{1}{M}\sum_{y=0}^{M-1} f(x,y)e^{-i2\pi\frac{vy}{M}}) \tag{7.25}$$

De esta manera se puede usar sucesivamente la transformada de Fourier unidimensional sobre filas y columnas de la matriz.

La implementación de la transformada de Fourier para dos dimensiones (FFT-2D ó Transformada Rápida de Fourier bidimensional) se puede realizar de la siguiente forma: FFT sobre filas → Multiplicación por M → FFT sobre columnas (Ecuación 7.25) donde N y M son potencias de dos.

Nota

El tamaño de las tablas, que albergarán los valores de la transformada de Fourier para cada componente, deben de tener un tamaño de 4 veces el tamaño de la imagen original, la cual debe de ser cuadrada (mismo ancho que alto) para un correcto funcionamiento de la función), debido a que los valores que representan dicha tabla están son reales (tamaño float = 4 * tamaño char) y 2 veces más porque cada tabla contiene la parte real y parte imaginaria de cada valor calculado, es decir, cada tabla debe de tener un tamaño de $Ancho^2 * 4 * 2$.

Ejemplo

```
I1.FFT2D(TablaRoja,TablaVerde,TablaAzul);
// Rellenaría las tres tablas con los valores de la transformada
// de Fourier en dos dimensiones para cada componente, donde
// previamente se han definido TablaRoja, TablaVerde y TablaAzul
// como Tabla = new char [Ancho*Ancho*4*2] siendo Ancho la anchura
// de la imagen I1
```

7.3.25. Representación de la Transformada de Fourier 2D

Descripción

Dadas tres tablas, una para cada componente de color, calcula la parte real, la parte imaginaria, el módulo o la fase de dichas tablas (a través de Opcion) y las representa a través de una Imagen cuadrada de tamaño.

Sintaxis

```
Imagen.CalculaFFT2D( int Tam_Imagen, char *Comp_Rojo, char *Comp_Verde,
                     char *Comp_Azul, char *Opcion )
```

donde:

- Imagen: es de clase tImagen.
- Tam_Imagen: es una variable entera.
- *Comp_Rojo, *Comp_Verde y *Comp_Azul: son punteros que apuntan a la dirección donde comienza cada tabla.
- *Opcion: es un puntero a una cadena de carácteres.

Parámetros de entrada

Necesitamos un total de cinco parámetros de llamada como son *Comp_Rojo, *Comp_Verde y *Comp_Azul que apuntan a la dirección de inicio de cada tabla para cada componente de color, la variable entera Tam_Imagen que indica el tamaño de la imagen resultante y el puntero *Opcion que apunta a la dirección de comienzo de una cadena de carácteres, y el contenido de ésta última será el que determine el tipo de representación a realizar, así:

Opcion=“char1char2char3”

$$char1 = \begin{cases} = \text{“}r\text{”} & \text{se representa la parte real} \\ = \text{“}i\text{”} & \text{se representa la parte imaginaria} \\ = \text{“}m\text{”} & \text{se representa el módulo} \\ = \text{“}f\text{”} & \text{se representa la fase} \end{cases}$$

$$char2 = \begin{cases} = \text{“}n\text{”}(parachar1 = \text{“}r\text{”}, \text{“}i\text{”}, \text{“}m\text{”}) & \text{representación normalizada} \\ = \text{“}l\text{”}(parachar1 = \text{“}r\text{”}, \text{“}i\text{”}, \text{“}m\text{”}) & \text{representación logarítmica} \\ = \text{“}n\text{”}(parachar1 = \text{“}f\text{”}) & \text{representación fase normalizada} \\ = \text{“}d\text{”}(parachar1 = \text{“}f\text{”}) & \text{representación fase desnormalizada} \end{cases}$$

$$char3 = \begin{cases} = \text{“}o\text{”} & \text{representación ordenada} \\ = \text{“}d\text{”} & \text{representación desordenada} \end{cases}$$

Errores de salida

Presenta varios errores de introducción de parámetros, dará error siempre y cuando alguno de los parámetros de la cadena de carácteres de Opcion no coincida con la sintaxis expuesta anteriormente. Además nos encontraremos con un error de FFT no válida cuando el tamaño de la imagen resultante no es múltiplo de dos.

Ejemplo

```
char *Tabla1,*Tabla2,*Tabla3; // Defínción de constantes
int Anchura=256;

Tabla1=new char[Anchura*Anchura*4*2]; // Reserva de memoria
Tabla2=new char[Anchura*Anchura*4*2]; // para las tablas
Tabla3=new char[Anchura*Anchura*4*2];

I1.FFT2D(Tabla1,Tabla2,Tabla3); // Rellenado de las tablas

// Distintas representaciones de la FFT

I2.CalculaFFT2D(Anchura,Tabla1,Tabla2,Tabla3,"mlo");
I3.CalculaFFT2D(Anchura,Tabla1,Tabla2,Tabla3,"fno");
I4.CalculaFFT2D(Anchura,Tabla1,Tabla2,Tabla3,"rlo");
I5.CalculaFFT2D(Anchura,Tabla1,Tabla2,Tabla3,"ilo");
```

A continuación vemos una serie de ejemplos de las representaciones que se pueden obtener a partir de la transformada de Fourier. Para esta serie de ejemplos se ha variado la Imagen 7.1 que a partir de ahora presenta un tamaño cuadrado de 256 x 256 pixeles de resolución.

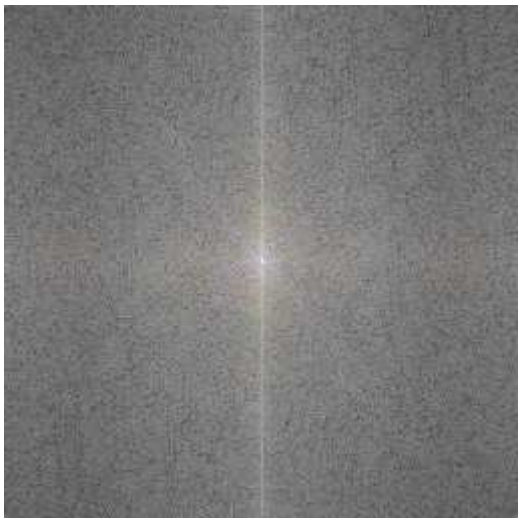

Figura 7.26: Ejemplo del módulo logarítmico ordenado de la FFT de la imagen 1 modificada

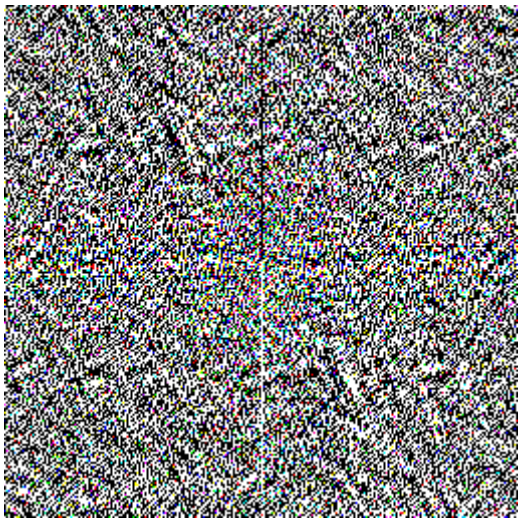

Figura 7.27: Ejemplo de la fase normalizada ordenada de la FFT de la imagen 1 modificada

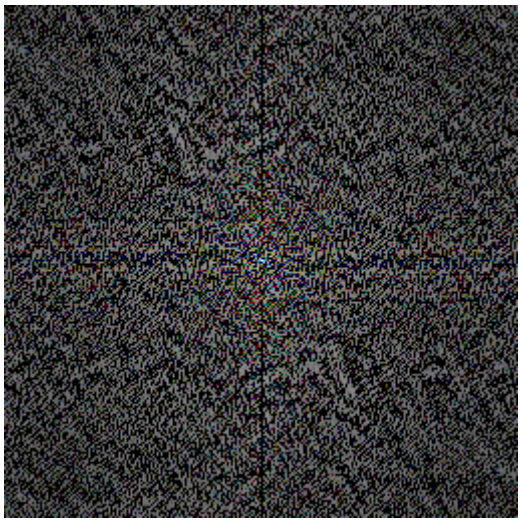

Figura 7.28: Ejemplo de la parte real logarítmica ordenada de la FFT de la imagen 1 modificada

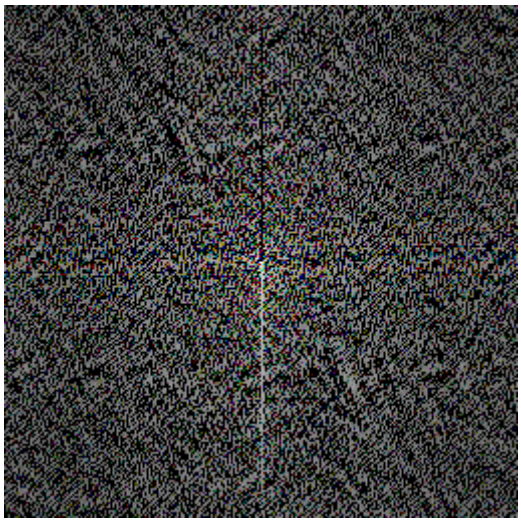

Figura 7.29: Ejemplo de la parte imaginaria logarítmica ordenada de la FFT de la imagen 1 modificada

Ejemplo de otras Transformadas de Fourier

Vamos a mostrar la transformada de Fourier de dos tipos de imágenes para comprobar el perfecto funcionamiento de la rutina desarrollada. Se representarán tres imágenes, la de la izquierda será la imagen inicial, la del medio el módulo de la transformada y la de la derecha la fase de dicha transformada.

En una primer imagen vamos a ver la transformada de Fourier, en cuanto a módulo y fase, de un rectángulo, variando la imagen inicial para comprobar alguna de las propiedades de dicha transformada.

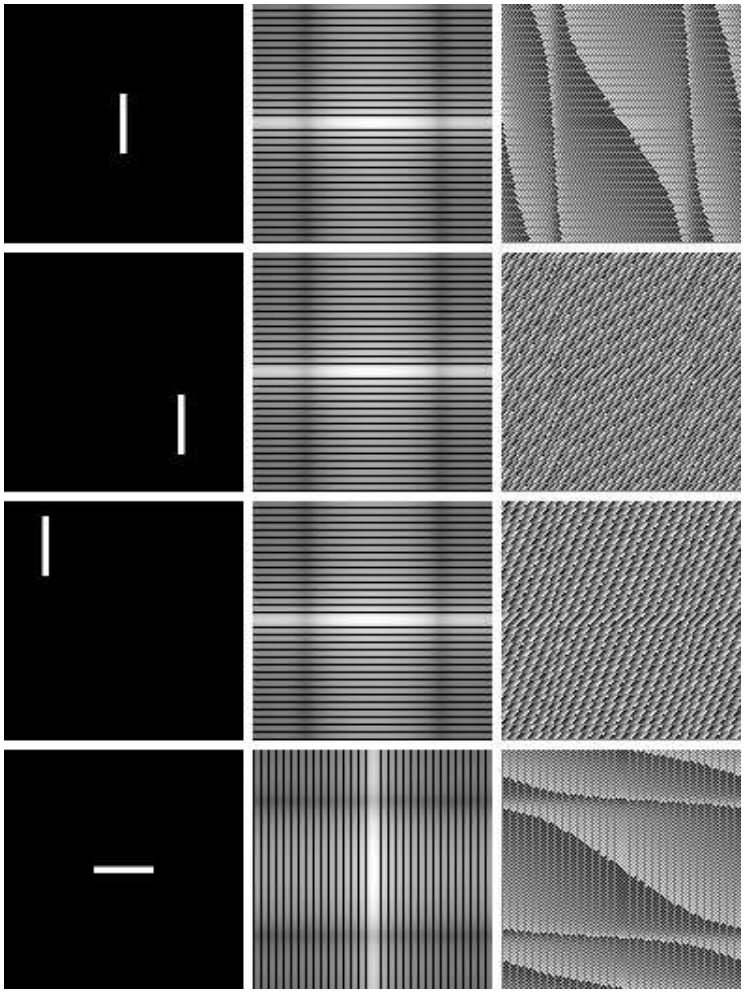

Figura 7.30: Ejemplo de Transformadas de Fourier de un rectángulo.

Se puede comprobar en la figura anterior que se verifican las propiedades de la transformada de Fourier de simetría conjugada, linealidad y desplazamiento.

Una segunda imagen va a ser una función seno que presenta distintas variaciones en cuanto a la frecuencia, observándose las diferentes variaciones en módulo y en fase que presentan las distintas representaciones de la transformada de Fourier.

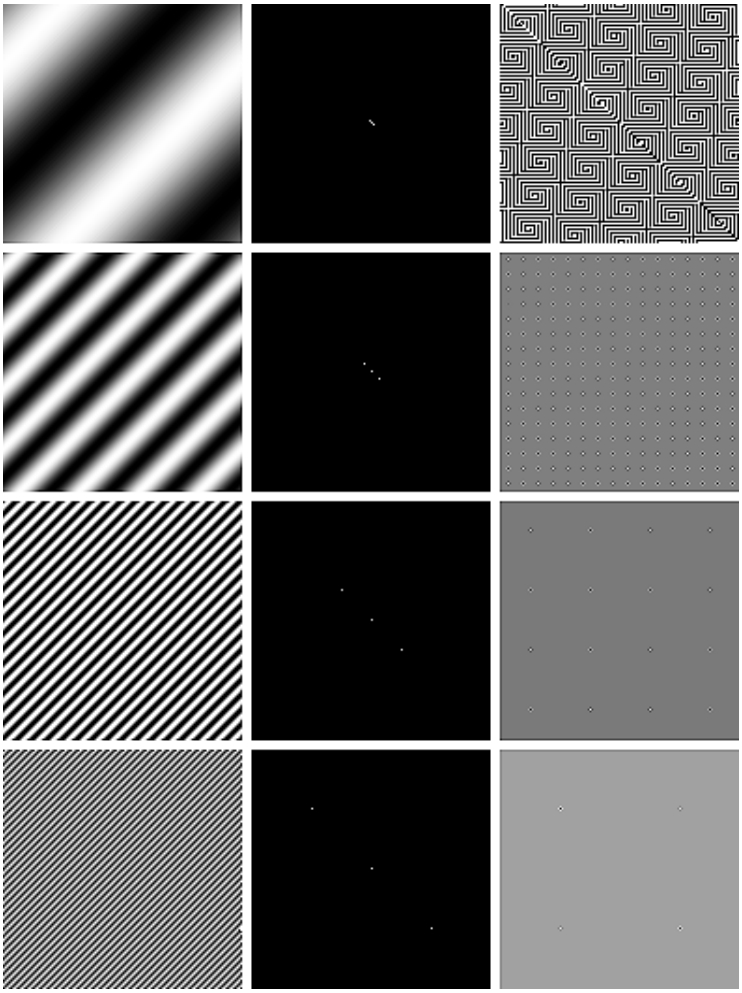

Figura 7.31: Ejemplo de Transformadas de Fourier de una función seno.

Se puede observar en la figura anterior que la transformada de un seno es un único punto a una determinada frecuancia, en nuestro caso son dos puntos, uno para la frecuencia postiva y otro para la frecuencia negativa y a medida que vamos aumentando la frecuencia los puntos se separan cada vez más.

7.3.26. Transformada Inversa de Fourier 2D

Descripción

Dadas tres tablas, una para cada componente de color (RGB) que contienen los valores de la transformada de Fourier en dos dimensiones y se representa a través de una imagen cuadrada de tamaño múltiplo de dos.

Sintaxis

```
Imagen.InvFFT2D( int Tam_Imagen, char *Comp_Rojo, char *Comp_Verde,
                 char *Comp_Azul )
```

donde:

- Imagen: es de clase tImagen.
- *Comp_Rojo, *Comp_Verde y *Comp_Azul: son punteros que apuntan a la dirección donde comienza cada tabla.
- Tam_Imagen: es una variable entera.

Parámetros de entrada

Necesitamos cuatros parámetros de llamada como son *Comp_Rojo, *Comp_Verde y *Comp_Azul que apuntan a la dirección de inicio de cada tabla para cada componente de color y la variable Tam_Imagen que indica el tamaño de la imagen resultante.

Errores de salida

Presenta un único error de introducción de parámetros cuando el tamaño de la imagen resultante o que se quiere obtener a partir de las tablas es múltiplo de dos.

Desarrollo teórico

Ya vimos en el apartado 7.3.24 el cálculo de la transformada de Fourier de 2D, pues a partir de ésta, se establece la 2D-IDFT ó Transformada Inversa de Fourier Bidimensional como:

$$2D - IDFT : M_{NxM}(C) \rightarrow M_{NxM}(C)$$

$$(F(u,v))_{NxM} \rightarrow (f(x,y))_{NxM'}$$

donde:

$$f(x,y) = \frac{1}{M} \sum_{u=0}^{N-1} \sum_{v=0}^{M-1} F(u,v) e^{i2\pi(\frac{ux}{N} + \frac{vy}{M})} \tag{7.26}$$

para x=0,1,...,N-1 e y=0,1,...,M-1.

La implementación de la 2D-IFFT es similar a la implementación de la 2D-FFT.

Ejemplo

```
I1.InvFFT2D(256,TablaRoja,TablaVerde,TablaAzul);
// Calcularía la imagen resultante al aplicar la transformada de
// Fourier Inversa a las tres tablas pasadas y cuya imagen tiene
// una anchura cuadrada de 256 x 256 pixeles
```

Capítulo 8

Pantalla

En este capítulo veremos el software desarrollado en el campo que tiene que ver con la visualización en pantalla de las imágenes.

Se desarrollarán los archivos de la biblioteca creada que gestionan la visualización de imágenes en pantalla.

A continuación, se explicará el contenido de los archivos anteriomente mencionados, que constarán de la definición de la clase empleada para tal efecto, con su conjunto de variables y funciones realizadas.

Por último, se comentarán las funciones realizadas que nos permiten trabajar con la pantalla, así podremos trabajar con la pantalla en modo texto y en modo gráfico.

8.1. Archivos

Se han incluido a la biblioteca de funciones generada dos archivos para la visualizacón de las imágenes en pantalla, estos archivos son:

- pantalla.h: contiene la definición de la clase tPantalla creada para tal efecto.
- pantalla.cpp: contiene la implementación de las funciones definidas en la clase tPantalla y la definición de dos estrcturas para la obtención de información del modo VESA seleccionado.

8.2. Clase tPantalla

Se define la clase tPantalla para gestionar la visualización de imágenes en pantalla, esta formada por un conjunto de variables para la caracterización del tipo de pantalla con la que se quiere trabajar y por dos funciones que seleccionan el modo de trabajo en pantalla, como veremos en la sección 8.3. Las variables que se definen se describen a continuación:

- Selector: selector de segmento empleado.
- Dirección: dirección del comienzo de la pantalla en el segmento anterior.
- DirFisica: dirección física del comienzo de la pantalla en memoria.
- AnchoPantalla: resolución horizontal de la pantalla en pixeles.
- AltoPantalla: resolución vertical de la pantalla en pixeles.
- TotalMemoriaDisponible: total de memoria de vídeo de la que disponemos.
- MemoriaRestante: memoria de vídeo de la que disponemos si quitamos lo que ocupa la pantalla en la memoria de vídeo.

8.3. Funciones

8.3.1. Poner Modo VESA

Descripción

Busca el Modo VESA correspondiente a los parámetros pasados a la función, como son el Ancho y el Alto, en pixeles, de la pantalla que queremos seleccionar y pone el Modo gráfico correspondiente. La función sólo trabaja para modos de 32 bits por pixel.

Sintaxis

```
Pantalla.PonModoVESA( int Ancho, int Alto )
```

donde:

- Pantalla: es de clase tPantalla.
- Ancho y Alto: son variables enteras.

Parámetros de entrada

Necesitamos dos parámetros de llamada como son, Ancho y Alto que indican el ancho y el alto de pantalla que queremos seleccionar, es decir, la resolución que queremos en pixeles.

Errores de salida

Presenta un único error de salida, cuando no encuentra ningún modo que se adapte a la resolución introducida.

Desarrollo de la función

Se definen dos estructuras, vistas anteriormente en la sección 4.4, como son ModeInfoBlock con la que se obtiene la información de un modo seleccionado y VbeInfoBlock

con la que se obtiene la información VESA deseada, para el cálculo de las variables TotalMemoriaDisponible y MemoriaRestante.

Ejemplo

```
P.PonModoVESA(800,600); // Pondría la pantalla P en un modo gráfico
                        // de 800 x 600 pixeles de resolución
```

8.3.2. Poner Modo Texto

Descripción

Pone la pantalla en modo texto, es decir, en un formato de 80 columnas por 24 filas.

Sintaxis

```
Pantalla.PonModoTexto()
```

donde Pantalla es de clase tPantalla.

Parámetros de entrada

No precisa ningún parámetro de llamada.

Errores de salida

No presenta ningún tipo de error de salida.

Ejemplo

```
P.PonModoTexto(); // Pondría la pantalla P en modo texto
```

Parte III

Aplicación práctica

Capítulo 9

Hardware desarrollado

En este capítulo se desarrollará una aplicación hardware específica, para mostrar el correcto funcionamiento de la librería creada.

En primer lugar se describirá el tema a tratar, como será el testeo de un display, y una posible solución al problema.

Más tarde se verán el display y los integrados de los que disponemos para abordar el tema a tratar.

A continuación se realizará el diseño a partir de una herramienta gráfica como es Proteus y se simulará el circuito realizado con dicha herramienta.

Seguidamente se implementará el circuito, diseñado anteriormente, y se desarrollará un programa escrito en C++ para la programación del trabajo con el display.

Y, por último, se verá un ejemplo de la aplicación práctica creada para tal efecto, y se comprobará su correcto funcionamiento.

9.1. Descripción y solución

Se pretende comprobar el correcto funcionamiento de un display, propiedad de la empresa Tecdis (Displays Iberica), cuyas características fundamentales se verán en la sección 9.2.

La solución expuesta para dicho problema será ir iluminando los segmentos del display de cuatro en cuatro, para ello se construirá un registro de desplazamiento, mediante integrados 74LS374, para conseguir dicha secuencia, e ir comprobando que se van iluminando los segmentos del display según la secuencia deseada. Para la comprobación de la iluminación de los segmentos del display se creará un programa escrito en lenguaje C++ y se utilizarán algunas de las funciones desarrolladas a lo largo del proyecto y vistas en los capítulos anteriores, que están definidas en la librería creada para tal efecto, es decir, el esquema que seguiremos para esta aplicación práctica será:

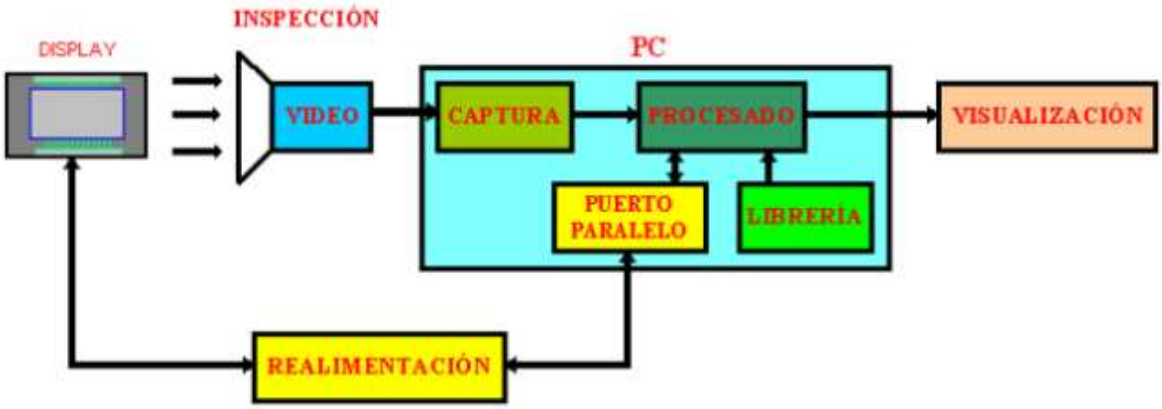

Figura 9.1: Ejemplo práctico.

Primeramente se reseteará el display, es decir, no se iluminará ningún segmento, a continuación iremos iluminando de cuatro en cuatro segmentos, por medio del hardware de realimentación construido para tal efecto. Se irán tomando las imágenes cada cierto intervalo de tiempo a través de una cámara de vídeo, estas imágenes llegarán a la etapa de procesado a través de la capturadora de vídeo que disponemos, una vez allí, las imágenes se compararán con unas imágenes de referencia y se comprobarán si se iluminan los segmentos que deseados, si el proceso es correcto se pasará a la comprobación de nuevos segmentos, en caso contrario, se mostrará un mensaje de error ó se desechará el display debido a su incorrecto funcionamiento.

9.2. Componentes necesarios

Como ya se ha visto en la sección 9.1 necesitaremos únicamente dos tipos de componentes, por un lado disponemos del display, del que queremos comprobar su correcto funcionamiento y, por otro lado, disponemos de latchs 74LS374 para la implementación de un registro de desplazamiento.

Display

El display que disponemos pertenece a la empresa Tecdis (Displays Iberica) y presenta las siguientes dimensiones:

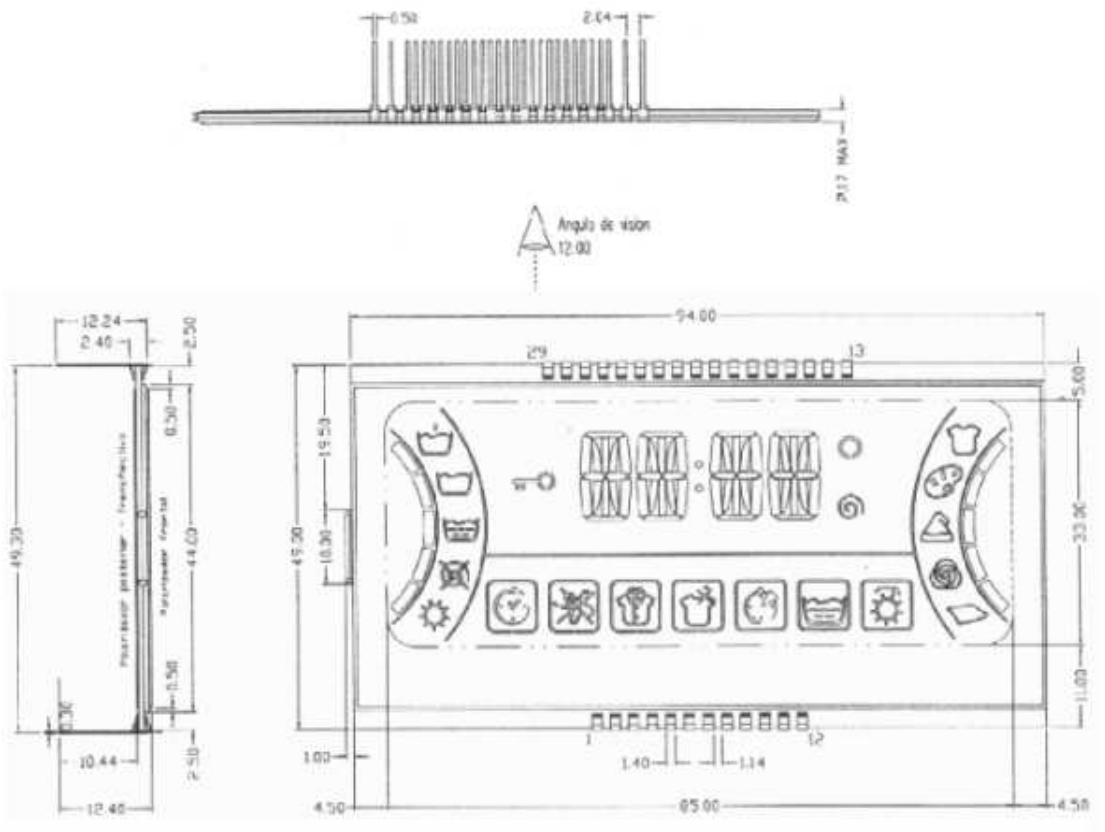

Figura 9.2: Dimensiones del display disponible (Escala 1:1 en mm).

El display que tenemos presenta las siguientes características técnicas de funcionamiento:

- Tipo de display: FSTN - Positivo - Reflectivo - Gris.
- Funcionamiento: 1/4 Duty, 1/3 Bias y V_{op}=5.0 v alterna.
- Op. Temp: -20ºC a 70ºC.
- St. Temp: -30ºC a 80ºC.

El display presenta la pantalla de cristal líquido (LCD) formada por un conjunto de segmentos, cuyo acceso es mediante filas y columnas, a través de los pines que presenta dicho display, así para el acceso a las filas tenemos los pines del 1 al 4 y para el acceso a columnas los pines del 5 al 29.

Para la iluminación de un segmento se debe producir una diferencia de potencial entre los pines que intervienen en dicha iluminación del segmento, los correspondientes a la fila y a la columna que deseemos iluminar.

A continuación se puede ver en una tabla la relación entre los pines y los segmentos que se iluminan.

PIN	1	2	3	4	5	6	7	8	9	10	11	12	13
COM1	X				W4	S1	T1	K1	L2	S7	S8	W9	S14
COM2		X			W5	S2	T2	K2	K7	T7	S9	W10	4C
COM3			X		S6	S3	T3	K3	K6	T6	S10	—	S13
COM4				X	S4	S5	T4	K4	K5	T5	S11	—	4B
PIN	1	2	3	4	14	15	16	17	18	19	20	21	22
COM1	X				W8	4D	W7	4E	W6	3D	L3	3E	COL
COM2		X			4M	4L	4K	3C	3M	3L	3K	2C	2M
COM3			X		4G2	4I	4G1	4F	3G2	3I	3G1	3F	2G2
COM4				X	4J	4A	4H	3B	3J	3A	3H	2B	2J
PIN	1	2	3	4	23	24	25	26	27	28	29		
COM1	X				2D	L1	2E	W1	1D	W2	W3		
COM2		X			2L	2K	1C	1M	1L	1K	1E		
COM3			X		2I	2G1	2F	1G2	1I	1G1	1G2		
COM4				X	2A	2H	1B	1J	1A	1H	1F		

En la siguiente figura se puede observar que segmentos del display se iluminan según la tabla anterior.

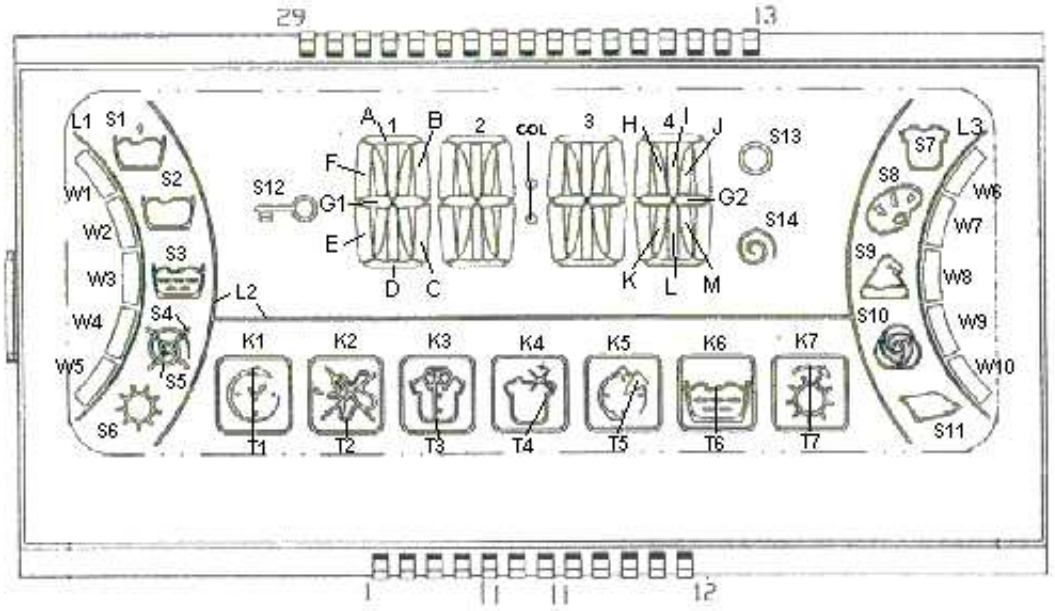

Figura 9.3: Relación entre el display y los segmentos a iluminar.

Integrado 74LS374

El integrado 74LS374 no es más que un conjunto de ocho biestables del tipo D con salida triestado, pensado para usos orientados al tratamiento de buses, que presenta una señal de reloj (CP) y una señal de habilitación (OE) común a todos.

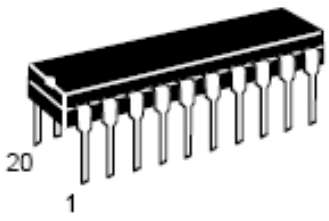

Figura 9.4: Encapsulado del integrado 74LS374.

El integrado 74LS374 presenta un total de 20 pines, distribuidos de la siguiente manera:

- 2 pines para alimentación (V_{cc}=5v) y masa (GND).
- 1 pin de habilitación de la señal de salida (OE) activo a nivel bajo.
- 1 pin de señal de reloj (CP) activa mediante el flanco de subida de dicha señal.
- 8 pines de salida de datos ($O_0 - O_7$).
- 8 pines de entrada de datos ($D_0 - D_7$).

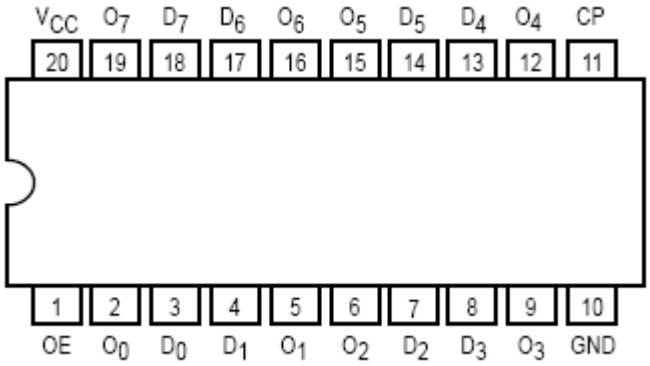

Figura 9.5: Diagrama de pines del integrado 74LS374.

El diagrama lógico que presenta este tipo de integrados, los 74LS374, se muestra a continuación:

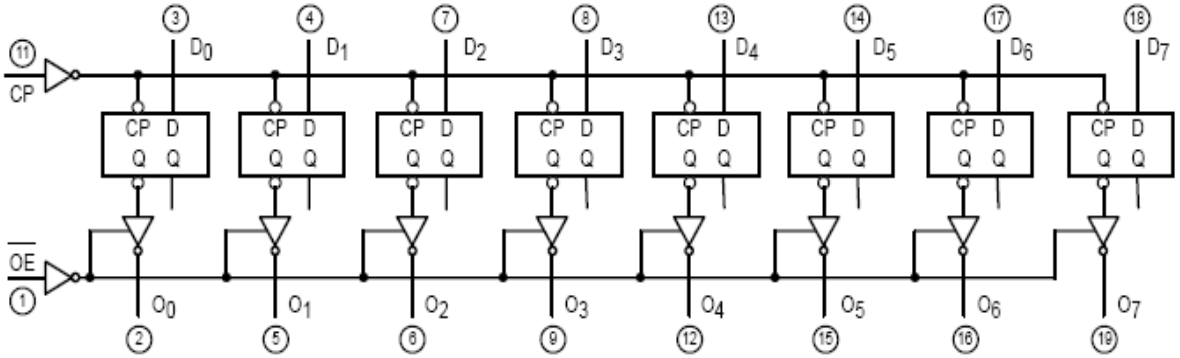

Figura 9.6: Diagrama lógico del integrado 74LS374.

Su funcionamiento se basa en una señal de reloj, cuando el integrado esta habilitado (pin OE activo a nivel bajo), las señales de entrada pasarán a la salida cuando nos llegue un flanco de subida de la señal de reloj por el pin CP, mientras el integado no este habilitado, éste presentará alta impedancia en su salida (Z^*). A continuación podemos ver la tabla de verdad de dicho integrado.

D_n	LE	$\overline{OE}$	O_n
H	⌐	L	H
L	⌐	L	L
X	X	H	Z*

Figura 9.7: Tabla de verdad del integrado 74LS374.

9.3. Diseño y simulación

Una vez expuestos los componentes de los que disponemos en la sección anterior, es hora de diseñar el circuito, para ello utilizamos la herramienta de diseño gráfico y simulación Proteus.

Primeramente diseñamos el registro de desplazamiento que queremos obtener utilizando los integrados 74LS374 de los que disponemos.

Queremos conseguir un registro de desplazamiento formado por 25 salidas de datos, una para cada columna del display expuesto anteriormente, con lo que necesitaremos un total de 4 integrados del tipo 74LS374, mientras las filas de dicho display estarán conectadas a una tensión negativa (-v) para una mejor visualización del display. Así cada vez que queramos iluminar un segmento del display tendremos que poner una diferencia de potencial en torno a 5 voltios en la patilla deseada, estara será alterna, conseguida mediante programación, por contra no iluminaremos un solo segmento, sino que iluminaremos 4 segmentos del display, debido a que hemos conectado las filas del display a masa, así que cuando pongamos una tensión en una patilla o pin que representa a una columna se iluminarán los 4 segmentos correspondientes a las 4 filas.

El funcionamiento del registro de desplazamiento debe de ser de tal forma que cada vez que demos un pulso de reloj (flanco de subida) por la patilla CP, los datos que se encuentren a la entrada del integrado pasen a su salida correspondiente.

La entrada del registro de desplazamiento será transmitida por el puerto paralelo del PC, al igual que la señal de reloj que gobierna este tipo de integrados.

El diseño realizado con la herramienta Proteus, incluyendo todas las características que debe presentar el registro de desplazamiento y su conexión con el puerto paralelo del PC y con el display, se puede observar a continuación.

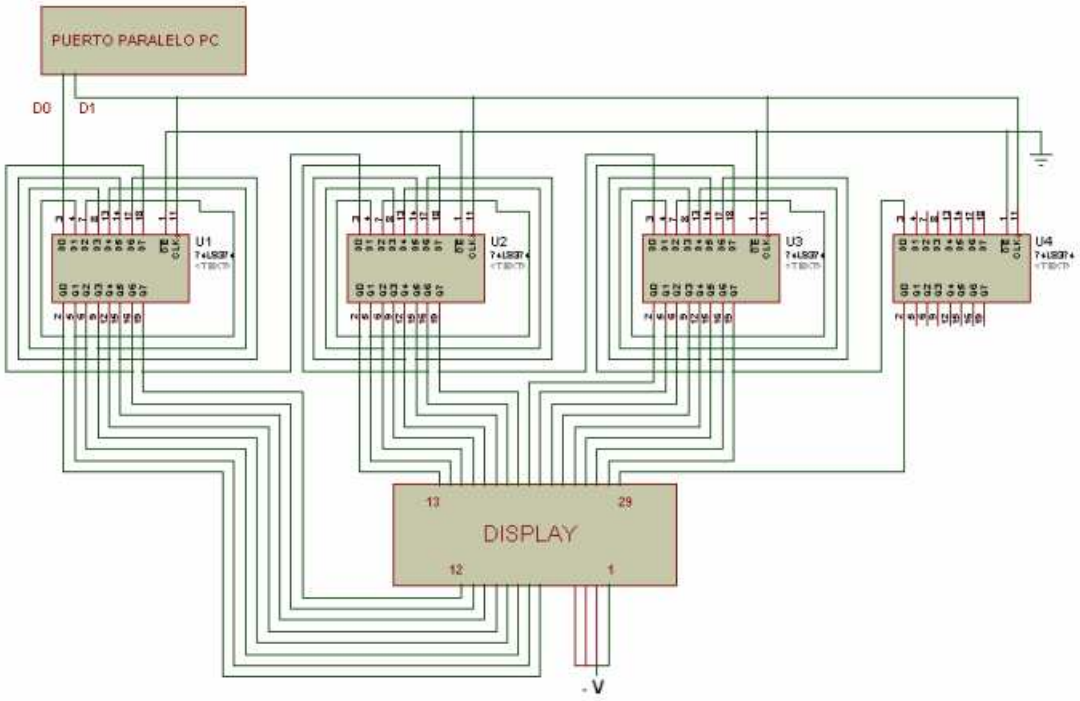

Figura 9.8: Esquema del circuito implementado.

Una vez diseñado el circuito, lo simulamos con Proteus y se observa que el diseño realizado es correcto y funciona correctamente, es decir, conseguimos todas las características que deseabamos.

9.4. Implementación del circuito

El siguiente paso en el diseño será la impementación del ciruito visto en la Figura 9.8. El circuito implementado se puede ver en la siguiente figura.

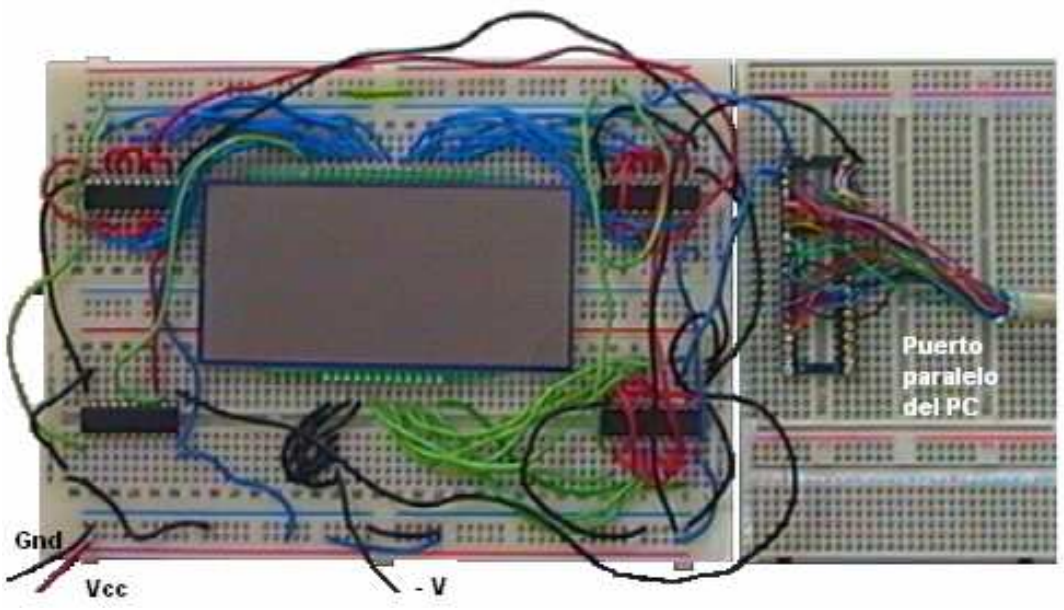

Figura 9.9: Implementación del circuito.

En dicho circuito se puede observar la disposición de las conexiones, y de las alimentaciones que disponemos expuestas en la sección 9.3.

9.5. Programación

El siguiente paso sera la programación, es decir, la creación de un programa que nos permita comprobar el correcto funcionamiento de un display dado, para ello se elabora un programa en C++ usando, además de las librerías estandares de dicho lenguaje de programación, la librería creada en el proyecto y ya vista anteriormente.

El programa realizado se puede dividir en tres partes principalmente:

- Programa principal.
- Funciones principales de muestreo, testeo y cambio de umbral.
- Función auxiliar de borrado de pantalla.

9.5.1. Programa principal

El programa principal se puede subdividir a su vez en dos partes, por un lado tenemos la cabecera de programa que incluye todas las librerías necesarias para la programación del ejemplo, como pueden ser las librerías imagen, video y pantalla creadas en dicho proyecto y otras librerías estándares del lenguaje de programación C++ como son stdlib, conio, stdio,...

Por otro lado tenemos la programación del menú principal, donde estan incluidas las definiciones de variables y de las funciones principales utilizadas a lo largo del programa principal, se inicializa el vídeo y se pone la pantalla en modo gráfico de 800x600 pixeles de resolución.

El menú principal es un bucle infinito, donde a partir de él podemos seleccionar lo que queremos hacer por medio de la intervención del usuario a partir de la pulsación de una tecla, así nos encontramos cuatro posibilidades:

- Pulsando la tecla numérica "1" saltamos a la función de muestreo del display.
- Pulsando la tecla numérica "2" saltamos a la función de testeo del display.
- Pulsando la tecla numérica "3" saltamos a la función de cambio de umbral para el testeo del display.
- Pulsando la tecla numérica "4" nos salimos de la ejecución del programa.

Si pulsamos cualquier otra tecla se repetirá el bucle, es decir, seguiremos en el menú principal esperando la pulsación de una tecla deseada.

Cuando nos salimos de la ejecución del programa, antes de salir, volvemos a poner la pantalla en modo gráfico.

9.5.2. Funciones principales

En el programa se han creado tres funciones principales como son el muestreo, testeo y cambio de umbral en el display para comprobar el correcto funcionamiento del display.

Función muestreo

Esta función realiza el muestreo de un display, es decir, toma las muestras o imágenes de un display que serán tomadas como referencia para un posterior testeo de displays.

La función presenta tres parámetros de entrada, una variable de clase tImagen BMP, que contiene la imagen con las letras necesarias para escribir texto en modo gráfico, una variable de clase tPantalla P, que indica el tipo de pantalla que tenemos y una variable de clase tVideo V, que contiene el vídeo empleado para la captura de imágenes.

El programa principal de dicha función primeramente borra la pantalla e inicaliza el display, es decir, apaga todos los segmentos del display. A continuación muestra la pantalla de colocación de la cámara y espera a la pulsación de una tecla y vuelve a borrar la pantalla.

A continuación pasamos a la parte de programación de toma de muestras, donde se configura dicha pantalla. Esta parte esta formada por un bucle de 27 repeticiones, que nos indica las 27 muestras que queremos tomar del display, una muestra con el reset del display, 25 muestras con los diferentes estados que puede presentar el display, una para cada pin, y una última muestra donde estan encendidos todos los segmentos del display.

Para la toma de cada muestra se construye un bucle infinito, donde aparecerá una pantalla que a la izquierda presenta la imagen capturada por la cámara de vídeo y a la derecha la imagen o muestra del display que queremos capturar, y esperará a la pulsación de una tecla, así si pulsamos la tecla s cuando la muestra de referencia tomada sea correcta, la guardaremos en un archivo con extension BMP y pasaremos a la muestra siguiente, si pulsamos cualquier otra tecla se actualizará la imagen o muestra capturada del display (imagen de la derecha de la pantalla).

Para la toma de cada muestra el display debe presentar un estado diferente, es decir, se tienen que iluminar determinados segmentos para cada muestra, esto se consigue mediante instrucciones for.

Para el correcto funcionamiento del display, la tensión que le llega a cada pin debe ser alterna, y lo conseguimos mediante programación o vía software, así cuando se ilumina un grupo de 4 segmentos correspondientes a un único pin del display lo que se hace es mandar un “1” al pin correspondiente durante un periodo de 10 milisegundos y a continuación resetear el display, con lo que los segmentos correspondientes se están apagando y encendiendo continuamente, pero el usuario ve estos segmentos como si estuvieran

continuamente encendidos, así es como se consigue una tensión alterna en el display para su correcto funcionamiento.

Una vez tomadas las 27 muestras se para la captura de vídeo y se vuelve al programa principal.

Función testeo

Esta función realiza el testeo de un display dado, es decir, toma las muestras o imágenes de un display y las compara con las muestras o imágenes tomadas como referencia en un proceso anterior de muestreo. Así nos indica la posibilidad de errores en el display, si dicho display presenta errores los mostrará por pantalla.

La función presenta cuatro parámetros de entrada, una variable de clase tImagen BMP, que contiene la imagen con las letras necesarias para escribir texto en modo gráfico, una variable de clase tPantalla P, que indica el tipo de pantalla que tenemos, una variable entera umbral, que indica el umbral requerido para que una muestra tomada sea correcta y una variable de clase tVideo V, que contiene el vídeo empleado para la captura de imágenes.

El programa principal de dicha función primeramente borra la pantalla e inicaliza el display, es decir, apaga todos los segmentos del display. A continuación muestra la pantalla de colocación de la cámara y espera a la pulsación de una tecla y vuelve a borrar la pantalla.

A continuación pasamos a la parte de programación de testeo del display, donde se configura dicha pantalla. Esta parte esta formada por un bucle de 27 repeticiones, que nos indica las 27 muestras que queremos comparar del display respecto a una serie de muestras de referencia y tomadas anteriormente en un proceso de muestreo.

Para el testeo de cada muestra se construye un bucle de 27 repeticiones, donde aparecerá una pantalla que a la izquierda presenta la imagen capturada por la cámara de vídeo, en la parte superior derecha la imagen capturada o muestra del display tomada y en la parte inferior derecha la muestra o imagen del display de referencia, tomada en un proceso anterior de muestreo. Durante el testeo el display estará encendido durante un tiempo de 100 msg y una vez realiza la captura el display se apagará, esto es así debido a que el display debe estar alimentado con una tensión alterna.

A cada muestra capturada se le resta la muestra referencia, esta muestra resultante es recorrida en todos sus valores para ver si alguna componente R, G ó B de dicha imagen supera el umbral establecido, si es así se produce un error y las imágenes referencia, capturada y resta son almacenadas en una matriz de imágenes para mostrar los errores pòsteriormente.

Una vez recorridas las 27 muestras, se para el vídeo y se borra la pantalla y se muestra una nueva pantalla indicando si ha habido o errores o no. Si el display no presenta errores se vuleve al programa principal. En caso contrario, se calcula una nueva imagen para cada error existente, dicha imagen presentará los pixeles que han superado el umbral en color rojo y ésta será almacenada en la matriz de imágenes mencionada anteriormente.

Por último se muestran todos las imágenes error en pantalla mediante un bucle infinito, para ello se crea una nueva pantalla, donde aparece en la parte superior izquierda la muestra referencia, en la parte inferior izquierda la muestra tomada o capturada, en la parte superior derecha la imagen diferencia y en la parte inferior izquierda la nueva imagen calculada con los errores indicados en color rojo. Esta última pantalla presenta una interfaz sencilla para el usuario, pulsando la tecla a pasamos al error anterior, pulsando la tecla b al error siguiente y si pulsamos la tecla s nos salimos de la muestra de errores y regresamos al programa principal.

Función cambio de umbral

Esta función realiza el cambio del umbral establecido para la comparación entre las muestras capturadas durante el testeo del display y las muestras referencia tomadas durante el muestreo del display.

La función presenta tres parámetros de entrada, una variable de clase tImagen BMP, que contiene la imagen con las letras necesarias para escribir texto en modo gráfico, una variable de clase tPantalla P, que indica el tipo de pantalla que tenemos y una variable entera umbral, que indica el umbral requerido para que una muestra tomada sea correcta y devuelve el nuevo umbral.

El programa principal de dicha función no es más que un bucle do-while, que pide al usuario que introduzca el nuevo umbral por teclado, cuya salida se establece cuando el umbral introducido es correcto, es decir, este comprendido entre 0 y 255. Cuando el umbral introducido es correcto regresamos al programa principal.

9.5.3. Función auxiliar

Es una única función de borrado de la pantalla (LimpiaPantalla) en modo gráfico, presenta dos parámetros de entrada, una variable de clase tImagen BMP, que contiene la imagen con las letras necesarias para escribir texto en modo gráfico, en nuestro caso con el espacio en blanco, una variable de clase tPantalla P, que indica el tipo de pantalla que tenemos.

El programa principal consta de dos bucles for, que lo que va haciendo es pintar el espacio en blanco para todas las filas y para todas las columnas.

9.5.4. Ejemplo gráfico

La realización y funcionamiento del programa realizado se puede observar de modo gráfico en la siguiente figura:

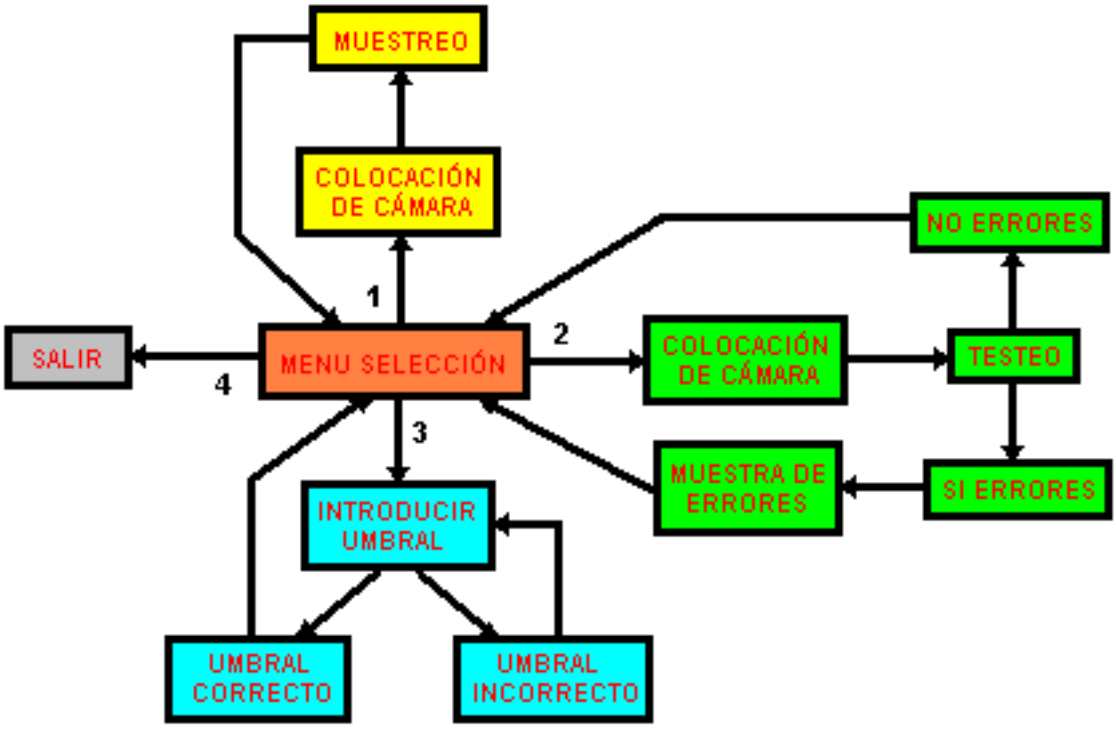

Figura 9.10: Ejemplo gráfico del programa realizado.

9.5.5. Listado del programa desarrollado

Listado 9.1: display.cpp

```
// Incluyo la cabecera de las clases empleadas
#include "pantalla.h"
#include "imagen.h"
#include "video.h"
#include "error.h"
// Cabeceras estandar
#include <iostream>
#include <stdlib.h>
#include <pc.h>
#include <conio.h>
#include <stdio.h>
#include <dos.h>
#include <ctype.h>
#include <math.h>
using namespace std;

int main()
{
   // Declaracion de funciones
   void muestreo(tImagen I, tPantalla P, tVideo V);
   void testeo(tImagen I, tPantalla P, int umbral, tVideo V);
   int cambiarumbral(tImagen I,tPantalla P,int umbral);
   void LimpiaPantalla(tImagen I, tPantalla P);
   // Declaracion de variables
   tVideo V;    // Declaracion de la variable de video
   tPantalla P; // Declaracion de la variable de pantalla
   tImagen BMP; // Llevara el Bmp con las letras
   char opcion;   // Declaracion de la variable de lectura del teclado
   int umbral=40; // Declaracion de la variable umbral
   int k;
   BMP.CargaBMP("letras.bmp"); // BMP contiene el archivo de letras
   V.InicializaVideo();        // Inicializacion del video
   P.PonModoVESA(800,600);     // Inicializacion de la pantalla

   for ( ; ; )   // Bucle infinito que muestra la pantalla de seleccion
   {
      BMP.Escribe("Bienvenido␣al␣testeo␣de␣displays",150,50,0,P);
      BMP.Escribe("Seleccione␣lo␣que␣quiere␣hacer",180,200,0,P);
      BMP.Escribe("1␣Tomar␣muestras␣del␣display",210,250,0,P);
      BMP.Escribe("2␣Testear␣display",210,300,0,P);
      BMP.Escribe("3␣Cambiar␣el␣umbral",210,350,0,P);
      BMP.Escribe("4␣Salir",210,400,0,P);
      opcion=getche(); // Espera hasta que es pulsada una tecla

      switch(opcion) // Segun la tecla pulsada saltamos a una funcion
      {              // o a otra
         case '1': muestreo(BMP,P,V); break;
         case '2': testeo(BMP,P,umbral,V); break;
         case '3': umbral=cambiarumbral(BMP,P,umbral); break;
         case '4': P.PonModoTexto(); exit(0);
      }
      LimpiaPantalla(BMP,P);  // Borrado de pantalla
   } // termina bucle for infinito*/
   return 0;
}

// Funcion que muestrea el display que tomamos como referencia, es decir,
// va tomando como muestras las 27 diferentes imagenes que puede mostrar
// el display y las va almacenando en archivos de extension BMP.

void muestreo(tImagen I, tPantalla P, tVideo V)
{
   // Declaracion de funciones
   void LimpiaPantalla(tImagen I, tPantalla P);
   // Declaracion de variables
   tImagen J;
   int i,j,k,h;
   char opcionmuestreo,numeromuestra[2];
```

```
      LimpiaPantalla(I,P); // Borra la pantalla

      for(k=1;k<=25;k++)
74    {
         outp(0x378,0);   // Reset o inicializacion del display
         outp(0x378,2);
      }
      outp(0x378,0);
79    delay(500);

      // Pantalla presentacion del modo muestreo, que pide colocar la
      // camara de video en posicion correcta, la colocacion del display
      // esta remarcada con cuatro señales en pantalla, para un perfecto
84    // funcionamiento de la toma de muestras.

      I.Escribe("Modo muestreo",295,50,0,P);
      V.Captura(P,200,150,400,300,1); // Arranca el video
      I.Escribe("M",184,212,0,P);
89    I.Escribe("M",600,212,0,P);     // Colocacion de las marcas en pantalla
      I.Escribe("M",184,370,0,P);
      I.Escribe("M",600,370,0,P);
      I.Escribe("Situe la camara y pulse una tecla",135,525,0,P);
      getkey();    // Esperamos hasta que se pulse una tecla
94    V.Paralo();  // Paramos el video

      LimpiaPantalla(I,P); // Borramos la pantalla

      // Va tomando las muestras y guardandolas en un archivo BMP, tenemos
99    // un total de 27 muestras, una muestra con el reset del display,
      // 25 muestras con los estados del display y una ultima muestra donde
      // estan encendidos todos los segmentos del display

      I.Escribe("Modo muestreo",295,50,0,P);
104   I.Escribe("Pulse s cuando la muestra sea correcta",100,475,0,P);
      I.Escribe("Pulse cualquier otra tecla para la",100,515,0,P);
      I.Escribe("actualizacion de la muestra",100,546,0,P);
      I.Escribe("Muestra",530,375,0,P);
      V.Captura(P,0,150,400,300,1);
109   delay(1000);

      for(k=1;k<=27;k++)
      {
         h=0;
114      itoa(k,numeromuestra,10);     // se sale del bucle
         I.Escribe(numeromuestra,650,375,1,P);
         for ( ; ; )
         {
            if (k!=1)
119         {
               if (k!=27)
               {
                  outp(0x378,1);
                  outp(0x378,3); // Dejo pasar un 1
124               outp(0x378,0);
                  for (j=1;j<k-1;j++)
                  {
                     outp(0x378,2); // Paso el 1 hasta el pin correspondiente
                     outp(0x378,0);
129               }
               }
               else
               {                          // Para el caso de que haya que encender
                  for (j=1;j<=26;j++) // todos los segmentos del display
134               {
                     outp(0x378,3);
                     outp(0x378,0);
                  }
               }
139         };
            if (h==0)        // Esperamos 120 mseg para la primera
            {                // actualizacion
               delay(120);
               J.Capturalo(5,218,390,150,P);
```

```
144             J.Dibujalo(410,225,P);
                }
                else delay (10); // Para que se vea continuamente el segmento
                h++;             // del display encendido
                if (kbhit())               // Si se pulsa una tecla se actualiza
149             {                          // la captura y se recoge la tecla
                   opcionmuestreo=getch(); // pulsada
                   J.Capturalo(5,218,390,150,P);
                   J.Dibujalo(410,225,P);
                };
154             for (j=1;j<=25;j++)
                {
                   outp(0x378,2);     // Reset o apagado del display
                   outp(0x378,0);
                }
159             if (opcionmuestreo=='s') break;
             } // bucle infinito

             opcionmuestreo='␣';
             if (k==1) J.SalvaBMP("m1.bmp");  // Va guardando la muestra en el
164          if (k==2) J.SalvaBMP("m2.bmp");  // BMP correspondiente
             if (k==3) J.SalvaBMP("m3.bmp");
             if (k==4) J.SalvaBMP("m4.bmp");
             if (k==5) J.SalvaBMP("m5.bmp");
             if (k==6) J.SalvaBMP("m6.bmp");
169          if (k==7) J.SalvaBMP("m7.bmp");
             if (k==8) J.SalvaBMP("m8.bmp");
             if (k==9) J.SalvaBMP("m9.bmp");
             if (k==10) J.SalvaBMP("m10.bmp");
             if (k==11) J.SalvaBMP("m11.bmp");
174          if (k==12) J.SalvaBMP("m12.bmp");
             if (k==13) J.SalvaBMP("m13.bmp");
             if (k==14) J.SalvaBMP("m14.bmp");
             if (k==15) J.SalvaBMP("m15.bmp");
             if (k==16) J.SalvaBMP("m16.bmp");
179          if (k==17) J.SalvaBMP("m17.bmp");
             if (k==18) J.SalvaBMP("m18.bmp");
             if (k==19) J.SalvaBMP("m19.bmp");
             if (k==20) J.SalvaBMP("m20.bmp");
             if (k==21) J.SalvaBMP("m21.bmp");
184          if (k==22) J.SalvaBMP("m22.bmp");
             if (k==23) J.SalvaBMP("m23.bmp");
             if (k==24) J.SalvaBMP("m24.bmp");
             if (k==25) J.SalvaBMP("m25.bmp");
             if (k==26) J.SalvaBMP("m26.bmp");
189          if (k==27) J.SalvaBMP("m27.bmp");
             I.Escribe("␣␣",650,375,1,P);
          }
          V.Paralo(); // Para el video
       }
194
       // Funcion que testea un display, comparandolo con otro que tomemos como
       // referencia, es decir, va comparando las 27 diferentes imagenes que
       // puede tomar un display con las 27 imagenes que tenemos de referencia

199    void testeo(tImagen I, tPantalla P,int umbral, tVideo V)
       {
          // Declaracion de funciones
          void LimpiaPantalla(tImagen I, tPantalla P);
          // Declaracion de variables
204       tImagen B,C,R;
          tImagen errorimagenes[28][4]; // Matriz de imagenes
          int k,l,j,m,numbyte,error,numerodeerrores=0,h=1;
          int vector[28],contador=1;
          char totalerrores[3],numeromuestra[3],numeroerror[3];
209       char numeropin[3],opcionerror;

          LimpiaPantalla(I,P); // Borramos la pantalla

          for(k=1;k<=25;k++)
214       {
             outp(0x378,0);   // Reset o inicializacion del display
             outp(0x378,2);
          }
```

```
      outp(0x378,0);
219   delay(500);

      // Pantalla presentacion del modo testeo, que pide colocar la
      // camara de video en posicion correcta, la colocacion del display
      // esta remarcada con cuatro señales en pantalla, para un perfecto
224   // funcionamiento del testeo de muestras.

      I.Escribe("Modo␣testeo",310,50,0,P);
      V.Captura(P,200,150,400,300,1);
      I.Escribe("M",184,212,0,P);
229   I.Escribe("M",600,212,0,P);
      I.Escribe("M",184,370,0,P);
      I.Escribe("M",600,370,0,P);
      I.Escribe("Situe␣la␣camara␣y␣pulse␣una␣tecla",135,525,0,P);
      getkey();   // Esperamos hasta que se pulse una tecla
234   V.Paralo(); // Para el video

      // Vamos capturando las 27 muestras y las vamos comparando con las
      // muestras que tenemos de referencia, si alguna muestra presenta
      // error se van almacenando en una matriz de errores.
239
      LimpiaPantalla(I,P); // Borra la pantalla
      I.Escribe("Modo␣testeo",310,50,0,P);
      I.Escribe("Imagen␣tomada",490,115,0,P);
      I.Escribe("Imagen␣referencia",470,463,0,P);
244   V.Captura(P,0,150,400,300,1); // Arranca el video
      delay(1500);

      h=1;
      for (k=1;k<=27;k++)  // Comparamos las 27 muestras
249   {
         error=0;
         if (k==1) B.CargaBMP("m1.bmp");   // Va cargando los diferentes BMP's
         if (k==2) B.CargaBMP("m2.bmp");
         if (k==3) B.CargaBMP("m3.bmp");
254      if (k==4) B.CargaBMP("m4.bmp");
         if (k==5) B.CargaBMP("m5.bmp");
         if (k==6) B.CargaBMP("m6.bmp");
         if (k==7) B.CargaBMP("m7.bmp");
         if (k==8) B.CargaBMP("m8.bmp");
259      if (k==9) B.CargaBMP("m9.bmp");
         if (k==10) B.CargaBMP("m10.bmp");
         if (k==11) B.CargaBMP("m11.bmp");
         if (k==12) B.CargaBMP("m12.bmp");
         if (k==13) B.CargaBMP("m13.bmp");
264      if (k==14) B.CargaBMP("m14.bmp");
         if (k==15) B.CargaBMP("m15.bmp");
         if (k==16) B.CargaBMP("m16.bmp");
         if (k==17) B.CargaBMP("m17.bmp");
         if (k==18) B.CargaBMP("m18.bmp");
269      if (k==19) B.CargaBMP("m19.bmp");
         if (k==20) B.CargaBMP("m20.bmp");
         if (k==21) B.CargaBMP("m21.bmp");
         if (k==22) B.CargaBMP("m22.bmp");
         if (k==23) B.CargaBMP("m23.bmp");
274      if (k==24) B.CargaBMP("m24.bmp");
         if (k==25) B.CargaBMP("m25.bmp");
         if (k==26) B.CargaBMP("m26.bmp");
         if (k==27) B.CargaBMP("m27.bmp");

279      if (k!=1)          // Va encendiendo el display
         {
            if (k!=27)
            {
               outp(0x378,1);
284            outp(0x378,3); // Dejo pasar un 1
               outp(0x378,0);
               for (j=1;j<k-1;j++)
               {
                  outp(0x378,2); // Paso el 1 hasta el pin correspondiente
289               outp(0x378,0);
               }
            }
```

```
            else
            {                          // Para el caso de que haya que encender
294             for (j=1;j<=26;j++) // todos los segmentos del display
                {
                    outp(0x378,3);
                    outp(0x378,0);
                }
299         }
        };
        if (h<27) delay(100); // Espera 100 msg para tomar la muestra o
        else delay(300);      // 300 msg para la ultima muestra

304     // C contendra la imagen capturada por el video, es decir, la muestra
        // real a comparar, B contendra la imagen de referencia, en R
        // tendremos la resta de C-B. Una vez capturada la muestra el display
        // se apaga

309     C.Capturalo(5,218,390,150,P);

        for(j=1;j<=25;j++)
        {
            outp(0x378,0);   // Reset o inicializacion del display
314         outp(0x378,2);
        }

        h++;
        C.Dibujalo(410,150,P);
319     B.Dibujalo(410,310,P);
        R.Copia(C);
        R.Resta(B);
        R.Escala(0.2);
        numbyte=0;
324
        // El proceso de comparacion de dos muestras consiste en comprobar
        // si la resta entre las dos imagenes a comparar (R=C-B) esta
        // comprendida dentro de un umbral, es decir, los pixeles del
        // resultado de la resta estan contenidos en un determinado intervalo
329     // que definimos desde 0 hasta un cierto umbral.
        // Se realiza un escalamiento de la imagen resta para que el
        // proceso de comparacion sea mas rapido

        for (l=0;l<(R.TotalBytes/4);l++)
334     {
            for (m=0;m<3;m++)
            {
                if (R.M[numbyte]>umbral) // Si hay error se guardan en la
                {                        // matriz de imagenes B, C y R.
339                 I.Escribe("Error␣de␣display",250,550,0,P);
                    numerodeerrores++;
                    errorimagenes[k][0].Copia(B);
                    errorimagenes[k][1].Copia(C);
                    errorimagenes[k][2].Copia(C);
344                 errorimagenes[k][2].Resta(B);
                    vector[contador]=k;
                    contador++;
                    error=1;
                    break;       // Si presenta error se sale del bucle de
349             }                // comparacion
                numbyte++;
            }
            numbyte++;
            if (error==1) break;
354     }
        I.Escribe("␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣",250,550,1,P);
    }

    V.Paralo();  // Para el video
359 LimpiaPantalla(I,P); // Borramos la pantalla

    // Tratamiento de errores
    // Si no ha habido errores se muestra por pantalla un mensaje de ello
    if (numerodeerrores==0)
364 {
        I.Escribe("No␣ha␣habido␣errores",150,200,0,P);
```

```
         I.Escribe("Display␣correcto",150,250,0,P);
         I.Escribe("Pulse␣una␣tecla␣para␣continuar",150,300,0,P);
         getkey(); // Espera a la pulsacion de una tecla
369   }
      // Si ha habido errores indica primeramente el numero de errores
      // totales que ha habido
      else
      {
374      I.Escribe("Ha␣habido␣errores",150,200,0,P);
         itoa(numerodeerrores,totalerrores,10);
         I.Escribe("Tenemos",150,250,0,P);
         I.Escribe(totalerrores,280,250,0,P);
         I.Escribe("errores",325,250,0,P);
379      I.Escribe("Pulse␣una␣tecla␣para␣continuar",150,300,0,P);
         getkey(); // Espera a la pulsacion de una tecla
         LimpiaPantalla(I,P); // Borra la pantalla

         // Calculamos las imagenes error, estas se mostraran en unas
384      // nuevas imagenes resaltando en rojo los pixeles que superan
         // el umbral

         error=0;
         for (k=1;k<=numerodeerrores;k++)
389      {
            errorimagenes[vector[k]][3].Copia(errorimagenes[vector[k]][2]);
            for (l=0;l<(errorimagenes[vector[k]][3].TotalBytes/4);l++)
            {
               numbyte=4*l;
394            for (m=0;m<3;m++)
               {
                  if (errorimagenes[vector[k]][3].M[numbyte]>umbral)
                  {
                     error=1;
399                  break;
                  }
                  numbyte++;
               }
               if (error==1)
404            {
                  numbyte=4*l;
                  errorimagenes[vector[k]][3].M[numbyte]=0;
                  errorimagenes[vector[k]][3].M[numbyte+1]=0;
                  errorimagenes[vector[k]][3].M[numbyte+2]=255;
409               error=0;
               }
            }
         }

414      contador=1;
         // Va mostrando todos los errores encontrados por pantalla, es decir,
         // se muestran las imagenes capturada, de referencia y de resta para
         // cada muestra tomada que ha presentado error
         I.Escribe("Muestra␣referencia",50,110,0,P);
419      I.Escribe("Muestra␣tomada",80,480,0,P);
         I.Escribe("Muestra␣diferencia",440,110,0,P);
         I.Escribe("Errores",525,480,0,P);
         for ( ; ; )
         {
424         I.Escribe("a␣anterior␣␣␣␣␣b␣siguiente␣␣␣␣␣s␣salir",0,560,0,P);
            if (contador==1) I.Escribe("␣␣␣␣␣␣␣␣␣␣",0,560,1,P);
            if (contador==numerodeerrores) I.Escribe("␣␣␣␣␣␣␣␣␣␣␣␣␣",240,560,1,P);
            I.Escribe("Error␣␣␣␣de",300,50,0,P);
            itoa(contador,numeroerror,10);
429         I.Escribe(numeroerror,400,50,0,P);
            itoa(numerodeerrores,numeroerror,10);
            I.Escribe(numeroerror,490,50,0,P);
            errorimagenes [vector[contador]][0].Dibujalo(0,150,P);
            errorimagenes [vector[contador]][1].Dibujalo(0,320,P);
434         errorimagenes [vector[contador]][2].Dibujalo(405,150,P);
            errorimagenes [vector[contador]][3].Dibujalo(405,320,P);
            I.Escribe("Muestra",280,520,0,P);
            itoa(vector[contador],numeromuestra,10);
            I.Escribe(numeromuestra,405,520,0,P);
439         if (vector[contador]!=1 and vector[contador]!=27)
```

```
            {
               I.Escribe("Pin",460,520,0,P);
               itoa(vector[contador]+3,numeropin,10);
               I.Escribe(numeropin,525,520,0,P);
            }
            opcionerror=getche();
            if (opcionerror=='a' and contador!=1)
               contador--;
            if (opcionerror=='b' and contador!=numerodeerrores)
               contador++;
            if (opcionerror=='s')
               break;
            I.Escribe("␣␣␣",400,50,1,P);
            I.Escribe("␣␣␣",405,520,1,P);
            I.Escribe("␣␣␣␣␣␣",460,520,1,P);
         }
      }
   }

// Funcion que nos sirve para la introduccion del umbral por teclado

   int cambiarumbral(tImagen I,tPantalla P,int umbral)
   {
      // Declaracion de funciones
      void LimpiaPantalla(tImagen I, tPantalla P);
      // Declaracion de variables
      int nuevoumbral,i,error=0;
      char opcionumbral[3],umbralseleccionado[3];

      // Se pide que se introduzca el nuevo umbral por teclado, este si
      // no esta comprendido entre 0 y 255 dara error

      do
      {
         LimpiaPantalla(I,P); // Borra la pantalla
         I.Escribe("Cambio␣de␣umbral",280,50,0,P);
         I.Escribe("El␣umbral␣que␣tenemos␣es␣",70,200,0,P);
         itoa(umbral,umbralseleccionado,10);
         I.Escribe(umbralseleccionado,470,200,0,P);
         I.Escribe("El␣nuevo␣umbral␣debe␣estar␣comprendido",70,250,0,P);
         I.Escribe("entre␣0␣y␣255",70,300,0,P);
         I.Escribe("Seleccione␣el␣nuevo␣umbral",70,350,0,P);
         for (i=0;i<3;i++) // Va leyendo las teclas pulsadas y mostrandolas
         {                 // por pantalla
             opcionumbral[i]=getche();
             I.Escribe("El␣umbral␣seleccionado␣es",70,400,0,P);
             I.Escribe(opcionumbral,500,400,0,P);
         }
         nuevoumbral=atoi(opcionumbral);
         if (nuevoumbral<0 or nuevoumbral>255) // Si el umbral no es correcto
         {                                     // dara un error
            error=1;
            opcionumbral[0]=0;opcionumbral[1]=0;opcionumbral[2]=0;
            I.Escribe("Umbral␣incorrecto",70,450,0,P);
         }
         else
         {
            I.Escribe("Umbral␣correcto",70,450,0,P);
            error=0;
         }
         I.Escribe("Pulse␣una␣tecla␣para␣continuar",70,500,0,P);
         getkey(); // Espera a la pulsacion de una tecla
      } while (error==1);
      return nuevoumbral;
   }

   // Funcion que borra la pantalla en modo grafico a traves de un archivo
   // BMP

   void LimpiaPantalla(tImagen I, tPantalla P)
   {
      // Definicion de variables
      int i,j;
```

```
514     // Se va pintando en pantalla de izquierda a derecha y de arriba a
        // abajo un espacio en blanco

        for (i=0;i<19;i++)
        {                               // Limpia pantalla
519         for (j=0;j<50;j++)
               I.Escribe("␣",j*16,i*31,1,P);
        }
        return;
    }
```

9.6. Ejemplo práctico

La ejecución del programa creado se realiza desde MS-DOS, en la línea de comandos escribimos **display** y pulsamos el enter, así entramos en el menú de selección que se muestra en la siguiente figura.

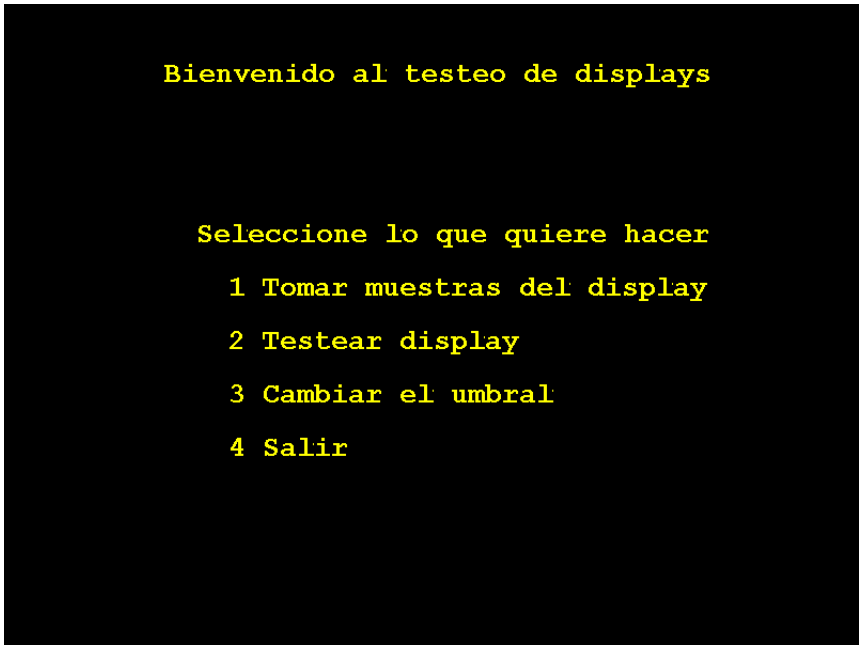

Figura 9.11: Menú de selección.

Este menú presenta cuatro opciones y esperará a la pulsación de una tecla. Las cuatro opciones que presenta el menú se enumerán a continuación:

- Muestreo del display: pulsando la tecla “1” (Ver sección 9.6.1).
- Testeo del display: pulsando la tecla “2” (Ver sección 9.6.2).
- Cambio del umbral de comparación: pulsando la tecla “3” (Ver sección 9.6.3).

- Salir: pulsando la tecla “4”, salimos del programa.

A tener en cuenta, si no se pulsa ninguna de las teclas mencionadas anteriormente, el programa no hará nada, es decir, nos mantendremos en el menú de selección indefinidamente, hasta que se pulse una tecla deseada.

9.6.1. Muestreo del display

En esta opción se reliza la toma de muestras de un display dado, que tomaremos como referencia y que serán guardadas en archivos con extensión BMP, para las comparaciones de otros displays posteriormente y que se realizará en la fase de testeo.

Una vez pulsada la opción 1 del menú de selección nos encontramos con una nueva pantalla que presenta la forma que se muestra en la siguiente figura.

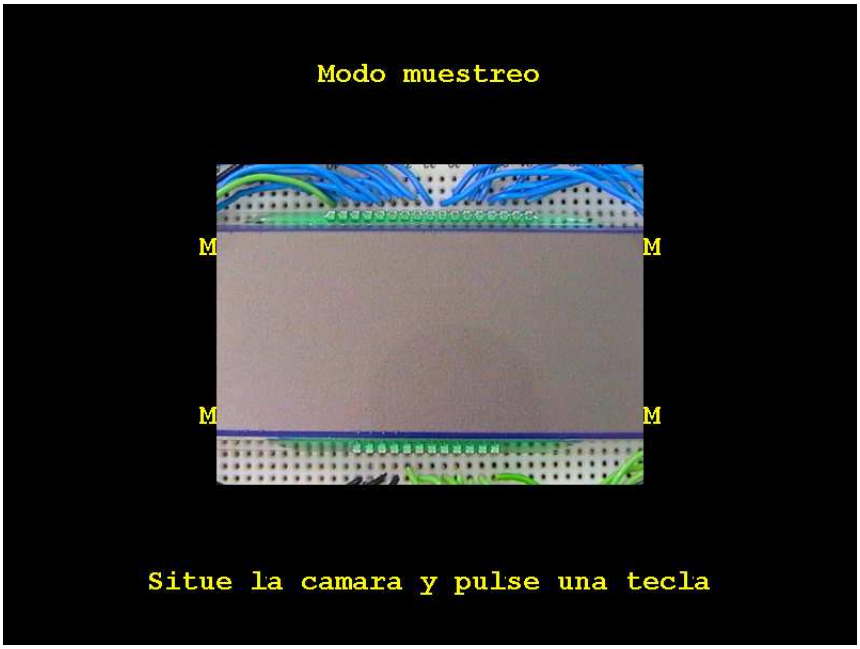

Figura 9.12: Colocación de cámara en el modo muestreo.

En esta situación tenemos que posicionar la cámara sobre el display, de tal forma que el display quede delimitado por las muescas que aparecen en la pantalla para un correcto funcionamiento de la rutina de muestreo. Una vez posicionada la cámara se pulsa una tecla para continuar y comienza verdaderamente el muestreo o toma de muestras del display. Nos aparecerá una pantalla como se muestra en la Figura 9.13.

Figura 9.13: Pantalla de muestreo del display.

En esta pantalla aparece a la izquierda la captura de vídeo en tiempo real y a la derecha la imagen capturada y que servirá de referencia. Se capturarán un total de 27 muestras, la primera con todos los segmentos del display apagados, después 25 muestras encendiendo los 4 segmentos correspondientes a cada pin del display, desde el pin 4 al pin 29 y por último, se capturará una imagen con todos los segmentos del display encendidos.

Mientras estemos capturando una imagen, ésta no se capturará en un archivo de extensión BMP, hasta que se pulse la letra "s", si durante este proceso se pulsa cualquier otra tecla la imagen a capturar (imagen derecha) se actualizará dependiendo de lo que estemos capturando con la camara de vídeo (imagen izquierda).

Cuando ya se hayan tomado las 27 muestras se vuelve al menú de selección visto en la Figura 9.11.

9.6.2. Testeo del display

En esta opción se reliza el testeo de un display dado, a partir de la comparación de las imágenes capturadas de dicho display con las imágenes tomadas como referencia en una fase de muestreo anteriormente realizada.

Una vez pulsada la opción 2 del menú de selección nos encontramos con una nueva pantalla que presenta la forma que se muestra en la siguiente figura.

Figura 9.14: Colocación de cámara en el modo testeo.

En esta situación tenemos que posicionar la cámara sobre el display, de tal forma que el display quede delimitado por las muescas que aparecen en la pantalla para un correcto funcionamiento de la rutina de testeo.

Una vez posicionada la cámara se pulsa una tecla para continuar y comienza el testeo del display. Nos aparecerá una pantalla como se muestra en la Figura 9.15.

En esta pantalla aparece a la izquierda la captura de vídeo en tiempo real y a la derecha en la parte superior veremos la imagen capturada y en la parte inferior veremos la imagen referencia.

Se irán viendo en tiempo real las diferentes imágenes capturadas y de referencia, atendiendo a las 27 muestras que se comparán y que ya se explicaron anteriormente. Una vez acabado el testeo del display nos podemos encontrar con dos posibles situaciones, que haya o no errores en el display testeado.

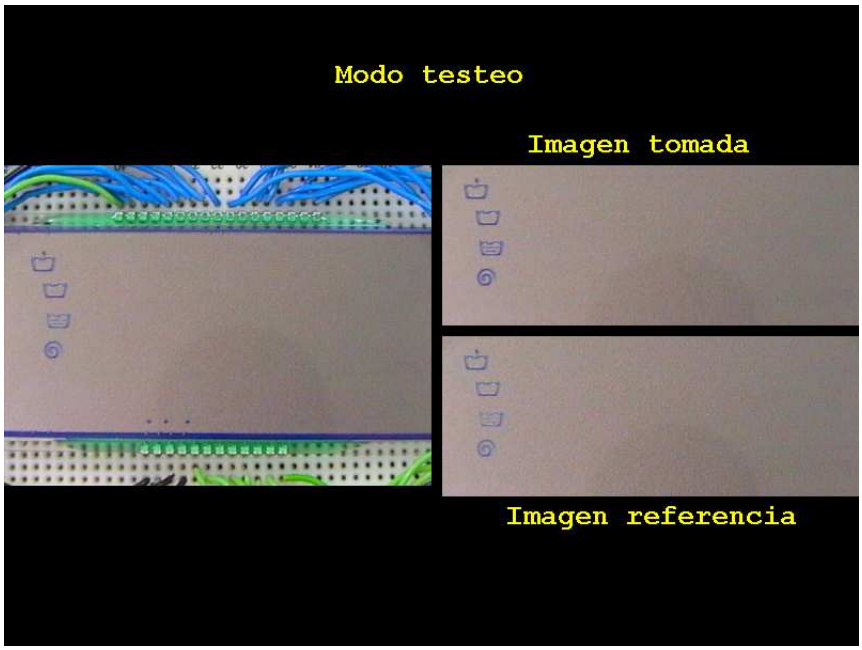

Figura 9.15: Pantalla de testeo del display.

No hay errores

Si el display testeado no presenta errores aparecerá una pantalla como se muestra en la siguiente figura.

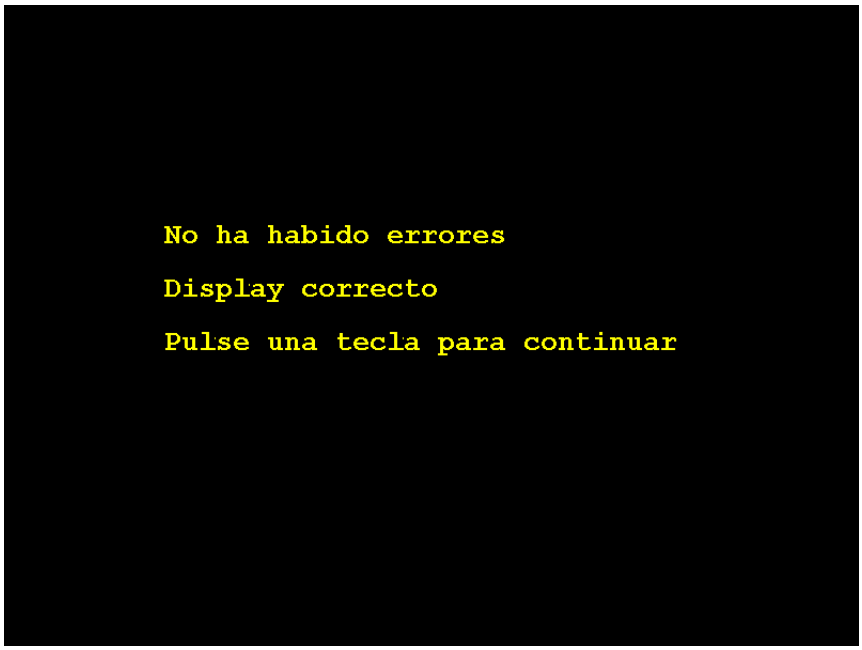

Figura 9.16: Pantalla que se muestra cuando no hay errores

Esperará la pulsación de una tecla y volvemos al menú de seleccion visto en la Figura 9.11.

Si hay errores

Si el display testeado presenta errores aparecerá una pantalla como se muestra en la siguiente figura, donde se indican el total de errores que ha habido.

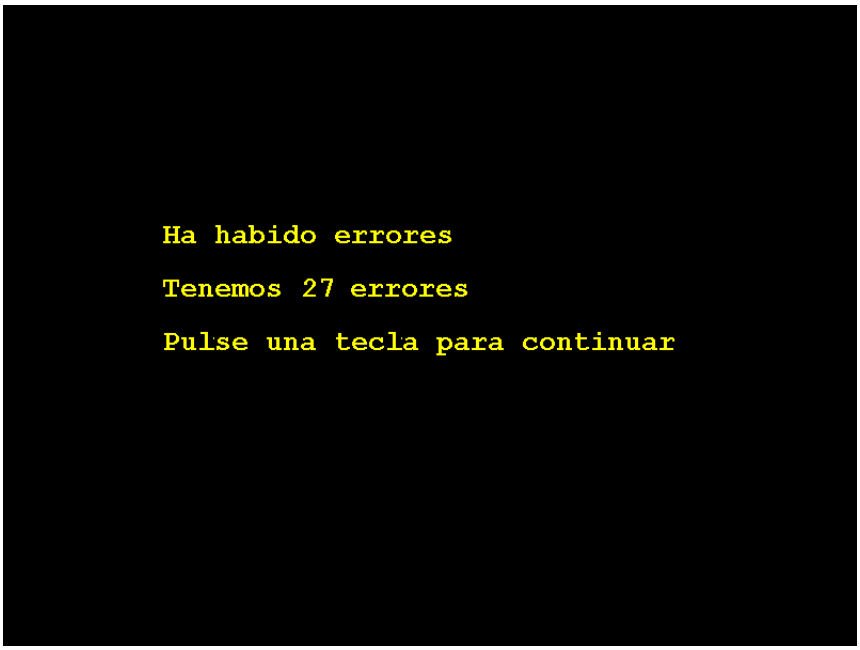

Figura 9.17: Pantalla que se muestra cuando hay errores.

A continuación pulsamos una tecla y se muestran todos los errores que ha habido uno a uno. Aparecerá una pantalla formada por cuatro imágenes o muestras, en la parte superior izquierda aparecerá la muestra que tenemos como referencia, en la parte inferior izquierda la muestra capturada durante el testeo del display, en la parte superior derecha la muestra diferencia entre las muestras capturada y referencia y por último en la parte inferior derecha una muestra en la que los pixeles, que han superado el umbral especificado o introducido por el usuario, aparecen en rojo. Dicha situación se puede observar en la Figura 9.18.

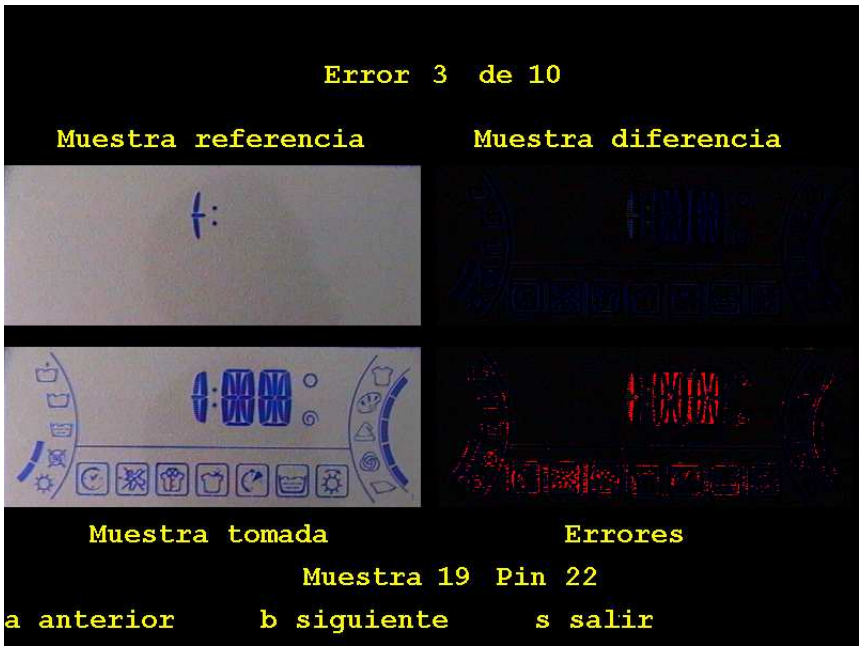

Figura 9.18: Muestra de errores.

En dicha pantalla si pulsamos la tecla "a" pasaremos a visualizar el error anterior, si pulsamos la tecla "b" pasaremos a visualizar el error siguiente y si pulsamos la tecla "s" salimos de la visualización de errores y volvemos al menú de selección visto anteriormente en la Figura 9.11.

9.6.3. Cambio de umbral de comparación de imágenes

En esta opción se realiza el cambio de umbral ya explicado en secciones anteriores, una vez seleccionada dicha opción nos aparecerá una pantalla como se muestra en la Figura 9.19 y el usuario deberá introducir el umbral que requiera para su aplicación o para su testeo de displays. A partir de aquí tenemos dos posibilidades respecto al umbral introducido por el usuario, así:

- el umbral introducido por el usuario será correcto si éste esta comprendido entre los valores de 0 y 255, correspondientes a los valores que pueden tomar la representación de cada componente de color R, G ó B. En este caso se mostrará la pantalla que aparece en la Figura 9.20.
- en caso contrario, el umbral introducido por el usuario será incorrecto y se mostrára la pantalla que aparece en la Figura 9.21.

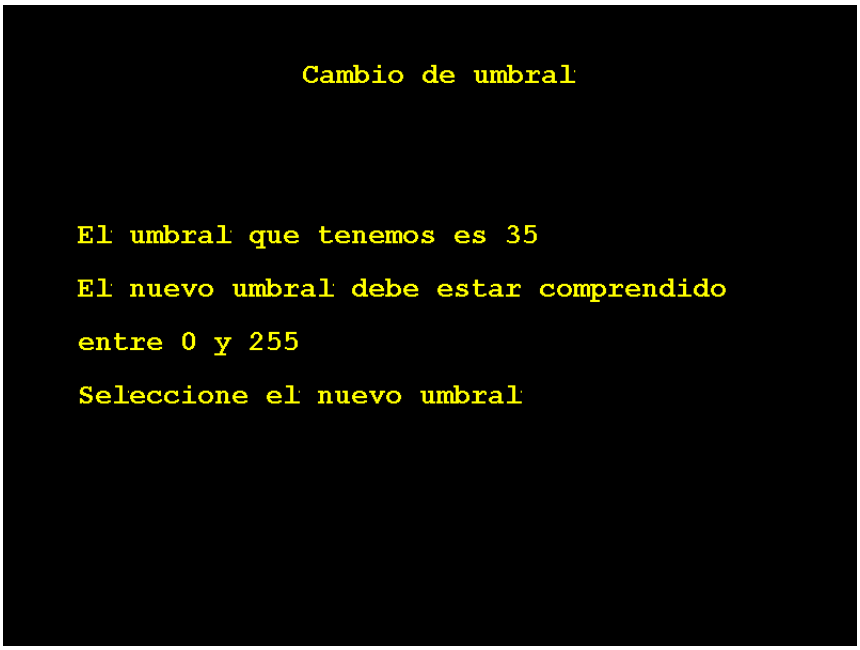

Figura 9.19: Pantalla de selección de umbral.

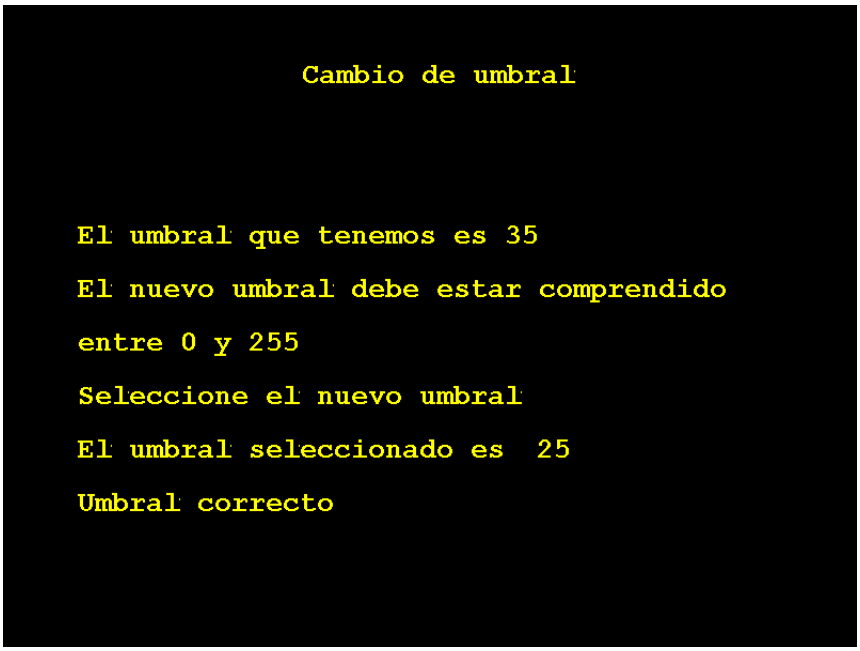

Figura 9.20: Ejemplo de umbral correcto.

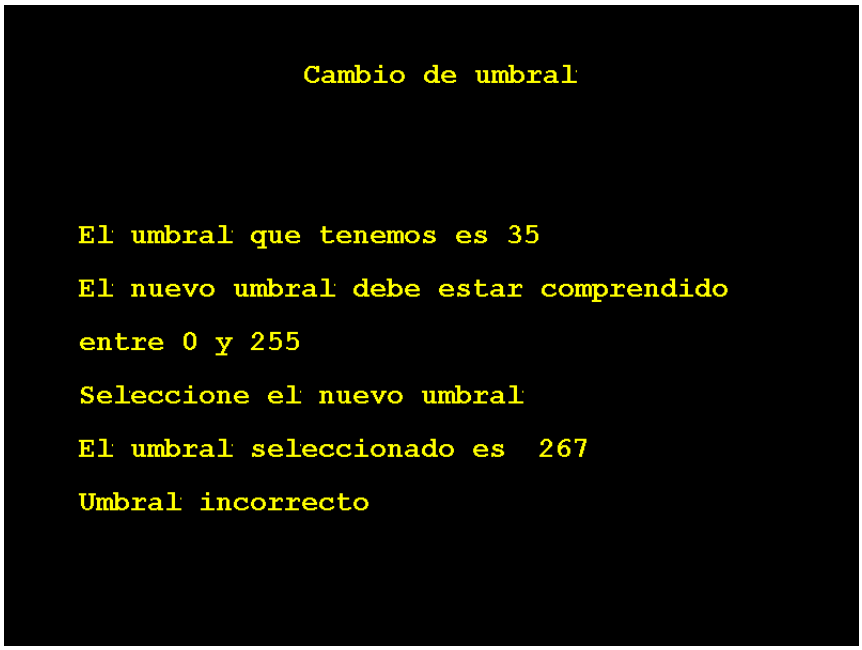

Figura 9.21: Ejemplo de umbral incorrecto.

Para ambas situaciones, vistas anteriormente, se espera a la pulsación de una tecla, una vez pulsada si el umbral introducido es correcto volvemos al menú principal (Figura 9.11) y en caso contrario volvemos al menú de selección de umbral(Figura 9.19).

Parte IV

Apéndice

Apéndice A

Archivos generados

Se presentan los diferentes listados de los archivos generados en la construcción de la biblioteca para el tratamiento de vídeo en tiempo real.

Primeramente veremos el listado de los archivos que tienen que ver con el tratamiento de vídeo, los archivos video.h y video.cpp, donde aparece la clase tVideo ya definida en el capítulo 6 con sus principales variables y funciones.

Seguidamente se mostrarán los listados de los archivos referente al tratamiento de imágenes, como son los archivos imagen.h e imagen.cpp, donde aparece la clase tImagen ya definida en el capítulo 7 con sus principales variables y funciones.

Más tarde se expondrán los listados de los archivos que hacen referencia a la visualización por pantalla, como son los archivos pantalla.h y pantalla.cpp, donde aparece la clase tPantalla ya defindia en el capítulo 8 con sus principales variables y funciones. Y aparecerá el archivo imagen.asm que contiene las rutinas para el tratamiento de imágenes escritas en lenguaje ensamblador, debido a que se busca la mayor rapidez del tratamiento.

A continuación se verá el fichero error.h que contiene los diferentes códigos de error de salida de programa que se pueden dar, cuando se utiliza la biblioteca generada en programación.

A.1. Archivos de tratamiento de vídeo

Listado A.1: video.h

```
// Fichero video.h
//
// Biblioteca de funciones para Tratamiento de
// Imágenes en Tiempo Real
//
// Esqueleto C++
//
// Incluimos las cabeceras
#include "imagen.h"
#include "pantalla.h"
#include "pcidecoder.h"

#ifndef _video_h_
#define _video_h_

// Clase Video

class tVideo
{
    private:
        PCI_DECODER Decoder;

    public:
        int Direccion;
        int Ancho;          // En pixel
        int Alto;           // En pixel

        int InicializaVideo();
        int Captura( tPantalla P, int x, int y, int Anchura, int Altura,
                     int ModoCaptura );
        int Captura( tImagen I , int ModoCaptura, tPantalla P);
        int Paralo();
};

#endif // _video_h_
```

Listado A.2: video.cpp

```
// Fichero: video.cpp
//
// Biblioteca de funciones para Tratamiento de
// Imágenes en Tiempo Real
//
// Esqueleto C++
//
// Incluimos las cabeceras
#include "video.h"
#include "error.h"
#include "scaler.h"
#include <stdio.h>
#include <stdlib.h>
#include <pc.h>
#include <dos.h>

#ifdef __cplusplus
extern "C" {
# endif
 extern int ASM_Copia_Video( unsigned, unsigned, unsigned, unsigned );
#ifdef __cplusplus
}
#endif

// Inicializa todos los parametros del chip BT878 para la captura de video
int tVideo::InicializaVideo()
{
 int valor;
 /* Localizacion de memoria para video risc programs (8K) */
 Decoder.Video->Even->AllocateRiscMemory( 8 * 1025 );
```

```
    Decoder.Video->Odd->AllocateRiscMemory( 8 * 1024 );
    /* Inicializacion de los parametros mas importantes del video, tales
       como la forma y entrada de la senal de video, saturacion, contraste
       y brillo de la imagen,... */
35  Decoder.Video->Reset();
    Decoder.Video->SetVideoInput(VideoInputComposite);
    Decoder.Video->SetVideoFormat(VideoFormatPAL);
    Decoder.Video->SetSaturation(160);
    Decoder.Video->SetBrightness(5);
40  Decoder.Video->SetContrast(80);
    Decoder.Video->SetWhiteCrush(On);
    Decoder.Video->SetGammaCorrection(Off);
    /* Configuracion del cristal para la generacion de frecuencias para el
       sistema PAL */
45  valor = Decoder.Video->ReadLocalDWORD(0x084);
    Decoder.Video->WriteLocalDWORD( 0x084, valor & 0xE7 ); // TGCTRL.TGCKI=00
    Decoder.Video->WriteLocalDWORD( 0x0F0, 0xF9 );
    Decoder.Video->WriteLocalDWORD( 0x0F4, 0xDC );// PLL_F=0xDCF9
    Decoder.Video->WriteLocalDWORD( 0x0F8, 0x8E );// PLL_X=1, PLL_C=0 y PLL_I=0x0E
50  do{                          // Repite mientras DSTATUS.PLOCK==1
       valor = Decoder.Video->ReadLocalDWORD(0x00);
       Decoder.Video->WriteLocalDWORD(0x00,valor&0xFB); // Se borra
    }while( valor&0x04 );
    valor = Decoder.Video->ReadLocalDWORD(0x084);
55  Decoder.Video->WriteLocalDWORD(0x84,valor | 0x08);  // TGCTRL.TGCKI=01
    return 0;
   };

   // Captura de video en la pantalla P a partir de la posicion (x,y) con una
60 // determinada Anchura y Altura, estas dos ultimas estan limitadas por la
   // resolucion maxima de la capturadora de video utilizada. Se permiten dos
   // modos de captura, uno continuo (ModoCaptura=1) y otro de captura de una
   // sola imagen (ModoCaptura=0) y por ultimo se arranca la captura de video
   int tVideo::Captura(tPantalla P, int x, int y, int Anchura, int Altura,
65                     int ModoCaptura)
   {
    if (ModoCaptura<0 or ModoCaptura>1)
       exit (ERROR_DE_INTRODUCCION_DE_PARAMETROS_EN_MODO_DE_CAPTURA);
    if (x>=P.AnchoPantalla or y>=P.AltoPantalla)
70     exit (ERROR_DE_INTRODUCCION_DE_PARAMETROS_EN_CAPTURA_VIDEO_EN_PANTALLA);
    // La imagen se adecua al tamano de la pantalla P, para que la senal
    // proveniente del video se vea entera
    if (x+Anchura>P.AnchoPantalla) Anchura=P.AnchoPantalla-x;
    if (y+Altura>P.AltoPantalla) Altura=P.AltoPantalla-y;
75
    if (Anchura>768) Anchura=768; // Limitaciones de resolucion
    if (Altura>568) Altura=568;
    if (ModoCaptura==0)
    {
80     Crea_RISC_WxH_Una_Sola( P, Decoder, Anchura, Altura, x, y);// Crea programa RISC
       Decoder.Video->RunRisc(); // Pone en marcha el programa Risc
       delay(30);
    }
    else
85  {
       Crea_RISC_WxH_Continuo( P, Decoder, Anchura, Altura, x, y);// Crea programa RISC
       Decoder.Video->RunRisc(); // Pone en marcha el programa Risc
    }
    if (Altura>284)
90  {
       Escalado(Decoder,Anchura,Altura/2,FieldOdd);  // Escalado de la imagen
       Escalado(Decoder,Anchura,Altura/2,FieldEven); // a capturar
    }
    else
95  {
       Escalado(Decoder,Anchura,Altura,FieldOdd);    // Escalado de la imagen
       Escalado(Decoder,Anchura,Altura,FieldEven);   // a capturar
    }
    return 0;
100 };

   // Captura de video en la imagen I con una determinado Ancho y Alto,
   // definidos en la imagen I, estas dos variables estan limitadas por la
   // resolucion maxima de la capturadora de video utilizada, si no se verifica
```

```
// se producira un error.
// Se permiten dos modos de captura, uno continuo (ModoCaptura=1) y otro de
// captura de una sola imagen (ModoCaptura=0)
// Se pasa a la funcion el parametro pantalla P, porque se captura el video
// en la memoria de pantalla no visible, siempre que haya suficiente espacio,
// y de ahi se copia en nuestra imagen I.
// Y por ultimo se arranca la captura de video.
int tVideo::Captura( tImagen I, int ModoCaptura, tPantalla P)
{
 if (ModoCaptura<0 or ModoCaptura>1)
    exit (ERROR_DE_INTRODUCCION_DE_PARAMETROS_EN_MODO_DE_CAPTURA);
 if (I.Ancho>768 or I.Alto>568)
    exit (ERROR_DE_INTRODUCCION_DE_PARAMETROS_CAPTURA_IMAGEN_DE_VIDEO);
 int offsett=(P.AnchoPantalla*P.AltoPantalla*4);
 // Comprobamos que la imagen a capturar nos cabe en la memoria de pantalla
 // restante, es decir, en el resto de memoria de pantalla que no utilizamos
 // para la visualizacion
 if (I.TotalBytes>P.MemoriaRestante)
    exit (ERROR_NO_HAY_MEMORIA_PANTALLA_SUFICIENTE_PARA_IMAGEN);
 I.DireccionFisica=P.DirFisica+offsett;
 if (ModoCaptura==0)
 {
    Crea_RISC_WxH_Una_Sola_Imagen( I, Decoder, I.Ancho, I.Alto);//Crea programa RISC
    Decoder.Video->RunRisc(); // Pone en marcha el programa Risc
    delay(30);
 }
 else
 {
    Crea_RISC_WxH_Imagen_Continua( I, Decoder, I.Ancho, I.Alto);//Crea programa RISC
    Decoder.Video->RunRisc(); // Pone en marcha el programa Risc
 }
 if (I.Alto>284)
 {
    Escalado(Decoder,I.Ancho,I.Alto/2,FieldOdd);  // Escalado de la imagen
    Escalado(Decoder,I.Ancho,I.Alto/2,FieldEven); // a capturar
 }
 else
 {
    Escalado(Decoder,I.Ancho,I.Alto,FieldOdd);    // Escalado de la imagen
    Escalado(Decoder,I.Ancho,I.Alto,FieldEven);   // a capturar
 }
 // Copiamos desde la memoria de pantalla no utilizada para la visualizacion
 // la imagen capturada en nuestra variable imagen I
 ASM_Copia_Video (P.Selector, offsett, I.Direccion,I.TotalBytes);
 return 0;
};

int tVideo::Paralo()  // Para la captura de video
{
 Decoder.Video->HaltRisc();     /* Stop Risc program */

 return 0;
};
```

A.2. Archivos de tratamiento de imagen

Listado A.3: imagen.h

```
// Fichero imagen.h
//
// Biblioteca de funciones para Tratamiento de
// Imágenes en Tiempo Real
//
// Esqueleto C++
//
// Incluimos las cabeceras
#include "pantalla.h"

#ifndef _imagen_h_
#define _imagen_h_

// Clase Imagen /

class tImagen
{
     public:
        unsigned long Direccion; // En el segmento DS
        int Ancho;               // En pixel
        int Alto;                // En pixel
        int TotalBytes;          // En byte
        char* M;                 // Puntero
        unsigned long DireccionFisica;

        tImagen( int AnchoMax=1, int AltoMax=1 );     // Constructor
        int Redimensiona(int Anchura, int Altura );
        int CargaBMP( char* NombreFichero );
        int SalvaBMP( char* NombreFichero );
        int Recortar( int X, int Y, int Anchura, int Altura);
        int Dibujalo( int X, int Y, tPantalla P );
        int Capturalo( int X, int Y, int Anchura, int Altura, tPantalla P );
        int Copia( tImagen Origen );
        int Suma( tImagen I );
        int Resta( tImagen I );
        int Convolucion( int *Matriz, int N );
        int Convolucion( double *Matriz, int N);
        int TransfAfin( double Matriz [3] [3] );
        int Rotacion( double Angulo );
        int Escala( double Factor );
        int Traslada( int X, int Y );
        int XOR( tImagen I );
        int OR( tImagen I );
        int NOT();
        int Negativo();
        int GeneraMatrizBezier( int MatrizEcualizado[255], int P0, int Q0,
              int P1, int Q1, int P2, int Q2, int P3, int Q3);
        int GeneraMatrizSplines( int MatrizEcualizado[255], int P0,int Q0,
              int P1, int Q1, int P2, int Q2, int P3, int Q3);
        int Ecualiza( int *MatrizEcualizado, int TipoEcualizado);
        int Escribe( char* Texto, int X, int Y, int ColorTransparente, tPantalla P);
        int CorrigeAberracion( double FactorDistorsion );
        int FFT2D( char *Comp_Rojo, char *Comp_Verde, char *Comp_Azul);
        int CalculaFFT2D( int Tam_Imagen, char *Comp_Rojo, char *Comp_Verde,
                          char *Comp_Azul, char *Opcion );
        int InvFFT2D( int Tam_Imagen, char *Comp_Rojo, char *Comp_Verde,
                          char *Comp_Azul);
        ~tImagen();  // Destructor


};
#endif // _imagen_h_
```

Listado A.4: imagen.cpp

```
// Fichero: imagen.cpp
//
// Biblioteca de funciones para Tratamiento de
// Imágenes en Tiempo Real
//
// Esqueleto C++
//
// Incluimos las cabeceras
#include "imagen.h"
#include "error.h"
#include "pantalla.h"
#include <stdio.h>
#include <stdlib.h>
#include <dpmi.h>
#include <sys/segments.h>
#include <conio.h>
#include <iostream>
#include <pc.h>
#include <math.h>
#include "mydefs.h"

#ifdef __cplusplus
extern "C" {
# endif
 extern int ASM_Dibuja(unsigned, unsigned, unsigned, unsigned, unsigned, unsigned,
                       unsigned);
 extern int ASM_Captura(unsigned, unsigned, unsigned, unsigned, unsigned, unsigned,
                       unsigned);
 extern int ASM_Copia(unsigned, unsigned, unsigned);
 extern int ASM_NOT (unsigned, unsigned);
 extern int ASM_Recorta (unsigned, unsigned, unsigned, unsigned, unsigned, unsigned,
                         unsigned);
 extern int ASM_Suma (unsigned, unsigned, unsigned);
 extern int ASM_Resta (unsigned, unsigned, unsigned);
 extern int ASM_Traslada (unsigned, unsigned, unsigned, unsigned, unsigned,
                          unsigned);
 extern int ASM_Rotacion (unsigned, unsigned, unsigned, unsigned, unsigned,unsigned,
                          double, double, double, double);
 extern int ASM_XOR (unsigned, unsigned, unsigned);
 extern int ASM_Magnificacion_Entera (unsigned, unsigned, unsigned, unsigned,
                                      unsigned);
 extern int ASM_Magnificacion_No_Entera (unsigned, unsigned, unsigned, unsigned,
                                         unsigned, double);
 extern int ASM_Minimizacion_Entera (unsigned, unsigned, unsigned, unsigned,
                                     unsigned, unsigned);
 extern int ASM_Minimizacion_No_Entera (unsigned, unsigned, unsigned, unsigned,
                                        unsigned, unsigned, double);
 extern int ASM_Convolucion_Entera (unsigned, unsigned, unsigned, unsigned,unsigned,
                                    unsigned);
 extern int ASM_Convolucion_Real (unsigned, unsigned, unsigned, unsigned, unsigned,
                                  unsigned);
 extern int ASM_OR (unsigned, unsigned, unsigned);
 extern int ASM_Redimensiona (unsigned, unsigned, unsigned, unsigned, unsigned,
                              unsigned);
 extern int ASM_TransfAfin (unsigned, unsigned, unsigned, unsigned, unsigned,
                            unsigned, double, double, double, double, double,
                            double);
 extern int ASM_Ecualizado (unsigned, unsigned, unsigned, unsigned);
 extern int ASM_Escribe (unsigned, unsigned, unsigned, unsigned, unsigned, unsigned,
                         unsigned, unsigned, unsigned, unsigned, unsigned,unsigned);
 extern int ASM_Corrige_Aberracion (unsigned, unsigned, unsigned, unsigned, double);
 extern int ASM_FFT2D (unsigned, unsigned, unsigned, unsigned, unsigned);
 extern int ASM_Crea_Tablas_Imagen_Real (unsigned, unsigned, unsigned, unsigned,
                                         unsigned);
 extern int ASM_Calcula_Parte_Real_Y_Maximo(unsigned, unsigned, unsigned, unsigned,
                                            unsigned);
 extern int ASM_Calcula_Parte_Imaginaria_Y_Maximo(unsigned, unsigned, unsigned,
                                                  unsigned, unsigned);
 extern int ASM_Calcula_Modulo_Y_Maximo(unsigned, unsigned, unsigned, unsigned,
                                        unsigned);
 extern int ASM_Calcula_Normalizacion (unsigned, unsigned, unsigned);
 extern int ASM_Calcula_Logaritmo (unsigned, unsigned, unsigned);
```

```
73   extern int ASM_Ordena_FFT (unsigned, unsigned);
     extern int ASM_Calcula_Fase_Maximo_Y_Minimo(unsigned, unsigned, unsigned, unsigned,
                                                 unsigned);
     extern int ASM_Calcula_Fase_Desnormalizada(unsigned, unsigned, unsigned);
     extern int ASM_Calcula_Fase_Normalizada(unsigned, unsigned, unsigned);
78   extern int ASM_Calcula_Imagen_Real_A_Partir_De_Tablas(unsigned, unsigned,
                                                           unsigned, unsigned);
     extern int ASM_Prueba(unsigned);
    #ifdef __cplusplus
    }
83  #endif

    // Constructor, se inicializa una imagen, es decir, se reserva la memoria
    // necesaria y se define su tamano
    tImagen::tImagen( int AnchoMax, int AltoMax )
88  {
     TotalBytes = AnchoMax*AltoMax*4;
     M = new char [TotalBytes];
     if (M==0) exit(ERROR_NO_HAY_MEMORIA);
     Direccion=(int)M;
93   Ancho=AnchoMax;
     Alto=AltoMax;
    };

    // Destructor, esta vacio ya que la memoria se crea y se elmina de
98  // forma manual a lo largo del programa
    tImagen::~tImagen()
    {
    };

103 // Carga un BMP en una Imagen
    int tImagen::CargaBMP( char* NombreFichero )
    {
     struct cabecera {
            char ident[2];
108         unsigned int tam;     // Para la extraccion de datos de la cabecera
            char reservado[4];    // que presenta un archivo BMP
            unsigned int offset;
     };
     struct info_general {
113         unsigned int tam_cabecera;
            unsigned int anchura;
            unsigned int altura;
            unsigned short int planos;
            unsigned short int tam_pixel;
118         unsigned int compresion;  // Para la extraccion de datos de la
            unsigned int tam_imagen;  // informacion general que presenta un
            unsigned int h_resolution;// archivo BMP
            unsigned int v_resolution;
            unsigned int num_color;
123         unsigned int color_imp;
     };

     struct cabecera bmp_cab;   // Definicion de variables
     struct info_general bmp_info;
128  FILE *f;
     int lineabytesleer,x,y,j,aux,i=0;
     char *linea;
     clrscr();
     f = fopen(NombreFichero, "rb" );  // Abrimos el fichero a leer
133  if (!f) exit (ERROR_DE_APERTURA_DE_FICHERO_PARA_LECTURA);
     fread(&bmp_cab.ident,2,1,f);
     fread(&bmp_cab.tam,4,1,f);        // Se extraen todos sus parametros
     fread(&bmp_cab.reservado,4,1,f);
     fread(&bmp_cab.offset,4,1,f);
138  fread(&bmp_info.tam_cabecera,4,1,f);
     fread(&bmp_info.anchura,4,1,f);
     fread(&bmp_info.altura,4,1,f);
     fread(&bmp_info.planos,2,1,f);
     fread(&bmp_info.tam_pixel,2,1,f);
143  fread(&bmp_info.compresion,4,1,f);
     fread(&bmp_info.tam_imagen,4,1,f);
     fread(&bmp_info.h_resolution,4,1,f);
     fread(&bmp_info.v_resolution,4,1,f);
```

```
    fread(&bmp_info.num_color,4,1,f);
148 fread(&bmp_info.color_imp,4,1,f);
    // Se ve si es un archivo BMP
    if (bmp_cab.ident[0]!='B' && bmp_cab.ident[1]!='M')
       exit (ERROR_NO_ES_UN_ARCHIVO_BMP);
    // Se comprueba si el tamano de la imagen es igual al del BMP, si no es asi
153 // se realiza una nueva reserva de memoria para que almacene el BMP
    if ((bmp_info.anchura*bmp_info.altura*4)!=TotalBytes)
    {
      delete M;
      TotalBytes=bmp_info.anchura*bmp_info.altura*4;
158   M= new char [TotalBytes];
      if (!M) exit (ERROR_NO_HAY_MEMORIA);
      Direccion=int(M);
    };
    Ancho = bmp_info.anchura;
163 Alto = bmp_info.altura;
    // El ancho de linea en bytes debe ser m£ltiplo de 4 siempre
    lineabytesleer=Ancho*3;
    aux=lineabytesleer/4;
    if ((aux*4)!=lineabytesleer) lineabytesleer+=(4+(aux*4)-lineabytesleer);
168 linea=new char [lineabytesleer]; // Albergara una linea del BMP
    if (!linea) exit (ERROR_NO_HAY_MEMORIA);

   /* printf("\n %35s%s>","<",NombreFichero);
    printf("\n Extension del archivo: %c%c",bmp_cab.ident[0],bmp_cab.ident[1]);
173 printf("\n Tamano del archivo: %ld bytes",bmp_cab.tam);
    printf("\n Offset del archivo: %ld",bmp_cab.offset);
    printf("\n Tamano de la cabecera: %d bytes",bmp_info.tam_cabecera);
    printf("\n Anchura: %d",bmp_info.anchura);
    printf("\n Altura: %d",bmp_info.altura);
178 printf("\n Numero de planos: %d",bmp_info.planos);
    printf("\n Numero de bits por pixel: %d bits",bmp_info.tam_pixel);
    printf("\n Compresion: %d",bmp_info.compresion);
    printf("\n Tamano de la imagen: %d bytes",bmp_info.tam_imagen);
    printf("\n Resolucion horizontal: %d pixels/metros",bmp_info.h_resolution);
183 printf("\n Resolucion vertical: %d pixels/metros",bmp_info.v_resolution);
    printf("\n Numero de colores utilizados: %d colores",bmp_info.num_color);
    printf("\n Numero de colores importantes: %d colores",bmp_info.color_imp);
    printf("\n Direccion donde copiarlo: %d",Direccion);
    getkey();
188 clrscr();*/

    // Se va leyendo el BMP de izquierda a derecha y de arriba a abajo y se van
    // introduciendo los datos en la imagen de izquierda a derecha y de abajo
    // a arriba
193 i=(Alto-1)*Ancho*4;
    for (y=0 ; y<Alto ; y++)
    {
      fread (linea, lineabytesleer,1,f);
      j=0;
198   for (x=0 ; x<Ancho ; x++)
      {
         M[i]=linea[j];
         M[i+1]=linea[j+1];
         M[i+2]=linea[j+2];
203      i=i+4;
         j=j+3;
      };
      i=i-(Ancho*2*4);
    };
208 delete linea;
    fclose(f); // Cerramos el fichero
    return 0;
   };

213 // Guarda una Imagen en formato BMP
   int tImagen::SalvaBMP( char* NombreFichero )
   {
    FILE *f;        // Definicion de variables
    int aux,tamanoarchivo,i=0,offset=54,tam_cab=40;
218 int planos=1,numbitpixel=24,compresion=0;
    char *linea;
    int lineaescribir,x,j,y;
```

```
     f=fopen(NombreFichero,"wb"); // Abrimos o creamos el fichero si no existe
     if (!f) exit (ERROR_DE_APERTURA_O_DE_CREACION_DE_FICHERO_PARA_ESCRITURA);
223  fwrite("BM",2,1,f);
     tamanoarchivo=((TotalBytes/4)*3)+54;
     fwrite(&tamanoarchivo,4,1,f);
     fwrite(&i,4,1,f);
     fwrite(&offset,4,1,f);    // Escritura en el archivo de su cabecera y de
228  fwrite(&tam_cab,4,1,f);   // de la informacion general que debe presentar
     fwrite(&Ancho,4,1,f);     // un archivo BMP
     fwrite(&Alto,4,1,f);
     fwrite(&planos,2,1,f);
     fwrite(&numbitpixel,2,1,f);
233  fwrite(&compresion,4,1,f);
     tamanoarchivo=tamanoarchivo-54;
     fwrite(&tamanoarchivo,4,1,f);
     fwrite(&i,4,1,f);
     fwrite(&i,4,1,f);
238  fwrite(&i,4,1,f);
     fwrite(&i,4,1,f);
     lineaescribir=Ancho*3;
     aux=lineaescribir/4;
     if ((aux*4)!=lineaescribir) lineaescribir+=(4+(aux*4)-lineaescribir);
243  // Reserva de memoria que albergara una linea a escribir en el BMP
     linea=new char [lineaescribir];
     if (!linea) exit (ERROR_NO_HAY_MEMORIA);
     // Se va leyendo la imagen de izquierda a derecha desde abajo hacia arriba
     // y se va introduciendo los datos en el archivo BMP, de izquierda a derecha
248  // y de arriba a abajo
     i=(Alto-1)*Ancho*4;
     for (y=0 ; y<Alto ; y++)
     {
       j=0;
253    for (x=0 ; x<Ancho ; x++)
       {
          linea[j]=M[i];
          linea[j+1]=M[i+1];
          linea[j+2]=M[i+2];
258       i=i+4;
          j=j+3;
       };
       fwrite(linea,lineaescribir,1,f);
       i=i-(Ancho*2*4);
263  };
     delete linea;
     fclose(f); // Cerramos el fichero
     return 0;
};
268
// Dibuja en Pantalla una Imagen a partir de la posicion (X,Y) indicada
int tImagen::Dibujalo( int X, int Y, tPantalla P )
{
 if (X<0 or X>P.AnchoPantalla or Y<0 or Y>P.AltoPantalla or
273      (X+Ancho)>P.AnchoPantalla or (Y+Alto)>P.AltoPantalla)
           exit (ERROR_DE_INTRODUCCION_DE_PARAMETROS_EN_DIBUJALO);
 ASM_Dibuja(P.Selector, P.AnchoPantalla, Direccion, Ancho, Alto, X, Y );
 return 0;
};
278
// Captura de Pantalla una Imagen a partir de la posicion (X,Y) con una
// determinada Altura y Anchura
int tImagen::Capturalo( int X, int Y, int Anchura, int Altura, tPantalla P )
{
283  int TotalBytesCaptura;  // Definicion de variables
     if (X<0 or X>P.AnchoPantalla or Y<0 or Y>P.AltoPantalla or
        (X+Anchura)>P.AnchoPantalla or (Y+Altura)>P.AltoPantalla)
        exit (ERROR_DE_INTRODUCCION_DE_PARAMETROS_EN_CAPTURALO);
     TotalBytesCaptura=Anchura*Altura*4;
288  if (TotalBytes!=TotalBytesCaptura)   // Se reserva el espacio necesario de
     {                                    // memoria para que albergue la Imagen
       delete M;                          // capturada
       M=new char[TotalBytesCaptura];
       if (!M) exit (ERROR_NO_HAY_MEMORIA);
293    Direccion=int(M);
       TotalBytes=TotalBytesCaptura;
```

```
 };
 ASM_Captura(P.Selector,P.AnchoPantalla,Direccion,Anchura,Altura,X,Y);
 Ancho=Anchura;
 Alto=Altura;
 return 0;
};

// Recorta una imagen a partir de la posicion (X,Y) con una determinada
// Anchura y Altura
int tImagen::Recortar( int X, int Y, int Anchura, int Altura )
{
 char *R;
 int TotalBytesRecorte,Dir_Recorte;

 if (X<0 or Y<0 or X>Ancho or Y>Alto or (Anchura+X)>Ancho or (Altura+Y)>Alto)
    exit (ERROR_DE_INTRODUCCION_DE_PARAMETROS_EN_RECORTAR);
 TotalBytesRecorte=Altura*Anchura*4;
 R = new char[TotalBytesRecorte];     // Se reserva el espacio necesario para
 if (!R) exit (ERROR_NO_HAY_MEMORIA);// que albergue a la Imagen Recorte
 Dir_Recorte=int(R);
 ASM_Recorta (Direccion,Ancho,X,Y,Anchura,Altura,Dir_Recorte);
 delete M;
 M=R;
 Direccion=Dir_Recorte;
 Ancho=Anchura;
 Alto=Altura;
 TotalBytes=TotalBytesRecorte;
 return 0;
};

// Copia una Imagen Origen en otra Imagen Destino
int tImagen::Copia( tImagen Origen )
{
   if (TotalBytes!=Origen.TotalBytes)  // Si la imagen destino es mas pequena
   {                                   // que la de Origen se reserva la
     delete M;                         // memoria suficiente
     M= new char [Origen.TotalBytes];
     if (!M) exit (ERROR_NO_HAY_MEMORIA);
     Direccion=int(M);
     TotalBytes=Origen.TotalBytes;
   };
   ASM_Copia (Origen.Direccion,Direccion,Origen.TotalBytes);
   Ancho=Origen.Ancho;
   Alto=Origen.Alto;
   return 0;
};

// Realizara la suma de dos imagenes del mismo tamano
int tImagen::Suma( tImagen I )
{
 if (Ancho!=I.Ancho or Alto!=I.Alto)
    exit (ERROR_IMAGENES_DE_DIFERENTE_TAMANO_EN_SUMA);
 ASM_Suma (Direccion, I.Direccion, TotalBytes);
 return 0;
};

// Realizara la resta de dos imagenes del mismo tamano
int tImagen::Resta( tImagen I )
{
  if (Ancho!=I.Ancho or Alto!=I.Alto)
    exit (ERROR_IMAGENES_DE_DIFERENTE_TAMANO_EN_RESTA);
 ASM_Resta (Direccion, I.Direccion, TotalBytes);
 return 0;
};

// Realiza la convolucion de una Imagen a traves de una matriz de tamano NxN
// de numeros enteros
int tImagen::Convolucion( int *Matriz, int N )
{
 char *C;    // Definicion de variables
 int TotalBytesConvolucion,AnchoConvolucion,AltoConvolucion,Dir_Convolucion;
 if (N<=0) exit (ERROR_DE_INTRODUCCION_DE_PARAMETROS_EN_CONVOLUCION_ENTERA);
 AnchoConvolucion=(Ancho-(N-1));
 AltoConvolucion=(Alto-(N-1));
```

```
     TotalBytesConvolucion=AnchoConvolucion*AltoConvolucion*4;
     C=new char [TotalBytesConvolucion]; // Reserva el espacio de memoria que
     Dir_Convolucion=int(C);             // albergara la convolucion entera
     ASM_Convolucion_Entera(Direccion,Dir_Convolucion,Ancho,Alto,N,int(Matriz));
373  delete M;
     M=C;
     Direccion=Dir_Convolucion;
     Ancho=AnchoConvolucion;
     Alto=AltoConvolucion;
378  TotalBytes=TotalBytesConvolucion;
     return 0;
    };

    // Realiza la convolucion de una Imagen a traves de una matriz de tamano NxN
383 // de numeros reales
    int tImagen::Convolucion( double *Matriz, int N)
    {
     char *C;     // Definicion de variables
     int TotalBytesConvolucion,AnchoConvolucion,AltoConvolucion,Dir_Convolucion;
388  if (N<=0) exit (ERROR_DE_INTRODUCCION_DE_PARAMETROS_EN_CONVOLUCION_REAL);
     AnchoConvolucion=(Ancho-(N-1));
     AltoConvolucion=(Alto-(N-1));
     TotalBytesConvolucion=AnchoConvolucion*AltoConvolucion*4;
     C=new char [TotalBytesConvolucion];// Reserva el espacio de memoria que
393  Dir_Convolucion=int(C);            // albergara la convolucion real
     ASM_Convolucion_Real (Direccion,Dir_Convolucion,Ancho,Alto,N,int(Matriz));
     delete M;
     M=C;
     Direccion=Dir_Convolucion;
398  Ancho=AnchoConvolucion;
     Alto=AltoConvolucion;
     TotalBytes=TotalBytesConvolucion;
     return 0;
    };
403
    // Realiza la Transformacion Afin de una Imagen, donde se pasa el parametro
    // Matriz[3][3] que presenta la siguiente forma:
    //          ( Sx*cos   -sen    Xo )
    //          (  sen    Sy*cos   Yo )
408 //          (   0       0       1 )
    // donde:
    //     -- Xo e Yo son el desplazamiento
    //     -- Sx y Sy los factores de escala en los ejes correspondientes
    int tImagen::TransfAfin( double Matriz [3] [3] )
413 {
     char *T;      // Definicion de variables
     int AlturaTransfAfin,AnchuraTransfAfin,TotalBytesTransfAfin,Dir_TransfAfin;
     double Angulo,Seno,Coseno,EscalaY,EscalaX,DesplazamientoX,DesplazamientoY;
     if ((Matriz[0][0]==0) or (Matriz[1][1]==0) or (Matriz[2][0]!=0) or
418     (Matriz[2][1]!=0) or (Matriz[2][2]!=1))
        exit (ERROR_DE_INTRODUCCION_DE_PARAMETROS_EN_TRANSFORMACION_AFIN);
     if (fabs(Matriz[0][1])!=fabs(Matriz[1][0]))
        exit (ERROR_DE_INTRODUCCION_DE_PARAMETROS_EN_TRANSFORMACION_AFIN);
     // Se van calculando los diferentes parametros necesarios para calcular la
423  // Transformacion Afin depediendo en que cuadrante nos encontremos
     if (Matriz[0][0]>0.0 and Matriz[0][1]>=0.0) // Estamos en el primer cuadrante
     {
        Seno=Matriz[0][1];
        Angulo=asin(Seno)*(180/3.141592654); // Repeticion de instuccion por
428     Angulo=asin(Seno)*(180/3.141592654); // problema
        Coseno=cos(Angulo*(3.141592654/180));
        EscalaY=Matriz[1][1]/Coseno;
        EscalaX=Matriz[0][0]/Coseno;
        AlturaTransfAfin=int(Matriz[1][2])+int((Ancho*(EscalaX)*Seno)+(Alto*(EscalaY)*
433     Coseno))+5;
        AnchuraTransfAfin=int(Matriz[0][2])+int((Alto*(EscalaY)*Seno)+(Ancho*(EscalaX)*
        Coseno))+5;
        DesplazamientoX=Matriz[0][2]+(Alto*(EscalaY)*Seno);
        DesplazamientoY=Matriz[1][2];
438  }
     else if ((Matriz[0][0]<0.0) and (Matriz[0][1]>0.0))//Estamos en el 2° cuadrante
     {
        Seno=Matriz[0][1];
        Angulo=180-(asin(Seno)*(180/3.141592654)); // Repeticion de instruccion
```

```
443      Angulo=180-(asin(Seno)*(180/3.141592654)); // por problema
         Coseno=cos(Angulo*(3.141592654/180));
         EscalaY=Matriz[1][1]/Coseno;
         EscalaX=Matriz[0][0]/Coseno;
         AlturaTransfAfin=int(Matriz[1][2])+int((Ancho*(EscalaX)*Seno)+(Alto*(EscalaY)*
448      (-Coseno)))+5;
         AnchuraTransfAfin=int(Matriz[0][2])+int((Alto*(EscalaY)*Seno)+(Ancho*(EscalaX)*
         (-Coseno)))+5;
         DesplazamientoX=Matriz[0][2]+(Alto*(EscalaY)*Seno)+(Ancho*(EscalaX)*(-Coseno));
         DesplazamientoY=Matriz[1][2]+(Alto*(EscalaY)*(-Coseno));
453   }
      else if ((Matriz[0][0]<0.0) and (Matriz[0][1]<=0.0))//Estamos en el 3 cuadrante
      {
         Seno=Matriz[0][1];
         Angulo=180-(asin(Seno)*(180/3.141592654)); // Repeticion de instruccion
458      Angulo=180-(asin(Seno)*(180/3.141592654)); // por problema
         Coseno=cos(Angulo*(3.141592654/180));
         EscalaY=Matriz[1][1]/Coseno;
         EscalaX=Matriz[0][0]/Coseno;
         AlturaTransfAfin=int(Matriz[1][2])+int((Ancho*(EscalaX)*(-Seno))+(Alto*
463      (EscalaY)*(-Coseno)))+5;
         AnchuraTransfAfin=int(Matriz[0][2])+int((Alto*(EscalaY)*(-Seno))+(Ancho*
         (EscalaX)*(-Coseno)))+5;
         DesplazamientoX=Matriz[0][2]+(Ancho*(EscalaX)*(-Coseno));
         DesplazamientoY=Matriz[1][2]+(Ancho*(EscalaX)*(-Seno))+(Alto*(EscalaY)*
468      (-Coseno));
      }
      else if ((Matriz[0][0]>0.0) and (Matriz[0][1]<0.0))//Estamos en el cuarto cuadrante
      {
         Seno=Matriz[0][1];
473      Angulo=360+(asin(Seno)*(180/3.141592654)); // Repeticion de instruccion
         Angulo=360+(asin(Seno)*(180/3.141592654)); // por problema
         Coseno=cos(Angulo*(3.141592654/180));
         EscalaY=Matriz[1][1]/Coseno;
         EscalaX=Matriz[0][0]/Coseno;
478      AlturaTransfAfin=int(Matriz[1][2])+int((Ancho*(EscalaX)*(-Seno))+(Alto*
         (EscalaY)*Coseno))+5;
         AnchuraTransfAfin=int(Matriz[0][2])+int((Alto*(EscalaY)*(-Seno))+(Ancho*
         (EscalaX)*Coseno))+5;
         DesplazamientoX=Matriz[0][2];
483      DesplazamientoY=Matriz[1][2]+(Ancho*(EscalaX)*(-Seno));
      };
      TotalBytesTransfAfin=AlturaTransfAfin*AnchuraTransfAfin*4;
      T=new char[TotalBytesTransfAfin];    // Reserva de memoria que albergara la
      if (!T) exit (ERROR_NO_HAY_MEMORIA); // Transformacion Afin
488   Dir_TransfAfin=int(T);
      ASM_TransfAfin(Direccion,Dir_TransfAfin,Alto,Ancho,AlturaTransfAfin,
          AnchuraTransfAfin,DesplazamientoX,DesplazamientoY,EscalaY,EscalaX,Seno,Coseno);
      delete M;
      M=T;
493   Direccion=Dir_TransfAfin;
      Ancho=AnchuraTransfAfin;
      Alto=AlturaTransfAfin;
      TotalBytes=TotalBytesTransfAfin;
      return 0;
498   };

     // Realiza la Rotacion de una Imagen, en un Angulo comprendido entre
     // 0 y 360 grados
     int tImagen::Rotacion( double Angulo )
503  {
      char *R;  // Definicion de variables
      int AlturaRotacion,AnchuraRotacion,TotalBytesRotacion,Dir_Rotacion;
      double Seno,Coseno,DesplazamientoX,DesplazamientoY;
      if ((Angulo<0.0) or (Angulo>=360.0))
508      exit (ERROR_DE_INTRODUCCION_DE_PARAMETROS_EN_ROTACION);
      Seno=sin(Angulo*(3.141592654/180));
      Coseno=cos(Angulo*(3.141592654/180));
      Seno=sin(Angulo*(3.141592654/180)); // Repeticion de instruccion por problema
      // Se van calculando los diferentes parametros necesarios para calcular la
513   // Rotacion depediendo en que cuadrante nos encontremos
      if (Angulo>=0 && Angulo<90)  // Estamos en el primer cuadrante
      {
          AlturaRotacion=int(Ancho*Seno)+int(Alto*Coseno)+1;
```

```
        AnchuraRotacion=int(Ancho*Coseno)+int(Alto*Seno)+1;
518     DesplazamientoX=Alto*Seno;
        DesplazamientoY=0;
    }
    else if (Angulo>=90 && Angulo<180) // Estamos en el segundo cuadrante
    {
523     AlturaRotacion=int(Ancho*Seno)+int(Alto*(-Coseno))+1;
        AnchuraRotacion=int(Ancho*(-Coseno))+int(Alto*Seno)+1;
        DesplazamientoX=(Ancho*(-Coseno))+(Alto*Seno);
        DesplazamientoY=Alto*(-Coseno);
    }
528 else if (Angulo>=180 && Angulo<270) // Estamos en el tercer cuadrante
    {
        AlturaRotacion=int(Ancho*(-Seno))+int(Alto*(-Coseno))+1;
        AnchuraRotacion=int(Ancho*(-Coseno))+int(Alto*(-Seno))+1;
        DesplazamientoX=Ancho*(-Coseno);
533     DesplazamientoY=(Ancho*(-Seno))+(Alto*(-Coseno));
    }
    else if (Angulo>=270 && Angulo<360) // Estamos en el cuarto cuadrante
    {
        AlturaRotacion=int(Ancho*(-Seno))+int(Alto*Coseno)+1;
538     AnchuraRotacion=int(Ancho*Coseno)+int(Alto*(-Seno))+1;
        DesplazamientoX=0;
        DesplazamientoY=Ancho*(-Seno);
    }
    TotalBytesRotacion=AlturaRotacion*AnchuraRotacion*4;
543 R = new char[TotalBytesRotacion];    // Reserva de memoria que albergara
    if (!R) exit (ERROR_NO_HAY_MEMORIA); // la Rotacion de la Imagen
    Dir_Rotacion=int(R);
    ASM_Rotacion (Direccion,Alto,Ancho,AlturaRotacion,AnchuraRotacion,Dir_Rotacion,
                  DesplazamientoX,DesplazamientoY,Seno,Coseno);
548 delete M;
    M=R;
    Direccion=Dir_Rotacion;
    Ancho=AnchuraRotacion;
    Alto=AlturaRotacion;
553 TotalBytes=TotalBytesRotacion;
    return 0;
   };

   // Realiza el Escalado de una Imagen a traves de un Factor de escalado
558 int tImagen::Escala( double Factor )
   {
    char *E;     // Definicion de variables
    int AnchoEscalado,AltoEscalado,Dir_Escalado,BytesNecesariosEscalado;
    int i,FactorEntero;
563 if (Factor<=0.0) exit (ERROR_DE_INTRODUCCION_DE_PARAMETROS_EN_ESCALADO);
    if (Factor!=1.0) // Si el Factor es 1 la Imagen se queda tal y como esta
    {
     AnchoEscalado=int(Ancho*Factor);
     AltoEscalado=int(Alto*Factor);
568  BytesNecesariosEscalado=AnchoEscalado*AltoEscalado*4;
     E=new char [BytesNecesariosEscalado];  // Reserva de memoria que albergara
     if (E==0) exit (ERROR_NO_HAY_MEMORIA); // la Imagen escalada
     Dir_Escalado=int(E);
     if (Factor>1.0)  // Haremos una magnificacion dependiendo del parametro Factor
573  {                // con dos posibles opciones que Factor sea entero o no
       FactorEntero=int(Factor);
       if (Factor==FactorEntero) ASM_Magnificacion_Entera (Direccion,Dir_Escalado,
                                                           Ancho, Alto,FactorEntero);
       else ASM_Magnificacion_No_Entera (Direccion,Dir_Escalado,Ancho,AnchoEscalado,
578                                      AltoEscalado,Factor);
     };
     if (Factor<1.0) // Haremos una minimizacion dependiendo del parametro Factor
     {               // con dos posibles opciones que 1/Factor sea un entero o no
       FactorEntero=int(1/Factor);
583    if ((FactorEntero*Factor)==1.0) ASM_Minimizacion_Entera (Direccion,Dir_Escalado,
                                       Ancho,AnchoEscalado,AltoEscalado,FactorEntero);
       else ASM_Minimizacion_No_Entera (Direccion,Dir_Escalado,Ancho,AnchoEscalado,
                                        AltoEscalado,FactorEntero,Factor);
     };
588  delete M;
     M=E;
     Ancho=AnchoEscalado;
```

```
      Alto=AltoEscalado;
      Direccion=Dir_Escalado;
593   TotalBytes=BytesNecesariosEscalado;
     };
     return 0;
    };

598 // Realiza una traslacion de una imagen al punto especificado por (X,Y)
    int tImagen::Traslada( int X, int Y )
    {
     char *T;   // Definicion de variables
     int TotalBytesTraslado, Dir_Traslado;
603  if (X<0 or Y<0) exit (ERROR_DE_INTRODUCCION_DE_PARAMETROS_EN_TRASLACION);
     TotalBytesTraslado=(Ancho+X)*(Alto+Y)*4;
     T = new char [TotalBytesTraslado];  // Reserva de memoria que albergara
     if (!T) exit (ERROR_NO_HAY_MEMORIA);// la Imagen trasladada
     Dir_Traslado=int(T);
608  ASM_Traslada (Direccion,Ancho,Alto,X,Y,Dir_Traslado);
     delete M;
     M=T;
     Direccion=Dir_Traslado;
     Ancho=Ancho+X;
613  Alto=Alto+Y;
     TotalBytes=TotalBytesTraslado;
     return 0;
    };

618 // Realiza la funcion logica XOR entre dos imagenes del mismo tamano
    int tImagen::XOR( tImagen I )
    {
     if ((Ancho!=I.Ancho) or (Alto!=I.Alto))
        exit (ERROR_IMAGENES_DE_DIFERENTE_TAMANO_EN_FUNCION_XOR);
623  ASM_XOR( Direccion, I.Direccion, TotalBytes);
     return 0;
    };

    // Realiza la funcion logica OR entre dos imagenes del mismo tamano
628 int tImagen::OR( tImagen I )
    {
     if ((Ancho!=I.Ancho) or (Alto!=I.Alto))
        exit (ERROR_IMAGENES_DE_DIFERENTE_TAMANO_EN_FUNCION_OR);
     ASM_OR( Direccion, I.Direccion, TotalBytes);
633  return 0;
    };

    // Realiza la funcion NOT de una Imagen
    int tImagen::NOT()
638 {
     ASM_NOT (Direccion,TotalBytes);
     return 0;
    };

643 // Realiza el Negativo de una Imagen
    int tImagen::Negativo()
    {
     ASM_NOT (Direccion,TotalBytes);
     return 0;
648 };

    // Dados 4 puntos se calcula la curva de Bezier, y cuyos valores son
    // almacenados en una MatrizEcualizado que servira para realizar dicho
    // ecualizado
653 int tImagen::GeneraMatrizBezier(int MatrizEcualizado[255],int P0,int Q0,
                   int P1,int Q1,int P2,int Q2,int P3,int Q3)
    {
     int a,b,c,d,e,f,g,h,i,xa,xb,Encontrado,Yt; // Definicion de variables
     double X1,X2,X3,fX1,fX2,fX3,t,Error;
658  a=(-P0+(3*P1)-(3*P2)+P3); b=((3*P0)-(6*P1)+(3*P2)); c=(3*(P1-P0)); d=P0;
     e=(-Q0+(3*Q1)-(3*Q2)+Q3); f=((3*Q0)-(6*Q1)+(3*Q2)); g=(3*(Q1-Q0)); h=Q0;
     xa=0; xb=255; Error=0.0000001;
     if (P0>0)    // Rellena la Matriz con ceros desde 0 hasta (P0-1)
     {
663     for(i=0;i<P0;i++) MatrizEcualizado[i]=0;
        xa=P0;
```

```
    };
    if (P3<255) // Rellena la Matriz con ceros desde (P3+1) hasta 255
    {
668     for(i=(P3+1);i<=255;i++) MatrizEcualizado[i]=0;
        xb=P3;
    };
    // Recorremos la Matriz desde xa hasta xb, para cada valor x buscamos
    // a traves del metodo de la secante la raiz que verifica la curva de
673 // Beizer para dicho valor de x (y formada con los coordenadas P),
    // este valor t que buscamos pertenecera al intervalo [0,1] SIEMPRE,
    // una vez calculado t calculamos su imagen a traves de la curva de Bezier
    // generada con las coordenadas Q.
    for (i=xa;i<=xb;i++)
678 {
        X1=0;
        X2=1;
        Encontrado=0;
        while (Encontrado==0)
683     {
            fX1=(a*X1*X1*X1)+(b*X1*X1)+(c*X1)+(d-i);
            fX2=(a*X2*X2*X2)+(b*X2*X2)+(c*X2)+(d-i);
            X3=((X1*fX2)-(X2*fX1))/(fX2-fX1);
            fX3=(a*X3*X3*X3)+(b*X3*X3)+(c*X3)+(d-i);
688         if (fX3==0) Encontrado=1;
            else
            {
               if (fabs(X3-X2)<Error) Encontrado=1;
               else
693            {
                  X1=X2;
                  X2=X3;
               };
            };
698     };
        t=X3;
        Yt=int((e*t*t*t)+(f*t*t)+(g*t)+h);
        if (Yt>255) Yt=255;  // Limita los valores entre 0 y 255
        if (Yt<0) Yt=0;
703     MatrizEcualizado[i]=Yt;
    };
    return 0;
};

708 // Dados 4 puntos se calcula la curva de Splines, y cuyos valores son
    // almacenados en una MatrizEcualizado que servira para realizar dicho
    // ecualizado
    int tImagen::GeneraMatrizSplines(int MatrizEcualizado[255], int P0,int Q0,
                int P1,int Q1,int P2,int Q2,int P3,int Q3)
713 {
    int Xa,Xb,i,Yt;
    double H0,H1,H2,Y0prima,Y1prima,Y2prima,a,b,c,Y,X,M1,M2;
    H0=P1-P0; H1=P2-P1; H2=P3-P2;
    Y0prima=(Q1-Q0)/H0; Y1prima=(Q2-Q1)/H1; Y2prima=(Q3-Q2)/H2;
718 a=(H0+H1)/3; b=H1/6; c=(H1+H2)/3;
    Y=Y1prima-Y0prima; X=Y2prima-Y1prima;
    M1=((Y*c)-(X*b))/((a*c)-(b*b)); M2=((a*X)-(Y*b))/((a*c)-(b*b));
    Xa=0; Xb=255;
    if (P0>0) // Se rellena la Matriz con ceros desde 0 hasta (P0-1)
723 {
        for (i=0;i<P0;i++) MatrizEcualizado[i]=0;
        Xa=P0;
    };
    if (P3<255) // Se rellena la Matriz con ceros desde (P3+1) hasta 255
728 {
        for (i=(P3+1);i<=255;i++) MatrizEcualizado[i]=0;
        Xb=P3;
    };
    // Se va rellenando la Matriz usando las curvas de Splines, de tal manera
733 // que los splines son diferentes para cada intervalo, siendo las curvas
    // diferentes para cada intervalo
    for (i=Xa;i<P1;i++)
    {
        Yt=int((((i-P0)*(i-P0)*(i-P0)*M1)/(6*H0))+((((P1-i)*Q0)+((i-P0)*Q1))/H0)-
738     ((H0/6)*((i-P0)*M1)));
```

```
        if (Yt>255) Yt=255;
        if (Yt<0) Yt=0;
        MatrizEcualizado[i]=Yt;
    };
    for (i=P1;i<P2;i++)
    {
        Yt=int(((((P2-i)*(P2-i)*(P2-i)*M1)+((i-P1)*(i-P1)*(i-P1)*M2))/(6*H1))+
        ((((P2-i)*Q1)+((i-P1)*Q2))/H1)-((H1/6)*(((P2-i)*M1)+((i-P1)*M2))));
        if (Yt>255) Yt=255;
        if (Yt<0) Yt=0;
        MatrizEcualizado[i]=Yt;
    };
    for (i=P2;i<=Xb;i++)
    {
        Yt=int((((P3-i)*(P3-i)*(P3-i)*M2)/(6*H2))+((((P3-i)*Q2)+((i-P2)*Q3))/H2)-
        ((H2/6)*(P3-i)*M2));
        if (Yt>255) Yt=255;
        if (Yt<0) Yt=0;
        MatrizEcualizado[i]=Yt;
    };
    return 0;
};

// Se ecualiza una Imagen a traves de una MatrizEcualizado calculada
// anteriormente a traves de la generacion de las curvas de Beizer o de
// Splines, ademas permite 8 tipos de ecualizados, asi:
// TipoEcualizado=  0 ("000") No ecualiza nada
//                  1 ("001") Solo ecualiza Componente Azul
//                  2 ("010") Solo ecualiza Componente Verde
//                  3 ("011") Ecualiza Componentes Verde y Azul
//                  4 ("100") Solo ecualiza Componente Roja
//                  5 ("101") Ecualiza Componentes Roja y Azul
//                  6 ("110") Ecualiza Componentes Roja y Verde
//                  7 ("111") Ecualiza las tres Componentes
int tImagen::Ecualiza(int *MatrizEcualizado,int TipoEcualizado)
{
 if ((TipoEcualizado<0) or (TipoEcualizado>7))
     exit (ERROR_DE_INTRODUCCION_DE_PARAMETROS_EN_ECUALIZADO);
 if (TipoEcualizado!=0)
 ASM_Ecualizado(Direccion,TotalBytes,TipoEcualizado,int(MatrizEcualizado));
 return 0;
};

// Realiza la correccion de una aberracion, que es la deformacion geometrica
// de una imagen, bien sea de tipo cojin o corse, que consiste en un aumento
// de la imagen con la distancia al ejeo (Distorsion positiva), o bien sea
// del tipo barril, que consiste en una disminucion de la imagen con la
// distancia al eje (Distorsion negativa), a traves de la introduccion de un
// factor de distorsion
int tImagen::CorrigeAberracion(double FactorDistorsion)
{
 char *CA;        // Definicion de variables
 int Dir_AberracionCorregida;
 CA=new char[TotalBytes];             // Reserva de memoria que albergara
 if (!CA) exit (ERROR_NO_HAY_MEMORIA);// la Imagen corregida
 Dir_AberracionCorregida=int(CA);
 ASM_Corrige_Aberracion(Direccion,Dir_AberracionCorregida,Ancho,Alto,
                        FactorDistorsion);
 delete M;
 M=CA;
 Direccion=Dir_AberracionCorregida;
 return 0;
};

// Dada una Imagen, rellena tres tablas, una para cada componente de color,
// con la transformada de Fourier en dos dimensiones
int tImagen::FFT2D( char *Comp_Rojo, char *Comp_Verde, char *Comp_Azul)
{
 ASM_Crea_Tablas_Imagen_Real(Direccion,int(Comp_Rojo),int(Comp_Verde),
                             int(Comp_Azul),Ancho);
 ASM_FFT2D (int(Comp_Rojo),int(Comp_Verde),int(Comp_Azul),Ancho,0);
 return 0;
};
```

```
// Dadas tres tablas, una para cada componente de color, calcula la parte
// real, la parte imaginaria, el modulo o la fase de dichas tablas (a traves
// de Opcion) y las representa a traves de una Imagen cuadrada de tamano
// Tam_Imagen.
// Opciones de calculo o representacion: Opcion="char1char2char3"
//    char1="r" parte real;="i" parte imaginaria;="m" modulo;="f" fase
//    char2(para char1="r","i","m")="n" normalizado;="l" logaritmico
//    char2(para char1="f")="n" normalizado;="d" desnormalizado
//    char3="o" ordenada;="d" desordenada
int tImagen::CalculaFFT2D( int Tam_Imagen, char *Comp_Rojo, char *Comp_Verde,
                           char *Comp_Azul, char *Opcion )
{
 char *Tabla_Real,*Tabla_Imag,*Tabla_Modulo,*Tabla_Fase;
 int Encontrado=0,i;     // Definicion de variables
 if ((Opcion[0]!='r' and Opcion[0]!='i' and Opcion[0]!='m' and Opcion[0]!='f')
     or (Opcion[0]!='f' and (Opcion[1]!='n' and Opcion[1]!='l'))
     or (Opcion[0]=='f' and (Opcion[1]!='n' and Opcion[1]!='d'))
     or (Opcion[2]!='o' and Opcion[2]!='d') or (Opcion[3]!=0))
     exit (ERROR_DE_INTRODUCCION_DE_PARAMETROS_EN_PINTA_FFT2D);
 if (Ancho!=Tam_Imagen or Alto!=Tam_Imagen)
 {
   delete M;
   TotalBytes=Tam_Imagen*Tam_Imagen*4; // Reserva de memoria que albergara
   M= new char [TotalBytes];           // la Imagen calculada
   if (!M) exit (ERROR_NO_HAY_MEMORIA);
   Direccion=int(M);
   Ancho=Tam_Imagen;
   Alto=Tam_Imagen;
 };
 // Comprobamos que el tamano de la Imagen sea multiplo de 2
 for (i=2;i<16384;i=i*2) if (Ancho==i) Encontrado=1;
 if (Encontrado==0) exit (ERROR_TAMANO_DE_FFT2D_NO_VALIDO);
 if (Opcion[0]=='r') // Pinta la Parte Real
 {
  Tabla_Real= new char [Ancho*Ancho*4*4];
  if (!Tabla_Real) exit (ERROR_NO_HAY_MEMORIA);
  ASM_Calcula_Parte_Real_Y_Maximo (int(Tabla_Real),int(Comp_Rojo),int(Comp_Verde),
                                   int(Comp_Azul),Ancho);
  if (Opcion[1]=='n') ASM_Calcula_Normalizacion (Direccion,int(Tabla_Real),Ancho);
  else ASM_Calcula_Logaritmo (Direccion,int(Tabla_Real),Ancho);
  if (Opcion[2]=='o') ASM_Ordena_FFT (Direccion,Ancho);
  delete Tabla_Real;
 }
 else
 {
    if (Opcion[0]=='i') // Pinta la Parte Imaginaria
    {
        Tabla_Imag= new char [Ancho*Ancho*4*4];
        if (!Tabla_Imag) exit (ERROR_NO_HAY_MEMORIA);
        ASM_Calcula_Parte_Imaginaria_Y_Maximo (int(Tabla_Imag),int(Comp_Rojo),
                                               int(Comp_Verde),int(Comp_Azul),Ancho);
        if (Opcion[1]=='n') ASM_Calcula_Normalizacion (Direccion,int(Tabla_Imag),
                                                       Ancho);
        else ASM_Calcula_Logaritmo (Direccion,int(Tabla_Imag),Ancho);
        if (Opcion[2]=='o') ASM_Ordena_FFT (Direccion,Ancho);
        delete Tabla_Imag;
    }
    else
    {
       if (Opcion[0]=='m') // Pinta el Modulo
       {
          Tabla_Modulo= new char [Ancho*Ancho*4*4];
          if (!Tabla_Modulo) exit (ERROR_NO_HAY_MEMORIA);
          ASM_Calcula_Modulo_Y_Maximo (int(Tabla_Modulo),int(Comp_Rojo),
                                       int(Comp_Verde),int(Comp_Azul),Ancho);
          if (Opcion[1]=='n') ASM_Calcula_Normalizacion(Direccion,int(Tabla_Modulo),
                                                        Ancho);
          else ASM_Calcula_Logaritmo (Direccion,int(Tabla_Modulo),Ancho);
          if (Opcion[2]=='o') ASM_Ordena_FFT (Direccion,Ancho);
          delete Tabla_Modulo;
       }
       else // Pinta la Fase
       {
         Tabla_Fase= new char [Ancho*Ancho*4*4];
```

```
            if (!Tabla_Fase) exit (ERROR_NO_HAY_MEMORIA);
888         ASM_Calcula_Fase_Maximo_Y_Minimo (int(Tabla_Fase),int(Comp_Rojo),
                                              int(Comp_Verde),int(Comp_Azul),Ancho);
            if (Opcion[1]=='n') ASM_Calcula_Fase_Normalizada(Direccion,int(Tabla_Fase),
                                                             Ancho);
            else ASM_Calcula_Fase_Desnormalizada (Direccion,int(Tabla_Fase),Ancho);
893         if (Opcion[2]=='o') ASM_Ordena_FFT (Direccion,Ancho);
            delete Tabla_Fase;
          };
       };
    };
898 return 0;
   };

   // Dadas tres tablas, una para cada componente de color, se calcula la
   // Transformada inversa de Fourier en dos dimensiones y se representa a
903 // traves de una imagen cuadrada de tamano Tam_Imagen
   int tImagen::InvFFT2D( int Tam_Imagen, char *Comp_Rojo, char *Comp_Verde,
                          char *Comp_Azul)
   {
    int Encontrado=0,i;  // Declaracion de variables
908 // Comprobamos que el tamano de la Imagen sea multiplo de 2
    for (i=2;i<16384;i=i*2) if (Tam_Imagen==i) Encontrado=1;
    if (Encontrado==0) exit (ERROR_TAMANO_DE_INVERSA_FFT2D_NO_VALIDO);
    if (Ancho!=Tam_Imagen or Alto!=Tam_Imagen)
    {
913    delete M;
       TotalBytes=Tam_Imagen*Tam_Imagen*4; // Reserva de memoria que albergara
       M=new char[TotalBytes];             // la transformada inversa
       if (!M) exit (ERROR_NO_HAY_MEMORIA);
       Direccion=int(M);
918    Ancho=Tam_Imagen;
       Alto=Tam_Imagen;
    };
    ASM_FFT2D(int(Comp_Rojo),int(Comp_Verde),int(Comp_Azul),Ancho,1);
    ASM_Calcula_Imagen_Real_A_Partir_De_Tablas(Direccion,int(Comp_Rojo),
923         int(Comp_Verde),int(Comp_Azul),Ancho);
    return 0;
   };

   // Dada una Imagen de tamano Ancho*Alto, esta se redimensiona, obteniendose
928 // una nueva Imagen de tamano Anchura*Altura
   int tImagen::Redimensiona( int Anchura, int Altura)
   {
    char *R;    // Definicion de variables
    int TotalBytesRedimensionado,Dir_Redimensionado;
933 if ((Anchura<Ancho) or (Altura<Alto))
        exit (ERROR_DE_INTRODUCCION_DE_PARAMETROS_EN_REDIMENSIONAMIENTO);
    TotalBytesRedimensionado=Altura*Anchura*4;
    R=new char[TotalBytesRedimensionado]; // Reserva de memoria que albergara
    if (!R) exit (ERROR_NO_HAY_MEMORIA);  // albergara la Imagen redimensionada
938 Dir_Redimensionado=int(R);
    ASM_Redimensiona (Direccion,Dir_Redimensionado,Ancho,Alto,Anchura,Altura);
    delete M;
    M=R;
    Direccion=Dir_Redimensionado;
943 Alto=Altura;
    Ancho=Anchura;
    TotalBytes=TotalBytesRedimensionado;
    return 0;
   };
948
   // Escribe Texto en modo grafico a traves de un BMP, a partir de la posicion
   // (X,Y) permitiendo un ColorTransparente, que implica que los puntos del
   // caracter que coincidan con ese color (combinacion RGB) no sea pintado.
   // La estructura que debe presentar el BMP es la siguiente:
953 //                            -------------
   //                           |0123456789   |
   //                           |ABCDEFGHIJKLM|
   //                           |NOPQRSTUVWXYZ|
   //                           |abcdefghijklm|
958 //                           |nopqrstuvwxyz|
   //                            -------------
   int tImagen::Escribe(char* Texto,int X,int Y,int ColorTransparente,tPantalla P)
```

```
{
 int AnchuraCaracter,AlturaCaracter; // Declaracion de variables
 if (X<0 or X>P.AnchoPantalla or Y<0 or Y>P.AltoPantalla)
        exit (ERROR_DE_INTRODUCCION_DE_PARAMETROS_EN_ESCRIBE_TEXTO);
 AnchuraCaracter=Ancho/13;
 AlturaCaracter=Alto/5;
 ASM_Escribe(int(Texto),Direccion,Ancho,Alto,AnchuraCaracter,AlturaCaracter,
             P.Selector,P.AnchoPantalla,P.AltoPantalla,X,Y,ColorTransparente);
 return 0;
};
```

Listado A.5: imagen.asm

```
; Fichero: imagen_.asm

[BITS 32]
[GLOBAL _ASM_Dibuja] ; 00
[GLOBAL _ASM_Captura]; 01
[GLOBAL _ASM_Copia]  ; 02
[GLOBAL _ASM_NOT] ;03
[GLOBAL _ASM_Recorta] ;04
[GLOBAL _ASM_Suma] ;05
[GLOBAL _ASM_Resta] ;06
[GLOBAL _ASM_Traslada] ;07
[GLOBAL _ASM_Rotacion] ;08
[GLOBAL _ASM_XOR] ;09
[GLOBAL _ASM_Magnificacion_Entera] ;10
[GLOBAL _ASM_Magnificacion_No_Entera] ;11
[GLOBAL _ASM_Minimizacion_Entera] ;12
[GLOBAL _ASM_Minimizacion_No_Entera] ;13
[GLOBAL _ASM_Convolucion_Entera] ;14
[GLOBAL _ASM_Convolucion_Real] ;15
[GLOBAL _ASM_OR] ;16
[GLOBAL _ASM_Redimensiona] ;17
[GLOBAL _ASM_TransfAfin] ;18
[GLOBAL _ASM_Ecualizado] ;19
[GLOBAL _ASM_Escribe] ;20
[GLOBAL _ASM_Corrige_Aberracion] ;21
[GLOBAL _ASM_FFT2D] ;22
[GLOBAL _ASM_Crea_Tablas_Imagen_Real] ;23
[GLOBAL _ASM_Calcula_Modulo_Y_Maximo] ;24
[GLOBAL _ASM_Calcula_Normalizacion] ;25
[GLOBAL _ASM_Calcula_Logaritmo] ;26
[GLOBAL _ASM_Ordena_FFT] ;27
[GLOBAL _ASM_Calcula_Fase_Maximo_Y_Minimo] ;28
[GLOBAL _ASM_Calcula_Fase_Desnormalizada] ;29
[GLOBAL _ASM_Calcula_Fase_Normalizada] ;30
[GLOBAL _ASM_Calcula_Imagen_Real_A_Partir_De_Tablas] ;31
[GLOBAL _ASM_Calcula_Parte_Real_Y_Maximo] ;32
[GLOBAL _ASM_Calcula_Parte_Imaginaria_Y_Maximo] ;33
[GLOBAL _ASM_Copia_Video] ;34
[SECTION .text]

_ASM_Dibuja:
; Dibuja en pantalla una Imagen de tamano Ancho por Alto a partir de la
; posicion en pantalla que indican las variables (coordenadas) X e Y.
; La pantalla comenzara en la direccion 0
Selector_00        equ  8
AnchoPantalla_00   equ 12
Dir_Dibuja_00      equ 16
Ancho_00           equ 20
Alto_00            equ 24
X_00               equ 28
Y_00               equ 32
       push ebp
       mov ebp,esp ; Prologo
       pushad
       push gs     ; Guardamos en la pila el selector de segmento gs
       mov ax,[ebp+Selector_00]
       mov gs,ax       ; Selector de segmento
       ; Comprueba el ancho de la imagen y segun sea este realizara un
       ; tratamiento diferente, optimizando cada subrrutina
       mov eax,[ebp+Ancho_00] ; eax=Ancho
```

```
        xor edx,edx
        mov ebx,4  ; ebx=4
        div ebx    ; eax=Cociente(Ancho/4) edx=Resto
        cmp eax,0
65      jz near .DibMenorDe4Pixeles ; Salta si el ancho es menor de 4
        cmp edx,0
        jz near .DibResto0 ; Salta si el ancho es multiplo de 4
        cmp edx,1
        jz near .DibResto1 ; Salta si la imagen es de la forma (W*4)+1
70      cmp edx,2     ; siendo W=Ancho/4
        jz near .DibResto2 ; Salta si la imagen es de la forma (W*4)+2
;Si no salta a ningun bucle el resto sera 3(=edx)
.DibResto3:
        mov eax,[ebp+AnchoPantalla_00] ; eax=AnchoPantalla
75      mov ebx,eax  ; ebx=AnchoPantalla
        mul dword [ebp+Y_00] ; edxeax=AnchoPantalla*Y
        add eax,[ebp+X_00]   ; eax=(AnchoPantalla*Y)+X
        shl eax,2            ; eax=[(AnchoPantalla*Y)+X]*4 -> Desplazamiento
        mov esi,eax          ; [gs:esi] apunta a la posicion en pantalla donde
80                           ; hay que comenzar a pintar
        mov eax,[ebp+Ancho_00] ;eax->ancho de la imagen
        sub ebx,eax  ; ebx=AnchoPantalla-Ancho
        inc ebx      ; ebx=AnchoPantalla-Ancho+1
        shl ebx,2    ; ebx=[AnchoPantalla-Ancho+1]*4 ->Salto de linea
85      shr eax,2    ; eax=Ancho/2 -> Bucle X
        mov ecx,[ebp+Alto_00] ; ecx=Alto -> Bucle Y
        mov edi,[ebp+Dir_Dibuja_00] ; [ds:edi] apunta a los datos
.DibYResto3:
        push ecx
90      mov ecx,eax
.DibXResto3:
        movdqu xmm0,[edi]     ; Va copiando la Imagen en pantalla fila
        movdqu [gs:esi],xmm0  ; a fila hasta completar todas filas,
        add esi,16            ; y para cada fila vamos copiando de 4 en
95      add edi,16            ; 4 pixeles
        loop .DibXResto3
        movq mm0,[edi]        ; Como el resto es 3 hay que pintar para cada
        movq [gs:esi],mm0     ; linea 3 pixeles mas
        add edi,8
100     add esi,8
        mov edx,[edi]
        mov [gs:esi],edx
        add edi,4
        add esi,ebx
105     pop ecx
        loop .DibYResto3
        pop gs ; Recuperamos de la pila el selector de segmento
        popad
        pop ebp ; Epilogo
110     ret
.DibResto2:
        mov eax,[ebp+AnchoPantalla_00] ; eax=AnchoPantalla
        mov ebx,eax  ; ebx=AnchoPantalla
        mul dword [ebp+Y_00] ; edxeax=AnchoPantalla*Y
115     add eax,[ebp+X_00]   ; eax=(AnchoPantalla*Y)+X
        shl eax,2            ; eax=[(AnchoPantalla*y)+X]*4 -> Desplazamiento
        mov esi,eax          ; [gs:esi] apunta a la posicion en pantalla donde
                             ; hay que comenzar a pintar
        mov eax,[ebp+Ancho_00] ; eax=Ancho
120     sub ebx,eax  ; ebx=AnchoPantalla-Ancho
        add ebx,2    ; ebx=AnchoPantalla-Ancho+2
        shl ebx,2    ; ebx=[AnchoPantalla-Ancho+2]*4 -> Salto de linea
        shr eax,2    ; eax=Ancho/2 -> Bucle X
        mov ecx,[ebp+Alto_00] ; ecx=Alto -> Bucle Y
125     mov edi,[ebp+Dir_Dibuja_00] ; [ds:edi] apunta a los datos
.DibYResto2:
        push ecx
        mov ecx,eax
.DibXResto2:
130     movdqu xmm0,[edi]     ; Vamos copiando la imagen en pantalla fila
        movdqu [gs:esi],xmm0  ; a fila y para cada fila vamos copiando
        add esi,16            ; de 4 en 4 pixeles
        add edi,16
        loop .DibXResto2
```

```
135         movq mm0,[edi]          ; Como el resto es 2 tenemos que ir pintando
            movq [gs:esi],mm0       ; dos pixeles mas por fila
            add edi,8
            add esi,ebx
            pop ecx
140         loop .DibYResto2
            pop gs ; Recuperamos de la pila el selector de segmento
            popad
            pop ebp ; Epilogo
            ret
145 .DibResto1:
            mov eax,[ebp+AnchoPantalla_00] ; eax=AnchoPantalla
            mov ebx,eax ; ebx=AnchoPantalla
            mul dword [ebp+Y_00] ; edxeax=AnchoPantalla*Y
            add eax,[ebp+X_00]   ; eax=(AnchoPantalla*Y)+X
150         shl eax,2            ; eax=[AnchoPantalla*Y)+X]*4 -> Desplazamiento
            mov esi,eax          ; [gs:esi] apunta a la posicion en pantalla donde
                                 ; hay que comenzar a pintar
            mov eax,[ebp+Ancho_00] ;eax=Ancho
            sub ebx,eax      ; ebx=AnchoPantalla-Ancho
155         inc ebx          ; ebx=AnchoPantalla-Ancho+1
            shl ebx,2        ; ebx=[AnchoPantalla-Ancho+1]*4 -> Salto de linea
            shr eax,2        ; eax=Ancho/2 -> Bucle X
            mov ecx,[ebp+Alto_00] ;ecx=Alto -> Bucle Y
            mov edi,[ebp+Dir_Dibuja_00] ; [ds:edi] apunta a los datos
160 .DibYResto1:
            push ecx
            mov ecx,eax
    .DibXResto1:
            movdqu xmm0,[edi]       ; Vamos copiando la Imagen en pantalla fila a
165         movdqu [gs:esi],xmm0    ; fila y para cada fila vamos copiando de
            add esi,16              ; 4 en 4 pixeles
            add edi,16
            loop .DibXResto1
            mov edx,[edi]           ; Como el resto es 1 tenemos que pintar un
170         mov [gs:esi],edx        ; pixel mas por fila
            add edi,4
            add esi,ebx
            pop ecx
            loop .DibYResto1
175         pop gs ; Recuperamos de la pila el selectro de segmento
            popad
            pop ebp ; Epilogo
            ret
    .DibResto0:
180         mov eax,[ebp+AnchoPantalla_00] ; eax=AnchoPantalla
            mov ebx,eax ; ebx=AnchoPantalla
            mul dword [ebp+Y_00] ; edxeax=AnchoPantalla*Y
            add eax,[ebp+X_00]   ; eax=(AnchoPantalla*Y)+X
            shl eax,2            ; eax=[(AnchoPantalla*Y)+X]*4 -> Desplazamiento
185         mov esi,eax          ; [gs:esi] apunta a la posicion en pantalla donde
                                 ; hay que comenzar a pintar
            mov eax,[ebp+Ancho_00] ; eax=Ancho
            sub ebx,eax      ; ebx=AnchoPantalla-Ancho
            shl ebx,2        ; ebx=(AnchoPantalla-Ancho)*4 -> Salto de linea
190         shr eax,2        ; eax=Ancho/2 -> Bucle X
            mov ecx,[ebp+Alto_00] ; ecx=Alto -> Bucle Y
            mov edi,[ebp+Dir_Dibuja_00] ; [ds:edi] apunta a los datos
    .DibYResto0:
            push ecx
195         mov ecx,eax
    .DibXResto0:
            movdqu xmm0,[edi]       ; Vamos copiando la imagen en pantalla fila a
            movdqu [gs:esi],xmm0    ; fila y para cada fila vamos copiando de
            add esi,16              ; 4 en 4 pixeles
200         add edi,16
            loop .DibXResto0
            add esi,ebx
            pop ecx
            loop .DibYResto0
205         pop gs ; Recuperamos de la pila el selector de segmento gs
            popad
            pop ebp ; Epilogo
            ret
```

```
    .DibMenorDe4Pixeles:
210         mov eax,[ebp+AnchoPantalla_00] ; eax=AnchoPantalla
            mov ebx,eax   ; ebx=AnchoPantalla
            mul dword [ebp+Y_00] ; edxeax=AnchoPantalla*Y
            add eax,[ebp+X_00]   ; eax=(AnchoPantalla*Y)+X
            shl eax,2            ; eax=[(AnchoPantalla*Y)+X]*4 -> Desplazamiento
215         mov esi,eax          ; [gs:esi] apunta a la posicion en pantalla donde
                                 ; hay que comenzar a pintar
            mov eax,[ebp+Ancho_00] ; eax=Ancho -> Bucle X
            sub ebx,eax   ; ebx=AnchoPantalla-Ancho
            shl ebx,2     ; ebx=(AnchoPantalla-Ancho)*4 -> Salto de linea
220         mov ecx,[ebp+Alto_00] ; ecx=Alto -> Bucle Y
            mov edi,[ebp+Dir_Dibuja_00] ; [ds:edi] -> apunta a los datos
    .DibYMenorDe4:
            push ecx
            mov ecx,eax
225 .DibXMenorDe4:               ; Vamos copiando la imagen en pantalla fila
            mov edx,[edi]        ; a fila y para cada fila vamos pixel a
            mov [gs:esi],edx     ; pixel
            add edi,4
            add esi,4
230         loop .DibXMenorDe4
            add esi,ebx
            pop ecx
            loop .DibYMenorDe4
            pop gs ; Recuperamos de la pila el selector de segmento
235         popad
            pop ebp ; Epilogo
            ret

    _ASM_Captura:
240 ; Captura de Pantalla una Imagen a partir de la posicion (X,Y) con una
    ; determinada Altura y Anchura. La pantalla comenzara en la direccion 0
    Selector_01         equ  8
    AnchoPantalla_01    equ 12
    Dir_Captura_01      equ 16
245 Ancho_01            equ 20
    Alto_01             equ 24
    X_01                equ 28
    Y_01                equ 32
            push ebp
250         mov ebp,esp ; Prologo
            pushad
            push gs ; Guardamos en la pila el selector de segmento
            mov ax,[ebp+Selector_01]
            mov gs,ax        ; Selector de segmento
255         mov edi,[ebp+Dir_Captura_01] ; edi -> Imagen Destino
            ; Comprueba el ancho de la imagen y segun sea este realizara un
            ; tratamiento diferente, optimizando cada subrrutina
            mov eax,[ebp+Ancho_01] ; eax=Ancho
            xor edx,edx
260         mov ebx,4    ; ebx=4
            div ebx      ; eax=Cociente(Ancho/4) edx=Resto
            cmp eax,0
            jz near .CapturaMenorDe4Pixeles ; Salta si el ancho es menor de 4
            cmp edx,0
265         je near .CapturaResto0 ; Salta si el ancho de captura es multiplo de 4
            cmp edx,1
            je near .CapturaResto1 ; Salta si el resto es 1
            cmp edx,2
            je near .CapturaResto2 ; Salta si el resto es 2
270 ; Si no ha saltado es porque el resto es 3(=edx)
    .CapturaResto3:
            mov eax,[ebp+AnchoPantalla_01] ; eax=AnchoPantalla
            mov ebx,eax ; ebx=AnchoPantalla
            mul dword [ebp+Y_01] ; edxeax=AnchoPantalla*Y
275         add eax,[ebp+X_01]   ; eax=(AnchoPantalla*Y)+X
            shl eax,2            ; eax=[(AnchoPantalla*Y)+X]*4 -> Desplazamiento
            mov esi,eax          ; [gs:esi] apunta a la posicion en pantalla donde
                                 ; hay que comenzar a coger los pixeles
            mov eax,[ebp+Ancho_01] ; eax=Ancho
280         sub ebx,eax  ; ebx=AnchoPantalla-Ancho
            inc ebx      ; ebx=AnchoPantalla-Ancho+1
            shl ebx,2    ; ebx=[AnchoPantalla-Ancho+1]*4 -> Salto de linea
```

```
            shr eax,2     ; eax=Ancho/2 -> Bucle X
            mov ecx,[ebp+Alto_01] ; ecx=Alto -> Bucle Y
.CapturaYResto3:
            push ecx
            mov ecx,eax
.CapturaXResto3:
            movdqu xmm0,[gs:esi]   ; Vamos copiando el contenido de la pantalla
            movdqu [edi],xmm0      ; fila a fila en nuestra Imagen Destino, y
            add esi,16             ; para cada fila se va haciendo de 4 en 4
            add edi,16             ; pixeles
            loop .CapturaXResto3
            movq mm0,[gs:esi]      ; Como el resto es 3 tenemos que ir copiando
            movq [edi],mm0         ; 3 pixeles de la pantalla en nuestra Imagen
            add esi,8              ; para cada fila
            add edi,8
            mov edx,[gs:esi]
            mov [edi],edx
            add edi,4
            add esi,ebx
            pop ecx
            loop .CapturaYResto3
            pop gs ; Recuperamos el selector de segmento
            popad
            pop ebp ; Epilogo
            ret
.CapturaResto2:
            mov eax,[ebp+AnchoPantalla_01] ; eax=AnchoPantalla
            mov ebx,eax ; ebx=AnchoPantalla
            mul dword [ebp+Y_01] ; edxeax=AnchoPantalla*Y
            add eax,[ebp+X_01]   ; eax=(AnchoPantalla*Y)+X
            shl eax,2            ; eax=[(AnchoPantalla*Y)+X]*4 -> Desplazamiento
            mov esi,eax          ; [gs:esi] apunta a la posicion en pantalla donde
                                 ; hay que comenzar a coger los pixeles
            mov eax,[ebp+Ancho_01] ; eax=Ancho
            sub ebx,eax  ; ebx=AnchoPantalla-Ancho
            add ebx,2    ; ebx=AnchoPantalla-Ancho+2
            shl ebx,2    ; ebx=[AnchoPantalla-Ancho+2]*4 -> Salto de linea
            shr eax,2    ; eax=Ancho/2 -> Bucle X
            mov ecx,[ebp+Alto_01] ; ecx=Alto -> Bucle Y
.CapturaYResto2:
            push ecx
            mov ecx,eax
.CapturaXResto2:
            movdqu xmm0,[gs:esi]   ; Vamos copiando el contenido de la pantalla
            movdqu [edi],xmm0      ; fila a fila en nuestra Imagen Destino, y
            add esi,16             ; para cada fila se va haciendo de 4 en 4
            add edi,16             ; pixeles
            loop .CapturaXResto2
            movq mm0,[gs:esi]      ; Como el resto es 2 hay que ir copiando 2
            movq [edi],mm0         ; pixeles mas para cada fila
            add edi,8
            add esi,ebx
            pop ecx
            loop .CapturaYResto2
            pop gs ; Recuperamos de la pila el selector de segmento
            popad
            pop ebp ; Epilogo
            ret
.CapturaResto1:
            mov eax,[ebp+AnchoPantalla_01] ; eax=AnchoPantalla
            mov ebx,eax ; ebx=AnchoPantalla
            mul dword [ebp+Y_01] ; edxeax=AnchoPantalla*Y
            add eax,[ebp+X_01]   ; eax=(AnchoPantalla*Y)+X
            shl eax,2            ; eax=[(AnchoPantalla*Y)+X]*4 -> Desplazamiento
            mov esi,eax          ; [gs:esi] apunta a la posicion en pantalla donde
                                 ; hay que comenzar a coger los pixeles
            mov eax,[ebp+Ancho_01] ; eax=Ancho
            sub ebx,eax  ; ebx=AnchoPantalla-Ancho
            inc ebx      ; ebx=AnchoPantalla-Ancho+1
            shl ebx,2    ; ebx=[AnchoPantalla-Ancho+1]*4 -> Salto de linea
            shr eax,2    ; eax=Ancho/2 -> Bucle X
            mov ecx,[ebp+Alto_01] ; ecx=Alto -> Bucle Y
.CapturaYResto1:
            push ecx
```

```
        mov ecx,eax
.CapturaXResto1:
        movdqu xmm0,[gs:esi] ; Vamos copiando el contenido de la pantalla
        movdqu [edi],xmm0    ; fila a fila a nuestra Imagen Destino, y para
        add esi,16           ; cada fila se va haciendo de 4 en 4 pixeles
        add edi,16
        loop .CapturaXResto1
        mov edx,[gs:esi]     ; Como el resto es 1 hay que ir copiando un
        mov [edi],edx        ; pixel mas para cada fila
        add edi,4
        add esi,ebx
        pop ecx
        loop .CapturaYResto1
        pop gs ; Recuperamos de la pila el selector de segmento gs
        popad
        pop ebp ; Epilogo
        ret
.CapturaResto0:
        mov eax,[ebp+AnchoPantalla_01] ; eax=AnchoPantalla
        mov ebx,eax ; ebx=AnchoPantalla
        mul dword [ebp+Y_01] ; edxeax=AnchoPantalla*Y
        add eax,[ebp+X_01]   ; eax=(AnchoPantalla*Y)+X
        shl eax,2            ; eax=[(AnchoPantalla*Y)+X]*4 -> Desplazamiento
        mov esi,eax          ; [gs:esi] apunta a la posicion en pantalla donde
                             ; hay que comenzar a coger los pixeles
        mov eax,[ebp+Ancho_01] ; eax=Ancho
        sub ebx,eax  ; ebx=AnchoPantalla-Ancho
        shl ebx,2    ; ebx=[AnchoPantalla-Ancho]*4 -> Salto de linea
        shr eax,2    ; eax=Ancho/2 -> Bucle X
        mov ecx,[ebp+Alto_01] ; ecx=Alto -> Bucle Y
.CapturaYResto0:
        push ecx
        mov ecx,eax
.CapturaXResto0:
        movdqu xmm0,[gs:esi]  ; Vamos copiando el contenido de la pantalla
        movdqu [edi],xmm0     ; fila a fila en nuestra Imagen Destino, y
        add esi,16            ; para cada fila se va haciendo de 4 en 4
        add edi,16            ; pixeles
        loop .CapturaXResto0
        add esi,ebx
        pop ecx
        loop .CapturaYResto0
        pop gs ; Recuperamos de la pila el selector de segmento gs
        popad
        pop ebp ; Epilogo
        ret
.CapturaMenorDe4Pixeles:
        mov eax,[ebp+AnchoPantalla_01] ; eax=AnchoPantalla
        mov ebx,eax ; ebx=AnchoPantalla
        mul dword [ebp+Y_01] ; edxeax=AnchoPantalla*Y
        add eax,[ebp+X_01]   ; eax=(AnchoPantalla*Y)+X
        shl eax,2            ; eax=[(AnchoPantalla*Y)+X]*4 -> Desplazamiento
        mov esi,eax          ; [gs:esi] apunta a la posicion en pantalla donde
                             ; hay que comenzar a coger los pixeles
        mov eax,[ebp+Ancho_01] ; eax=Ancho -> Bucle X
        sub ebx,eax  ; ebx=AnchoPantalla-Ancho
        shl ebx,2    ; ebx=[AnchoPantalla-Ancho]*4 -> Salto de linea
        mov ecx,[ebp+Alto_01] ; ecx=Alto -> Bucle Y
.CapturaYMenorDe4Pixeles:
        push ecx
        mov ecx,eax
.CapturaXMenorDe4Pixeles:  ; Vamos copiando el contenido de la pantalla
        mov edx,[gs:esi]   ; fila a fila en nuestra Imagen Destino, y
        mov [edi],edx      ; para cada fila se va haciendo pixel a pixel
        add esi,4
        add edi,4
        loop .CapturaXMenorDe4Pixeles
        add esi,ebx
        pop ecx
        loop .CapturaYMenorDe4Pixeles
        pop gs ; Recuperamos de la pila el selector de segmento gs
        popad
        pop ebp ; Epilogo
        ret
```

```
_ASM_Copia:
; Copia un bloque de datos o Imagen de una direccion Origen (OrgDespl)
; en una direccion Destino (DestDespl)
Dir_Origen_02   equ  8
Dir_Destino_02 equ 12
TotalBytes_02   equ 16
        push ebp
        mov ebp,esp ; Prologo
        pushad
        mov edi,[ebp+Dir_Origen_02] ; edi -> Bloque de Datos Origen
        mov esi,[ebp+Dir_Destino_02]; esi -> Bloque de Datos Destino
        mov eax,[ebp+TotalBytes_02] ; eax=TotalBytes a copiar
        ; Compruebo el numero de bytes que hay que copiar y segun el resto
        ; haremos un tratamiento diferente de copia
        xor edx,edx
        mov ebx,16  ; ebx=16
        div ebx     ; eax=Cociente(TotalBytes/16) edx=Resto
        cmp eax,0
        jz near .CopMenorDe16Bytes ; Salta si el TotalBytes es menor de 16
        mov ecx,eax   ; ecx=TotalBytes/16 -> Bucle
        cmp edx,0
        jz .CopResto0 ; Salta si el TotalBytes es multiplo de 16
        cmp edx,4
        jz .CopResto4 ; Salta si el TotalBytes es de la forma (W*16)+4
        cmp edx,8     ; siendo W=TotalBytes/16
        jz .CopResto8 ; Salta si el TotalBytes es de la forma (W*16)+8
; Si no salta a ninguna etiqueta es que el resto es 12
.CopResto12:
        movdqu xmm0,[edi] ; Vamos copiando de 4 en 4 pixeles
        movdqu [esi],xmm0
        add edi,16
        add esi,16
        loop .CopResto12  ; Como el resto es 12 hay que copiar 3 pixeles
        movq mm0,[edi]    ; mas
        movq [esi],mm0
        add edi,8
        add esi,8
        mov edx,[edi]
        mov [esi],edx
        popad
        pop ebp ; Epilogo
        ret
.CopResto8:
        movdqu xmm0,[edi] ; Vamos copiando de 4 en 4 pixeles
        movdqu [esi],xmm0
        add edi,16
        add esi,16
        loop .CopResto8
        movq mm0,[edi]    ; Como el resto es 8 hay que copiar 2 pixeles
        movq [esi],mm0    ; mas
        popad
        pop ebp ; Epilogo
        ret
.CopResto4:
        movdqu xmm0,[edi] ; Vamos copiando de 4 en 4 pixeles
        movdqu [esi],xmm0
        add edi,16
        add esi,16
        loop .CopResto4
        mov edx,[edi]     ; Como el resto es 4 hay que copiar un pixel
        mov [esi],edx     ; mas
        popad
        pop ebp ; Epilogo
        ret
.CopResto0:
        movdqu xmm0,[edi] ; Vamos copiando de 4 en 4 pixeles
        movdqu [esi],xmm0
        add edi,16
        add esi,16
        loop .CopResto0
        popad
        pop ebp ; Epilogo
        ret
```

```
505 .CopMenorDe16Bytes:
        mov ecx,edx    ; ecx=Resto
        shr ecx,2      ; ecx=Resto/4 -> Pixeles que hay que copiar, ya que
    .CopRepetir:       ; el Resto solo puede valer 4,8 o 12, asi el numero
        mov edx,[edi] ; de pixeles a copiar sera 1,2 o 3
510     mov [esi],edx
        add edi,4
        add esi,4
        loop .CopRepetir
        popad
515     pop ebp ; Epilogo
        ret

    _ASM_NOT:
    ; Realiza la funcion logica NOT de un bloque de datos o Imagen
520 Dir_NOT_03   equ  8
    Num_Bytes_03 equ 12
        push ebp
        mov ebp,esp ; Prologo
        pushad
525     mov edi,[ebp+Dir_NOT_03] ; edi -> Direccion Imagen
        movdqu xmm1,[TODO_UNOS] ; xmm1=FFFF|FFFF|FFFF|FFFF para realizar el NOT
        movdq2q mm1,xmm1 ; mm1=FFFF|FFFF para realizar el NOT
        xor edx,edx
        ; Compruebo el numero de bytes que hay que tratar y segun el resto
530     ; haremos un tratamiento diferente
        mov eax,[ebp+Num_Bytes_03] ; eax=NumBytes
        mov ebx,16  ; ebx=16
        div ebx     ; eax=Cociente(NumBytes/16) edx=Resto
        cmp eax,0
535     jz near .NOTMenorDe16Bytes ; Salta si Num_Bytes menor de 16
        mov ecx,eax ; ecx=NumBytes/16 -> Bucle
        cmp edx,0
        je near .NOTResto0 ; Salta si el resto es 0
        cmp edx,4
540     je near .NOTResto4 ; Salta si el resto es 4
        cmp edx,8
        je near .NOTResto8 ; Salta si el resto es 8
    ; Si no salta a ninguna etiqueta es por que el resto es 12(=edx)
    .NOTResto12:
545     movdqu xmm0,[edi]  ; Vamos haciendo el NOT de 4 en 4 pixeles
        pandn xmm0,xmm1
        movdqu [edi],xmm0
        add edi,16
        loop .NOTResto12
550     movq mm0,[edi]     ; Como el resto es 12 habra que realizar el
        pandn mm0,mm1      ; NOT a 3 pixeles mas
        movq [edi],mm0
        add edi,8
        mov edx,[edi]
555     not edx
        mov [edi],edx
        popad
        pop ebp ; Epilogo
        ret
560 .NOTResto8:
        movdqu xmm0,[edi]  ; Vamos haciendo el NOT de 4 en 4 pixeles
        pandn xmm0,xmm1
        movdqu [edi],xmm0
        add edi,16
565     loop .NOTResto8
        movq mm0,[edi]     ; Como el resto es 8 habra que realizar el
        pandn mm0,mm1      ; NOT a 2 pixeles mas
        movq [edi],mm0
        popad
570     pop ebp ; Epilogo
        ret
    .NOTResto4:
        movdqu xmm0,[edi]  ; Vamos realizando el NOT de 4 en 4 pixeles
        pandn xmm0,xmm1
575     movdqu [edi],xmm0
        add edi,16
        loop .NOTResto4
        mov edx,[edi]      ; Como el resto es 4 habra que realizar el
```

```
            not edx             ; NOT a un pixel mas
            mov [edi],edx
            popad
            pop ebp ; Epilogo
            ret
    .NOTResto0:
            movdqu xmm0,[edi]   ; Vamos realizando el NOT de 4 en 4 pixeles
            pandn xmm0,xmm1
            movdqu [edi],xmm0
            add edi,16
            loop .NOTResto0
            popad
            pop ebp ; Epilogo
            ret
    .NOTMenorDe16Bytes:
            mov ecx,edx    ; ecx=Resto
            shr ecx,2      ; ecx=Resto/4 -> Pixeles que hay que copiar, ya que
    .NOTRepetir:           ; el Resto solo puede valer 4,8 o 12, asi el numero
            mov edx,[edi] ; de pixeles a copiar sera 1,2 o 3
            not edx
            mov [edi],edx
            add edi,4
            loop .NOTRepetir
            popad
            pop ebx ; Epilogo
            ret

    _ASM_Recorta:
    ; Dada una Imagen Origen o un bloque de datos situado en una direccion
    ; Dir_Origen, se copia una parte de dicha Imagen con un tamano Anchura por
    ; Altura a partir de la posicion X e Y de dicha Imagen a una nueva direccion
    ; DirRecorte
    Dir_Origen_04 equ  8
    Ancho_04      equ 12
    X_04          equ 16
    Y_04          equ 20
    Anchura_04    equ 24
    Altura_04     equ 28
    DirRecorte_04 equ 32
           push ebp
           mov ebp,esp ; Prologo
           pushad
           ; Compruebo la Anchura del recorte y segun sea esta y el resto
           ; haremos un tratamiento diferente
           mov eax,[ebp+Anchura_04] ; eax=Anchura del recorte
           xor edx,edx
           mov ebx,4  ; ebx=4
           div ebx    ; eax=Cociente(Anchura/4) edx=Resto
           cmp eax,0
           jz near .RecMenorDe4Pixeles ; Salta si la Anchura es menor de 4
           cmp edx,0
           jz near .RecResto0 ; Salta si la Anchura es multiplo de 4
           cmp edx,1
           jz near .RecResto1 ; Salta si el recorte es de la forma (W*4)+1
           cmp edx,2     ; siendo W=X_04/4
           jz near .RecResto2 ; Salta si el recorte es de la forma (W*4)+2
    ;Si no salta a ningun bucle el resto sera 3(=edx)
    .RecResto3:
           mov eax,[ebp+Ancho_04] ; eax=Ancho
           mov ebx,eax  ; ebx=Ancho
           mul dword [ebp+Y_04] ; edxeax=Ancho*Y
           add eax,[ebp+X_04]   ; eax=(Ancho*Y)+X
           shl eax,2            ; eax=[(Ancho*Y)+X]*4
           mov edi,eax
           add edi,[ebp+Dir_Origen_04] ; edi -> Apunta al primer dato a recortar
           mov eax,[ebp+Anchura_04] ; eax=Anchura
           sub ebx,eax  ; ebx=Ancho-Anchura
           inc ebx      ; ebx=Ancho-Anchura+1
           shl ebx,2    ; ebx=(Ancho-Anchura+1)*4 -> Salto de linea
           shr eax,2    ; eax=Anchura/4 -> Bucle X
           mov ecx,[ebp+Altura_04] ; ecx=Altura -> Bucle Y
           mov esi,[ebp+DirRecorte_04] ; esi -> Imagen Destino
    .RecYResto3:
           push ecx
```

```
        mov ecx,eax
    .RecXResto3:
655     movdqu xmm0,[edi] ; Vamos copiando la Imagen Origen en la Imagen
        movdqu [esi],xmm0 ; Destino de fila en fila, hasta completar todas
        add edi,16        ; filas, y para cada fila se va copiando una imagen
        add esi,16        ; en la otro de 4 en 4 pixeles
        loop .RecXResto3
660     movq mm0,[edi]    ; Como el resto es 3 habra que copiar 3 pixeles
        movq [esi],mm0    ; mas por fila
        add edi,8
        add esi,8
        mov edx,[edi]
665     mov [esi],edx
        add esi,4
        add edi,ebx
        pop ecx
        loop .RecYResto3
670     popad
        pop ebp ; Epilogo
        ret
    .RecResto2:
        mov eax,[ebp+Ancho_04] ; eax=Ancho
675     mov ebx,eax  ; ebx=Ancho
        mul dword [ebp+Y_04] ; edxeax=Ancho*Y
        add eax,[ebp+X_04]   ; eax=(Ancho*Y)+X
        shl eax,2            ; eax=[(Ancho*Y)+X]*4
        mov edi,eax
680     add edi,[ebp+Dir_Origen_04] ; edi -> Apunta al primer dato a recortar
        mov eax,[ebp+Anchura_04] ; eax=Anchura
        sub ebx,eax  ; ebx=Ancho-Anchura
        add ebx,2    ; ebx=Ancho-Anchura+2
        shl ebx,2    ; ebx=(Ancho-Anchura+2)*4 -> Salto de linea
685     shr eax,2    ; eax=Anchura/4 -> Bucle X
        mov ecx,[ebp+Altura_04] ; ecx=Altura -> Bucle Y
        mov esi,[ebp+DirRecorte_04] ; esi -> Imagen Destino
    .RecYResto2:
        push ecx
690     mov ecx,eax
    .RecXResto2:
        movdqu xmm0,[edi] ; Vamos copiando la imagen Origen en la Imagen
        movdqu [esi],xmm0 ; Destino fila a fila, y para cada fila vamos
        add edi,16        ; copiando de 4 en 4 pixeles
695     add esi,16
        loop .RecXResto2
        movq mm0,[edi]    ; Como el resto es 2 habra que copiar 2 pixeles
        movq [esi],mm0    ; mas por fila
        add esi,8
700     add edi,ebx
        pop ecx
        loop .RecYResto2
        popad
        pop ebp ; Epilogo
705     ret
    .RecResto1:
        mov eax,[ebp+Ancho_04] ; eax=Ancho
        mov ebx,eax  ; ebx=Ancho
        mul dword [ebp+Y_04] ; edxeax=Ancho*Y
710     add eax,[ebp+X_04]   ; eax=(Ancho*Y)+X
        shl eax,2            ; eax=[(Ancho*Y)+X]*4
        mov edi,eax
        add edi,[ebp+Dir_Origen_04] ; edi -> Apunta al primer dato a recortar
        mov eax,[ebp+Anchura_04] ; eax=Anchura
715     sub ebx,eax  ; ebx=Ancho-Anchura
        inc ebx      ; ebx=Ancho-Anchura+1
        shl ebx,2    ; ebx=(Ancho-Anchura+1)*4 -> Salto de linea
        shr eax,2    ; eax=Anchura/4 -> Bucle X
        mov ecx,[ebp+Altura_04] ; ecx=Altura -> Bucle Y
720     mov esi,[ebp+DirRecorte_04] ; esi -> Imagen Destino
    .RecYResto1:
        push ecx
        mov ecx,eax
    .RecXResto1:
725     movdqu xmm0,[edi] ; Vamos copiando la Imagen Origen en la Imagen
        movdqu [esi],xmm0 ; Destino fila a fila, y para cada fila vamos
```

```
        add edi,16          ; copiando de 4 en 4 pixeles
        add esi,16
        loop .RecXResto1    ; Como el resto es 1 habra que copiar un pixel
        mov edx,[edi]       ; mas cada fila
        mov [esi],edx
        add esi,4
        add edi,ebx
        pop ecx
        loop .RecYResto1
        popad
        pop ebp ; Epilogo
        ret
.RecResto0:
        mov eax,[ebp+Ancho_04] ; eax=Ancho
        mov ebx,eax   ; ebx=Ancho
        mul dword [ebp+Y_04] ; edxeax=Ancho*Y
        add eax,[ebp+X_04]   ; eax=(Ancho*Y)+X
        shl eax,2            ; eax=[(Ancho*Y)+X]*4
        mov edi,eax
        add edi,[ebp+Dir_Origen_04] ; edi -> Apunta al primer dato a recortar
        mov eax,[ebp+Anchura_04] ; eax=Anchura
        sub ebx,eax   ; ebx=Ancho-Anchura
        shl ebx,2     ; ebx=(Ancho-Anchura)*4 -> Salto de linea
        shr eax,2     ; eax=Anchura/4 -> Bucle X
        mov ecx,[ebp+Altura_04] ; ecx=Altura -> Bucle Y
        mov esi,[ebp+DirRecorte_04] ; esi -> Imagen Destino
.RecYResto0:
        push ecx
        mov ecx,eax
.RecXResto0:
        movdqu xmm0,[edi] ; Vamos copiando la Imagen Origen en la Imagen
        movdqu [esi],xmm0 ; Destino fila a fila, y para cada fila vamos
        add edi,16        ; copiando de 4 en 4 pixeles
        add esi,16
        loop .RecXResto0
        add edi,ebx
        pop ecx
        loop .RecYResto0
        popad
        pop ebp ; Epilogo
        ret
.RecMenorDe4Pixeles:
        mov eax,[ebp+Ancho_04] ; eax=Ancho
        mov ebx,eax   ; ebx=Ancho
        mul dword [ebp+Y_04] ; edxeax=Ancho*Y
        add eax,[ebp+X_04]   ; eax=(Ancho*Y)+X
        shl eax,2            ; eax=[(Ancho*Y)+X]*4
        mov edi,eax
        add edi,[ebp+Dir_Origen_04] ; edi -> Apunta al primer dato a recortar
        mov eax,[ebp+Anchura_04] ; eax=Anchura -> Bucle X
        sub ebx,eax   ; ebx=Ancho-Anchura
        shl ebx,2     ; ebx=(Ancho-Anchura)*4 -> Salto de linea
        mov ecx,[ebp+Altura_04] ; ecx=Altura -> Bucle Y
        mov esi,[ebp+DirRecorte_04] ; esi -> Imagen Destino
.RecYMenorDe4Pixeles:
        push ecx
        mov ecx,eax
.RecXMenorDe4Pixeles:  ; Vamos copiando la Imagen Original en la Imagen
        mov edx,[edi]    ; Destino fila a fila, y para cada fila vamos
        mov [esi],edx    ; copiando pixel a pixel
        add esi,4
        add edi,4
        loop .RecXMenorDe4Pixeles
        add edi,ebx
        pop ecx
        loop .RecYMenorDe4Pixeles
        popad
        pop ebp ; Epilogo
        ret

_ASM_Suma:
; Suma un total de Num_Bytes de dos bloques de datos o Imagenes presentes
; en las direcciones Dir_Origen y Dir_Destino
Dir_Destino_05  equ  8 ; Direccion donde quiero la suma
```

```
    Dir_Origen_05    equ 12 ; Direccion donde esta lo que quiero sumar
    Num_Bytes_05     equ 16
          push ebp
          mov ebp,esp ; Prologo
805       pushad
          mov edi,[ebp+Dir_Destino_05] ; edi -> Bloque Destino
          mov esi,[ebp+Dir_Origen_05]  ; esi -> Bloque Origen
          ; Compruebo el numero de bytes que hay que tratar y segun el resto
          ; haremos un tratamiento diferente
810       mov eax,[ebp+Num_Bytes_05]   ; eax=NumBytes
          xor edx,edx
          mov ebx,16  ; ebx=16
          div ebx     ; eax=Cociente(NumBytes/16) edx=Resto
          cmp eax,0
815       jz near .SumaMenorDe16Bytes ; Salta si el NumBytes es menor de 16
          mov ecx,eax ; ecx=NumBytes/16 -> Bucle
          cmp edx,0
          je near .SumaResto0 ; Salta si el resto es 0
          cmp edx,4
820       je near .SumaResto4 ; Salta si el resto es 4
          cmp edx,8
          je near .SumaResto8 ; Salta si el resto es 8
    ; Si no salta a ninguna etiqueta es porque el resto es 12(=edx)
    .SumaResto12:
825       movdqu xmm0,[edi]  ; Vamos sumando de 16 en 16 bytes, que corresponden
          movdqu xmm1,[esi]  ; a cuatro pixeles
          paddusb xmm0,xmm1  ; La suma se realiza de byte en byte con saturacion
          movdqu [edi],xmm0  ; y sin signo
          add edi,16
830       add esi,16
          loop .SumaResto12
          movq mm0,[edi]     ; Como el resto es 12 habra que sumar otros 12
          movq mm1,[esi]     ; bytes que se corresponden con 3 pixeles
          paddusb mm0,mm1
835       movq [edi],mm0
          add edi,8
          add esi,8
          movd mm0,[edi]
          movd mm1,[esi]
840       paddusb mm0,mm1
          movd [edi],mm0
          popad
          pop ebp ; Epilogo
          ret
845 .SumaResto8:
          movdqu xmm0,[edi] ; Vamos sumando byte a byte de 4 en 4 pixeles
          movdqu xmm1,[esi]
          paddusb xmm0,xmm1
          movdqu [edi],xmm0
850       add edi,16
          add esi,16
          loop .SumaResto8
          movq mm0,[edi]    ; Como el resto es 8 habra que sumar 2 pixeles
          movq mm1,[esi]    ; mas
855       paddusb mm0,mm1
          movq [edi],mm0
          popad
          pop ebp ; Epilogo
          ret
860 .SumaResto4:
          movdqu xmm0,[edi] ; Vamos sumando byte a byte de 4 en 4 pixeles
          movdqu xmm1,[esi]
          paddusb xmm0,xmm1
          movdqu [edi],xmm0
865       add edi,16
          add esi,16
          loop .SumaResto4
          movd mm0,[edi]    ; Como el resto es 1 habra que sumar un pixel
          movd mm1,[esi]    ; mas
870       paddusb mm0,mm1
          movd [edi],mm0
          popad
          pop ebp ; Epilogo
          ret
```

```
875 .SumaResto0:
        movdqu xmm0,[edi] ; Vamos sumando byte a byte de 4 en 4 pixeles
        movdqu xmm1,[esi]
        paddusb xmm0,xmm1
        movdqu [edi],xmm0
880     add edi,16
        add esi,16
        loop .SumaResto0
        popad
        pop ebp ; Epilogo
885     ret
    .SumaMenorDe16Bytes:
        mov ecx,edx     ; ecx=Resto
        shr ecx,2       ; ecx=Resto/4 -> El resto solo puede valer 4,8 o 12 bytes
    .SumaRepetir:       ; con lo que los pixeles que hay que sumar byte a byte
890     movd mm0,[edi] ; unicamente pueden ser 1,2 o 3.
        movd mm1,[esi]
        paddusb mm0,mm1
        movd [edi],mm0
        add edi,4
895     add esi,4
        loop .SumaRepetir
        popad
        pop ebp ; Epilogo
        ret
900
    _ASM_Resta:
    ; Resta un total de Num_Bytes de dos bloques de datos o Imagenes presentes
    ; en las direcciones Dir_Origen y Dir_Destino
    Dir_Destino_06  equ  8 ; Direccion donde quiero la resta
905 Dir_Origen_06   equ 12 ; Direccion donde se encuentra lo que quiero restar
    Num_Bytes_06    equ 16
        push ebp
        mov ebp,esp ; Prologo
        pushad
910     mov edi,[ebp+Dir_Destino_06] ; edi -> Bloque Destino
        mov esi,[ebp+Dir_Origen_06]  ; esi -> Bloque Origen
        ; Compruebo el numero de bytes que hay que tratar y segun el resto
        ; haremos un tratamiento diferente
        mov eax,[ebp+Num_Bytes_06]   ; eax=NumBytes
915     xor edx,edx
        mov ebx,16  ; ebx=16
        div ebx     ; eax=Cociente(NumBytes/16) edx=Resto
        cmp eax,0
        jz near .RestaMenorDe16Bytes ; Si el NumBytes es menor de 16
920     mov ecx,eax ; ecx=NumBytes/16 -> Bucle
        cmp edx,0
        je near .RestaResto0 ; Salta si el resto es 0
        cmp edx,4
        je near .RestaResto4 ; Salta si el resto es 4
925     cmp edx,8
        je near .RestaResto8 ; Salta si el resto es 8
    ; Si no ha saltado a ninguna etiqueta es porque el resto es 12(=edx)
    .RestaResto12:
        movdqu xmm0,[edi] ; Vamos restando byte a byte de 16 en 16 bytes
930     movdqu xmm1,[esi] ; que se corresponden con 4 pixeles
        psubusb xmm0,xmm1 ; La resta se realiza con saturacion y sin signo
        movdqu [edi],xmm0
        add edi,16
        add esi,16
935     loop .RestaResto12
        movq mm0,[edi]    ; Como el resto es 12 habra que realizar la resta
        movq mm1,[esi]    ; de 3 pixeles mas
        psubusb mm0,mm1
        movq [edi],mm0
940     add edi,8
        add esi,8
        movd mm0,[edi]
        movd mm1,[esi]
        psubusb mm0,mm1
945     movd [edi],mm0
        popad
        pop ebp ; Epilogo
        ret
```

```
    .RestaResto8:
950         movdqu xmm0,[edi] ; Vamos restando byte a byte de 4 en 4 pixeles
            movdqu xmm1,[esi]
            psubusb xmm0,xmm1
            movdqu [edi],xmm0
            add edi,16
955         add esi,16
            loop .RestaResto8
            movq mm0,[edi]    ; Como el resto es 8 restaremos 2 pixeles mas
            movq mm1,[esi]
            psubusb mm0,mm1
960         movq [edi],mm0
            popad
            pop ebp ; Epilogo
            ret
    .RestaResto4:
965         movdqu xmm0,[edi] ; Vamos restando byte a byte de 4 en 4 pixeles
            movdqu xmm1,[esi]
            psubusb xmm0,xmm1
            movdqu [edi],xmm0
            add edi,16
970         add esi,16
            loop .RestaResto4
            movd mm0,[edi]    ; Como el resto es 4 restaremos un pixel mas
            movd mm1,[esi]
            psubusb mm0,mm1
975         movd [edi],mm0
            popad
            pop ebp ; Epilogo
            ret
    .RestaResto0:
980         movdqu xmm0,[edi] ; Vamos restando byte a byte de 4 en 4 pixeles
            movdqu xmm1,[esi]
            psubusb xmm0,xmm1
            movdqu [edi],xmm0
            add edi,16
985         add esi,16
            loop .RestaResto0
            popad
            pop ebp ; Epilogo
            ret
990 .RestaMenorDe16Bytes:
            mov ecx,edx     ; ecx=Resto
            shr ecx,2       ; ecx=Resto/4 -> El resto solo puede valer 4,8 o 12 bytes
    .RestaRepetir:          ; con lo que los pixeles que hay que sumar byte a byte
            movd mm0,[edi] ; unicamente pueden ser 1,2 o 3.
995         movd mm1,[esi]
            psubusb mm0,mm1
            movd [edi],mm0
            add edi,4
            add esi,4
1000        loop .RestaRepetir
            popad
            pop ebp ; Epilogo
            ret

1005 _ASM_Traslada:
    ; Dada una Imagen o un bloque de datos lo que se hace es crear una nueva
    ; Imagen que contendra a la anterior desplazada a unas nuevas coordenadas
    ; X e Y y se rellenaran estas nuevas posiciones con pixeles nulo, es decir:
    ;                                 ------------------
1010 ;       -----------             Y | 0000000000000000 | <- Cuadro 1
    ;      | HHHHHHHHH |  ------>     <--X-->------------
    ;      | HHHHHHHHH |  ------>     | 0000 | HHHHHHHHH |
    ;       -----------   Cuadro 2 -->| 0000 | HHHHHHHHH |
    ;     Imagen Original               ------------------
1015 ;                                   Imagen Destino
    ; siendo:
    ;        -- H pixeles de la Imagen Original
    ;        -- 0 pixeles nulos de la Imagen Destino
    Dir_Origen_07   equ  8
1020 Ancho_07        equ 12
    Alto_07         equ 16
    X_07            equ 20
```

```
    Y_07            equ 24
    Dir_Traslado_07 equ 28
1025        push ebp
            mov ebp,esp ; Prologo
            pushad
            ; Primeramente se pinta con pixeles nulos, es decir, de valor 0
            ; el Cuadro 1
1030        ; Inicializacion de registros para pintar de negro el Cuadro1
            pxor xmm0,xmm0    ; xmm0 -> Para pintar en negro de 4 en 4 pixeles
            pxor mm0,mm0      ; mm0 -> Para pintar de negro de 2 en 2 pixeles
            mov edi,[ebp+Dir_Traslado_07] ; edi-> Imagen Destino
            mov ebx,[ebp+Y_07]      ; ebx=Y
1035        cmp ebx,0               ; Compruebo si hay que pintar el Cuadro 1
            jz .PintarCuadro2       ; para ello Y>0
            mov eax,[ebp+X_07]      ; eax=X
            add eax,[ebp+Ancho_07] ; eax=X+Ancho
            mul ebx                 ; eax=(X+Ancho)*Y -> Numero de pixeles a pintar
1040        ; Compruebo el tamano del Cuadro 1, para poder pintar de 4 en 4 pixeles
            mov ebx,4     ; ebx=4
            xor edx,edx
            div ebx       ; eax=Cociente(NumPixeles/4) edx=Resto
            cmp eax,0
1045        jz .Cuadro1MenorDe4Pixeles ; Salta si el Cuadro 1 es menor de 4 pixeles
            mov ecx,eax  ; ecx=NumPixeles/4 -> Veces a realizar el bucle
    .PintaCuadro1Negro:
            movdqu [edi],xmm0  ; Vamos pintando el Cuadro 1 de 4 en 4 pixeles
            add edi,16
1050        loop .PintaCuadro1Negro
            cmp edx,0          ; Compruebo si hay resto
            jz .PintarCuadro2
    .Cuadro1MenorDe4Pixeles:
            mov ecx,edx        ; ecx=Resto
1055        xor eax,eax        ; eax=0 -> Para ir pintando en negro los pixeles
    .RepetirPintarPixel:       ;          resultantes
            mov [edi],eax
            add edi,4
            loop .RepetirPintarPixel
1060    ; Ya he pintado de negro el Cuadro1, ahora pinto de negro el Cuadro2
    .PintarCuadro2:
            mov eax,[ebp+X_07] ; eax=X
            shl eax,2       ; eax=X*4
            add edi,eax     ; Dejo apuntando edi al punto donde tengo que
1065        push edi        ; copiar la Imagen Origen
            sub edi,eax
            shr eax,2       ; Restauro eax -> eax=X
            cmp eax,0       ; Compruebo si hay que pintar el Cuadro 2 para
            jz .NoMasNegro ; ello X>0
1070        ; Compruebo el tamano del Cuadro 2, para poder pintar de 4 en 4 pixeles
            mov ebx,4       ; ebx=4
            xor edx,edx
            div ebx         ; eax=Cociente(X/4) edx=Resto
            cmp eax,0       ; Compruebo si el Cuadro 2 es menor de 4 pixeles (X<4)
1075        jz .Cuadro2MenorDe4PixelesDeAncho ; Salta si X<4 pixeles
            mov ecx,[ebp+Alto_07]  ; ecx=Alto
            mov ebx,[ebp+Ancho_07] ; ebx=Ancho
            shl ebx,2     ; ebx=Ancho*4 -> Salto de linea
    ; Iremos pintando linea a linea, hasta pintar todas las lineas
1080    .PintaCuadro2NegroY:
            push ecx
            mov ecx,eax   ; ecx=X/4 -> Veces a realizar el bucle X
    .PintaCuadro2NegroX:
            movdqu [edi],xmm0 ; Vamos pintando de negro de 4 en 4 pixeles
1085        add edi,16        ; linea a linea
            loop .PintaCuadro2NegroX
            cmp edx,0         ; Compruebo si hay resto
            jz .Salir
            mov ecx,edx       ; Pinto de negro el resto de pixeles para cada
1090    .RepetirPintarPixel2:    ; linea
            movd [edi],mm0
            add edi,4
            loop .RepetirPintarPixel2
    .Salir:
1095        add edi,ebx
            pop ecx
```

```
        loop .PintaCuadro2NegroY
        jmp .NoMasNegro
.Cuadro2MenorDe4PixelesDeAncho:
        mov ecx,[ebp+Alto_07]  ; ecx=Alto
        mov ebx,[ebp+Ancho_07] ; ebx=Ancho
        shl ebx,2      ; ebx=Ancho*4 -> Salto de linea
.PintarNegroMenorDe4PixelesDeAnchoEjeY:
        push ecx
        mov ecx,edx        ; ecx=Resto
.PintarNegroMenorDe4PixelesDeAnchoEjeX:
        movd [edi],mm0     ; Vamos pintando de negro de pixel en pixel
        add edi,4          ; fila a fila hasta completar todas las filas
        loop .PintarNegroMenorDe4PixelesDeAnchoEjeX
        add edi,ebx
        pop ecx
        loop .PintarNegroMenorDe4PixelesDeAnchoEjeY
; Ya he pintando los dos cuadros negros, ahora pinto la imagen
.NoMasNegro:
        pop edi ; Recupero edi -> Posicion del primer pixel de la imagen a pintar
        mov esi,[ebp+Dir_Origen_07] ; esi -> Imagen Origen
        mov eax,[ebp+Ancho_07] ; eax=Ancho
        ; Compruebo el Ancho de la Imagen Origen, y segun sea este realizaremos
        ; un tratamiento u otro
        mov ebx,4     ; ebx=4
        xor edx,edx
        div ebx       ; eax=Cociente(Ancho/4) edx=Resto
        mov ecx,[ebp+Alto_07] ; ecx=Alto
        mov ebx,[ebp+X_07]    ; ebx=X
        shl ebx,2             ; ebx=X*4
        cmp eax,0             ; Compruebo si el Ancho es menor de 4 pixeles
        jz .ImagenMenorDe4Pixeles ; Salta si el Ancho es menor de 4 pixeles
; Voy copiando la Imagen Original en la Imagen Destino fila a fila hasta
; completar todas las filas, es decir, el Alto de la Imagen Origen
.PintaImagenEjeY:
        push ecx
        mov ecx,eax ; ecx=X/4 -> Veces a repetir el bucle X
.PintaImagenEjeX:
        movdqu xmm0,[esi] ; Voy copiando de 4 en 4 pixeles
        movdqu [edi],xmm0
        add esi,16
        add edi,16
        loop .PintaImagenEjeX
        cmp edx,0         ; Comprueba si hay Resto
        jz .Salir2        ; Si no hay Resto salta
        mov ecx,edx       ; ecx=Resto
.RepetirPintarPixel3:
        movd mm0,[esi]    ; Copio el Resto pixel a pixel para cada fila
        movd [edi],mm0
        add esi,4
        add edi,4
        loop .RepetirPintarPixel3
.Salir2:
        add edi,ebx
        pop ecx
        loop .PintaImagenEjeY
        jmp .SalirTotalmente
; Si la Imagen Origen tiene un Ancho menor de 4 pixeles, iremos copiando
; fila a fila de la Imagen Original a la Imagen Destino y pixel a pixel,
; hasta completar todas las filas
.ImagenMenorDe4Pixeles:
        push ecx
        mov ecx,edx  ; ecx=Resto
.RepetirPintarPixel4:
        movd mm0,[esi] ; Voy copiando pixel a pixel para cada fila
        movd [edi],mm0
        add esi,4
        add edi,4
        loop .RepetirPintarPixel4
        add edi,ebx
        pop ecx
        loop .ImagenMenorDe4Pixeles
.SalirTotalmente:
        popad
        pop ebp ; Epilogo
```

```
        ret

_ASM_Rotacion:
; Dada una Imagen Origen o un bloque de datos que nos llega en Dir_Origen,
; se obtendra una nueva Imagen Destino o bloques de datos que contendra la
; Imagen Original rotada un cierto angulo, del que conocemos su Seno
; y su Coseno
Dir_Origen_08      equ  8
Alto_08            equ 12
Ancho_08           equ 16
AltoRot_08         equ 20
AnchoRot_08        equ 24
Dir_Rotacion_08    equ 28
DesplazamientoX_08 equ 32
DesplazamientoY_08 equ 40
Seno_08            equ 48
Coseno_08          equ 56
        push ebp
        mov ebp,esp ; Prologo
        pushad
        ; Inicializacion de registros XMM para calculos posteriores
        movsd xmm0,[ebp+Seno_08]  ; xmm0=*|sen (pfdp)
        pxor xmm1,xmm1            ; xmm1=0|0|0|0
        cvtsd2ss xmm1,xmm0        ; xmm1=0|0|0|sen (pfsp)
        movsd xmm0,[ebp+Coseno_08]; xmm0=*|cos (pfdp)
        pxor mm2,mm2              ; xmm2=0|0|0|0
        cvtsd2ss xmm2,xmm0        ; xmm2=0|0|0|cos (pfsp)
        pslldq xmm2,4             ; xmm2=0|0|cos|0 (pfsp)
        orps xmm1,xmm2            ; xmm1=0|0|cos|sen (pfsp)
        pshufd xmm7,xmm1,44h      ; xmm7=cos|sen|cos|sen (pfsp)
        movsd xmm0,[ebp+DesplazamientoX_08] ; xmm0=*|Dx (pfdp)
        cvtsd2ss xmm1,xmm0        ; xmm1=*|*|*|Dx (pfsp)
        pxor xmm0,xmm0            ; xmm0=0|0|0|0
        movss xmm0,xmm1           ; xmm0=0|0|0|Dx (pfsp)
        movups xmm1,xmm0          ; xmm1=0|0|0|Dx (pfsp)
        pslldq xmm1,12            ; xmm1=Dx|0|0|0 (pfsp)
        subps xmm0,xmm1           ; xmm0=-Dx|0|0|Dx (pfsp)
        movups xmm1,xmm0          ; xmm1=-Dx|0|0|Dx (pfsp)
        movsd xmm2,[ebp+DesplazamientoY_08] ; xmm2=*|Dy (pfdp)
        pxor xmm3,xmm3            ; xmm3=0|0|0|0
        cvtsd2ss xmm2,xmm2        ; xmm2=*|*|*|Dy (pfsp)
        subss xmm3,xmm2           ; xmm3=0|0|0|-Dy (pfsp)
        pslldq xmm3,4             ; xmm3=0|0|-Dy|0 (pfsp)
        por xmm1,xmm3             ; xmm1=-Dx|0|-Dy|Dx (pfsp)
        pslldq xmm3,4             ; xmm3=0|-Dy|0|0 (pfsp)
        por xmm1,xmm3             ; xmm1=-Dx|-Dy|-Dy|Dx (pfsp)
        movdqu xmm4,[MASC_AND_ROTACION] ; xmm4=0|F|F|0
        cvtdq2ps xmm5,[MASC_ROTACION_Y] ; xmm5=0|1|1|0 (pfsp)
        cvtdq2ps xmm6,[MASC_ROTACION_X] ; xmm6=1|0|0|-1 (pfsp)
        ; Inicializacion registros MMX para posteriores calculos
        movd mm5,[ebp+Dir_Origen_08] ; mm5=0|Dir
        mov eax,[ebp+Ancho_08]  ; eax=Ancho
        dec eax                 ; eax=Ancho-1
        movd mm7,eax            ; mm7=0|(Ancho-1)
        psllq mm7,32             ; mm7=(Ancho-1)|0
        inc eax                 ; eax=Ancho
        movd mm3,eax            ; mm3=0|Ancho
        mov eax,1               ; eax=1
        shl eax,16              ; eax=1|0
        movd mm6,eax            ; mm6=0||1|0
        psubsw mm6,mm3          ; mm6=0||1|-Ancho
        mov ebx,[ebp+Alto_08]   ; ebx=Alto
        dec ebx                 ; ebx=Alto-1
        movd mm4,ebx            ; mm4=0|(Alto-1)
        por mm7,mm4             ; mm7=(Ancho-1)|(Alto-1)
        ; Inicializacion del bloque de datos a recorrer
        mov eax,[ebp+AnchoRot_08] ; eax=AnchoRot
        mov ebx,eax               ; ebx=AnchoRot -> Bucle X
        mov ecx,[ebp+AltoRot_08]  ; ecx=AltoRot
        dec ecx                   ; ecx=AltoRot-1
        mul ecx                   ; eax=AnchoRot*(AltoRot-1)
        inc ecx                   ; ecx=AltoRot -> Bucle Y
        shl eax,2                 ; eax=AnchoRot*(AltoRot-1)*4
        mov esi,eax               ; esi-> bloque de datos origen o Imagen Origen
```

```
1245          add esi,[ebp+Dir_Rotacion_08]
              mov eax,ebx                ; eax=AnchoRot
              shl eax,3                  ; eax=AnchoRot*8->Salto de linea
      ; Para realizar la rotacion lo que iremos haciendo es recorrer la Imagen
      ; Destino, de izquierda a derecha y de abajo a arriba, partiendo de la
1250  ; coordenada (0,0) e incrementando dichas coordenadas durante la realizacion
      ; del recorrido, para obtener el valor de las coordenadas que le corresponden
      ; en la Imagen Origen. Las ecuaciones utilizadas para la obtencion de dichas
      ; coordenadas son:
      ;       -- x=(x'-Dx)*cos(a)+(y'-Dy)*sen(a)
1255  ;       -- y=(y'-Dy)*cos(a)+(Dx-x')*sen(a) o y=(y'-Dy)*cos(a)-(x'-Dx)*sen(a)
      ; siendo:
      ;       -- (x,y) las coordenadas de la Imagen Origen
      ;       -- (x',y') las coordenadas de la Imagen Destino
      ;       -- a el angulo de rotacion
1260  ;       -- Dx y Dy los desplazamientos que son pasados a la rutina
      ;          y que tienen que ver con lo que se desplaza la Imagen Origen
      ;          a la hora de hacer la rotacion en cuanto al eje X y al eje Y
      .RotEjeY:
              push ecx
1265          mov ecx,ebx
      .RotEjeX:
              movups xmm2,xmm7   ; xmm2=cos|sen|cos|sen (pfsp)
              mulps xmm2,xmm1    ; xmm2=(x'-Dx)cos|(y-Dy)'sen|(y'-Dy)cos|(Dx-x')sen (pfsp)
              haddps xmm2,xmm1   ; xmm2=*|*|x|y (pfsp)
1270          cvtps2pi mm0,xmm2  ; mm0=x|y (enteros 32 bits)
              pxor mm3,mm3       ; Compruebo que las coordenadas obtenidas pertenecen
              pcmpgtd mm3,mm0    ; a alguna coordenada de la Imagen Origen, es decir:
              movq mm1,mm0       ;                0<=X<=(Ancho-1)
              pcmpgtd mm1,mm7    ;                0<=Y<=(Alto-1)
1275          por mm1,mm3
              movq mm2,mm1
              psrlq mm2,32
              por mm2,mm1
              movd edx,mm2       ; Si mm2 o edx es igual a todo unos la coordenada
1280          cmp edx,0          ; obtenida no pertenece a la Imagen Origen con lo
              neg edx            ; que pintare ese pixel en negro
              jnz .RotacionPixelNulo
              ; La coordenada es valida calculamos el valor del pixel para esa
              ; coordenada en la Imagen Origen y le copiamos en la Imagen Destino
1285          packssdw mm0,mm0   ; mm0=*|xy (enteros de 32 bits)
              psubsw mm0,mm4     ; mm0=*|x (y-Alto)
              pmaddwd mm0,mm6    ; mm0=*|x+(Alto-y)*Ancho
              pslld mm0,2        ; mm0=*|[x+(Alto-y)*Ancho]*4
              paddd mm0,mm5      ; mm0=*|[[x+(Alto-y)*Ancho]*4]+Dir
1290          movd edi,mm0       ; edi=Dir_Pixel
              mov edx,[edi]      ; edx=[Dir_Pixel]
      .RotacionPixelNulo:
              mov [esi],edx       ; [esi]=[Dir_Pixel]
              add esi,4
1295          addps xmm1,xmm6     ; xmm1 -> Incremento x
              loop .RotEjeX
              andps xmm1,xmm4     ; xmm1 -> Deja solo pasar la componente y
              addps xmm1,xmm5     ; xmm1 -> Incremento y
              addps xmm1,xmm0     ; xmm1 -> Inicializacion de la componente X
1300          sub esi,eax
              pop ecx
              loop .RotEjeY
              popad
              pop ebp ; Epilogo
1305          ret

      _ASM_XOR:
      ; Realiza la funcion logica XOR para un total de Num_Bytes de dos bloques
      ; de datos o Imagenes presentes en las direcciones Dir y DirXOR
1310  Dir_Destino_09 equ  8 ; Direccion donde quiero la funcion logica XOR
      Dir_Origen_09  equ 12
      Num_Bytes_09   equ 16
              push ebp
              mov ebp,esp ; Prologo
1315          pushad
              mov edi,[ebp+Dir_Destino_09]    ; edi -> Imagen Destino
              mov esi,[ebp+Dir_Origen_09] ; esi -> Imagen Origen
              ; Vemos si el Numero de Bytes (NumBytes) a tratar es multiplo de
```

```
            ; 16, y segunsea este realizaremos un tratamiento u otro
1320        xor edx,edx
            mov eax,[ebp+Num_Bytes_09] ; eax=NumBytes
            mov ebx,16   ; ebx=16
            div ebx      ; eax=Cociente(NumBytes/16) edx=Resto
            mov ecx,edx  ; ecx=Resto
1325        shr ecx,2    ; ecx=Resto/4
            cmp eax,0    ; Comprueba si el numero de bytes a tratar es menor de 16
            jz near .XORMenorDe16Bytes ; Salta si el numero de Bytes es menor
            mov ecx,eax ; ecx=NumBytes/16
            cmp edx,0
1330        je near .XORResto0 ; Salta si el Resto es 0
            cmp edx,4
            je .XORResto4 ; Salta si el Resto es 4
            cmp edx,8
            je .XORResto8 ; Salta si el Resto es 8
1335    ; Si no salta a ninguna etiqueta es por que el resto es 12(=edx)
    .XORResto12:
            movdqu xmm0,[edi] ; Vamos realizando la operacion logica XOR
            movdqu xmm1,[esi] ; entre ambos bloques de datos de 16 en 16 bytes,
            pxor xmm0,xmm1    ; o lo que es lo mismo de 4 en 4 pixeles
1340        movdqu [edi],xmm0
            add edi,16
            add esi,16
            loop .XORResto12
            movq mm0,[edi]    ; Como el Resto es 12 habra que realizar la funcion
1345        pxor mm0,[esi]    ; logica XOR para 3 pixeles mas
            movq [edi],mm0
            add edi,8
            add esi,8
            mov edx,[edi]
1350        xor edx,[esi]
            mov [edi],edx
            popad
            pop ebp ; Epilogo
            ret
1355    .XORResto8:
            movdqu xmm0,[edi] ; Vamos realizando la operacion logica XOR
            movdqu xmm1,[esi] ; entre ambos bloques de datos de 16 en 16 bytes,
            pxor xmm0,xmm1    ; o lo que es lo mismo de 4 en 4 pixeles
            movdqu [edi],xmm0
1360        add edi,16
            add esi,16
            loop .XORResto8
            movq mm0,[edi]    ; Como el Resto es 8 habra que realizar la funcion
            pxor mm0,[esi]    ; logica XOR para 2 pixeles mas
1365        movq [edi],mm0
            popad
            pop ebp ; Epilogo
            ret
    .XORResto4:
1370        movdqu xmm0,[edi] ; Vamos realizando la operacion logica XOR
            movdqu xmm1,[esi] ; entre ambos bloques de datos de 16 en 16 bytes,
            pxor xmm0,xmm1    ; o lo que es lo mismo de 4 en 4 pixeles
            movdqu [edi],xmm0
            add edi,16
1375        add esi,16
            loop .XORResto4   ; Como el Resto es 4 habra que realizar la funcion
            mov edx,[edi]     ; logica XOR para un pixel mas
            xor edx,[esi]
            mov [edi],edx
1380        popad
            pop ebp ; Epilogo
            ret
    .XORResto0:
            movdqu xmm0,[edi] ; Vamos realizando la operacion logica XOR
1385        movdqu xmm1,[esi] ; entre ambos bloques de datos de 16 en 16 bytes,
            pxor xmm0,xmm1    ; o lo que es lo mismo de 4 en 4 pixeles
            movdqu [edi],xmm0
            add edi,16
            add esi,16
1390        loop .XORResto0
            popad
            pop ebp ; Epilogo
```

```
        ret
.XORMenorDe16Bytes:
        mov eax,[edi]     ; Vamos realizando la operacion logica XOR
        xor eax,[esi]     ; entre ambos bloques de datos de 4 en 4 bytes,
        add edi,4         ; o lo que es lo mismo de pixel en pixel
        add esi,4
        loop .XORMenorDe16Bytes
        popad
        pop ebp ; Epilogo
        ret

_ASM_Magnificacion_Entera:
; Dada una Imagen Origen lo que se hace es crear otra Imagen Destino de
; tal forma que Imagen_Destino=Imagen_Orien*Factor siendo este Factor
; un numero entero y mayor de 1, es decir, lo que hacemos es hacer Factor
; veces mas grande la Imagen Origen
Dir_Orig_10   equ  8
Dir_Dest_10   equ 12
Ancho_Orig_10 equ 16
Alto_Orig_10  equ 20
Factor_10     equ 24
        push ebp
        mov ebp,esp ; Prologo
        pushad
        mov esi,[ebp+Dir_Orig_10]  ; esi -> Imagen Origen
        mov edi,[ebp+Dir_Dest_10]  ; edi -> Imagen Destino
        mov eax,[ebp+Ancho_Orig_10]; eax=Ancho
        mov ebx,eax ; ebx=Ancho -> Bucle X
        shl eax,2   ; eax=Ancho*4 -> Salto de linea
        mov ecx,[ebp+Alto_Orig_10] ; ecx=Alto -> Bucle Y
        mov edx,[ebp+Factor_10] ; edx=Factor -> Para el bucle
; Para obtener la Imagen Destino vamos creandola fila a fila, teniendo en
; cuenta que cada pixel de la Imagen Origen se repite Factor veces en la
; Imagen Destino, y que cada fila de la Imagen Origen se repite Factor veces
; en la Imagen Destino
.MagEntEjeY:       ; Vamos recorriendo la Imagen Origen fila a fila
        push ecx
        push esi
        mov ecx,edx ; ecx=Factor -> Control del bucle
.RepeticionFila:   ; Cada fila se repetira Factor veces
        push ecx
        mov ecx,ebx ; ecx=Ancho -> Control del bucle
.MagEntEjeX:       ; Vamos recorriendo una fila de la Imagen Origen
        push ecx
        mov ecx,edx    ; ecx=Factor -> Control del bucle
        movd mm0,[esi]
.RepeticionPixel:     ; Cada pixel se repite Factor veces en la Imagen Destino
        movd [edi],mm0
        add edi,4
        loop .RepeticionPixel
        add esi,4
        pop ecx
        loop .MagEntEjeX
        sub esi,eax
        pop ecx
        loop .RepeticionFila
        pop esi
        add esi,eax
        pop ecx
        loop .MagEntEjeY
        popad
        pop ebp ; Epilogo
        ret

_ASM_Magnificacion_No_Entera:
; Dada una Imagen Origen lo que se hace es crear otra Imagen Destino de
; tal forma que Imagen_Destino=Imagen_Orien*Factor siendo este Factor
; un numero no entero y mayor de 1, es decir, lo que hacemos es hacer Factor
; veces mas grande la Imagen Origen
Dir_Orig_11   equ  8
Dir_Dest_11   equ 12
Ancho_Orig_11 equ 16
Ancho_Dest_11 equ 20
Alto_Dest_11  equ 24
```

```
Factor_11     equ 28
        push ebp
        mov ebp,esp ; Prologo
        pushad
        ; Inicializacion de registros para su posterior utilizacion
        movsd xmm0,[ebp+Factor_11]    ; xmm0=*|F (pfdp)
        cvtsd2ss xmm0,xmm0            ; xmm0=*|*|*|F (pfsp)
        pxor xmm7,xmm7                ; xmm7=0|0|0|0
        rcpss xmm7,xmm0               ; xmm7=0|0|0|1/F (pfsp)
        cvtdq2ps xmm6,[MASC_MAGNIFICACION] ; xmm6=0|0|0|1 (pfsp)
        pxor xmm5,xmm5                ; xmm5=0|0|0|0 (entremezclado)
        pxor xmm0,xmm0                ; xmm0=0|0|0|0 (coord actual)
        movq mm7,[MASC_MAGNIFICACION_INC_X] ; mm7=0|1 (enteros) -> INC X
        movq mm6,[MASC_MAGNIFICACION_INC_Y] ; mm6=1|0 (enteros) -> INC Y
        movd mm5,[ebp+Ancho_Dest_11] ; mm5=0|AnchoDest -> bucle x
        movq mm3,[MASC_MAGNIFICACION_AND] ; mm3=F|0
        pxor mm1,mm1                  ; mm1=0|0 (coordenadas)
        mov eax,[ebp+Ancho_Orig_11]   ; eax=AnchoOrig
        shl eax,2                     ; eax=AnchoOrig*4
        mov ecx,[ebp+Alto_Dest_11]    ; ecx=AltoDest -> Bucle y
        xor edx,edx                   ; edx=0 -> INC esi
        mov esi,[ebp+Dir_Orig_11]     ; esi -> Imagen Origen
        mov edi,[ebp+Dir_Dest_11]     ; edi -> Imagen Destino
; Se va creando la Imagen Destino fila a fila a partir de la Imagen Origen,
; sabemos que cada pixel de la Imagen Origen ocupa Factor pixeles de la
; Imagen Destino y esto es lo que se va haciendo. Se van llevando las
; coordenadas de las Imagendes Origen y Destino por separado y cada pixel
; se define por 4 estados o distancias:
;                          --- ---
;                      YB |   |   |
;                          --- ---
;                      YA |   |   |
;                          --- ---
;                           XA  XB
; Siendo cada una de estas distancias las diferencias de las coordenadas
; de la Imagen Origen y de la Imagen Destino, cada distancia servira de
; contribuccion al pixel que estemos calculando. Estas distancias se van
; llevando en el registro xmm4, de tal forma que presenta la siguiente
; forma: xmm4= XB|XA|YB|YA
.MagNoEntEjeY:
        push ecx
        push esi
        movd ebx,mm5 ; ebx=AnchoDest
        cvtps2pi mm0,xmm0 ; mm0 (coordenadas actuales)
        movq mm4,mm0      ; mm4=mm0 (coordenadas actuales)
        pcmpeqd mm0,mm1   ; Se comprueba si la coordenada Y actual es la misma
        psrlq mm0,32      ; que en el estado anterior
        movq mm2,mm0
        pxor xmm4,xmm4    ; Si es igula la Y entonces la distancia YA sera 1
        movd ecx,mm0      ; en caso contrario calculamos esa YA y YB a traves
        cmp ecx,0         ; de las coordenadas
        jz .DistintaY
.MismaY:
        movups xmm4,xmm6  ; xmm4=0|0|0|1 (pfsp)
        jmp .MagNoEntEjeX
.DistintaY:
        cvtpi2ps xmm1,mm4 ; xmm1=*|*|y|x (pfsp)
        subps xmm1,xmm0   ; xmm1=*|*|YA|* (pfsp)
        pslldq xmm1,8     ; xmm1=YA|*|0|0 (pfsp)
        psrldq xmm1,12    ; xmm1=0|0|0|YA (pfsp)
        movups xmm2,xmm6  ; xmm2=0|0|0|1 (pfsp)
        subss xmm2,xmm1   ; xmm2=0|0|0|YB (pfsp) YB=1-YA
        pslldq xmm2,4     ; xmm2=0|0|YB|0 (pfsp)
        orps xmm2,xmm1    ; xmm2=0|0|YB|YA (pfsp)
        movups xmm4,xmm2  ; xmm4=0|0|YB|YA (pfsp)
        paddd mm1,mm6     ; INC Y
.MagNoEntEjeX:            ; Realizamos lo mismo para la coordenada X, es decir,
        cvtps2pi mm0,xmm0 ; si la coordenada actual X no ha cambiado XA=1 (XB=0),
        movq mm4,mm0      ; en caso contrario calculariamos las distancias
        pcmpeqd mm0,mm1   ; XA y XB
        movd ecx,mm0
        cmp ecx,0
        jz .DistintaX
.MismaX:
```

```
            movups xmm1,xmm6   ; xmm1=0|0|0|1 (pfsp)
            pslldq xmm1,8      ; xmm1=0|1|0|0 (pfsp)
            orps xmm4,xmm1     ; xmm4=XB|XA|YB|YA (pfsp)
            jmp .CalculoPixel
.DistintaX:
            cvtpi2ps xmm1,mm4  ; xmm1=*|*|y|x (pfsp)
            subps xmm1,xmm0    ; xmm1=*|*|*|XA (pfsp)
            movups xmm2,xmm6   ; xmm2=0|0|0|1 (pfsp)
            subss xmm2,xmm1    ; xmm2=0|0|0|XB (pfsp) XB=1-XA
            pslldq xmm1,12     ; xmm1=XA|0|0|0 (pfsp)
            psrldq xmm1,4      ; xmm1=0|XA|0|0 (pfsp)
            pslldq xmm2,12     ; xmm2=XB|0|0|0 (pfsp)
            orps xmm4,xmm1     ; xmm4=0|XA|YB|YA (pfsp)
            orps xmm4,xmm2     ; xmm4=XB|XA|YB|YA (pfsp)
            paddd mm1,mm7      ; INC X
            mov edx,4          ; edx=4 Indica que hay que incrementar esi
    ; Una vez calculadas las distancia XA, XB, YA e YB, lo que vamos calculando
    ; es el pixel, a traves de las contribuciones de area que producen dichas
    ; distancias calculadas
.CalculoPixel:
            movd xmm3,[esi]        ; xmm3=*|*|*|*RGB (enteros)
            punpcklbw xmm3,xmm5    ; xmm3=*|*|0*0R0G0B (enteros)
            punpcklwd xmm3,xmm5    ; xmm3=*|R|G|B (enteros 32 bits)
            cvtdq2ps xmm3,xmm3     ; xmm3=*|R|G|B (pfsp)
            pshufd xmm2,xmm4,0h    ; xmm2=YA|YA|YA|YA (pfsp)
            mulps xmm3,xmm2        ; xmm3=Pixel*YA (pfsp)
            pshufd xmm2,xmm4,0AAh; xmm2=XA|XA|XA|XA (pfsp)
            mulps xmm3,xmm2        ; xmm3=Pixel*YA*XA (pfsp)
            movups xmm1,xmm3       ; xmm1 -> Llevara la suma del valor del pixel
            add esi,eax
            movd xmm3,[esi]        ; xmm3=*|*|*|*RGB (enteros)
            punpcklbw xmm3,xmm5    ; xmm3=*|*|0*0R0G0B (enteros)
            punpcklwd xmm3,xmm5    ; xmm3=*|R|G|B (enteros 32 bits)
            cvtdq2ps xmm3,xmm3     ; xmm3=*|R|G|B (pfsp)
            mulps xmm3,xmm2        ; xmm3=Pixel*XA (pfsp)
            pshufd xmm2,xmm4,55h ; xmm2=YB|YB|YB|YB (pfsp)
            mulps xmm3,xmm2        ; xmm3=Pixel*XA*YB (pfsp)
            addps xmm1,xmm3        ; xmm1 -> xmm1+xmm3
            add esi,4
            movd xmm3,[esi]        ; xmm3=*|*|*|*RGB (enteros)
            punpcklbw xmm3,xmm5    ; xmm3=*|*|0*0R0G0B (enteros)
            punpcklwd xmm3,xmm5    ; xmm3=*|R|G|B (enteros 32 bits)
            cvtdq2ps xmm3,xmm3     ; xmm3=*|R|G|B (pfsp)
            mulps xmm3,xmm2        ; xmm3=Pixel*YB (pfsp)
            pshufd xmm2,xmm4,0FFh; xmm2=XB|XB|XB|XB (pfsp)
            mulps xmm3,xmm2        ; xmm3=Pixel*YB*XB (pfsp)
            addps xmm1,xmm3        ; xmm1 -> xmm1+xmm3
            sub esi,eax
            movd xmm3,[esi]        ; xmm3=*|*|*|*RGB (enteros)
            punpcklbw xmm3,xmm5    ; xmm3=*|*|0*0R0G0B (enteros)
            punpcklwd xmm3,xmm5    ; xmm3=*|R|G|B (enteros 32 bits)
            cvtdq2ps xmm3,xmm3     ; xmm3=*|R|G|B (pfsp)
            mulps xmm3,xmm2        ; xmm3=Pixel*XB (pfsp)
            pshufd xmm2,xmm4,0h    ; xmm2=YA|YA|YA|YA (pfsp)
            mulps xmm3,xmm2        ; xmm3=Pixel*XB*YA (pfsp)
            addps xmm1,xmm3        ; xmm1 -> xmm1+xmm3
            sub esi,4
            cvtps2dq xmm1,xmm1     ; xmm1=*|R|G|B (enteros 32 bits)
            packssdw xmm1,xmm5     ; xmm1=*|*|0*0R0G0B
            packuswb xmm1,xmm5     ; xmm1=*|*|*|*RGB
            movd [edi],xmm1        ; [edi] <- xmm1
            add edi,4              ; Actualizacion de punteros
            add esi,edx
            xor edx,edx
            andps xmm4,[MASC_MAGNIFICACION_XMM4]
            addps xmm0,xmm7      ; Incremento X
            dec ebx
            cmp ebx,0
            jnz .MagNoEntEjeX
            ; Si la coordenada Y eran iguales hay que decrementar esi en una linea
            pop esi
            movd ecx,mm2
            cmp ecx,0
            jnz .CoordYNoCambia
```

```
1615        add esi,eax
    .CoordYNoCambia:
            andps xmm0,[MASC_MAGNIFICACION_AND_XMM] ; Solo deja pasar la Y
            pslldq xmm7,4    ; xmm7=0|0|1/F|0 (pfsp)
            addps xmm0,xmm7 ; Incremento Y
1620        psrldq xmm7,4    ; xmm7=0|0|0|1/F (pfsp)
            pand mm1,mm3     ; Pone la coordenada X a 0
            pop ecx
            dec ecx
            cmp ecx,0
1625        jnz .MagNoEntEjeY
            popad
            pop ebp ; Epilogo
            ret

1630 _ASM_Minimizacion_Entera:
    ; Dada una Imagen Origen lo que se hace es crear otra Imagen Destino de
    ; tal forma que Imagen_Destino=Imagen_Orien*Factor siendo este Factor
    ; un numero entero y menor de 1, es decir, lo que hacemos es hacer Factor
    ; veces mas pequena la Imagen Origen
1635 Dir_Orig_12   equ  8
    Dir_Dest_12   equ 12
    Ancho_Orig_12 equ 16
    Ancho_Dest_12 equ 20
    Alto_Dest_12  equ 24
1640 Factor_12     equ 28
            push ebp
            mov ebp,esp ; Prologo
            pushad
            ;Inicializacion de registros para posteriores calculos
1645        movd mm0,[ebp+Factor_12] ; mm0=0|F (enteros de 32 bits)
            cvtpi2ps xmm6,mm0        ; xmm6=*|*|0|F (pfsp)
            pshufd xmm7,xmm6,0h      ; xmm7=F|F|F|F (pfsp)
            mulps xmm7,xmm7          ; xmm7=F2|F2|F2|F2 (pfsp) (F2=F*F)
            pxor xmm5,xmm5           ; xmm5 -> Para entremezclado
1650        movd mm1,[ebp+Ancho_Dest_12]; mm1=0|AnchoDest (enteros) -> Bucle x
            mov eax,[ebp+Ancho_Orig_12] ; eax=AnchoOrig
            mov ecx,eax          ; ecx=AnchoOrig
            movd ebx,mm0         ; ebx=Factor
            mul ebx              ; eax=AnchoOrig*Factor
1655        mov edx,ecx          ; edx=AnchoOrig
            sub edx,ebx          ; edx=AnchoOrig-Factor
            shl edx,2            ; edx=(AnchoOrig-Factor)*4 -> Salto de linea
                                 ; para cuando cambiamos de linea en la Imagen
                                 ; Origen
1660        shl ebx,2            ; ebx=Factor*4 -> Salto para cuando cambiamos
                                 ; de pixel en una misma linea de la Imagen
                                 ; Destino
            shl eax,2            ; eax=AnchoOrig*(Factor)*4 -> Salto de linea
                                 ; para cuando cambianos de linea de pixeles en
1665                             ; la Imagen Destino
            mov ecx,[ebp+Alto_Dest_12] ; ecx=AltoDest -> Bucle Y
            mov esi,[ebp+Dir_Orig_12]  ; esi -> Imagen Origen
            mov edi,[ebp+Dir_Dest_12]  ; edi -> Imagen Destino
    ; Para obtener la Imagen Destino vamos creandola fila a fila, teniendo en
1670 ; cuenta que cada pixel de la Imagen Destino esta formado por Factor*Factor
    ; pixeles de la Imagen Origen
    .MiniEntEjeY:
            push ecx
            push esi
1675        movd ecx,mm1  ; ecx=AnchoDest -> Control de bucle
    .MiniEntEjeX:
            push ecx
            push esi
            movd ecx,mm0   ; ecx=Factor -> Control de bucle
1680        pxor xmm1,xmm1 ; xmm1 -> Llevara la suma del pixel
    .MiniEntColumna:
            push ecx
            movd ecx,mm0  ; ecx=Factor -> Control de bucle
    .MiniEntFila:
1685        movd xmm0,[esi]     ; xmm0=*|*|*|*RGB (enteros)
            punpcklbw xmm0,xmm5 ; xmm0=*|*|0*0R0G0B (enteros)
            paddusw xmm1,xmm0   ; xmm1 -> Va llevando la suma
            add esi,4
```

```
        loop .MiniEntFila
        add esi,edx
        pop ecx
        loop .MiniEntColumna
        punpcklwd xmm1,xmm5 ; xmm1=*|SumaR|SumaG|SumaB (enteros 32 bits)
        cvtdq2ps xmm2,xmm1  ; xmm2=*|SumaR|SumaG|SumaB (pfsp)
        divps xmm2,xmm7     ; xmm2=*|SumaR/A|SumaG/A|SumaB/A siendo A el
                            ; area del pixel y de valor A=F*F (F=Factor)
        cvtps2dq xmm1,xmm2  ; xmm1=*|R|G|B (enteros de 32 bits)
        packssdw xmm1,xmm5  ; xmm1=*|*|0*0R0G0B
        packuswb xmm1,xmm5  ; xmm1=*|*|*|*RGB
        movd [edi],xmm1     ; [edi] <- xmm1
        add edi,4
        pop esi
        add esi,ebx
        pop ecx
        loop .MiniEntEjeX
        pop esi
        add esi,eax
        pop ecx
        loop .MiniEntEjeY
        popad
        pop ebp ; Epilogo
        ret

_ASM_Minimizacion_No_Entera:
; Dada una Imagen Origen lo que se hace es crear otra Imagen Destino de
; tal forma que Imagen_Destino=Imagen_Orien*Factor siendo este Factor
; un numero no entero y menor de 1, es decir, lo que hacemos es hacer Factor
; veces mas pequena la Imagen Origen
Dir_Orig_13              equ  8
Dir_Dest_13              equ 12
Ancho_Orig_13            equ 16
Ancho_Dest_13            equ 20
Alto_Dest_13             equ 24
Factor_Inverso_Entero_13 equ 28
Factor_13                equ 32
        push ebp
        mov ebp,esp ; Prologo
        pushad
        ; Inicializacion de registros para posteriores calculos
        xor eax,eax ; eax=0
        ; Se inicializan las variables HAY_INCREMENTO_EJEX e INCREMENTO_EJEY
        ; que nos indicaran una situacion especial cuando se va construyendo
        ; la Imagen Destino a partir de la Imagen Origen
        mov dword [HAY_INCREMENTO_EJEX],eax
        mov dword [HAY_INCREMENTO_EJEY],eax
        movsd xmm0,[ebp+Factor_13]  ; xmm0=*|F (pfdp)
        pxor xmm7,xmm7              ; xmm7=0|0|0|0
        cvtsd2ss xmm7,xmm0          ; xmm7=0|0|0|F (pfsp)
        mov eax,[ebp+Factor_Inverso_Entero_13] ; eax=K
        pxor xmm6,xmm6              ; xmm6=0|0|0|0
        cvtsi2ss xmm6,eax           ; xmm6=0|0|0|K (pfsp)
        mulss xmm6,xmm7             ; xmm6=0|0|0|K*F (pfsp)
        pxor xmm5,xmm5              ; xmm5 -> Entremezclado
        pxor xmm0,xmm0              ; xmm0=0|0|Y|X (Coordenadas)
        movq mm7,[MASC_MINI_INC_X]  ; mm7=0|1 (enteros)
        movq mm6,[MASC_MINI_AND]    ; mm6=F|0 (Solo pasa componente y)
        movq mm5,[MASC_MINI_INC_Y]  ; mm5=1|0 (enteros)
        movd mm4,[ebp+Ancho_Dest_13]; mm4=0|AnchoDest
        movd mm3,eax                ; mm3=0|K
        psubusw mm3,mm7             ; mm3=0|(K-1)
        mov ebx,[ebp+Ancho_Orig_13] ; ebx=AnchoOrig
        movd mm1,ebx                ; mm1=0|AnchoOrig
        pxor mm0,mm0                ; mm0=0|0 (Coordenadas)
        sub ebx,eax                 ; ebx=AnchoOrig-K
        movd mm2,ebx                ; mm2=0|(AnchoOrig-K)
        shl eax,2                   ; eax=K*2
        mov esi,[ebp+Dir_Orig_13]   ; esi -> Imagen Origen
        mov edi,[ebp+Dir_Dest_13]   ; edi -> Imagen Destino
        mov ecx,[ebp+Alto_Dest_13]  ; ecx=Alto -> Bucle Y
        xor ebx,ebx                 ; Inicializacion
        xor edx,edx
; Se va creando la Imagen Destino fila a fila a partir de la Imagen Origen,
```

```
    ; sabemos que cada pixel de la Imagen Destino esta formada por Factor pixeles
    ; de la Imagen Origen y esto es lo que se va haciendo. Se van llevando las
1765 ; coordenadas de las Imagendes Origen y Destino por separado y cada pixel
    ; se define por 4 estados o distancias:
    ;                          --- ---
    ;                    Y1 |   |   |
    ;                        --- ---
1770 ;                    Y0 |   |   |
    ;                        --- ---
    ;                         X0  X1
    ; Siendo cada una de estas distancias las diferencias de las coordenadas
    ; de la Imagen Origen y de la Imagen Destino, cada distancia servira de
1775 ; contribuccion al pixel que estemos calculando. Estas distancias se van
    ; llevando en el registro xmm4, de tal forma que presenta la siguiente
    ; forma: xmm4= X1|X0|Y1|Y0
    ; El calculo de las distancias presenta el siguiente algoritmo:
    ;   -- Calculo de Y0 --> Si Y0=F (factor) --> Y0=1
1780 ;                       Si Y0!=F --> Y0 lo que se calcule
    ;   -- Incremento de Y --> Y=Y+KF
    ;   -- Calculo de Y1 --> Si Y1=F --> INC Y --> Y=Y+F --> Y1=1 -->
    ;                                                    --> HAY_INCREMENTO_EJEY
    ;                       Si Y1>F --> INC Y --> Y=Y+F --> Nueva Y1 --> INC ebx
1785 ;                       Si Y1<F --> Y1 lo que se calcule
    .MiniNoEntEjeY:
            push ecx
            push esi
            cvtpi2ps xmm2,mm0    ; Calculo de Y0
1790        subps xmm2,xmm0
            psrldq xmm2,4
            movss xmm1,xmm7
            subss xmm1,xmm2
            movss xmm2,xmm1
1795        cmpss xmm2,xmm7,0b
            movd ecx,xmm2        ; Comprobamos si Y0 es igual al factor
            cmp ecx,0
            jz .Y0NoFactor
    .Y0Factor:                   ; Y0!=0 --> Y0=1
1800        pxor xmm4,xmm4       ; xmm4=0|0|0|0
            cvtpi2ps xmm4,mm7    ; xmm4=0|0|0|1=Y0 (pfsp)
            jmp .CalculoDeY1
    .Y0NoFactor:                 ; Y0!=F
            pxor xmm4,xmm4       ; xmm4=0|0|0|0
1805        divss xmm1,xmm7      ; Contribucion de Y0
            movss xmm4,xmm1      ; xmm4=0|0|0|Y0!=0 (pfsp)
    .CalculoDeY1:                ; Calculo de Y1
            paddd mm0,mm5        ; Incremento de Y
            pslldq xmm6,4        ; xmm6=0|0|(K*F)|0 (pfsp)
1810        addps xmm0,xmm6      ; xmm6=0|0|Y+(K*F)|X (pfsp)
            psrldq xmm6,4        ; xmm6=0|0|0|(K*F) (pfsp)
            cvtpi2ps xmm1,mm0    ; xmm1=0|0|(y+1)|x (pfsp)
            subps xmm1,xmm0      ; xmm1=*|*|Y1|* (pfsp)
            psrldq xmm1,4        ; xmm1=0|*|*|Y1 (pfsp)
1815        movss xmm2,xmm1      ; xmm2=0|*|*|Y1 (pfsp)
            cmpss xmm2,xmm7,0b   ; xmm2=*|*|*|"F..F" si Y1=F y "0..0" si Y1!=F
            movd ecx,xmm2
            cmp ecx,0            ; Comprobamos si Y1 es igual al Factor
            jz .Y1DistintaDeFactor
1820 .Y1IgualAFactor:            ; Y1=F
            cvtpi2ps xmm2,mm7    ; xmm2=*|*|0|1 (pfsp)
            pslldq xmm2,12       ; xmm2=1|0|0|0 (pfsp)
            psrldq xmm2,8        ; xmm2=0|0|1|0 (pfsp)
            por xmm4,xmm2        ; xmm4=0|0|Y1|Y0 (pfsp)
1825        pslldq xmm7,4
            addps xmm0,xmm7
            psrldq xmm7,4
            mov ebx,1
            mov dword [HAY_INCREMENTO_EJEY],ebx
1830        xor ebx,ebx
            jmp .AntesDeBucleX
    .Y1DistintaDeFactor:         ; Y1!=F
            movss xmm2,xmm1      ; xmm2=0|*|*|Y1 (pfsp)
            cmpss xmm2,xmm7,01b  ; xmm2=*|*|*|"F..F" si Y1<F y "0..0" si Y1>F
1835        movd ecx,xmm2
            cmp ecx,0
```

```
        jnz .Y1MenorDeFactor
    .Y1MayorDeFactor:           ; Y1>F
        subss xmm1,xmm7         ; xmm1=*|*|*|Y1 (nueva) (pfsp)
        divss xmm1,xmm7         ; Contribucion de Y1
        pslldq xmm1,12          ; xmm1=Y1|0|0|0 (pfsp)
        psrldq xmm1,8           ; xmm1=0|0|Y1|0 (pfsp)
        por xmm4,xmm1           ; xmm4=0|0|Y1|Y0 (pfsp)
        pslldq xmm7,4           ; xmm7=0|0|F|0 (pfsp)
        addps xmm0,xmm7         ; xmm0=0|0|Y+F|X (pfsp)
        psrldq xmm7,4           ; xmm7=0|0|0|F (pfsp)
        inc ebx
        jmp .AntesDeBucleX
    .Y1MenorDeFactor:           ; Y1<F
        divss xmm1,xmm7         ; Contribucion de Y1
        pslldq xmm1,12          ; xmm1=Y1|0|0|0 (pfsp)
        psrldq xmm1,8           ; xmm1=0|0|Y1|0 (pfsp)
        por xmm4,xmm1           ; xmm4=0|0|Y1|Y0 (pfsp)
    ; Para la coordenada X se aplica un algortimo similar al de la coordenada
    ; Y, dicho algortimo resulta:
    ;   -- Calculo de X0 --> Si X0=F (factor) --> X0=1
    ;                       Si X0!=F --> X0 lo que se calcule
    ;   -- Incremento de X --> X=X+KF
    ;   -- Calculo de X1 --> Si X1=F --> INC X --> X=X+F --> X1=1 -->
    ;                                                   --> HAY_INCREMENTO_EJEY
    ;                       Si X1>F --> INC X --> X=X+F --> Nueva X1 --> INC edx
    ;                       Si X1<F --> X1 lo que se calcule
    .AntesDeBucleX:
        movd ecx,mm4
    .MiniNoEntEjeX:
        push ecx
        push esi
        cvtpi2ps xmm2,mm0       ; Calculo de X0
        subss xmm2,xmm0
        movss xmm1,xmm7
        subss xmm1,xmm2
        movss xmm2,xmm1
        cmpss xmm2,xmm7,0b
        movd ecx,xmm2           ; Comprobamos si X0 es igual a Factor
        cmp ecx,0
        jz .X0NoFactor
    .X0Factor:                  ; X0 igual a Factor
        cvtpi2ps xmm2,mm7       ; xmm2=*|*|0|1=X0 (pfsp)
        pslldq xmm2,8           ; xmm2=0|1=X0|0|0 (pfsp)
        por xmm4,xmm2           ; xmm4=0|X0|Y1|Y0 (pfsp)
        jmp .CalculoDeX1
    .X0NoFactor:                ; X0!=Factor
        divss xmm1,xmm7
        pslldq xmm1,12          ; xmm1=X0|0|0|0 (pfsp)
        psrldq xmm1,4           ; xmm1=0|X0|0|0 (pfsp)
        por xmm4,xmm1           ; xmm4=0|X0|Y1|Y0 (pfsp)
    .CalculoDeX1:               ; Calculo de X1
        paddd mm0,mm7
        addss xmm0,xmm6         ; xmm6=0|0|Y|X+(K*F) (pfsp)
        cvtpi2ps xmm1,mm0       ; xmm1=0|0|y|(x+1) (pfsp)
        subss xmm1,xmm0         ; xmm1=*|*|*|X1 (pfsp)
        movss xmm2,xmm1         ; xmm2=*|*|*|X1 (pfsp)
        cmpss xmm2,xmm7,0b      ; xmm2=*|*|*|"F..F" si X1=F y "0..0" si X1!=F
        movd ecx,xmm2
        cmp ecx,0
        jz .X1DistintaDeFactor
    .X1IgualAFactor:            ; X1=Factor
        cvtpi2ps xmm2,mm7       ; xmm2=*|*|0|1 (pfsp)
        pslldq xmm2,12          ; xmm2=1|0|0|0 (pfsp)
        por xmm4,xmm2           ; xmm4=X1|X0|Y1|Y0 (pfsp)
        addss xmm0,xmm7
        mov edx,1
        mov dword [HAY_INCREMENTO_EJEX],edx
        xor edx,edx
        jmp .MiniCalculoPixel
    .X1DistintaDeFactor:        ; X1!=Factor
        movss xmm2,xmm1         ; xmm2=0|*|*|X1 (pfsp)
        cmpss xmm2,xmm7,01b     ; xmm2=*|*|*|"F..F" si X1<F y "0..0" si X1>F
        movd ecx,xmm2
        cmp ecx,0
```

```
        jnz .X1MenorDeFactor
.X1MayorDeFactor:           ; X1>Factor
        subss xmm1,xmm7     ; xmm1=*|*|*|X1 (nueva) (pfsp)
        divss xmm1,xmm7
        pslldq xmm1,12      ; xmm1=X1|0|0|0 (pfsp)
        por xmm4,xmm1       ; xmm4=X1|X0|Y1|Y0 (pfsp)
        addss xmm0,xmm7     ; xmm0=0|0|Y|X+F (pfsp)
        inc edx
        jmp .MiniCalculoPixel
.X1MenorDeFactor:           ; X1<Factor
        divss xmm1,xmm7
        pslldq xmm1,12      ; xmm1=X1|0|0|0 (pfsp)
        por xmm4,xmm1       ; xmm4=X1|X0|Y1|Y0 (pfsp)
; Tenemos xmm4=X1|X0|Y1|Y0 (pfsp), ahora vamos calculando cada pixel de la
; Imagen Destino, sabiendo que esta formado por Factor pixeles de la Imagen
; Origen y conociendo las distancia entre coordenadas
.MiniCalculoPixel:
        pxor xmm1,xmm1      ; xmm1 -> Llevara la cuenta del pixel
        ; Calculo del primer pixel de la primera fila
        movd xmm3,[esi]     ; xmm3=*|*|*|*RGB (enteros)
        add esi,4
        punpcklbw xmm3,xmm5 ; xmm3=*|*|0*0R0G0B (enteros)
        punpcklwd xmm3,xmm5 ; xmm3=*|R|G|B (enteros 32 bits)
        cvtdq2ps xmm3,xmm3  ; xmm3=*|R|G|B (pfsp)
        pshufd xmm2,xmm4,0AAh ; xmm2=X0|X0|X0|X0 (pfsp)
        mulps xmm3,xmm2     ; xmm3=Pixel*X0
        pshufd xmm2,xmm4,0h ; xmm2=Y0|Y0|Y0|Y0 (pfsp)
        mulps xmm3,xmm2     ; xmm3=Pixel*X0*Y0
        movups xmm1,xmm3    ; xmm1=xmm3
        movd ecx,mm3
        add ecx,edx
        cmp ecx,0
        jz .PixelUltimoPrimeraFila
.PixelesCentralesPrimeraFila:
        movd xmm3,[esi]     ; xmm3=*|*|*|*RGB (enteros)
        add esi,4
        punpcklbw xmm3,xmm5 ; xmm3=*|*|0*0R0G0B (enteros)
        punpcklwd xmm3,xmm5 ; xmm3=*|R|G|B (enteros 32 bits)
        cvtdq2ps xmm3,xmm3  ; xmm3=*|R|G|B (pfsp)
        mulps xmm3,xmm2     ; xmm3=Pixel*Y0 (ya que X=1)
        addps xmm1,xmm3     ; xmm1 --> xmm1+xmm3
        loop .PixelesCentralesPrimeraFila
.PixelUltimoPrimeraFila:
        movd xmm3,[esi]     ; xmm3=*|*|*|*RGB (enteros)
        punpcklbw xmm3,xmm5 ; xmm3=*|*|0*0R0G0B (enteros)
        punpcklwd xmm3,xmm5 ; xmm3=*|R|G|B (enteros 32 bits)
        cvtdq2ps xmm3,xmm3  ; xmm3=*|R|G|B (pfsp)
        mulps xmm3,xmm2     ; xmm3=Pixel*Y0
        pshufd xmm2,xmm4,0FFh ; xmm2=X1|X1|X1|X1 (pfsp)
        mulps xmm3,xmm2     ; xmm3=Pixel*Y0*X1
        addps xmm1,xmm3     ; xmm1 --> xmm1+xmm3
        movd ecx,mm2
        sub ecx,edx
        shl ecx,2           ; Salto de linea=[AnchoOrig-K-edx]*4
        add esi,ecx         ; Actualizacion de puntero
        movd ecx,mm3
        add ecx,ebx
        cmp ecx,0
        jz .PixelPrimeroUltimaFila
.CalculoFilaIntermedia:
        push ecx
        ; Primer pixel de fila intermedia
        movd xmm3,[esi]     ; xmm3=*|*|*|*RGB (enteros)
        add esi,4
        punpcklbw xmm3,xmm5 ; xmm3=*|*|0*0R0G0B (enteros)
        punpcklwd xmm3,xmm5 ; xmm3=*|R|G|B (enteros 32 bits)
        cvtdq2ps xmm3,xmm3  ; xmm3=*|R|G|B (pfsp)
        pshufd xmm2,xmm4,0AAh ; xmm2=X0|X0|X0|X0 (pfsp)
        mulps xmm3,xmm2     ; xmm3=Pixel*X0 (ya que Y=1)
        addps xmm1,xmm3     ; xmm1 --> xmm1+xmm3
        movd ecx,mm3
        add ecx,edx
        cmp ecx,0
        jz .PixelUltimoFilaCentral
```

```
1985 .PixelesCentralesFilaCentral:
        movd xmm3,[esi]         ; xmm3=*|*|*|*RGB (enteros)
        add esi,4
        punpcklbw xmm3,xmm5     ; xmm3=*|*|0*0R0G0B (enteros)
        punpcklwd xmm3,xmm5     ; xmm3=*|R|G|B (enteros 32 bits)
1990    cvtdq2ps xmm3,xmm3      ; xmm3=*|R|G|B (pfsp)
        addps xmm1,xmm3         ; xmm3=Pixel (ya que X=1 e Y=1)
        loop .PixelesCentralesFilaCentral
    .PixelUltimoFilaCentral:
        movd xmm3,[esi]         ; xmm3=*|*|*|*RGB (enteros)
1995    punpcklbw xmm3,xmm5     ; xmm3=*|*|0*0R0G0B (enteros)
        punpcklwd xmm3,xmm5     ; xmm3=*|R|G|B (enteros 32 bits)
        cvtdq2ps xmm3,xmm3      ; xmm3=*|R|G|B (pfsp)
        pshufd xmm2,xmm4,0FFh   ; xmm2=X1|X1|X1|X1 (pfsp)
        mulps xmm3,xmm2         ; xmm3=Pixel*X1 (ya que Y=1)
2000    addps xmm1,xmm3         ; xmm1 --> xmm1+xmm3
        movd ecx,mm2
        sub ecx,edx
        shl ecx,2               ; Salto de linea=[Ancho-K-ebx]*4
        add esi,ecx             ; Actualizacion de puntero
2005    pop ecx
        dec ecx
        cmp ecx,0
        jnz .CalculoFilaIntermedia
    .PixelPrimeroUltimaFila:
2010    movd xmm3,[esi]         ; xmm3=*|*|*|*RGB (enteros)
        add esi,4
        punpcklbw xmm3,xmm5     ; xmm3=*|*|0*0R0G0B (enteros)
        punpcklwd xmm3,xmm5     ; xmm3=*|R|G|B (enteros 32 bits)
        cvtdq2ps xmm3,xmm3      ; xmm3=*|R|G|B (pfsp)
2015    pshufd xmm2,xmm4,0AAh   ; xmm2=X0|X0|X0|X0 (pfsp)
        mulps xmm3,xmm2         ; xmm3=Pixel*X0
        pshufd xmm2,xmm4,055h   ; xmm2=Y1|Y1|Y1|Y1 (pfsp)
        mulps xmm3,xmm2         ; xmm3=Pixel*X0*Y1
        addps xmm1,xmm3         ; xmm1 --> xmm1+xmm3
2020    movd ecx,mm3
        add ecx,edx
        cmp ecx,0
        jz .UltimoPixelUltimaFila
    .PixelesCentralesUltimaFila:
2025    movd xmm3,[esi]         ; xmm3=*|*|*|*RGB (enteros)
        add esi,4
        punpcklbw xmm3,xmm5     ; xmm3=*|*|0*0R0G0B (enteros)
        punpcklwd xmm3,xmm5     ; xmm3=*|R|G|B (enteros 32 bits)
        cvtdq2ps xmm3,xmm3      ; xmm3=*|R|G|B (pfsp)
2030    mulps xmm3,xmm2         ; xmm3=Pixel*Y1 (ya que X=1)
        addps xmm1,xmm3         ; xmm1 --> xmm1+xmm3
        loop .PixelesCentralesUltimaFila
    .UltimoPixelUltimaFila:
        movd xmm3,[esi]         ; xmm3=*|*|*|*RGB (enteros)
2035    punpcklbw xmm3,xmm5     ; xmm3=*|*|0*0R0G0B (enteros)
        punpcklwd xmm3,xmm5     ; xmm3=*|R|G|B (enteros 32 bits)
        cvtdq2ps xmm3,xmm3      ; xmm3=*|R|G|B (pfsp)
        mulps xmm3,xmm2         ; xmm3=Pixel*Y1
        pshufd xmm2,xmm4,0FFh   ; xmm2=X1|X1|X1|X1 (pfsp)
2040    mulps xmm3,xmm2         ; xmm3=Pixel*Y1*X1 (ya que Y=1)
        addps xmm1,xmm3         ; xmm1 --> xmm1+xmm3
        ; Ya tengo en xmm1=*|R|G|B (pfsp)
        pshufd xmm2,xmm7,0h     ; xmm2=F|F|F|F (pfsp)
        rcpps xmm2,xmm2         ; xmm2=A|A|A|A siendo A=1/F (pfsp)
2045    mulps xmm2,xmm2         ; xmm2=A2|A2|A2|A2 siendo A2=A*A (pfsp)
        divps xmm1,xmm2         ; xmm1=*|R|G|B final (pfsp)
        cvtps2dq xmm1,xmm1      ; xmm1=*|R|G|B (enteros 32 bits)
        packssdw xmm1,xmm5      ; xmm1=*|*|0*0R0G0B
        packuswb xmm1,xmm5      ; xmm1=*|*|*|*RGB
2050    movd [edi],xmm1         ; [edi] <- xmm1
        add edi,4
        pop esi
        add esi,eax             ; Actualizacion de puntero
    ; Dependiendo de las situaciones que nos hemos encontrado al calcular el
2055 ; pixel, es decir, si ha habido incremento en las variable HAY_INCREMENTO_EJEX
    ; o incremento en el registro edx tendremos que actualizar el puntero
        cmp edx,0
        jz .Siguiente
```

```
        add esi,4
.Siguiente:
        mov edx,[HAY_INCREMENTO_EJEX]
        cmp edx,0
        jz .Siguiente2
        add esi,4
        xor edx,edx
        mov dword [HAY_INCREMENTO_EJEX],edx
.Siguiente2:
; Tengo que limpiar o eliminar o dejar xmm4 de la forma xmm4=0|0|Y1|Y0
        pand xmm4,[MASC_MINI_AND_XMM4]
        pop ecx
        dec ecx
        cmp ecx,0
        jnz .MiniNoEntEjeX
; Tengo que anular la componente X de mm0, mm1 y mm2 , aumentar la componente
; Y de mm1 y aumentar la componente x de mm1
        pand mm0,mm6
; Tengo que anular la X de xmm0
        pand xmm0,[MASC_MINI_XMM0]
        pop esi
        mov ecx,eax
        movd eax,mm1
        mul ecx
        add esi,eax
        xor edx,edx
        mov eax,ecx
; Dependiendo de las situaciones que nos hemos encontrado al calcular el
; pixel, es decir, si ha habido incremento en las variable HAY_INCREMENTO_EJEY
; o incremento en el registro ebx tendremos que actualizar el puntero
        cmp ebx,0
        jz .Siguiente3
        movd ecx,mm1
        shl ecx,2
        add esi,ecx
.Siguiente3:
        mov ebx,[HAY_INCREMENTO_EJEY]
        cmp ebx,0
        jz .Siguiente4
        movd ecx,mm1
        shl ecx,2
        add esi,ecx
        xor ebx,ebx
        mov dword [HAY_INCREMENTO_EJEY],ebx
.Siguiente4:
        pop ecx
        dec ecx
        cmp ecx,0
        jnz .MiniNoEntEjeY
        popad
        pop ebp ; Epilogo
        ret

_ASM_Convolucion_Entera:
; Dada una Imagen Origen o un Bloque de datos origen lo que realiza
; es la convolucion de dicha Imagen Origen con una Matriz de enteros para
; obtener una nueva Imagen Destino
Dir_Orig_14   equ  8
Dir_Dest_14   equ 12
Ancho_Orig_14 equ 16
Alto_Orig_14  equ 20
N_14          equ 24
Matriz_14     equ 28
        push ebp
        mov ebp,esp ; Prologo
        pushad
        mov esi,[ebp+Dir_Orig_14] ; esi -> Imagen Origen
        mov edi,[ebp+Dir_Dest_14] ; edi -> Imagen Destino
        mov ebx,[ebp+N_14] ; ebx=N
        movd mm7,ebx       ; mm7=0|N
        mov eax,[ebp+Ancho_Orig_14] ; eax=AnchoOrig
        sub eax,ebx  ; eax=AnchoOrig-N
        inc eax      ; eax=[AnchoOrig-(N-1)]
        movd mm6,eax ; mm6=0|[(AnchoOrig-(N-1)] -> Bucle X
```

```
        dec eax       ; eax=AnchoOrig-N
        shl eax,2     ; eax=(AnchoOrig-N)*4 -> Salto de linea
        mov ecx,[ebp+Alto_Orig_14] ; ecx=AltoOrig
        inc ecx       ; ecx=AltoOrig+1
        sub ecx,ebx   ; ecx=AltoOrig-(N-1) -> Bucle Y
        dec ebx       ; ebx=N-1
        shl ebx,2     ; ebx=(N-1)*4 -> Para el Incremento
        pxor xmm7,xmm7 ; xmm7 -> Para la entremezclado
        movd mm5,[ebp+Matriz_14] ; mm5=0|Dir_Matriz
; Lo que vamos haciendo es trasladar la Matriz de dimension N*N por toda
; la Imagen Origen para obtener la multiplicacion de matrices y por lo
; tanto la convolucion, para asi obtener la Imagen Destino
.ConvEntEjeY:
        push ecx
        movd ecx,mm6    ; ecx=AnchoOrig-(N-1)
.ConvEntEjeX:
        push ecx
        push esi
        movd ecx,mm7    ; ecx=N
        movd edx,mm5    ; edx=Dir_Matriz
        pxor xmm5,xmm5 ; xmm5 -> Llevara la suma de la convolucion
.ConvolucionPixelColumna:
        push ecx
        movd ecx,mm7    ; ecx=N
.ConvolucionPixelFila:
        movd xmm0,[esi]      ; xmm0=0|0|0|*RGB (enteros)
        punpcklbw xmm0,xmm7 ; xmm0=0|0|0*0R0G0B (enteros 16 bits)
        punpcklwd xmm0,xmm7 ; xmm0=*|R|G|B (enteros 32 bits)
        cvtdq2ps xmm0,xmm0  ; xmm0=*|R|G|B (pfsp)
        movd xmm1,[edx]      ; xmm1 -> Un valor entero de la Matriz=M
        pshufd xmm1,xmm1,0h ; xmm1=M|M|M|M (enteros)
        cvtdq2ps xmm1,xmm1  ; xmm1=M|M|M|M (pfsp)
        mulps xmm0,xmm1      ; xmm0=*|R*M|G*M|B*M (pfsp)
        addps xmm5,xmm0      ; xmm5 -> Lleva la suma de la convolucion
        add edx,4
        add esi,4
        loop .ConvolucionPixelFila
        add esi,eax
        pop ecx
        loop .ConvolucionPixelColumna
        pxor xmm2,xmm2        ; Compruebo si algun valor obtenido de la
        cmpps xmm2,xmm5,001b ; convolucion es menor de 0 si es asi pongo
        andps xmm2,xmm5       ; esa componente a 0
        cvtps2dq xmm2,xmm2    ; xmm2=*|R|G|B (enteros 32 bits)
        packssdw xmm2,xmm7    ; xmm2=0|0|0*0R0G0B (enteros 16 bits)
        packuswb xmm2,xmm7    ; xmm2=0|0|0|*RGB
        movd [edi],xmm2       ; [edi] <- xmm2
        add edi,4
        pop esi
        add esi,4
        pop ecx
        loop .ConvEntEjeX
        add esi,ebx
        pop ecx
        loop .ConvEntEjeY
        popad
        pop ebp ; Epilogo
        ret

_ASM_Convolucion_Real:
; Dada una Imagen Origen o un Bloque de datos origen lo que realiza
; es la convolucion de dicha Imagen Origen con una Matriz de numeros reales
; para obtener una nueva Imagen Destino
Dir_Orig_15   equ  8
Dir_Dest_15   equ 12
Ancho_Orig_15 equ 16
Alto_Orig_15  equ 20
N_15          equ 24
Matriz_15     equ 28
        push ebp
        mov ebp,esp ; Prologo
        pushad
        mov esi,[ebp+Dir_Orig_15] ; esi -> Imagen Origen
        mov edi,[ebp+Dir_Dest_15] ; edi -> Imagen Destino
```

```
        mov ebx,[ebp+N_15] ; ebx=N
        movd mm7,ebx       ; mm7=0|N
        mov eax,[ebp+Ancho_Orig_15] ; eax=AnchoOrig
        sub eax,ebx  ; eax=AnchoOrig-N
        inc eax      ; eax=[AnchoOrig-(N-1)]
        movd mm6,eax ; mm6=0|[(AnchoOrig-(N-1)] -> Bucle X
        dec eax      ; eax=AnchoOrig-N
        shl eax,2    ; eax=(AnchoOrig-N)*4 -> Salto de linea
        mov ecx,[ebp+Alto_Orig_15] ; ecx=AltoOrig
        inc ecx      ; ecx=AltoOrig+1
        sub ecx,ebx  ; ecx=AltoOrig-(N-1) -> Bucle Y
        dec ebx      ; ebx=N-1
        shl ebx,2    ; ebx=(N-1)*4 -> Para el Incremento
        pxor xmm7,xmm7 ; xmm7 -> Para la entremezclado
        movd mm5,[ebp+Matriz_15] ; mm5=0|Dir_Matriz
; Lo que vamos haciendo es trasladar la Matriz de dimension N*N por toda
; la Imagen Origen para obtener la multiplicacion de matrices y por lo
; tanto la convolucion, para asi obtener la Imagen Destino
.ConvRealEjeY:
        push ecx
        movd ecx,mm6   ; ecx=AnchoOrig-(N-1) -> Control de bucle
.ConvRealEjeX:
        push ecx
        push esi
        movd ecx,mm7   ; ecx=N -> Control de bucle
        movd edx,mm5   ; edx=Dir_Matriz
        pxor xmm5,xmm5 ; xmm5 -> Llevara la suma de la convolucion
.ConvolucionRealPixelColumna:
        push ecx
        movd ecx,mm7   ; ecx=N -> Control de bucle
.ConvolucionRealPixelFila:
        movd xmm0,[esi]     ; xmm0=0|0|0|*RGB (enteros)
        punpcklbw xmm0,xmm7 ; xmm0=0|0|0*0R0G0B (enteros 16 bits)
        punpcklwd xmm0,xmm7 ; xmm0=*|R|G|B (enteros 32 bits)
        cvtdq2ps xmm0,xmm0  ; xmm0=*|R|G|B (pfsp)
        movupd xmm1,[edx]   ; xmm1=*|M (pfdp) siendo M=valor de la Matriz
        cvtsd2ss xmm1,xmm1  ; xmm1=*|*|*|M (pfsp)
        pshufd xmm1,xmm1,0h ; xmm1=M|M|M|M (pfsp)
        mulps xmm0,xmm1     ; xmm0=*|R*M|G*M|B*M (pfsp)
        addps xmm5,xmm0     ; xmm5 -> Lleva la suma de la convolucion
        add edx,8
        add esi,4
        loop .ConvolucionRealPixelFila
        add esi,eax
        pop ecx
        loop .ConvolucionRealPixelColumna
        pxor xmm2,xmm2       ; Compruebo si algun valor obtenido de la
        cmpps xmm2,xmm5,001b ; convolucion es menor de 0 si es asi pongo
        andps xmm2,xmm5      ; esa componente a 0
        cvtps2dq xmm2,xmm2   ; xmm2=*|R|G|B (enteros 32 bits)
        packssdw xmm2,xmm7   ; xmm2=0|0|0*0R0G0B (enteros 16 bits)
        packuswb xmm2,xmm7   ; xmm2=0|0|0|*RGB
        movd [edi],xmm2      ; [edi] <- xmm2
        add edi,4
        pop esi
        add esi,4
        pop ecx
        loop .ConvRealEjeX
        add esi,ebx
        pop ecx
        loop .ConvRealEjeY
        popad
        pop ebp ; Epilogo
        ret

_ASM_OR:
; Realiza la funcion logica OR para un total de Num_Bytes de dos bloques
; de datos o Imagenes presentes en las direcciones Dir y DirOR
Dir_Destino_16 equ  8 ; Direccion donde quiero la funcion logica OR
Dir_Origen_16  equ 12
Num_Bytes_16   equ 16
        push ebp
        mov ebp,esp ; Prologo
        pushad
```

```
        mov edi,[ebp+Dir_Destino_16]; edi -> Imagen Destino
        mov esi,[ebp+Dir_Origen_16] ; esi -> Imagen Origen
        xor edx,edx
        ; Comprueba si el numero de bytes es multiplo de 16, y segun sea
        ; este se realizara un tratamiento u otro
        mov eax,[ebp+Num_Bytes_16] ; eax=NumBytes
        mov ebx,16  ; ebx=16
        div ebx     ; eax=Cociente(NumBytes/16) edx=Resto
        mov ecx,edx ; ecx=Resto
        shr ecx,2   ; ecx=Resto/4
        cmp eax,0   ; Comprueba si el numero de bytes es menor de 16
        jz near .ORMenorDe16Bytes ; Salta si el numero de bytes es menor de 16
        mov ecx,eax ; ecx=NumBytes/16 -> Veces a realizar el bucle
        cmp edx,0
        je near .ORResto0 ; Salta si el Resto es 0
        cmp edx,4
        je .ORResto4 ; Salta si el Resto es 4
        cmp edx,8
        je .ORResto8 ; Salta si el Resto es 8
; Si no salta a ninguna etiqueta es por que el resto es 12(=edx)
.ORResto12:
        movdqu xmm0,[edi] ; Vamos realizando la operacion logica OR
        movdqu xmm1,[esi] ; entre ambos bloques de datos de 16 en 16 bytes,
        por xmm0,xmm1     ; o lo que es lo mismo de 4 en 4 pixeles
        movdqu [edi],xmm0
        add edi,16
        add esi,16
        loop .ORResto12
        movq mm0,[edi]    ; Como el resto es 12 habra que realizar la funcion
        por mm0,[esi]     ; logica OR para 3 pixeles mas
        movq [edi],mm0
        add edi,8
        add esi,8
        mov edx,[edi]
        or edx,[esi]
        mov [edi],edx
        popad
        pop ebp ; Epilogo
        ret
.ORResto8:
        movdqu xmm0,[edi] ; Vamos realizando la operacion logica OR
        movdqu xmm1,[esi] ; entre ambos bloques de datos de 16 en 16 bytes,
        por xmm0,xmm1     ; o lo que es lo mismo de 4 en 4 pixeles
        movdqu [edi],xmm0
        add edi,16
        add esi,16
        loop .ORResto8
        movq mm0,[edi]    ; Como el resto es 8 habra que realizar la funcion
        por mm0,[esi]     ; logica OR para 2 pixeles mas
        movq [edi],mm0
        popad
        pop ebp ; Epilogo
        ret
.ORResto4:
        movdqu xmm0,[edi] ; Vamos realizando la operacion logica OR
        movdqu xmm1,[esi] ; entre ambos bloques de datos de 16 en 16 bytes,
        por xmm0,xmm1     ; o lo que es lo mismo de 4 en 4 pixeles
        movdqu [edi],xmm0
        add edi,16
        add esi,16
        loop .ORResto4
        mov edx,[edi]     ; Como el resto es 1 habra que realizar la funcion
        or edx,[esi]      ; logica OR para un pixel mas
        mov [edi],edx
        popad
        pop ebp ; Epilogo
        ret
.ORResto0:
        movdqu xmm0,[edi] ; Vamos realizando la operacion logica OR
        movdqu xmm1,[esi] ; entre ambos bloques de datos de 16 en 16 bytes,
        por xmm0,xmm1     ; o lo que es lo mismo de 4 en 4 pixeles
        movdqu [edi],xmm0
        add edi,16
        add esi,16
```

```
2355            loop .ORResto0
                popad
                pop ebp ; Epilogo
                ret
    .ORMenorDe16Bytes:
2360            mov eax,[edi]     ; Vamos realizando la operacion logica OR
                or eax,[esi]      ; entre ambos bloques de datos de 4 en 4 bytes,
                add edi,4         ; o lo que es lo mismo de pixel en pixel
                add esi,4
                loop .ORMenorDe16Bytes
2365            popad
                pop ebp ; Epilogo
                ret

    _ASM_Redimensiona:
2370 ; Dada una Imagen o un bloque de datos lo que se hace es crear una nueva
    ; Imagen que presentara una nueva Anchura y una nueva Altura y que
    ; contendra a la Imagen Original. Las nuevas posiciones o pixeles de la
    ; Imagen Destino se rellenaran con pixeles nulo, es decir:
    ;                                  ------------------
2375 ;       -----------                | HHHHHHHHH | OOOO | <- Cuadro 1
    ;      | HHHHHHHHH |  ------>      | HHHHHHHHH | OOOO |
    ;      | HHHHHHHHH |  ------>      | ------------------
    ;       -----------                | OOOOOOOOOOOOOOO | <- Cuadro 2
    ;     Imagen Original               ------------------
2380 ;                                   Imagen Destino
    ; siendo:
    ;        -- H pixeles de la Imagen Original
    ;        -- O pixeles nulos de la Imagen Destino
    Dir_Origen_17          equ  8
2385 Dir_Redimensionado_17 equ 12
    Ancho_17               equ 16
    Alto_17                equ 20
    Anchura_17             equ 24
    Altura_17              equ 28
2390            push ebp
                mov ebp,esp ; Prologo
                pushad
                mov esi,[ebp+Dir_Origen_17] ; esi -> Imagen Origen
                mov edi,[ebp+Dir_Redimensionado_17] ; edi -> Imagen Destino
2395            ; Primeramente se copiara la Imagen Original en la Imagen Destino
                ; y dependiendo del tamano de la Imagen Original se hara un
                ; tratamiento u otro
                mov eax,[ebp+Ancho_17]  ; eax=Ancho
                mov ecx,eax   ; ecx=Ancho
2400            mov ebx,4     ; mov ebx,4
                xor edx,edx
                div ebx       ; eax=Cociente(Ancho/4) edx=Resto
                mov ebx,[ebp+Anchura_17]; ebx=Anchura
                sub ebx,ecx             ; ebx=Anchura-Ancho
2405            shl ebx,2               ; ebx=[Anchura-Ancho]*4 -> Salto de linea
                mov ecx,[ebp+Alto_17]   ; ecx=Alto -> Bucle Y
                cmp eax,0               ; Comprobamos si el Ancho es menor de 4 pixeles
                jz .ImagenMenorDe4Pixeles ; Salta si el Ancho es menor de 4 pixeles
    ; Vamos copiando la Imagen Original en la Imagen Destino fila a fila, hasta
2410 ; completar el numero de filas, es decir, el Alto de la Imagen Origen
    .CopiaImagenEjeY:
                push ecx
                mov ecx,eax ; ecx=Ancho/4 -> Veces a repetir el bucle
    .CopiaImagenEjeX:
2415            movdqu xmm0,[esi] ; Vamos copiando de 4 en 4 pixeles
                movdqu [edi],xmm0
                add edi,16
                add esi,16
                loop .CopiaImagenEjeX
2420            cmp edx,0         ; Comprueba si hay Resto
                jz .NoMasCopiarEnLinea ; Salta si no hay Resto
                mov ecx,edx       ; ecx=Resto
    .CopiaImagenRestanteEnLinea:
                movd mm0,[esi]    ; Vamos copiando pixel a pixel el Resto de la
2425            movd [edi],mm0    ; Imagen Origen a la Imagen Destino
                add esi,4
                add edi,4
                loop .CopiaImagenRestanteEnLinea
```

```
    .NoMasCopiarEnLinea:
2430        add edi,ebx
            pop ecx
            loop .CopiaImagenEjeY
            jmp .PintaNegroCuadroUno
            ; Si la Imagen Origen tiene un Ancho menor de 4 pixeles, iremos
2435        ; copiando fila a fila y pixel a pixel hasta completar todas las
            ; filas, es decir, el Alto de la Imagen Origen
    .ImagenMenorDe4Pixeles:
            push ecx
            mov ecx,edx ; ecx=Resto
2440    .CopiaImagenMenorDe4Pixeles:
            movd mm0,[esi] ; Vamos copiando pixel a pixel de la Imagen Origen
            movd [edi],mm0 ; a la Imagen Destino
            add esi,4
            add edi,4
2445        loop .CopiaImagenMenorDe4Pixeles
            add edi,ebx
            pop ecx
            loop .ImagenMenorDe4Pixeles
    ; Una vez copiada la Imagen Origen en la Imagen Destino pintamos de negro
2450    ; el Cuadro 1 en la Imagen Destino
    .PintaNegroCuadroUno:
            pxor xmm0,xmm0 ; xmm0 -> Para pintar de negro de 4 en 4 pixeles
            shr ebx,2   ; ebx=Anchura-Ancho
            mov eax,ebx ; eax=Anchura-Ancho
2455        cmp eax,0   ; Si la anchura del Cuadro 1 es nula no hay que pintar
            jz .PintaNegroCuadroDos ; dicho cuadro
            ; Vemos si la anchura del Cuadro 1(=Anchura-Ancho) es multiplo de
            ; 4 para ir pintando de negro de 4 en 4 pixeles
            xor edx,edx
2460        mov ebx,4   ; ebx=4
            div ebx     ; eax=Cociente((Anchura-Ancho)/4) edx=Resto
            mov ebx,[ebp+Ancho_17]  ; ebx=Ancho
            shl ebx,2               ; ebx=Ancho*4
            mov edi,[ebp+Dir_Redimensionado_17]
2465        add edi,ebx             ; edi-> apunta al primer dato a pintar en negro
            mov ecx,[ebp+Alto_17]   ; ecx=Alto
            cmp eax,0   ; Comprueba si el Cuadro 1 tiene una anchura menor de 4 pixeles
            jz .CuadroUnoMenorDe4Pixeles ; Salta si la anchura es menor de 4 pixeles
            ; Vamos pintando el Cuadro 1 fila a fila y de 4 en 4 pixeles hasta
2470        ; completar todas las filas, es decir, la altura de la Imagen Origen
    .PintaNegroCuadroUnoEjeY:
            push ecx
            mov ecx,eax ; ecx=((Anchura-Ancho)/4) -> Veces a realizar el bucle X
    .PintaNegroCuadroUnoEjeX:
2475        movdqu [edi],xmm0 ; Vamos pintando en negro de 4 en 4 pixeles
            add edi,16
            loop .PintaNegroCuadroUnoEjeX
            cmp edx,0   ; Comprueba si hay Resto
            jz .NoPintarMasEnLineaDeCuadroUno ; Salta si no hay Resto
2480        mov ecx,edx ; ecx=Resto
    .PintaNegroRestanteEnLineaCuadroUno:
            movd [edi],xmm0 ; Vamos pintando pixel a pixel el Resto
            add edi,4
            loop .PintaNegroRestanteEnLineaCuadroUno
2485    .NoPintarMasEnLineaDeCuadroUno:
            add edi,ebx
            pop ecx
            loop .PintaNegroCuadroUnoEjeY
            jmp .PintaNegroCuadroDos
2490    ; Si el Cuadro 1 tiene una anchura menor de 4 pixeles, iremos pintando
    ; en negro fila a fila y pixel a pixel hasta completar todas las
    ; filas, es decir, el Alto de la Imagen Origen
    .CuadroUnoMenorDe4Pixeles:
            push ecx
2495        mov ecx,edx ; ecx=Resto
    .PintaNegroMenorDe4Pixeles:
            movd [edi],xmm0 ; Vamos pintando en negro pixel a pixel
            add edi,4
            loop .PintaNegroMenorDe4Pixeles
2500        add edi,ebx
            pop ecx
            loop .CuadroUnoMenorDe4Pixeles
```

```
    ; Una vez pintado el Cuadro 1 pintamos el Cuadro 2 de pixeles negros
    .PintaNegroCuadroDos:
2505        pxor xmm0,xmm0 ; xmm0 -> Para pintar de negro de 4 en 4 pixeles
            mov eax,[ebp+Anchura_17] ; eax=Anchura
            mul dword [ebp+Alto_17]  ; edxeax=Anchura*Alto
            shl eax,2                ; eax=(Anchura*Alto)*4
            mov edi,[ebp+Dir_Redimensionado_17]
2510        add edi,eax ; edi -> Apunta al primer dato a pintar en negro
            mov eax,[ebp+Altura_17]  ; eax=Altura
            sub eax,[ebp+Alto_17]    ; eax=Altura-Alto
            mul dword [ebp+Anchura_17] ; edxeax=(Altura-Alto)*Anchura
            ; eax -> Numero de pixeles a pintar de negro
2515        cmp eax,0 ; Si el numero de pixeles a pintar es nulo no hay que
            jz .SalirCompletamente ; pintar el Cuadro 2
            ; Comprueba si el numero de pixeles a pintar es multiplo de 4
            ; y segun sea este se realizara un tratamiento u otro
            mov ebx,4 ; ebx=4
2520        xor edx,edx
            div ebx   ; eax=Cociente(NumPixeles/4) edx=Resto
            cmp eax,0 ; Comprueba si el Cuadro 2 es menor de 4 pixeles
            jz .CuadroDosMenorDe4Pixeles ; Salta si es menor de 4 pixeles
            mov ecx,eax ; ecx=NumPixeles/4 -> Veces a realizar el bucle
2525    .PintaCuadroDosNegro:
            movdqu [edi],xmm0 ; Vamos pintando de negro el Cuadro 2 de 4 en
            add edi,16        ; 4 pixeles
            loop .PintaCuadroDosNegro
            cmp edx,0
2530        jz .SalirCompletamente
    ; Si el Cuadro2 es menor de 4 pixeles, pintaremos de negro pixel a pixel
    .CuadroDosMenorDe4Pixeles:
            mov ecx,edx ; ecx=Resto
            xor eax,eax ; eax=0 -> Para ir pintando los pixeles en negro
2535    .PintaNegroRestanteCuadroDos:
            mov [edi],eax ; Vamos pintando en negro pixel a pixel
            add edi,4
            loop .PintaNegroRestanteCuadroDos
    .SalirCompletamente:
2540        popad
            pop ebp ; Epilogo
            ret

    _ASM_TransfAfin:
2545    ; Dada una Imagen Origen o un bloque de datos que nos llega en Dir_Origen, se
    ; obtendra una nueva Imagen Destino o bloques de datos que contendra la
    ; Transformacion Afin de la Imagen Original
    Dir_Origen_18      equ  8
    Dir_TransfAfin_18  equ 12
2550Alto_18            equ 16
    Ancho_18           equ 20
    AltoTransfAfin_18  equ 24
    AnchoTransfAfin_18 equ 28
    DesplazamientoX_18 equ 32
2555DesplazamientoY_18 equ 40
    EscalaY_18         equ 48
    EscalaX_18         equ 56
    Seno_18            equ 64
    Coseno_18          equ 72
2560        push ebp
            mov ebp,esp ; Prologo
            pushad
            ; Inicializacion de registros XMM para posteriores calculos
            movsd xmm0,[ebp+Coseno_18] ; xmm0=*|cos (pfdp)
2565        pxor xmm7,xmm7             ; xmm7=0|0|0|0
            cvtsd2ss xmm7,xmm0         ; xmm7=0|0|0|cos (pfsp)
            movups xmm6,xmm7           ; xmm6=0|0|0|cos (pfsp)
            movsd xmm0,[ebp+EscalaY_18]; xmm0=*|Sy (pfdp)
            cvtsd2ss xmm0,xmm0         ; xmm0=*|*|*|Sy (pfsp)
2570        movsd xmm1,[ebp+EscalaX_18]; xmm1=*|Sx (pfdp)
            cvtsd2ss xmm1,xmm1         ; xmm1=*|*|*|Sx (pfsp)
            movsd xmm2,[ebp+Seno_18]   ; xmm2=*|sen (pfdp)
            pxor xmm3,xmm3             ; xmm3=0|0|0|0
            cvtsd2ss xmm3,xmm2         ; xmm3=0|0|0|sen (pfsp)
2575        movups xmm2,xmm3           ; xmm2=0|0|0|sen (pfsp)
            divss xmm7,xmm1            ; xmm7=0|0|0|cos/Sx
```

```
            pslldq xmm7,12              ; xmm7=cos/Sx|0|0|0
            divss xmm2,xmm1             ; xmm2=0|0|0|sen/Sx
            pslldq xmm2,8               ; xmm2=0|sen/Sx|0|0
2580        por xmm7,xmm2               ; xmm7=cos/Sx|sen/Sx|0|0
            divss xmm6,xmm0             ; xmm6=0|0|0|cos/Sy
            pslldq xmm6,4               ; xmm6=0|0|cos/Sy|0
            por xmm7,xmm6               ; xmm7=cos/Sx|sen/Sx|cos/Sy|0
            divss xmm3,xmm0             ; xmm3=0|0|0|sen/Sy
2585        por xmm7,xmm3               ; xmm7=cos/Sx|sen/Sx|cos/Sy|sen/Sy
            movsd xmm0,[ebp+DesplazamientoX_18] ; xmm0=*|Dx (pfdp)
            pxor xmm1,xmm1              ; xmm1=0|0|0|0
            cvtsd2ss xmm1,xmm0          ; xmm1=0|0|0|Dx (pfsp)
            movups xmm0,xmm1            ; xmm0=0|0|0|Dx (pfsp)
2590        pslldq xmm1,12              ; xmm1=Dx|0|0|0 (pfsp)
            subps xmm0,xmm1             ; xmm0=-Dx|0|0|Dx (pfsp)
            movups xmm1,xmm0            ; xmm1=-Dx|0|0|Dx (pfsp)
            movsd xmm2,[ebp+DesplazamientoY_18] ; xmm2=*|Dy (pfdp)
            pxor xmm3,xmm3              ; xmm3=0|0|0|0
2595        cvtsd2ss xmm2,xmm2          ; xmm2=*|*|*|Dy (pfsp)
            subss xmm3,xmm2             ; xmm3=0|0|0|-Dy (pfsp)
            pslldq xmm3,4               ; xmm3=0|0|-Dy|0 (pfsp)
            por xmm1,xmm3               ; xmm1=-Dx|0|-Dy|Dx (pfsp)
            pslldq xmm3,4               ; xmm3=0|-Dy|0|0 (pfsp)
2600        por xmm1,xmm3               ; xmm1=-Dx|-Dy|-Dy|Dx (pfsp)
            movdqu xmm4,[MASC_AND_ROTACION] ; xmm4=0|F|F|0
            cvtdq2ps xmm5,[MASC_ROTACION_Y] ; xmm5=0|1|1|0 (pfsp)
            cvtdq2ps xmm6,[MASC_ROTACION_X] ; xmm6=1|0|0|-1 (pfsp)
            ; Inicializacion registros MMX para posteriores calculos
2605        movd mm5,[ebp+Dir_Origen_18] ; mm5=0|Dir
            mov eax,[ebp+Ancho_18]  ; eax=Ancho
            dec eax                 ; eax=Ancho-1
            movd mm7,eax            ; mm7=0|(Ancho-1)
            psllq mm7,32            ; mm7=(Ancho-1)|0
2610        inc eax                 ; eax=Ancho
            movd mm3,eax            ; mm3=0|Ancho
            mov eax,1               ; eax=1
            shl eax,16              ; eax=1|0
            movd mm6,eax            ; mm6=0||1|0
2615        psubsw mm6,mm3          ; mm6=0||1|-Ancho
            mov ebx,[ebp+Alto_18]   ; ebx=Alto
            dec ebx                 ; ebx=Alto-1
            movd mm4,ebx            ; mm4=0|(Alto-1)
            por mm7,mm4             ; mm7=(Ancho-1)|(Alto-1)
2620        ; Inicializacion del bloque de datos a recorrer
            mov eax,[ebp+AnchoTransfAfin_18] ; eax=AnchoRot
            mov ebx,eax             ; ebx=AnchoRot -> Bucle X
            mov ecx,[ebp+AltoTransfAfin_18]  ; ecx=AltoRot
            dec ecx                 ; ecx=AltoRot-1
2625        mul ecx                 ; eax=AnchoRot*(AltoRot-1)
            inc ecx                 ; ecx=AltoRot -> Bucle Y
            shl eax,2               ; eax=AnchoRot*(AltoRot-1)*4
            mov esi,eax             ; esi-> Bloque de datos origen
            add esi,[ebp+Dir_TransfAfin_18]
2630        mov eax,ebx             ; eax=AnchoRot
            shl eax,3               ; eax=AnchoRot*8 -> Salto de linea
    ; Para realizar la transformacion afin lo que iremos haciendo es recorrer la
    ; Imagen Destino, de izquierda a derecha y de abajo a arriba, partiendo de la
    ; coordenada (0,0) e incrementando dichas coordenadas durante la realizacion
2635 ; del recorrido, para obtener el valor de las coordenadas que le corresponden
    ; en la Imagen Origen. Las ecuaciones utilizadas para la obtencion de dichas
    ; coordenadas son:
    ;       -- x=[(x'-Dx)*cos(a)+(y'-Dy)*sen(a)]/Sx
    ;       -- y=[(y'-Dy)*cos(a)+(Dx-x')*sen(a)]/Sy
2640 ; siendo:
    ;       -- (x,y) las coordenadas de la Imagen Origen
    ;       -- (x',y') las coordenadas de la Imagen Destino
    ;       -- a el angulo de rotacion
    ;       -- Dx y Dy los desplazamientos que son pasados a la rutina
2645 ;          y que tienen que ver con lo que se desplaza la Imagen Origen
    ;          a la hora de hacer la rotacion en cuanto al eje X y al eje Y
    ;       -- Sx y Sy los factores de escala para el eje X y para el eje Y
    ;          respectivamente
    .TransfAfinEjeY:
2650        push ecx
```

```
        mov ecx,ebx
.TransfAfinEjeX:
        movups xmm2,xmm7   ; xmm2=Sxcos|Sxsen|Sycos|Sysen (pfsp)
        mulps xmm2,xmm1    ; xmm2=(x'-Dx)Sxcos|(y'-Dy)Sysen|(y'-Dy)Sycos|
                           ;      |(Dx-x')Sysen (pfsp)
        haddps xmm2,xmm1   ; xmm2=*|*|x|y (pfsp)
        cvtps2pi mm0,xmm2  ; mm0=x|y (enteros 32 bits)
        pxor mm3,mm3       ; Compruebo que las coordenadas obtenidas pertenecen
        pcmpgtd mm3,mm0    ; a alguna coordenada de la Imagen Origen, es decir:
        movq mm1,mm0       ;                0<=X<=(Ancho-1)
        pcmpgtd mm1,mm7    ;                0<=Y<=(Alto-1)
        por mm1,mm3
        movq mm2,mm1
        psrlq mm2,32
        por mm2,mm1
        movd edx,mm2       ; Si edx o mm2 es todo unos la coordenada obtenida
        cmp edx,0          ; no es valida o no pertenece a la Imagen Origen,
        neg edx            ; en este caso se pinta el pixel en negro
        jnz .TransfAfinPixelNulo
        ; La coordenada es valida calculamos el valor del pixel para esa
        ; coordenada en la Imagen Origen y le copiamos en la Imagen Destino
        packssdw mm0,mm0   ; mm0=*|xy (enteros de 32 bits)
        psubsw mm0,mm4     ; mm0=*|x (y-Alto)
        pmaddwd mm0,mm6    ; mm0=*|x+(Alto-y)*Ancho
        pslld mm0,2        ; mm0=*|[x+(Alto-y)*Ancho]*4
        paddd mm0,mm5      ; mm0=*|[[x+(Alto-y)*Ancho]*4]+Dir
        movd edi,mm0       ; edi=Dir_Pixel
        mov edx,[edi]      ; edx=[Dir_Pixel]
.TransfAfinPixelNulo:
        mov [esi],edx      ; [esi]=[Dir_Pixel]
        add esi,4
        addps xmm1,xmm6    ; xmm1 -> Incremento x
        loop .TransfAfinEjeX
        andps xmm1,xmm4    ; xmm1 -> Deja solo pasar la componente y
        addps xmm1,xmm5    ; xmm1 -> Incremento y
        addps xmm1,xmm0    ; xmm1 -> Inicializacion de la componente X
        sub esi,eax
        pop ecx
        loop .TransfAfinEjeY
        popad
        pop ebp ; Epilogo
        ret

_ASM_Ecualizado:
; Dada una Imagen que llega a traves de Dir se ecualiza a partir de una
; MatrizCualizado (calculada anteriormente a traves de las curvas de Beizer
; o de Splines), ademas se permiten 8 tipos de ecualizados, dicho ecualizado
; nos llega a traves de la variable TipoEcualizado, asi:
; TipoEcualizado=  0 ("000") No ecualiza nada
;                  1 ("001") Solo ecualiza Componente Azul
;                  2 ("010") Solo ecualiza Componente Verde
;                  3 ("011") Ecualiza Componentes Verde y Azul
;                  4 ("100") Solo ecualiza Componente Roja
;                  5 ("101") Ecualiza Componentes Roja y Azul
;                  6 ("110") Ecualiza Componentes Roja y Verde
;                  7 ("111") Ecualiza las tres Componentes
Dir_Origen_19       equ  8
NumBytes_19         equ 12
TipoEcualizado_19   equ 16
MatrizEcualizado_19 equ 20
      push ebp
      mov ebp,esp ; Prologo
      pushad
      mov esi,[ebp+MatrizEcualizado_19] ; esi -> MatrizEcualizado
      xor edx,edx
      mov eax,[ebp+NumBytes_19] ; eax=Num_Bytes
      mov ebx,16  ; ebx=16
      ; Dividimos el Numero de Bytes que tenemos que ecualizar entre 16
      ; ya que podemos ir ecualizando de 16 en 16 bytes
      div ebx     ; eax=Cociente y edx=Resto
      shr edx,2   ; El resto solo puede ser 0,4,8 o 12 (multiplo de 4)
      mov ebx,[ebp+TipoEcualizado_19] ; ebx=TipoEcualizado
      push ebx    ; Guardamos en la pila ebx=TipoEcualizado
      bt ebx,0    ; Comprobamos si hay que ecualizar la Componente Azul
```

```
2725        jnc near .EcualizadoVerde
    .EcualizadoAzul:
            movdqu xmm7,[MASC_ECUALIZADO_AZUL_0] ; xmm7=000F|000F|000F|000F
            movdqu xmm6,[MASC_ECUALIZADO_AZUL_1] ; xmm6=0FF0|0FF0|0FF0|0FF0
            mov edi,[ebp+Dir_Origen_19] ; edi -> Imagen a ecualizar
2730        cmp eax,0    ; Si el Num_Bytes es menor de 16
            jz near .HayRestoComponenteAzul
            mov ecx,eax ; ecx=Num_Bytes/16 -> Veces a repetir el bucle
    .ComponenteAzul:
            movdqu xmm0,[edi]        ; xmm0=*RGB(4)|*RGB(3)|*RGB(2)|*RGB(1)
2735        movups xmm1,xmm0         ; xmm1=*RGB(4)|*RGB(3)|*RGB(2)|*RGB(1)
            pand xmm1,xmm7           ; xmm1=000B(4)|000B(3)|000B(2)|000B(1)
            pand xmm0,xmm6           ; xmm0=0RG0(4)|0RG0(3)|0RG0(2)|0RG0(1)
            pshufd xmm1,xmm1,1Bh     ; xmm1=000B(1)|000B(2)|000B(3)|000B(4)
            movd ebx,xmm1            ; ebx=000B(4)
2740        movd xmm2,[esi+(ebx*4)] ; xmm2=*|*|*|000X(4)
            pslldq xmm2,4            ; xmm2=*|*|000X(4)|0
            psrldq xmm1,4            ; xmm1=0|000B(1)|000B(2)|000B(3)
            movd ebx,xmm1            ; ebx=000B(3)
            movd xmm3,[esi+(ebx*4)] ; xmm3=0|0|0|000X(3)
2745        por xmm2,xmm3            ; xmm2=*|*|000X(4)|000X(3)
            pslldq xmm2,4            ; xmm2=*|000X(4)|000X(3)|0
            psrldq xmm1,4            ; xmm1=0|0|000B(1)|000B(2)
            movd ebx,xmm1            ; ebx=000B(2)
            movd xmm3,[esi+(ebx*4)] ; xmm3=0|0|0|000X(2)
2750        por xmm2,xmm3            ; xmm2=*|000X(4)|000X(3)|000X(2)
            pslldq xmm2,4            ; xmm2=000X(4)|000X(3)|000X(2)|0
            psrldq xmm1,4            ; xmm1=0|0|0|000B(1)
            movd ebx,xmm1            ; ebx=000B(1)
            movd xmm3,[esi+(ebx*4)] ; xmm3=0|0|0|000X(1)
2755        por xmm2,xmm3            ; xmm2=000X(4)|000X(3)|000X(2)|000X(1)
            por xmm0,xmm2            ; xmm0=0RGX(4)|0RGX(3)|0RGX(2)|0RGX(1)
            movdqu [edi],xmm0
            add edi,16
            loop .ComponenteAzul
2760    .HayRestoComponenteAzul:
            cmp edx,0                ; Comprobamos si hay resto
            jz .EcualizadoVerde
            mov ecx,edx              ; ecx=Resto de pixeles a ecualizar(1,2 o 3)
    .RestoComponenteAzul:
2765        movd xmm0,[edi]          ; xmm0=*|*|*|*RGB
            movups xmm1,xmm0         ; xmm1=*|*|*|*RGB
            pand xmm1,xmm7           ; xmm1=*|*|*|000B
            pand xmm0,xmm6           ; xmm0=*|*|*|0RG0
            movd ebx,xmm1            ; ebx=000B
2770        movd xmm1,[esi+(ebx*4)] ; xmm1=*|*|*|000X
            por xmm0,xmm1            ; xmm0=*|*|*|0RGX
            movd [edi],xmm0
            add edi,4
            loop .RestoComponenteAzul
2775    .EcualizadoVerde:
            pop ebx    ; Recuperamos de la pila ebx=TipoEcualizado
            push ebx   ; Guardamos en la pila ebx=tipoEcualizado
            bt ebx,1   ; Comprobamos si hay que ecualizar la Componente Verde
            jnc near .EcualizadoRojo
2780        movdqu xmm7,[MASC_ECUALIZADO_VERDE_0] ; xmm7=00F0|00F0|00F0|00F0
            movdqu xmm6,[MASC_ECUALIZADO_VERDE_1] ; xmm6=0F0F|0F0F|0F0F|0F0F
            mov edi,[ebp+Dir_Origen_19] ; edi -> Imagen a ecualizar
            cmp eax,0 ; Si el Num_Bytes es menor de 16
            jz near .HayRestoComponenteVerde
2785        mov ecx,eax ; ecx=Num_Bytes/16 -> Veces a repetir el bucle
    .ComponenteVerde:
            movdqu xmm0,[edi]        ; xmm0=*RGB(4)|*RGB(3)|*RGB(2)|*RGB(1)
            movups xmm1,xmm0         ; xmm1=*RGB(4)|*RGB(3)|*RGB(2)|*RGB(1)
            pand xmm1,xmm7           ; xmm1=00G0(4)|00G0(3)|00G0(2)|00G0(1)
2790        pand xmm0,xmm6           ; xmm0=0R0B(4)|0R0B(3)|0R0B(2)|0R0B(1)
            psrld xmm1,8             ; xmm1=000G(4)|000G(3)|000G(2)|000G(1)
            pshufd xmm1,xmm1,1Bh     ; xmm1=000G(1)|000G(2)|000G(3)|000G(4)
            movd ebx,xmm1            ; ebx=000G(4)
            movd xmm2,[esi+(ebx*4)] ; xmm2=*|*|*|000X(4)
2795        pslldq xmm2,4            ; xmm2=*|*|000X(4)|0
            psrldq xmm1,4            ; xmm1=0|000G(1)|000G(2)|000G(3)
            movd ebx,xmm1            ; ebx=000G(3)
            movd xmm3,[esi+(ebx*4)] ; xmm3=0|0|0|000X(3)
```

```
        por xmm2,xmm3           ; xmm2=*|*|000X(4)|000X(3)
        pslldq xmm2,4           ; xmm2=*|000X(4)|000X(3)|0
        psrldq xmm1,4           ; xmm1=0|0|000G(1)|000G(2)
        movd ebx,xmm1           ; ebx=000G(2)
        movd xmm3,[esi+(ebx*4)] ; xmm3=0|0|0|000X(2)
        por xmm2,xmm3           ; xmm2=*|000X(4)|000X(3)|000X(2)
        pslldq xmm2,4           ; xmm2=000X(4)|000X(3)|000X(2)|0
        psrldq xmm1,4           ; xmm1=0|0|0|000G(1)
        movd ebx,xmm1           ; ebx=000G(1)
        movd xmm3,[esi+(ebx*4)] ; xmm3=0|0|0|000X(1)
        por xmm2,xmm3           ; xmm2=000X(4)|000X(3)|000X(2)|000X(1)
        pslld xmm2,8            ; xmm2=00X0(4)|00X0(3)|00X0(2)|00X0(1)
        por xmm0,xmm2           ; xmm0=0RXB(4)|0RXB(3)|0RXB(2)|0RXB(1)
        movdqu [edi],xmm0
        add edi,16
        loop .ComponenteVerde
.HayRestoComponenteVerde:
        cmp edx,0               ; Comprobamos si hay resto
        jz .EcualizadoRojo
        mov ecx,edx             ; ecx=Resto de pixeles a ecualizar(1,2 o 3)
.RestoComponenteVerde:
        movd xmm0,[edi]         ; xmm0=*|*|*|*RGB
        movups xmm1,xmm0        ; xmm1=*|*|*|*RGB
        pand xmm1,xmm7          ; xmm1=*|*|*|00G0
        pand xmm0,xmm6          ; xmm0=*|*|*|0R0B
        psrld xmm1,8            ; xmm1=*|*|*|000G
        movd ebx,xmm1           ; ebx=000G
        movd xmm1,[esi+(ebx*4)] ; xmm1=*|*|*|000X
        pslld xmm1,8            ; xmm1=*|*|*|00X0
        por xmm0,xmm1           ; xmm0=*|*|*|0RXB
        movd [edi],xmm0
        add edi,4
        loop .RestoComponenteVerde
.EcualizadoRojo:
        pop ebx  ; Recuperamos de la pila ebx=TipoEcualizado
        push ebx ; Guardamos en la pila ebx=TipoEcualizado
        bt ebx,2 ; Comprobamos si hay que ecualizar la Componente Roja
        jnc near .SalirCompletamente
        movdqu xmm7,[MASC_ECUALIZADO_ROJO_0] ; xmm7=0F00|0F00|0F00|0F00
        movdqu xmm6,[MASC_ECUALIZADO_ROJO_1] ; xmm6=00FF|00FF|00FF|00FF
        mov edi,[ebp+Dir_Origen_19] ; edi -> Imagen a ecualizar
        cmp eax,0   ; Si Num_Bytes es menor de 16
        jz near .HayRestoComponenteRoja
        mov ecx,eax ; ecx=Num_Bytes/16 -> Veces a repetir el bucle
.ComponenteRoja:
        movdqu xmm0,[edi]       ; xmm0=*RGB(4)|*RGB(3)|*RGB(2)|*RGB(1)
        movups xmm1,xmm0        ; xmm1=*RGB(4)|*RGB(3)|*RGB(2)|*RGB(1)
        pand xmm1,xmm7          ; xmm1=0R00(4)|0R00(3)|0R00(2)|0R00(1)
        pand xmm0,xmm6          ; xmm0=00GB(4)|00GB(3)|00GB(2)|00GB(1)
        psrld xmm1,16           ; xmm1=000R(4)|000R(3)|000R(2)|000R(1)
        pshufd xmm1,xmm1,1Bh    ; xmm1=000R(1)|000R(2)|000R(3)|000R(4)
        movd ebx,xmm1           ; ebx=000R(4)
        movd xmm2,[esi+(ebx*4)] ; xmm2=*|*|*|000X(4)
        pslldq xmm2,4           ; xmm2=*|*|000X(4)|0
        psrldq xmm1,4           ; xmm1=0|000R(1)|000R(2)|000R(3)
        movd ebx,xmm1           ; ebx=000R(3)
        movd xmm3,[esi+(ebx*4)] ; xmm3=0|0|0|000X(3)
        por xmm2,xmm3           ; xmm2=*|*|000X(4)|000X(3)
        pslldq xmm2,4           ; xmm2=*|000X(4)|000X(3)|0
        psrldq xmm1,4           ; xmm1=0|0|000R(1)|000R(2)
        movd ebx,xmm1           ; ebx=000R(2)
        movd xmm3,[esi+(ebx*4)] ; xmm3=0|0|0|000X(2)
        por xmm2,xmm3           ; xmm2=*|000X(4)|000X(3)|000X(2)
        pslldq xmm2,4           ; xmm2=000X(4)|000X(3)|000X(2)|0
        psrldq xmm1,4           ; xmm1=0|0|0|000R(1)
        movd ebx,xmm1           ; ebx=000R(1)
        movd xmm3,[esi+(ebx*4)] ; xmm3=0|0|0|000X(1)
        por xmm2,xmm3           ; xmm2=000X(4)|000X(3)|000X(2)|000X(1)
        pslld xmm2,16           ; xmm2=0X00(4)|0X00(3)|0X00(2)|0X00(1)
        por xmm0,xmm2           ; xmm0=0XGB(4)|0XGB(3)|0XGB(2)|0XGB(1)
        movdqu [edi],xmm0
        add edi,16
        loop .ComponenteRoja
.HayRestoComponenteRoja:
```

```
        cmp edx,0               ; Comprobamos si hay resto
        jz .SalirCompletamente
        mov ecx,edx             ; ecx=Resto de pixeles a ecualizar(1,2 o 3)
.RestoComponenteRoja:
        movd xmm0,[edi]         ; xmm0=*|*|*|*RGB
        movups xmm1,xmm0        ; xmm1=*|*|*|*RGB
        pand xmm1,xmm7          ; xmm1=*|*|*|0R00
        pand xmm0,xmm6          ; xmm0=*|*|*|00GB
        psrld xmm1,16           ; xmm1=*|*|*|000R
        movd ebx,xmm1           ; ebx=000R
        movd xmm1,[esi+(ebx*4)] ; xmm1=*|*|*|000X
        pslld xmm1,16           ; xmm1=*|*|*|0X00
        por xmm0,xmm1           ; xmm0=*|*|*|0XGB
        movd [edi],xmm0
        add edi,4
        loop .RestoComponenteRoja
.SalirCompletamente:
        pop ebx
        popad
        pop ebp  ; Epilogo
        ret

_ASM_Escribe:
; Escribe en pantalla en modo grafico un texto contenido en Dir_Texto,
; a partir de la posicion indicada por las variables (coordenadas) X e Y.
; Ademas, tiene en cuenta el incremento automatico de las coordenadas de
; caracter de forma que si el texto se sale por el lado derecho de la pantalla,
; entre por el lado izquierdo de la misma pero en la fila de caracter
; inmediatamente inferior. Si el texto se sale por la parte inferior de la
; pantalla asomara por la parte superior.
; Tambien presenta la variable Color_Transparente que implica que los puntos
; del caracter que coincidan con ese color (combinacion RGB) no deberan ser
; pintados, con lo que se conservara el pixel de fondo
Dir_Texto_20            equ  8
Dir_BMP_20              equ 12
Ancho_BMP_20            equ 16
Alto_BMP_20             equ 20
AnchuraCaracter_BMP_20  equ 24
AlturaCaracter_BMP_20   equ 28
Selector_Pantalla_20    equ 32
Ancho_Pantalla_20       equ 36
Alto_Pantalla_20        equ 40
X_20                    equ 44
Y_20                    equ 48
ColorTransparente_20    equ 52
        push ebp
        mov ebp,esp ; Prologo
        pushad
        push gs ; Guardamos en la pila el selector de segmento gs
        ; Inicializacion de la pantalla
        mov ax,[ebp+Selector_Pantalla_20]
        mov gs,ax ; gs -> apunta a la pantalla
        ; Inicializacion de registros
        mov ebx,[ebp+AnchuraCaracter_BMP_20] ; ebx=AnchuraCaracter
        movd mm3,ebx            ; mm3=0|AnchuraCaracter
        movd xmm0,ebx           ; xmm0=0|0|0|AnchuraCaracter
        pslld xmm0,2            ; xmm0=0|0|0|AnchuraCaracter*4
        mov ecx,[ebp+Ancho_BMP_20] ; ecx=Ancho_BMP
        mov eax,ecx             ; eax=Ancho_BMP
        sub ecx,ebx             ; ecx=Ancho_BMP-AnchuraCaracter
        shl ecx,2               ; ecx=(Ancho_BMP-AnchuraCaracter)*4
        movd xmm5,ecx           ; xmm5=0|0|0|(Ancho_BMP-AnchuraCaracter)*4
        mov ecx,[ebp+AlturaCaracter_BMP_20] ; ecx=AlturaCaracter
        mul ecx                 ; edxeax=Ancho_BMP*AlturaCaracter
        shl eax,2               ; eax=Ancho_BMP*AlturaCaracter*4
        movd xmm1,eax           ; xmm1=0|0|0|(Ancho_BMP*AlturaCaracter*4)
        movd mm6,ecx            ; mm6=0|AlturaCaracter
        psllq mm6,32            ; mm6=AlturaCaracter|0
        por mm3,mm6             ; mm3=AlturaCaracter|AnchuraCaracter
        dec ecx                 ; ecx=AlturaCaracter-1
        mov ebx,[ebp+Ancho_Pantalla_20] ; ebx=AnchoPantalla
        movd xmm7,ebx           ; xmm7=0|0|0|AnchoPantalla
        psllq xmm7,2            ; xmm7=0|0|0|AnchoPantalla*4
        mov eax,ebx             ; eax=AnchoPantalla
```

```
            mul ecx                 ; edxeax=AnchoPantalla*(AlturaCaracter-1)
            shl eax,2               ; eax=AnchoPantalla*(AlturaCaracter-1)*4
            movd xmm2,eax           ; xmm2=0|0|0|AnchoPantalla*(AlturaCaracter-1)*4
2950        mov ecx,[ebp+Alto_Pantalla_20] ; ecx=AltoPantalla
            dec ecx                 ; ecx=AltoPantalla-1
            movd mm6,ecx            ; mm6=0|(AltoPantalla-1)
            psllq mm6,32            ; mm6=(AltoPantalla-1)|0
            inc ecx                 ; ecx=AltoPantalla
2955        dec ebx                 ; ebx=AnchoPantalla-1
            movd mm1,ebx            ; mm1=0|(AnchoPantalla-1)
            por mm6,mm1             ; mm6=(AltoPantalla-1)|(AnchoPantalla-1)
            inc ebx                 ; ebx=AnchoPantalla
            mov eax,ebx             ; eax=AnchoPantalla
2960        mul ecx                 ; edxeax=AnchoPantalla*AltoPantalla
            shl eax,2               ; eax=AnchoPantalla*AltoPantalla*4
            movd xmm3,eax           ; xmm3=0|0|0|(AnchoPantalla*AltoPantalla*4)
            mov eax,[ebp+Y_20]      ; eax=Y
            movd mm1,eax            ; mm1=0|Y
2965        psllq mm1,32            ; mm1=Y|0
            mul ebx                 ; edxeax=Y*AnchoPantalla
            mov ebx,[ebp+X_20]      ; ebx=X
            movd mm0,ebx            ; mm0=0|X
            por mm0,mm1             ; mm0=Y|X (coordenadas)
2970        add eax,ebx             ; eax=(Y*AnchoPantalla)+X
            shl eax,2               ; eax=((Y*AnchoPantalla)+X)*4
            mov edi,eax             ; [gs:esi]-> pantalla a pintar
            movq mm4,[MASC_ESCRIBE_INC_X] ; mm4=0|1
            movq mm5,[MASC_ESCRIBE_INC_Y] ; mm5=1|0
2975        movd mm7,[ebp+ColorTransparente_20] ; mm7=0|ColorTransparente
            pand mm7,[MASC_ESCRIBE_COLOR_TRANSPARENTE] ;mm7=0|0RGB (color transp)
            xor edx,edx
            mov ebx,[ebp+Dir_Texto_20] ; ebx=Direccion de la cadena
    .Inicio:
2980    ; Vamos leyendo caracter a caracter y vemos que tipo de caracter es, los
    ; caracteres estan divididos en 5 grupos, cada grupo se corresponde con
    ; una fila del BMP, una vez obtenido el caracter que es, se calcula su
    ; posicion dentro del BMP
            xor eax,eax
2985        mov al,[ebx+edx] ; eax=Caracter leido
            cmp eax,0        ; Si el Caracter leido es 0 hemos acabado la cadena
            jz near .SalirCompletamenteDeEscribe ; y salimos de la rutina
            push edx
            push edi
2990        movd ecx,xmm0    ; ecx=AnchoCaracter*4
            mov esi,[ebp+Dir_BMP_20]
            cmp eax,20h      ; Veo si el caracter es un espacio en blanco
            jnz .SeraUnNumero
            mov eax,10       ; Es un espacio en blanco
2995        mul ecx          ; Calculo el desplazamiento
            add esi,eax
            jmp .PintaCaracter
    .SeraUnNumero:
            cmp eax,30h      ; Veo si el caracter es un numero
3000        jb near .CaracterNoValido
            cmp eax,39h
            jg .SeraUnaMayusculaLinea1
            sub eax,30h      ; Es un numero
            mul ecx          ; Calculo el desplazamiento
3005        add esi,eax
            jmp .PintaCaracter
    .SeraUnaMayusculaLinea1:
            cmp eax,41h      ; Veo si el caracter es una mayuscula de la linea 1 del BMP
            jb .CaracterNoValido
3010        cmp eax,4dh
            jg .SeraUnaMayusculaLinea2
            sub eax,41h      ; Es una mayuscula de la primera linea del BMP
            mul ecx
            add esi,eax      ; Calculo el desplazamiento
3015        movd ecx,xmm1
            add esi,ecx
            jmp .PintaCaracter
    .SeraUnaMayusculaLinea2:
            cmp eax,5Ah      ; Veo si el caracter es una mayuscula de la linea 2 del BMP
3020        jg .SeraUnaMinusculaLinea1
```

```
        sub eax,4eh      ; Es una mayuscula de la segunda linea del BMP
        mul ecx
        add esi,eax      ; Calculo el desplazamiento
        movd ecx,xmm1
        shl ecx,1
        add esi,ecx
        jmp .PintaCaracter
.SeraUnaMinusculaLinea1:
        cmp eax,61h      ; Veo si el caracter es una minuscula de la linea 1 del BMP
        jb .CaracterNoValido
        cmp eax,6dh
        jg .SeraUnaMinusculaLinea2
        sub eax,61h      ; Es una minuscula de la primera linea del BMP
        mul ecx
        add esi,eax      ; Calculo el desplazamiento
        movd ecx,xmm1
        shl ecx,1
        add esi,ecx
        shr ecx,1
        add esi,ecx
        jmp .PintaCaracter
.SeraUnaMinusculaLinea2:
        cmp eax,7ah      ; Veo si el caracter es una minuscula de la linea 2 del BMP
        jg .CaracterNoValido
        sub eax,6eh      ; Es una minuscula de la segunda linea del BMP
        mul ecx
        add esi,eax      ; Calculo el desplazamiento
        movd ecx,xmm1
        shl ecx,2
        add esi,ecx
        jmp .PintaCaracter
.CaracterNoValido:
        pop edi          ; Si el caracter leido no es ninguno de los que
        pop edx          ; tenemos en el BMP no se hace nada y se va a
        inc edx          ; buscar el siguiente caracter
        jmp .Inicio
; Una vez obtenido el caracter y la posicion que ocupa en el BMP se pinta
; en pantalla en unas determinadas coordenadas
.PintaCaracter:
        mov ecx,[ebp+AlturaCaracter_BMP_20]
        mov eax,[ebp+AnchuraCaracter_BMP_20]
        movq2dq xmm6,mm0 ; Copia de coordenadas antes de pintar caracter
.PintaCaracterEjeY:
        push ecx
        push edi
        movq2dq xmm4,mm0 ; Copia de coordenadas antes de entrar en el eje x
        mov ecx,eax
.PintaCaracterEjeX:
        push ecx
        movd mm1,[esi]   ; mm1=*|*RGB
        psllq mm1,40     ; mm1=RGB0|0
        psrlq mm1,40     ; mm1=0|0RGB
        movq mm2,mm1     ; mm2=0|0RGB
        pcmpeqd mm2,mm7 ; mm2="F..F" si Pixel=ColorTransparente
                         ; mm2="0..0" si Pixel!=ColorTransparente
        movd ecx,mm2
        cmp ecx,0
        jnz .NoPintarPixel
        movd [gs:edi],mm1 ; Si no coincide el pixel con el color transparente
                          ; pinta el pixel, si coincide con el color
.NoPintarPixel:           ; transparente no pinta nada, es decir, deja el
        add esi,4         ; color del pixel que tengamos en la pantalla
        add edi,4
        paddd mm0,mm4     ; Incremento de X
        movq mm1,mm0      ; mm1=Y|X (coordenadas)
        pcmpgtd mm1,mm6   ; mm1="F..F" si Y<AltoP|"F..F" si X<AnchoP
                          ;     "0..0" si Y>AltoP|"0..0" si X>AnchoP
        movd ecx,mm1      ; Comprobacion de X
        cmp ecx,0
        jz .NoHayMasIncrementos
        ; Si X>AnchoPantalla hay que actulizar X que sera igual
        ; a X-AnchoPantalla
        movq mm2,mm6      ; mm2=(AltoPantalla-1)|(AnchoPantalla-1)
        psllq mm2,32      ; mm2=(AnchoPantalla-1)|0
```

```
3095          psrlq mm2,32      ; mm2=0|(AnchoPantalla-1)
              psubd mm0,mm2     ; mm0=0|X-(AnchoPantalla-1)
              psubd mm0,mm4     ; mm0=0|X-AnchoPantalla
              movd ecx,xmm2
              add edi,ecx       ; Actualizacion del puntero
3100          ; Como X>AnchoPantalla hay que actualizar Y que sera igual
              ; a Y+AlturaCaracter
              movq mm1,mm3      ; mm1=AlturaCaracter|AnchuraCaracter
              psrlq mm1,32      ; mm1=0|AlturaCaracter
              psllq mm1,32      ; mm1=AlturaCaracter|0
3105          paddd mm0,mm1     ; mm0=Y+AltoCaracter|X
              movq mm1,mm0      ; mm1=Y|X
              pcmpgtd mm1,mm6   ; mm1="F..F" si Y<AltoP|"F..F" si X<AnchoP
                                ;     "0..0" si Y>AltoP|"0..0" si X>AnchoP
              psrlq mm1,32
3110          movd ecx,mm1      ; Comprobacion de Y
              cmp ecx,0
              jz .NoHayMasIncrementos
              ; Si Y>AltoPantalla hay que actualizar Y que sera igual
              ; a Y-AltoPantalla
3115          movq mm1,mm6      ; mm1=(AltoPantalla-1)|(AnchoPantalla-1)
              psrlq mm1,32      ; mm1=0|(AltoPantalla-1)
              psllq mm1,32      ; mm1=(AltoPantalla-1)|0
              psubd mm0,mm1     ; mm0=Y-(AltoPantalla-1)|0
              psubd mm0,mm5     ; mm0=Y-AltoPantalla|0
3120          movd ecx,xmm3
              sub edi,ecx       ; Actualizacion del puntero
    .NoHayMasIncrementos:
              pop ecx
              dec ecx
3125          cmp ecx,0
              jnz .PintaCaracterEjeX
              movdq2q mm0,xmm4  ; mm0=Y|X antes de entrar en el bucle X
              paddd mm0,mm5     ; Incremento Y
              pop edi
3130          movd ecx,xmm7
              add edi,ecx       ; Actualizacion del puntero
              movd ecx,xmm5
              add esi,ecx       ; Actualizacion del puntero
              movq mm1,mm0      ; Compruebo si Y se sale de la pantalla
3135          pcmpgtd mm1,mm6   ; mm1="F..F" si Y<AltoP|"F..F" si X<AnchoP
                                ;     "0..0" si Y>AltoP|"0..0" si X>AnchoP
              psrlq mm1,32
              movd ecx,mm1
              cmp ecx,0
3140          jz .NoHayIncrementoDeY
              ; Si Y>AltoPantalla hay que actualizar Y que sera igual
              ; a Y-AltoPantalla
              movq mm1,mm6
              psrlq mm1,32
3145          psllq mm1,32
              psubd mm0,mm1
              psubd mm0,mm5
              movd ecx,xmm3
              sub edi,ecx       ; Actualizacion del puntero
3150  .NoHayIncrementoDeY:
              pop ecx
              dec ecx
              cmp ecx,0
              jnz .PintaCaracterEjeY
3155          movdq2q mm0,xmm6  ; mm0=Y|X antes de pintar un caracter
              ; Hay que incrementar la coordenada X, siendo la nueva X igual
              ; a X+AnchuraCaracter
              movq mm1,mm3      ; mm1=AlturaCaracter|AnchuraCaracter
              psllq mm1,32      ; mm1=AnchuraCaracter|0
3160          psrlq mm1,32      ; mm1=0|AnchuraCaracter
              paddd mm0,mm1     ; mm0=Y|X+AnchuraCaracter
              pop edi
              movd ecx,xmm0
              add edi,ecx       ; Actualizacion del puntero
3165          movq mm1,mm0      ; Compruebo si X>AnchoPantalla
              pcmpgtd mm1,mm6
              movd ecx,mm1
              cmp ecx,0
```

```
        jz .NoHayMasIncrementos2
        ; Si X>AnchoPantalla hay que actualizar X que sera igual
        ; a X-AnchoPantalla
        movq mm2,mm6
        psllq mm2,32      ; mm2=(AnchoPantalla-1)|0
        psrlq mm2,32      ; mm2=0|(AnchoPantalla-1)
        psubd mm0,mm2     ; mm0=0|X-(AnchoPantalla-1)
        psubd mm0,mm4     ; mm0=0|X-AnchoPantalla
        movd ecx,xmm2
        add edi,ecx       ; Actualizacion del puntero
        ; Como X>AnchoPantalla hay que actualizar Y que sera igual
        ; a Y+AlturaCaracter
        movq mm1,mm3      ; mm1=AlturaCaracter|AnchuraCaracter
        psrlq mm1,32      ; mm1=0|AlturaCaracter
        psllq mm1,32      ; mm1=AlturaCaracter|0
        paddd mm0,mm1     ; mm0=Y+AlturaCaracter|X
        movq mm1,mm0      ; Compruebo si Y>AltoPantalla
        pcmpgtd mm1,mm6
        psrlq mm1,32
        movd ecx,mm1
        cmp ecx,0
        jz .NoHayMasIncrementos2
        ; Si Y>AltoPantalla hay que actualizar Y que sera igual
        ; a Y-AltoPantalla
        movq mm1,mm6      ; mm2=(AltoPantalla-1)|(AnchoPantalla-1)
        psrlq mm1,32      ; mm2=0|(AltoPantalla-1)
        psllq mm1,32      ; mm2=(AltoPantalla-1)|0
        psubd mm0,mm1     ; mm0=Y-(AltoPantalla-1)|0
        psubd mm0,mm5     ; mm0=Y-AltoPantalla|0
        movd ecx,xmm3
        sub edi,ecx       ; Actualizacion del puntero
.NoHayMasIncrementos2:
        pop edx
        inc edx
        jmp .Inicio       ; Vamos a por el siguiente caracter

.SalirCompletamenteDeEscribe:
        pop gs  ; Recuperamos de la pila el selector de segmento gs
        popad
        pop ebp ; Epilogo
        ret

_ASM_Corrige_Aberracion:
; Dada una imagen aberrada, es decir, una imagen que presenta una deformacion
; geometrica, es corregida dicha aberracion obteniendose una Imagen Destino.
; Para corregir dicha aberracion que puede ser positiva (aumento de la imagen
; con la distancia al eje) o negativa (disminucion de la imagen con la
; distancia al eje) se implementan las siguientes ecuaciones:
;        -- Xu=[1+(K*(r*r))]*Xd
;        -- Yu=[1+(K*(r*r))]*Yd  donde r=sqrt[(Xd*Xd)+(Yd*Yd)]
; siendo:
;        -- (Xd,Yd) las coordenadas distorsionadas o aberradas
;        -- (Xu,Yu) las coordenadas del punto corregido
;        -- K el factor de distorsion
Dir_Orig_21 equ  8
Dir_Dest_21 equ 12
Ancho_21    equ 16
Alto_21     equ 20
Factor_21   equ 24
        push ebp
        mov ebp,esp ; Prologo
        pushad
        ;Inicializacion de registros
        mov ebx,[ebp+Ancho_21] ; ebx=Ancho
        pxor xmm1,xmm1
        cvtsi2ss xmm1,ebx      ; xmm1=0|0|0|Ancho (pfsp)
        mov ecx,[ebp+Alto_21]  ; ecx=Alto
        pxor xmm2,xmm2
        cvtsi2ss xmm2,ecx      ; xmm2=0|0|0|Alto (pfsp)
        mov eax,2              ; eax=2 (calculo del punto medio de la imagen)
        pxor xmm3,xmm3
        cvtsi2ss xmm3,eax      ; xmm3=0|0|0|2 (pfsp)
        divss xmm1,xmm3        ; xmm1=0|0|0|(Ancho/2)=X0 (pfsp)
        divss xmm2,xmm3        ; xmm2=0|0|0|(Alto/2)=Y0 (pfsp)
```

```
            pxor xmm5,xmm5          ; xmm5=0|0|0|0
            subps xmm5,xmm2         ; xmm5=0|0|0|-Y0
3245        pslldq xmm5,4           ; xmm5=0|0|-Y0|0
            addps xmm5,xmm1         ; xmm5=0|0|-Y0|X0
            pshufd xmm1,xmm1,44h    ; xmm1=0|X0|0|X0
            pshufd xmm2,xmm2,11h    ; xmm2=Y0|0|Y0|0
            pxor xmm0,xmm0          ; xmm0=0|0|0|0
3250        subps xmm0,xmm1         ; xmm0=0|-X0|0|-X0
            movups xmm7,xmm0        ; xmm7=0|-X0|0|-X0
            subps xmm0,xmm2         ; xmm0=-Y0|-X0|-Y0|-X0
            movsd xmm1,[ebp+Factor_21] ; xmm1=*|K (pfdp)
            pxor xmm2,xmm2
3255        cvtsd2ss xmm2,xmm1      ; xmm2=0|0|0|K (pfsp)
            pshufd xmm2,xmm2,50h    ; xmm2=0|0|K|K (pfsp)
            cvtdq2ps xmm3,[MASC_DISTORSION_SUMA] ; xmm3=0|0|1|1 (pfsp)
            cvtdq2ps xmm4,[MASC_DISTORSION_INC]  ; xmm4=0|1|0|1 (pfsp)
            movdqu xmm6,[MASC_DISTORSION_AND]    ; xmm6=F..F|0..0|F..F|0..0
3260        dec ebx         ; ebx=Ancho-1
            movd mm4,ebx    ; mm4=0|Ancho-1
            inc ebx         ; ebx=Ancho -> Bucle X
            mov edx,ecx     ; edx=Alto
            neg edx         ; edx=-Alto
3265        inc edx         ; edx=1-Alto
            movd mm5,edx    ; mm5=0|(1-Alto)
            psllq mm5,32    ; mm5=(1-Alto)|0
            mov eax,ebx     ; eax=Ancho
            mul ecx         ; edxeax=Ancho*Alto
3270        inc ecx         ; ecx=Alto+1 -> Bucle Y
            shl eax,2       ; eax=Ancho*Alto*4=Desplazamiento edi
            mov edi,eax
            add edi,[ebp+Dir_Dest_21] ; Voy a recorrer la imagen de izquierda
                                      ; a derecha y de abajo a arriba
3275        mov eax,ebx     ; eax=Ancho
            shl eax,3       ; eax=Ancho*4*2 -> Salto de linea
            mov edx,ebx     ; edx=Ancho
            neg edx         ; edx=-Ancho
            movd mm1,edx    ; mm1=0|-Ancho
3280        pslld mm1,16    ; mm1=0||-Ancho|0
            mov edx,1       ; edx=1
            movd mm7,edx    ; mm7=0|1
            paddw mm1,mm7   ; mm1=0||-Ancho|1
            movd mm2,[ebp+Dir_Orig_21] ; mm2=0|Dir_Orig (Imagen Original)
3285        pxor mm3,mm3    ; mm3=0|0 (entremezclado)
    .AberracionEjeY:
            push ecx
            mov ecx,ebx
    .AberracionEjeX:
3290        movups xmm1,xmm0    ; xmm1=Y|X|Y|X (coordenadas)
            mulps xmm1,xmm1     ; xmm1=Y*Y|X*X|Y*Y|X*X
            haddps xmm1,xmm1    ; xmm1=r|r|r|r  r=(Y*Y)+(X*X)
            mulps xmm1,xmm2     ; xmm1=0|0|r*K|r*K
            addps xmm1,xmm3     ; xmm1=0|0|1+(r*K)|1+(r*K)
3295        mulps xmm1,xmm0     ; xmm1=0|0|Yd+Y0|Xd-X0  Yd=[1+(r*K)]*Y  Xd=[1+(r*K)]*X
            addps xmm1,xmm5     ; xmm1=0|0|Yd|Xd
            cvtps2pi mm0,xmm1   ; mm0=Yd|Xd (enteros 32 bits)
            movq mm7,mm0        ; Las coordenadas obtenidas para los puntos a
            pcmpgtd mm7,mm4     ; pintar deben de estar comprendidos:
3300        movq mm6,mm7        ;           (-Alto)<=Y<=0
            psrlq mm6,32        ;                 0<=X<=Ancho
            por mm7,mm6
            movq mm6,mm5
            pcmpgtd mm6,mm0
3305        por mm7,mm6
            psrlq mm6,32
            por mm7,mm6
            movd edx,mm7        ; Si las coordenadas obtenidas no son validas
            cmp edx,0           ; tendremos todo unos en mm7, con lo que deberemos
3310        neg edx             ; el pixel negro edx=0
            jnz .AberracionPintarPixelNegro
            packssdw mm0,mm3    ; mm0=0||Yd|Xd (enteros 16 bits)
            pmaddwd mm0,mm1     ; mm0=0|[Yd*(-Ancho)]+Xd
            pslld mm0,2         ; mm0=0|[[Yd*(-Ancho)]+Xd]*4=Desplaz
3315        paddd mm0,mm2       ; mm0=0|Desplaz+Dir_Orig
            movd esi,mm0        ; esi -> Desplaz+Dir_Orig
```

```
        mov edx,[esi]
.AberracionPintarPixelNegro:
        mov [edi],edx
        add edi,4
        addps xmm0,xmm4   ; Incremento de x
        loop .AberracionEjeX
        andps xmm0,xmm6   ; Deja pasar solo la componente y
        pslldq xmm4,4     ; xmm4=1|0|1|0 (pfsp)
        addps xmm0,xmm4   ; Incremento de la y
        psrldq xmm4,4     ; xmm4=0|1|0|1 (pfsp)
        addps xmm0,xmm7   ; Inicializacion de la x para el nuevo y
        sub edi,eax       ; Actualizacion del puntero de la Imagen Destino
        pop ecx
        loop .AberracionEjeY
        popad
        pop ebp
        ret

_ASM_FFT2D:
; Dadas tres tablas, una para cada componente de color, realiza la
; Transformada de Fourier directa (si la Opcion es 0) o la Transformada
; Inversa (si la opcion es 1) de dichas tablas
Dir_Tabla_Rojo_22  equ  8
Dir_Tabla_Verde_22 equ 12
Dir_Tabla_Azul_22  equ 16
Ancho_FFT_22       equ 20
Opcion_22          equ 24
        push ebp
        mov ebp,esp ; Prologo
        pushad
        mov eax,[ebp+Ancho_FFT_22] ; eax=AnchoFFT
        mov ebx,eax     ; ebx=AnchoFFT
        shl ebx,3       ; ebx=AnchoFFT*4*2 -> Salto de linea
        mov ecx,eax     ; ecx=AnchoFFT
        mov edx,eax     ; edx=AnchoFFT
        movq mm7,[MASC_FFT_Fila_1] ; mm7=0|2 (enteros); Para trabajar con la FFT
        movq mm6,[MASC_FFT_Fila_2] ; mm6=0|1 (enteros); Para trabajar con la FFT
; Primeramente se hace la Transformada de Fourier directa o inversa para
; cada fila de cada tabla de componentes, lo que se realiza en dos pasos,
; el primero haciendo lo que se llama el bit reversal, que es la colocacion
; de los datos para la realizacion posterior de la Transformada de Fourier
; directa o inversa
; Hacemos la Transformada de Fourier directa o inversa para la componente roja
        mov esi,[ebp+Dir_Tabla_Rojo_22] ; esi -> TABLA_ROJO
.Bit_Reversal_Rojo_Fila:
        call .Bit_Reversal_Fila
        call .FFT_Fila
        add esi,ebx
        loop .Bit_Reversal_Rojo_Fila
        mov ecx,eax       ; ecx=AnchoFFT -> Bucle
; Hacemos la Transformada de Fourier directa o inversa para la componente verde
        mov esi,[ebp+Dir_Tabla_Verde_22] ; esi -> TABLA_VERDE
.Bit_Reversal_Verde_Fila:
        call .Bit_Reversal_Fila
        call .FFT_Fila
        add esi,ebx
        loop .Bit_Reversal_Verde_Fila
        mov ecx,eax       ; ecx=AnchoFFT -> Bucle
; Hacemos la Transformada de Fourier directa o inversa para la componente azul
        mov esi,[ebp+Dir_Tabla_Azul_22] ; esi -> TABLA_AZUL
.Bit_Reversal_Azul_Fila:
        call .Bit_Reversal_Fila
        call .FFT_Fila
        add esi,ebx
        loop .Bit_Reversal_Azul_Fila
; Tengo en eax=AnchoFFT (me da igual para los calculos posteriores), en
; ebx=AnchoFFT*4*2 -> Salto de linea
; Realizo un cambio de columnas por filas para hacer la FFT unidimensional
; de cada columna, pero empleando el metodo utilizado anteriormente que
; consistia en hacer la FFT unidimensional de cada fila
        mov ecx,eax       ; ecx=AnchoFFT -> Bucle Y
        mov edx,eax       ; edx=AnchoFFT -> Bucle X
        mov esi,[ebp+Dir_Tabla_Rojo_22] ; esi -> TABLA_ROJO
        call .Cambia_Columnas_Por_Filas
```

```
            mov esi,[ebp+Dir_Tabla_Verde_22]; esi -> TABLA_VERDE
            call .Cambia_Columnas_Por_Filas
            mov esi,[ebp+Dir_Tabla_Azul_22] ; esi -> TABLA_AZUL
            call .Cambia_Columnas_Por_Filas
; Tengo en eax=ecx=edx=AnchoFFT y en ebx=AnchoFFT*4*2 -> Salto de linea
; Hacemos la Transformada de Fourier directa o inversa para la componente roja
            mov esi,[ebp+Dir_Tabla_Rojo_22] ; esi -> TABLA_ROJO
.Bit_Reversal_Rojo_Columna:
            call .Bit_Reversal_Fila
            call .FFT_Fila
            add esi,ebx
            loop .Bit_Reversal_Rojo_Columna
            mov ecx,eax ; ecx=AnchoFFT
; Hacemos la Transformada de Fourier directa o inversa para la componente verde
            mov esi,[ebp+Dir_Tabla_Verde_22] ; esi -> TABLA_VERDE
.Bit_Reversal_Verde_Columna:
            call .Bit_Reversal_Fila
            call .FFT_Fila
            add esi,ebx
            loop .Bit_Reversal_Verde_Columna
            mov ecx,eax ; ecx=AnchoFFT
; Hacemos la Transformada de Fourier directa o inversa para la componente azul
            mov esi,[ebp+Dir_Tabla_Azul_22] ; esi -> TABLA_AZUL
.Bit_Reversal_Azul_Columna:
            call .Bit_Reversal_Fila
            call .FFT_Fila
            add esi,ebx
            loop .Bit_Reversal_Azul_Columna
; Tengo en eax=edx=AnchoFFT y en ebx=AnchoFFT*4*2 -> Salto de linea
            mov ecx,eax       ; ecx=AnchoFFT
; Tengo que deshacer el cambio de columnas por filas hecho anteriormente
            mov esi,[ebp+Dir_Tabla_Rojo_22] ; esi -> TABLA_ROJO
            call .Cambia_Columnas_Por_Filas
            mov esi,[ebp+Dir_Tabla_Verde_22]; esi -> TABLA_VERDE
            call .Cambia_Columnas_Por_Filas
            mov esi,[ebp+Dir_Tabla_Azul_22] ; esi -> TABLA_AZUL
            call .Cambia_Columnas_Por_Filas
            popad
            pop ebp ; Epilogo
            ret

.Bit_Reversal_Fila:
; Realiza el bit reversal de un conjunto de datos, en este caso de una fila,
; los parametros de entrada son esi que apunta al primer dato de la fila y eax
; que contenga el tamano de la fila o el numero de datos a tratar
            pushad      ; Guardamos todos los registros
            shr eax,1   ; eax=AnchoFFT/2 -> eax=mtmp
            xor ecx,ecx ; ecx=0=j
            xor ebx,ebx ; ebx=0=i
.Comienzo_For:
            cmp ecx,ebx
            jbe .NoHacer_If ; si (j<=i) salta
            movq xmm0,[esi+(ecx*8)]
            movq xmm1,[esi+(ebx*8)]  ; Intercambio de partes reales e imaginarias
            movq [esi+(ecx*8)],xmm1  ; entre posiciones de la tabla
            movq [esi+(ebx*8)],xmm0
.NoHacer_If:
            mov edx,eax  ; (edx=k) -> edx=mtmp
.Comienzo_While:
            cmp edx,ecx        ; Mientras (k<=j y k>0) hacer
            jg .NoHacer_While
            cmp edx,0
            jbe .NoHacer_While ; (k>0)
            sub ecx,edx        ; (j=j-k)
            shr edx,1          ; (k=k/2)
            jmp .Comienzo_While
.NoHacer_While:
            add ecx,edx  ; (j=j+m)
            inc ebx
            cmp ebx,[ebp+Ancho_FFT_22] ; Repetir si (i<AnchoFFT)
            jb .Comienzo_For
            popad  ; Recuperamos de la pila los registros
            ret
```

```
3465 .FFT_Fila:
     ; Calcula la FFT de un conjunto de datos, en nuestro caso el conjunto de
     ; datos sera una fila de datos, el dato de entrada es esi que apunta al
     ; primer dato del conjunto de datos. El desarrollo de la FFT se realiza
     ; a traves de la implementacion del algoritmo de Cooley-Tukey (algoritmo
3470 ; de decimacion en frecuencia)
            pushad ; Guardamos todos los registros
            cvtdq2ps xmm0,[MASC_FFT_PHASE_REAL] ; xmm0 -> phase.real (C1)
            pxor xmm1,xmm1 ; xmm1 -> phase.imag (C2)
            mov dword [MMAX],1  ; Inicializacion de la variable MMAX
3475        mov dword [ISTEP],2 ; Inicializacion de la variable ISTEP
     .Comienzo_While_FFT:
            cmp dword eax,[MMAX]    ; Mientras (MMAX<AnchoFFT) hacer
            jbe near .Salida_De_FFT
            cvtdq2ps xmm2,[MASC_FFT_TWIDDLE.REAL] ; xmm2 -> twiddle.real (U1)
3480        pxor xmm3,xmm3 ; xmm3 -> twiddle.imag (U2)
            mov ebx,0      ; ebx=m
     .Comienzo_For_1:
            mov ecx,ebx    ; ecx=i (i=m)
     .Comienzo_For_2:
3485        mov edx,ecx    ; edx=j  (j=i)
            add dword edx,[MMAX] ; (j=i+mmax)
            movss xmm4,[esi+(edx*8)]    ; xmm4 -> data[j].real
            movss xmm5,xmm4             ; xmm5 -> data[j].real
            movss xmm6,[esi+(edx*8)+4] ; xmm6 -> data[j].imag
3490        movss xmm7,xmm6             ; xmm7 -> data[j].imag
            mulss xmm4,xmm2             ; xmm4 -> data[j].real * U1
            mulss xmm6,xmm3             ; xmm6 -> data[j].imag * U2
            mulss xmm5,xmm3             ; xmm5 -> data[j].real * U2
            mulss xmm7,xmm2             ; xmm7 -> data[j].imag * U1
3495        subss xmm4,xmm6             ; xmm4 -> temp1 o t1 o temp.real
            addss xmm5,xmm7             ; xmm5 -> temp2 o t2 o temp.imag
            movss xmm6,[esi+(ecx*8)]    ; xmm6 -> data[i].real
            movss xmm7,[esi+(ecx*8)+4] ; xmm7 -> data[i].imag
            subss xmm6,xmm4             ; xmm6 -> data[i].real - temp.real
3500        subss xmm7,xmm5             ; xmm7 -> data[i].imag - temp.imag
            movss [esi+(edx*8)],xmm6    ; data[j].real=data[i].real-temp.real
            movss [esi+(edx*8)+4],xmm7 ; data[j].imag=data[i].imag-temp.imag
            addss xmm4,[esi+(ecx*8)]    ; xmm4 -> temp.real+data[i].real
            movss [esi+(ecx*8)],xmm4    ; data[i].real=data[i].real+temp.real
3505        addss xmm5,[esi+(ecx*8)+4] ; xmm5 -> temp.imag+data[i].imag
            movss [esi+(ecx*8)+4],xmm5 ; data[i].imag=data[i].imag+temp.imag
            add dword ecx,[ISTEP]       ; (i=i+ISTEP)
            cmp ecx,[ebp+Ancho_FFT_22] ; Repetir si (i<AnchoFFT)
            jb .Comienzo_For_2
3510        movss xmm4,xmm3             ; xmm4 -> U2
            movss xmm5,xmm3             ; xmm5 -> U2
            movss xmm3,xmm2             ; xmm3 -> U1
            mulss xmm2,xmm0             ; xmm2 -> U1 * C1
            mulss xmm3,xmm1             ; xmm3 -> U1 * C2
3515        mulss xmm4,xmm1             ; xmm4 -> U2 * C2
            mulss xmm5,xmm0             ; xmm5 -> U2 * C1
            subss xmm2,xmm4             ; xmm2 ->(U1*C1)-(U2*C2) -> Nuevo U1
            addss xmm3,xmm5             ; xmm3 ->(U1*C2)+(U2*C1) -> Nuevo U2
            inc ebx                     ; (m++)
3520        cmp dword ebx,[MMAX]        ; Repetir si (m<MMAX)
            jb .Comienzo_For_1
            cvtpi2ps xmm1,mm6           ; xmm1=0|0|0|1 (pfsp)
            movss xmm4,xmm1             ; xmm4=0|0|0|1 (pfsp)
            subss xmm1,xmm0             ; xmm1 -> 1.0-C1
3525        addss xmm0,xmm4             ; xmm0 -> 1.0+C1
            cvtpi2ps xmm4,mm7           ; xmm4=0|0|0|2 (pfsp)
            divss xmm0,xmm4             ; xmm0 -> (1.0+C1)/2
            divss xmm1,xmm4             ; xmm1 -> (1.0-C1)/2
            sqrtss xmm0,xmm0            ; xmm0 -> Nuevo C1
3530        sqrtss xmm1,xmm1            ; xmm1 -> Nuevo C2
            cmp dword [ebp+Opcion_22],0; Si Opcion=1 (transformada inversa)
            jz .NoInversa1              ; Nuevo C2= -C2
            cvtdq2ps xmm4,[MASC_FFT_PHASE_REAL] ; xmm4=0|0|0|-1 (pfsp)
            mulps xmm1,xmm4             ; xmm1 -> C2*(-1) -> Nuevo C2
3535 .NoInversa1:
            shl dword [MMAX],1          ; (mmax=2*mmax)
            shl dword [ISTEP],1         ; (istep=2*istep)
            jmp .Comienzo_While_FFT
```

```
    .Salida_De_FFT:
3540        cmp dword [ebp+Opcion_22],0
            jz .NoInversa2
    ; Si Opcion=1 (transformada inversa) hay que escalar los valores obtenidos
            mov eax,[ebp+Ancho_FFT_22] ; eax=AnchoFFT
            cvtsi2ss xmm0,eax ; xmm0=*|*|*|AnchoFFT=A (pfsp)
3545        pshufd xmm0,xmm0,0h ; xmm0=A|A|A|A
            mov ecx,eax  ; ecx=AnchoFFT
            xor edx,edx  ; edx -> Para recorrer la tabla de datos
    .Escalado_Inversa:
            movq xmm1,[esi+(edx*8)] ; xmm1=A|B|C|D siendo A,B,C y D datos en pfsp
3550        divps xmm1,xmm0 ; xmm1=A/W|B/W|C/W|D/W siendo W=AnchoFFT
            movq [esi+(edx*8)],xmm1 ; Lo metemos en memoria
            inc edx
            loop .Escalado_Inversa
    .NoInversa2:
3555        popad  ; Recuperamos los registros
            ret

    .Cambia_Columnas_Por_Filas:
    ; Dada una matriz de datos, cambia las columnas por filas siendo los
3560 ; parametros de entrada:
    ;        -- esi -> apunta al primer dato de la matriz
    ;        -- ecx -> bucle Y
    ;        -- ebx -> salto de linea
    ;        -- edx -> bucle X
3565        pushad ; guardamos los registros
    .Cambia_Columna_Por_Fila_EjeY:
            push ecx
            mov ecx,edx
            xor eax,eax
3570        mov edi,esi
    .Cambia_Columna_Por_Fila_EjeX:
            movq xmm0,[esi+(eax*8)]
            movq xmm1,[edi]
            movq [esi+(eax*8)],xmm1
3575        movq [edi],xmm0
            inc eax
            add edi,ebx ; Incrementamos el puntero
            loop .Cambia_Columna_Por_Fila_EjeX
            add esi,ebx ; Incrementamos el puntero
3580        add esi,8
            dec edx
            pop ecx
            loop .Cambia_Columna_Por_Fila_EjeY
            popad ; Recuperamos los registros
3585        ret

    _ASM_Crea_Tablas_Imagen_Real:
    ; Crea a partir de una Imagen Real Origen tres tablas con parte real y parte
    ; imaginaria, una para cada componente de color, que contienen respectivamente
3590 ; el valor en pfsp de la Imagen Origen en su parte real y su parte imaginaria
    ; nula
    Dir_Orig_23        equ  8
    Dir_Tabla_Rojo_23  equ 12
    Dir_Tabla_Verde_23 equ 16
3595 Dir_Tabla_Azul_23  equ 20
    Ancho_FFT_23       equ 24
            push ebp
            mov ebp,esp ; Prologo
            pushad
3600        mov eax,[ebp+Ancho_FFT_23] ; eax=AnchoFFT
            pxor mm0,mm0
            movd mm0,eax    ; mm0=0|eax=0|AnchoFFT -> Bucle X
            mov ebx,eax     ; ebx=AnchoFFT
            shl ebx,3       ; ebx=AnchoFFT*4*2 -> Salto de linea
3605        pxor mm1,mm1
            movd mm1,ebx    ; mm1=0|Salto de linea
            mov ecx,eax     ; ecx=AnchoFFT
            dec eax         ; eax=AnchoFFT-1
            mul ecx         ; edxeax=AnchoFFT*(AnchoFFT-1)
3610        shl eax,2       ; eax=AnchoFFT*(AnchoFFT-1)*4
            mov esi,eax
            add esi,[ebp+Dir_Orig_23] ; esi->Imagen Original
```

```
        mov edi,[ebp+Dir_Tabla_Rojo_23] ; edi -> Tabla_Roja
        mov eax,[ebp+Dir_Tabla_Verde_23]; eax -> Tabla_Verde
        mov ebx,[ebp+Dir_Tabla_Azul_23] ; ebx -> Tabla_Azul
        xor edx,edx
        pxor xmm7,xmm7 ; xmm7=0 -> Para entremezclado
        movdqu xmm6,[MASC_FFT_AND_CREA_TABLAS] ; Solo dejara pasar el pixel (RGB)
.Crea_Tablas_EjeY:
        push ecx
        movd ecx,mm0
.Crea_Tablas_EjeX:
        movd xmm0,[esi]      ; xmm0=*|*|*|*RGB
        pand xmm0,xmm6       ; xmm0=0|0|0|0RGB
        punpcklbw xmm0,xmm7 ; xmm0=0|0|0R0G0B
        punpcklwd xmm0,xmm7 ; xmm0=0|R|G|B (enteros 32 bits)
        cvtdq2ps xmm0,xmm0  ; xmm0=0|R|G|B (pfsp)
        pshufd xmm1,xmm0,0fdh ; xmm1=0|0|0|G (pfsp)
        pshufd xmm2,xmm0,0fch ; xmm2=0|0|0|B (pfsp)
        psrldq xmm0,8        ; xmm0=0|0|0|R (pfsp)
        movq [edi+(edx*8)],xmm0 ; Tabla Rojo <- xmm0
        movq [eax+(edx*8)],xmm1 ; Tabla Verde <- xmm1
        movq [ebx+(edx*8)],xmm2 ; Tabla Azul <- xmm2
        inc edx
        add esi,4
        loop .Crea_Tablas_EjeX
        movd ecx,mm1
        sub esi,ecx ; Actualizacion del puntero de Imagen Origen
        pop ecx
        loop .Crea_Tablas_EjeY
        popad
        pop ebp ; Epilogo
        ret

_ASM_Calcula_Modulo_Y_Maximo:
; Dadas tres tablas, una para cada componente de color, calcula el modulo
; de cada tabla de color y la introduce en una nueva tabla Tabla_Modulo,
; es decir, esta ultima Tabla_Modulo contiene el modulo que presenta cada
; componente de color, ademas se calcula tambien el maximo de todos los
; modulos y es introducido en la variable MAXIMO, para posteriores calculos
Dir_Tabla_Modulo_24 equ  8
Dir_Tabla_Rojo_24   equ 12
Dir_Tabla_Verde_24  equ 16
Dir_Tabla_Azul_24   equ 20
Ancho_FFT_24        equ 24
        push ebp
        mov ebp,esp ; Prologo
        pushad
        mov eax,[ebp+Ancho_FFT_24] ; eax=AnchoFFT
        mov ecx,eax  ; ecx=AnchoFFT
        mul ecx      ; edxeax=AnchoFFT*AnchoFFT
        mov ecx,eax  ; ecx=Ancho*Ancho -> Bucle
        pxor xmm6,xmm6 ; xmm6 -> Llevara el maximo
        mov edi,[ebp+Dir_Tabla_Modulo_24]; edi -> Tabla Modulo
        mov esi,[ebp+Dir_Tabla_Rojo_24]  ; esi -> Tabla Rojo
        mov eax,[ebp+Dir_Tabla_Verde_24] ; eax -> Tabla Verde
        mov ebx,[ebp+Dir_Tabla_Azul_24]  ; ebx -> Tabla Azul
        xor edx,edx  ; edx -> Para recorrer las tablas
.Calcular_Modulo_Y_Buscar_Maximo:
        movq xmm0,[esi+(edx*8)] ; xmm0=0|0|Im|Re (R)
        movq xmm1,[eax+(edx*8)] ; xmm1=0|0|Im|Re (G)
        movq xmm2,[ebx+(edx*8)] ; xmm2=0|0|Im|Re (B)
        mulps xmm0,xmm0     ; xmm0=0|0|Im2|Re2 (R) siendo 2=cuadrado
        mulps xmm1,xmm1     ; xmm1=0|0|Im2|Re2 (G)
        mulps xmm2,xmm2     ; xmm2=0|0|Im2|Re2 (B)
        pslldq xmm1,8       ; xmm1=Im2|Re2|0|0 (G)
        por xmm1,xmm2       ; xmm1=Im2|Re2(G)|Im2|Re2(B)
        haddps xmm1,xmm0    ; xmm1=*|Im2+Re2(B)|Im2+Re2(G)|Im2+Re2(B)
        sqrtps xmm1,xmm1    ; xmm1=*|R|G|B (pfsp)
        movups [edi],xmm1   ; edi <- xmm1
        maxps xmm6,xmm1     ; xmm6 -> Maximo de xmm6 y xmm1
        inc edx
        add edi,16
        loop .Calcular_Modulo_Y_Buscar_Maximo
        ; Tengo el maximo en xmm6=*|max(R)|max(G)|max(B)
        movups xmm5,xmm6 ; xmm5=*|max(R)|max(G)|max(B)
```

```
        movups xmm4,xmm6 ; xmm4=*|max(R)|max(G)|max(B)
        psrldq xmm5,4    ; xmm5=0|*|max(R)|max(G)
        psrldq xmm4,8    ; xmm4=0|0|*|max(R)
        maxss xmm4,xmm5  ; xmm4=*|*|*|max(R),max(G)
        maxss xmm6,xmm4  ; xmm6=*|*|*|max(R),max(G),max(B)=maximo=M
        movd [MAXIMO_FFT],xmm6 ; xmm6 -> [MAXIMO_FFT]
        popad
        pop ebp ; Epilogo
        ret

_ASM_Calcula_Normalizacion:
; Dada una Tabla que puede contener el modulo, la parte real o la parte
; imaginaria para cada componente de color, se calcula la representacion
; normalizada de dicha tabla, y es representada en una imagen destino
; (Dir_FFT), es decir, se implementa la ecuacion:
;                                  255
;   Representacion_Normalizada=--------- * X
;                                MAXIMO
; donde X es el valor de una componente y MAXIMO nos llega a traves de una
; variable. Asi el MAXIMO se correspondera con la maxima representacion
Dir_FFT_25   equ  8
Dir_Tabla_25 equ 12
Ancho_FFT_25 equ 16
        push ebp
        mov ebp,esp ; Prologo
        pushad
        movd xmm6,[MAXIMO_FFT] ; xmm6=0|0|0|M siendo M=Maximo
        pshufd xmm6,xmm6,0h ; xmm6=M|M|M|M
        cvtdq2ps xmm5,[MASC_FFT_MAXIMO] ; xmm5=0|255|255|255
        divps xmm5,xmm6 ; xmm5=0|A|A|A siendo A=255/M
        mov eax,[ebp+Ancho_FFT_25] ; eax=AnchoFFT
        mov ebx,eax  ; ebx=AnchoFFT
        shl ebx,3    ; ebx=AnchoFFT*4*2 -> Salto de linea
        mov ecx,eax  ; ecx=AnchoFFT -> Bucle Y
        dec eax      ; eax=(AnchoFFT-1)
        mul ecx      ; edxeax=(AnchoFFT-1)*AnchoFFT
        shl eax,2    ; eax=(AnchoFFT-1)*AnchoFFT*4
        mov edi,eax
        add edi,[ebp+Dir_FFT_25]  ; edi -> Imagen Destino
        mov esi,[ebp+Dir_Tabla_25]; esi -> Tabla Origen
        mov eax,ecx  ; eax=AnchoFFT -> Bucle X
        xor edx,edx  ; edx -> Para recorrer las tablas
        pxor xmm4,xmm4 ; xmm4 -> Para entremezclado
.Modulo_EjeY:
        push ecx
        mov ecx,eax
.Modulo_EjeX:
        movups xmm0,[esi+(edx*8)] ; xmm0=*|R|G|B (pfsp)
        mulps xmm0,xmm5    ; xmm0=*|R/A|G/A|B/A=*|R|G|B normalizado (pfsp)
        cvtps2dq xmm0,xmm0 ; xmm0=*|R|G|B (enteros 32 bits)
        packssdw xmm0,xmm4 ; xmm0=*|*|0*0R0G0B (enteros 16 bits)
        packuswb xmm0,xmm4 ; xmm1=*|*|*|*RGB
        movd [edi],xmm0    ; edi <- xmm0
        add edi,4
        add edx,2
        loop .Modulo_EjeX
        sub edi,ebx        ; Actualizacion de puntero
        pop ecx
        loop .Modulo_EjeY
        popad
        pop ebp ; Epilogo
        ret

_ASM_Calcula_Logaritmo:
; Dada una Tabla que puede contener el modulo, la parte real o la parte
; imaginaria para cada componente de color, se calcula la representacion
; logaritmica de dicha tabla, y es representada en una imagen destino
; (Dir_FFT), es decir, se implementa la ecuacion:
;                                  log (1+|F(X)|)
;   Representacion_Logaritmico=------------------ * 255
;                              log (1+|MAXIMO|)
; donde X es el valor de una componente y MAXIMO nos llega a traves de una
; variable. Asi el log (1+|MAXIMO|) se correspondera con la maxima representacion
Dir_FFT_26   equ  8
```

```
    Dir_Tabla_26 equ 12
    Ancho_FFT_26 equ 16
           push ebp
           mov ebp,esp ; Prologo
3765       pushad
           mov eax,[ebp+Ancho_FFT_26] ; eax=AnchoFFT
           mov ecx,eax  ; ecx=AnchoFFT -> Bucle Y
           mov ebx,eax  ; ebx=AnchoFFT
           shl ebx,3    ; ebx=AnchoFFT*4*2 -> Salto de linea
3770       dec eax      ; eax=(AnchoFFT-1)
           mul ecx      ; edxeax=(AnchoFFT-1)*AnchoFFT
           shl eax,2    ; eax=(AnchoFFT-1)*AnchoFFT*4
           mov edi,eax
           add edi,[ebp+Dir_FFT_26] ; edi -> Imagen Destino
3775       mov esi,[ebp+Dir_Tabla_26] ; esi -> Tabla Origen
           mov eax,ecx  ; eax=AnchoFFT -> Bucle X
           xor edx,edx  ; edx -> Para recorrer las tablas
           finit        ; Inicializacion de la FPU
           ; Calculo del log(10)Maximo=logaritmo en base 10 del maximo
3780       fld1      ; 1
           fld dword [MAXIMO_FFT] ; x,1 siendo x=Modulo_Azul
           fadd st1  ; x+1,1
           fyl2x     ; log(2)(x+1)
           fldl2t    ; log(2)10,log(2)(x+1)
3785       fdivp st1 ; log(10)(x+1)=log(2)(x+1)/log(2)10
           fstp dword [MAXIMO_FFT]
           movss xmm6,[MAXIMO_FFT]
           pshufd xmm6,xmm6,0h ; xmm6=log(M)|log(M)|log(M)|log(M)
           cvtdq2ps xmm5,[MASC_FFT_MAXIMO] ; xmm5=0|255|255|255 (pfsp)
3790       divps xmm5,xmm6 ; xmm5=0|A|A|A siendo A=255/log(M)
           pxor xmm4,xmm4  ; xmm4 -> Entremezclado
    .Logaritmo_EjeY:
           push ecx
           mov ecx,eax
3795  .Logaritmo_EjeX:
           fld1      ; 1
           fld dword [esi+(edx*8)] ; x,1 siendo x=Modulo_Azul
           fadd st1  ; x+1,1
           fyl2x     ; log(2)x
3800       fldl2t    ; log(2)10,log(2)x
           fdivp st1 ; log(10)x=log(2)x/log(2)10
           fstp dword [MODULO_LOG_AZUL]
           fld1      ; 1
           fld dword [esi+(edx*8)+4] ; x,1 siendo x=Modulo_Verde
3805       fadd st1  ; x+1,1
           fyl2x     ; log(2)x
           fldl2t    ; log(2)10,log(2)x
           fdivp st1 ; log(10)x=log(2)x/log(2)10
           fstp dword [MODULO_LOG_VERDE]
3810       fld1      ; 1
           fld dword [esi+(edx*8)+8] ; x,1 siendo x=Modulo_Rojo
           fadd st1  ; x+1,1
           fyl2x     ; log(2)x
           fldl2t    ; log(2)10,log(2)x
3815       fdivp st1 ; log(10)x=log(2)x/log(2)10
           fstp dword [MODULO_LOG_ROJO]
           movss xmm0,[MODULO_LOG_ROJO] ; xmm0=0|0|0|R
           pslldq xmm0,8  ; xmm0=0|R|0|0
           movss xmm1,[MODULO_LOG_VERDE]; xmm1=0|0|0|G
3820       pslldq xmm1,4  ; xmm1=0|0|G|0
           por xmm0,xmm1  ; xmm0=0|R|G|0
           movss xmm1,[MODULO_LOG_AZUL] ; xmm1=0|0|0|B
           por xmm0,xmm1  ; xmm0=0|R|G|B (pfsp)
           mulps xmm0,xmm5    ; xmm0=0|R*A|G*A|B*A (pfsp) siendo A=255/log(M)
3825       cvtps2dq xmm0,xmm0 ; xmm0=0|R|G|B (enteros 32 bits)
           packssdw xmm0,xmm4 ; xmm0=0|0|0R0G0B (enteros 16 bits)
           packuswb xmm0,xmm4 ; xmm0=0|0|0|0RGB
           movd [edi],xmm0    ; edi <- xmm0
           add edi,4
3830       add edx,2
           dec ecx
           cmp ecx,0
           jnz .Logaritmo_EjeX
           sub edi,ebx  ; Actualizacion de puntero
```

```
3835        pop ecx
            dec ecx
            cmp ecx,0
            jnz .Logaritmo_EjeY
            popad
3840        pop ebp ; Epilogo
            ret

    _ASM_Ordena_FFT:
    ; Dada una Imagen (Dir_FFT) cuadrada de tamano Ancho_FFT, lo que se hace
3845 ; es lo siguiente:
    ;             Imagen Original             Imagen Destino
    ;             ---------------             ---------------
    ;           |   1   |   2   |           |   3   |   4   |
    ;             ---------------             ---------------
3850 ;           |   4   |   3   |           |   2   |   1   |
    ;             ---------------             ---------------
    Dir_FFT_27   equ  8
    Ancho_FFT_27 equ 12
            push ebp
3855        mov ebp,esp ; Prologo
            pushad
            mov eax,[ebp+Ancho_FFT_27] ; eax=AnchoFFT
            ; Comprueba el Ancho de la Imagen y segun sea este el tratamiento
            ; sera diferente
3860        cmp eax,2
            jz near .AnchoIgualA2
            cmp eax,4
            jz near .AnchoIgualA4
            mov ebx,eax ; ebx=AnchoFFT
3865        shl ebx,1   ; ebx=AnchoFFT*2 -> Salto de linea
            mov ecx,eax ; ecx=AnchoFFT
            shr ecx,1   ; ecx=AnchoFFT/2 -> Bucle Y
            inc eax     ; eax=(AnchoFFT+1)
            mul ebx     ; edxeax=(AnchoFFT+1)*AnchoFFT*2
3870        mov edi,eax
            add edi,[ebp+Dir_FFT_27] ; edi -> Imagen Destino
            mov esi,[ebp+Dir_FFT_27] ; esi -> Imagen Originen
            mov eax,ecx ; eax=Ancho/2
            mov edx,ecx ; edx=Ancho/2
3875        shr eax,2   ; eax=(Ancho/2)/4 -> Bucle X debido a que el Ancho
                        ; va a ser >=8 y por lo tanto multiplo de 4 con lo
                        ; que cambiaremos los pixeles de 4 en 4
    .Cambia_1_Por_3_EjeY:
            push ecx       ; Intercambia los bloques 1 y 3
3880        mov ecx,eax
    .Cambia_1_Por_3_EjeX:
            movdqu xmm0,[esi]
            movdqu xmm1,[edi]
            movdqu [esi],xmm1
3885        movdqu [edi],xmm0
            add esi,16
            add edi,16
            loop .Cambia_1_Por_3_EjeX
            add esi,ebx
3890        add edi,ebx
            pop ecx
            loop .Cambia_1_Por_3_EjeY
            mov esi,ebx ; esi=AnchoFFT*2
            add esi,[ebp+Dir_FFT_27]
3895        mov ecx,edx ; ecx=AnchoFFT/2
            mov eax,edx ; eax=AnchoFFT/2
            shl eax,1   ; eax=AnchoFFT
            mul ebx     ; edxeax=AnchoFFT*AnchoFFT*2
            mov edi,eax
3900        add edi,[ebp+Dir_FFT_27]
            mov eax,ecx ; eax=AnchoFFT/2
            shr eax,2   ; eax=(AnchoFFT/2)/4
    .Cambia_2_Por_4_EjeY:
            push ecx         ; Intercambia los bloques 2 y 4
3905        mov ecx,eax
    .Cambia_2_Por_4_EjeX:
            movdqu xmm0,[esi]
            movdqu xmm1,[edi]
```

```
        movdqu [esi],xmm1
        movdqu [edi],xmm0
        add esi,16
        add edi,16
        loop .Cambia_2_Por_4_EjeX
        add esi,ebx
        add edi,ebx
        pop ecx
        loop .Cambia_2_Por_4_EjeY
        popad
        pop ebp  ; Epilogo
        ret
.AnchoIgualA2:
        mov esi,[ebp+Dir_FFT_27]  ; Cuando la Imagen cuadrada tiene un
        movd mm0,[esi]            ; Ancho de 2, tenemos la situacion:
        movd mm1,[esi+12]         ;
        movd [esi],mm1            ; (Incial) 1 2 ---->  4 3 (Destino)
        movd [esi+12],mm0         ;          3 4 ---->  1 2
        movd mm0,[esi+4]
        movd mm1,[esi+8]
        movd [esi+8],mm1
        movd [esi+4],mm0
        popad
        pop ebp  ; Epilogo
        ret
.AnchoIgualA4:
        mov esi,[ebp+Dir_FFT_27] ; Caso similar al de Ancho=2 pero ahora
        movq mm0,[esi]           ; tenemos que el Ancho es de 4 pixeles
        movq mm1,[esi+40]
        movq [esi],mm1
        movq [esi+40],mm0
        movq mm0,[esi+16]
        movq mm1,[esi+56]
        movq [esi+16],mm1
        movq [esi+56],mm0
        movq mm0,[esi+8]
        movq mm1,[esi+32]
        movq [esi+8],mm1
        movq [esi+32],mm0
        movq mm0,[esi+24]
        movq mm1,[esi+48]
        movq [esi+24],mm1
        movq [esi+48],mm0
        popad
        pop ebp ; Epilogo
        ret

_ASM_Calcula_Fase_Maximo_Y_Minimo:
; Dadas tres tablas, una para cada componente de color, calcula la
; fase de cada tabla de color y la introduce en una nueva tabla Tabla_Fase,
; es decir, esta ultima Tabla_Fase contiene la fase que presenta cada
; componente de color, ademas se calcula tambien el maximo y el minimo de
; todas las fases y son introducidos en las variables MAXIMO y MINIMO,
; para posteriores calculos
Dir_Tabla_Fase_28  equ  8
Dir_Tabla_Rojo_28  equ 12
Dir_Tabla_Verde_28 equ 16
Dir_Tabla_Azul_28  equ 20
Ancho_FFT_28       equ 24
        push ebp
        mov ebp,esp  ; Prologo
        pushad
        pxor xmm7,xmm7 ; xmm7 -> Llevara el maximo
        cvtdq2ps xmm6,[MASC_FFT_MAXIMO] ; xmm6 -> Llevara el minimo
                                        ; Meto la mascara porque lleve algo
        mov eax,[ebp+Ancho_FFT_28] ; eax=AnchoFFT
        mov ecx,eax  ; ecx=AnchoFFT
        mul ecx      ; edxeax=AnchoFFT*AnchoFFT
        mov ecx,eax  ; ecx=Ancho*Ancho -> Bucle
        mov edi,[ebp+Dir_Tabla_Fase_28] ; edi -> Tabla Fase
        mov esi,[ebp+Dir_Tabla_Rojo_28] ; esi -> Tabla Rojo
        mov eax,[ebp+Dir_Tabla_Verde_28]; eax -> Tabla Verde
        mov ebx,[ebp+Dir_Tabla_Azul_28] ; ebx -> Tabla Azul
        xor edx,edx  ; edx -> Para recorrer la tabla
```

```
        finit ; Inicializacion de la FPU
.Calcular_Fase_Maximo_Y_Minimo:
        fld dword [esi+(edx*8)+4] ; Ir
        fld dword [esi+(edx*8)]   ; Rr,Ir
        fpatan                    ; arctg (Ir/Rr)
        fstp dword [ANGULO]
        movss xmm0,[ANGULO]       ; xmm0=0|0|0|Ar
        pslldq xmm0,8             ; xmm0=0|Ar|0|0
        fld dword [eax+(edx*8)+4] ; Ig
        fld dword [eax+(edx*8)]   ; Rg,Ig
        fpatan                    ; arctg (Ig/Rg)
        fstp dword [ANGULO]
        movss xmm1,[ANGULO]       ; xmm1=0|0|0|Ag
        pslldq xmm1,4             ; xmm1=0|0|Ag|0
        fld dword [ebx+(edx*8)+4] ; Ib
        fld dword [ebx+(edx*8)]   ; Rb,Ib
        fpatan                    ; arctg (Ib/Rb)
        fstp dword [ANGULO]
        movss xmm2,[ANGULO]       ; xmm2=0|0|0|Ab
        orps xmm0,xmm1            ; xmm0=0|Ar|Ag|0
        orps xmm0,xmm2            ; xmm0=0|Ar|Ag|Ab
        movups [edi],xmm0         ; edi <- xmm0
        maxps xmm7,xmm0           ; xmm7 -> Maximo de xmm7 y xmm0
        minps xmm6,xmm0           ; xmm6 -> Minimo de xmm6 y xmm0
        inc edx
        add edi,16
        loop .Calcular_Fase_Maximo_Y_Minimo
        ; Tengo en xmm7=0|MAX(R)|MAX(G)|MAX(B) siendo MAX=Maximo y
        ; en xmm6=0|MIN(R)|MIN(G)|MIN(B) siendo MIN=Minimo
        movups xmm5,xmm7 ; xmm5=0|MAX(R)|MAX(G)|MAX(B)
        movups xmm4,xmm7 ; xmm4=0|MAX(R)|MAX(G)|MAX(B)
        psrldq xmm5,4    ; xmm5=0|0|MAX(R)|MAX(G)
        psrldq xmm4,8    ; xmm4=0|0|0|MAX(R)
        maxss xmm4,xmm5  ; xmm4=*|*|*|MAX(R,G)
        maxss xmm7,xmm4  ; xmm7=*|*|*|MAX(R,G,B)
        movd [MAXIMO_FFT],xmm7 ; MAX(R,G,B) -> [MAXIMO]
        movups xmm5,xmm6 ; xmm5=0|MIN(R)|MIN(G)|MIN(B)
        movups xmm4,xmm6 ; xmm4=0|MIN(R)|MIN(G)|MIN(B)
        psrldq xmm5,4    ; xmm5=0|0|MIN(R)|MIN(G)
        psrldq xmm4,8    ; xmm5=0|0|0|MIN(R)
        minss xmm4,xmm5  ; xmm5=*|*|*|MIN(R,G)
        minss xmm6,xmm4  ; xmm6=*|*|*|MIN(R,G,B)
        movd [MINIMO_FFT],xmm6 ; MIN(R,G,B) -> [MINIMO]
        popad
        pop ebp ; Epilogo
        ret

_ASM_Calcula_Fase_Desnormalizada:
; Dada una Tabla_Fase que contiene las fases para cada componente de color
; se calcula la fase desnormalizada de dicha tabla, y es representada en una
; imagen destino (Dir_FFT), es decir, se implementa la ecuacion:
;                              255
;         Fase_Normalizada=---------------- ( X - MINIMO )
;                          (MAXIMO-MINIMO)
; donde X esta en radianes y hay que pasarla a grados X(GRADOS)=X*(180/PI)
; y siendo el MAXIMO=180 y MINIMO=-180 ambas en grados
Dir_FFT_29        equ  8
Dir_Tabla_Fase_29 equ 12
Ancho_FFT_29      equ 16
        push ebp
        mov ebp,esp ; Prologo
        pushad
        pxor xmm7,xmm7 ; xmm7 -> Para entremezclado
        cvtdq2ps xmm1,[MASC_FFT_FASE_1] ; xmm1=0|360|360|360 (pfsp)
        cvtdq2ps xmm3,[MASC_FFT_MAXIMO] ; xmm3=0|255|255|255 (pfsp)
        divps xmm3,xmm1     ; xmm3=0|A|A|A (pfsp) siendo A=255/360
        movsd xmm4,[MASC_FFT_FASE_PI] ; xmm2=*|PI (pfdp)
        cvtsd2ss xmm4,xmm4  ; xmm2=*|*|*|PI (pfsp)
        pshufd xmm4,xmm4,0h ; xmm2=PI|PI|PI|PI (pfsp)
        cvtdq2ps xmm1,[MASC_FFT_FASE_2] ; xmm1=0|180|180|180 (pfsp)
        movups xmm2,xmm1 ; xmm2=0|180|180|180 (pfsp)
        divps xmm1,xmm4  ; xmm1=0|B|B|B (pfsp) siendo B=180/PI
        mov eax,[ebp+Ancho_FFT_29] ; eax=AnchoFFT
        mov ebx,eax    ; ebx=AnchoFFT
```

```
        shl ebx,3      ; ebx=AnchoFFT*4*2 -> Salto de linea
        mov ecx,eax    ; ecx=AnchoFFT -> Bucle Y
        dec eax        ; eax=(AnchoFFT-1)
        mul ecx        ; edxeax=AnchoFFT*(AnchoFFT-1)
        shl eax,2      ; eax=AnchoFFT*(AnchoFFT-1)*4
        mov edi,eax
        add edi,[ebp+Dir_FFT_29] ; edi -> Imagen Destino
        mov esi,[ebp+Dir_Tabla_Fase_29]
        mov eax,ecx    ; eax=AnchoFFT -> Bucle X
        xor edx,edx    ; edx -> Para recorrer la tabla
.Calcula_Fase_Desnormalizada_EjeY:
        push ecx
        mov ecx,eax
.Calcula_Fase_Desnormalizada_EjeX:
        movups xmm0,[esi+(edx*8)] ; xmm0=*|Ar|Ag|Ab
        mulps xmm0,xmm1   ; xmm0=0|Ar*(180/PI)|Ag*(180/PI)|Ab*(180/PI)
        addps xmm0,xmm2   ; xmm0=0|(Ar*(180/PI))+180|(Ag*(180/PI))+180|
                                |(Ab*(180/PI))+180
        mulps xmm0,xmm3   ; xmm0=0|R|G|B (pfsp)
        cvtps2dq xmm0,xmm0 ; xmm0=0|R|G|B (enteros 32 bits)
        packssdw xmm0,xmm7 ; xmm0=0|0|000R0G0B (enterso 16 bits)
        packuswb xmm0,xmm7 ; xmm0=0|0|0|0RGB
        movd [edi],xmm0    ; Imagen Destino <- xmm0
        add edi,4
        add edx,2
        loop .Calcula_Fase_Desnormalizada_EjeX
        sub edi,ebx  ; Situamos el puntero de la Imagen DEstino
        pop ecx
        loop .Calcula_Fase_Desnormalizada_EjeY
        popad
        pop ebp  ; Epilogo
        ret

_ASM_Calcula_Fase_Normalizada:
; Dada una Tabla_Fase que contiene las fases para cada componente de color
; se calcula la fase normalizada de dicha tabla, y es representada en una
; imagen destino (Dir_FFT), es decir, se implementa la ecuacion:
;                                  255
;         Fase_Normalizada=--------------- ( X - MINIMO )
;                           (MAXIMO-MINIMO)
; donde X esta en radianes y hay que pasarla a grados X(GRADOS)=X*(180/PI)
Dir_FFT_30        equ  8
Dir_Tabla_Fase_30 equ 12
Ancho_FFT_30      equ 16
        push ebp
        mov ebp,esp ; Prologo
        pushad
        pxor xmm7,xmm7 ; xmm7 -> Para entremezclado
        cvtdq2ps xmm1,[MASC_FFT_FASE_1]; xmm1=0|180|180|180 (pfsp)
        movsd xmm2,[MASC_FFT_FASE_PI]  ; xmm2=*|PI (pfdp)
        cvtsd2ss xmm2,xmm2  ; xmm2=*|*|*|PI (pfsp)
        pshufd xmm2,xmm2,0h ; xmm2=PI|PI|PI|PI (pfsp)
        divps xmm1,xmm2     ; xmm1=0|B|B|B (pfsp) siendo B=180/PI
        movss xmm2,[MINIMO_FFT] ; xmm2=*|*|*|m siendo m=minimo
        pshufd xmm2,xmm2,0h ; xmm2=m|m|m|m
        movss xmm4,[MAXIMO_FFT] ; xmm4=*|*|*|M siendo M=Maximo
        pshufd xmm4,xmm4,0h ; xmm4=M|M|M|M
        subps xmm4,xmm2     ; xmm4=M-m|M-m|M-m|M-m
        cvtdq2ps xmm3,[MASC_FFT_MAXIMO] ; xmm3=0|255|255|255 (pfsp)
        divps xmm3,xmm4     ; xmm3=0|255(M-m)|255(M-m)|255(M-m)
        mov eax,[ebp+Ancho_FFT_30] ; eax=AnchoFFT
        mov ebx,eax    ; ebx=AnchoFFT
        shl ebx,3      ; ebx=AnchoFFT*4*2 -> Salto de linea
        mov ecx,eax    ; ecx=AnchoFFT -> Bucle Y
        dec eax        ; eax=(AnchoFFT-1)
        mul ecx        ; edxeax=AnchoFFT*(AnchoFFT-1)
        shl eax,2      ; eax=AnchoFFT*(AnchoFFT-1)*4
        mov edi,eax
        add edi,[ebp+Dir_FFT_30] ; edi -> Imagen Destino
        mov esi,[ebp+Dir_Tabla_Fase_30]
        mov eax,ecx    ; eax=AnchoFFT -> Bucle X
        xor edx,edx    ; edx -> Para recorrer la tabla
.Calcula_Fase_Normalizada_EjeY:
        push ecx
```

```
        mov ecx,eax
.Calcula_Fase_Normalizada_EjeX:
        movups xmm0,[esi+(edx*8)] ; xmm0=*|Ar|Ag|Ab
        mulps xmm0,xmm1   ; xmm0=0|Ar*(180/PI)|Ag*(180/PI)|Ab*(180/PI)
        subps xmm0,xmm2   ; xmm0=0|(Ar*(180/PI))-m|(Ag*(180/PI))-m|(Ab*(180/PI))-m
        mulps xmm0,xmm3   ; xmm0=0|R|G|B (pfsp)
        cvtps2dq xmm0,xmm0 ; xmm0=0|R|G|B (enteros 32 bits)
        packssdw xmm0,xmm7 ; xmm0=0|0|000R0G0B (enteros 16 bits)
        packuswb xmm0,xmm7 ; xmm0=0|0|0|0RGB
        movd [edi],xmm0    ; Imagen Destino <- xmm0
        add edi,4
        add edx,2
        loop .Calcula_Fase_Normalizada_EjeX
        sub edi,ebx ; Situamos el puntero de la Imagen Destino
        pop ecx
        loop .Calcula_Fase_Normalizada_EjeY
        popad
        pop ebp ; Epilogo
        ret

_ASM_Calcula_Imagen_Real_A_Partir_De_Tablas:
; Dadas tres tablas, una para cada componente de color, se calcula la
; Imagen Real que se obtiene a partir de las tablas (la imagen real se
; construye a traves del modulo de las componentes)
Dir_Dest_31        equ  8
Dir_Tabla_Rojo_31  equ 12
Dir_Tabla_Verde_31 equ 16
Dir_Tabla_Azul_31  equ 20
Ancho_FFT_31       equ 24
        push ebp
        mov ebp,esp ; Prologo
        pushad
        mov eax,[ebp+Ancho_FFT_31] ; eax=AnchoFFT
        pxor mm0,mm0
        movd mm0,eax ; mm0=0|AnchoFFT -> Bucle X
        mov ebx,eax  ; ebx=AnchoFFT
        mov ecx,eax  ; ecx=AnchoFFT -> Bucle Y
        dec eax      ; eax=(AnchoFFT-1)
        mul ecx      ; edxeax=AnchoFFT*(AnchoFFT-1)
        shl eax,2    ; eax=AnchoFFT*(AnchoFFT-1)*4
        mov edi,eax
        add edi,[ebp+Dir_Dest_31] ; edi -> Imagen Destino
        shl ebx,3    ; ebx=AnchoFFT*4*2 -> Salto de linea
        pxor mm1,mm1
        movd mm1,ebx ; mm1=0|Salto de linea
        mov esi,[ebp+Dir_Tabla_Rojo_31] ; esi -> Tabla Rojo
        mov eax,[ebp+Dir_Tabla_Verde_31]; eax -> Tabla Verde
        mov ebx,[ebp+Dir_Tabla_Azul_31] ; ebx -> Tabla Azul
        xor edx,edx  ; edx -> Para recorrer las tablas
        pxor xmm7,xmm7 ; xmm7 -> Para entremezclado
        pxor xmm5,xmm5
.Calcula_Imagen_Real_EjeY:
        push ecx
        movd ecx,mm0
.Calcula_Imagen_Real_EjeX:
        movq xmm0,[esi+(edx*8)] ; xmm0=0|0|Im|Re (R)
        movq xmm1,[eax+(edx*8)] ; xmm1=0|0|Im|Re (G)
        movq xmm2,[ebx+(edx*8)] ; xmm2=0|0|Im|Re (B)
        mulps xmm0,xmm0    ; xmm0=0|0|Im2|Re2 (R) siendo 2=cuadrado
        mulps xmm1,xmm1    ; xmm1=0|0|Im2|Re2 (G)
        mulps xmm2,xmm2    ; xmm2=0|0|Im2|Re2 (B)
        pslldq xmm1,8      ; xmm1=Im2|Re2|0|0 (G)
        por xmm1,xmm2      ; xmm1=Im2|Re2(G)|Im2|Re2(B)
        haddps xmm1,xmm0   ; xmm1=0|Im2+Re2(B)|Im2+Re2(G)|Im2+Re2(B)
        sqrtps xmm1,xmm1   ; xmm1=0|R|G|B (pfsp)
        cvtps2dq xmm1,xmm1 ; xmm1=0|R|G|B (enteros 32 bits)
        packssdw xmm1,xmm7 ; xmm1=0|0|000R0G0B (enteros 16 bits)
        packuswb xmm1,xmm7 ; xmm1=0|0|0|0RGB
        movd [edi],xmm1    ; Tabla Destino <- xmm1
        add edi,4
        inc edx
        loop .Calcula_Imagen_Real_EjeX
        movd ecx,mm1       ; Situamos el puntero de la Imagen Destino
        sub edi,ecx
```

```
4205        pop ecx
            loop .Calcula_Imagen_Real_EjeY
            popad
            pop ebp  ; Epilogo
            ret
4210
    _ASM_Calcula_Parte_Real_Y_Maximo:
    ; Dadas tres tablas, una para cada componente de color, calcula la
    ; parte real de cada tabla de color y la introduce en una nueva
    ; tabla Tabla_Real, es decir, esta ultima Tabla_Real contiene la parte
4215 ; Real de cada componente de color, ademas se calcula tambien
    ; el maximo de todas las partes reales y es introducido en la
    ; variable MAXIMO, para posteriores calculos
    Dir_Tabla_Real_32  equ  8
    Dir_Tabla_Rojo_32  equ 12
4220 Dir_Tabla_Verde_32 equ 16
    Dir_Tabla_Azul_32  equ 20
    Ancho_FFT_32       equ 24
            push ebp
            mov ebp,esp ; Prologo
4225        pushad
            mov eax,[ebp+Ancho_FFT_32] ; eax=AnchoFFT
            mov ecx,eax ; ecx=AnchoFFT
            mul ecx     ; edxeax=AnchoFFT*AnchoFFT
            mov ecx,eax ; ecx=AnchoFFT*AnchoFFT -> Bucle
4230        pxor xmm7,xmm7 ; xmm7 -> Llevara el maximo
            mov edi,[ebp+Dir_Tabla_Real_32] ; edi -> Tabla Real
            mov esi,[ebp+Dir_Tabla_Rojo_32] ; esi -> Tabla Rojo
            mov eax,[ebp+Dir_Tabla_Verde_32]; eax -> Tabla Verde
            mov ebx,[ebp+Dir_Tabla_Azul_32] ; ebx -> Tabla Azul
4235        xor edx,edx ; edx -> Para recorrer las tablas
    .Calcula_Parte_Real_Y_Busca_Maximo:
            movd xmm0,[esi+(edx*8)] ; xmm0=0|0|0|R (pfsp)
            pslldq xmm0,8 ; xmm0=0|R|0|0 (pfsp)
            movd xmm1,[eax+(edx*8)] ; xmm1=0|0|0|G (pfsp)
4240        pslldq xmm1,4 ; xmm1=0|0|G|0 (pfsp)
            movd xmm2,[ebx+(edx*8)] ; xmm2=0|0|0|B (pfsp)
            por xmm0,xmm1 ; xmm0=0|R|G|0 (pfsp)
            por xmm0,xmm2 ; xmm0=0|R|G|B (pfsp)
            maxps xmm7,xmm0   ; xmm7 -> Maximo de xmm7 y xmm0
4245        movups [edi],xmm0 ; Tabla Real <- xmm0
            add edi,16 ; Incremento de la Tabla Real
            inc edx
            loop .Calcula_Parte_Real_Y_Busca_Maximo
            ; Tengo en xmm7=0|MAX(R)|MAX(G)|MAX(B) siendo MAX=Maximo
4250        movups xmm6,xmm7 ; xmm6=0|MAX(R)|MAX(G)|MAX(B)
            movups xmm5,xmm7 ; xmm5=0|MAX(R)|MAX(G)|MAX(B)
            psrldq xmm6,4    ; xmm6=0|0|MAX(R)|MAX(G)
            psrldq xmm5,8    ; xmm7=0|0|0|MAX(R)
            maxss xmm7,xmm6  ; xmm7=*|*|*|MAX(R,G)
4255        maxss xmm7,xmm5  ; xmm7=*|*|*|MAX(R,G,B)
            movd [MAXIMO_FFT],xmm7 ; MAX(R,G,B) -> [MAXIMO]
            popad
            pop ebp ; Epilogo
            ret
4260
    _ASM_Calcula_Parte_Imaginaria_Y_Maximo:
    ; Dadas tres tablas, una para cada componente de color, calcula la
    ; parte imaginaria de cada tabla de color y la introduce en una nueva
    ; tabla Tabla_Imag, es decir, esta ultima Tabla_Imag contiene la parte
4265 ; Imaginaria de cada componente de color, ademas se calcula tambien
    ; el maximo de todas las partes imaginarias y es introducido en la
    ; variable MAXIMO, para posteriores calculos
    Dir_Tabla_Imag_33  equ  8
    Dir_Tabla_Rojo_33  equ 12
4270 Dir_Tabla_Verde_33 equ 16
    Dir_Tabla_Azul_33  equ 20
    Ancho_FFT_33       equ 24
            push ebp
            mov ebp,esp ; Prologo
4275        pushad
            mov eax,[ebp+Ancho_FFT_33] ; eax=AnchoFFT
            mov ecx,eax ; ecx=AnchoFFT
            mul ecx     ; edxeax=AnchoFFT*AnchoFFT
```

```
        mov ecx,eax ; ecx=AnchoFFT*AnchoFFT -> Bucle
        pxor xmm7,xmm7 ; xmm7 -> Llevara el maximo
        mov edi,[ebp+Dir_Tabla_Imag_33] ; edi -> Tabla Imaginaria
        mov esi,[ebp+Dir_Tabla_Rojo_33] ; esi -> Tabla Rojo
        mov eax,[ebp+Dir_Tabla_Verde_33]; eax -> Tabla Verde
        mov ebx,[ebp+Dir_Tabla_Azul_33] ; ebx -> Tabla Azul
        xor edx,edx ; edx -> Para recorrer las tablas
.Calcula_Parte_Imag_Y_Busca_Maximo:
        movd xmm0,[esi+(edx*8)+4] ; xmm0=0|0|0|R (pfsp)
        pslldq xmm0,8 ; xmm0=0|R|0|0 (pfsp)
        movd xmm1,[eax+(edx*8)+4] ; xmm1=0|0|0|G (pfsp)
        pslldq xmm1,4 ; xmm1=0|0|G|0 (pfsp)
        movd xmm2,[ebx+(edx*8)+4] ; xmm2=0|0|0|B (pfsp)
        por xmm0,xmm1 ; xmm0=0|R|G|0 (pfsp)
        por xmm0,xmm2 ; xmm0=0|R|G|B (pfsp)
        maxps xmm7,xmm0   ; xmm7 -> Maximo de xmm7 y xmm0
        movups [edi],xmm0 ; Tabla Imaginaria <- xmm0
        add edi,16 ; Incrementamos la tabla
        inc edx
        loop .Calcula_Parte_Imag_Y_Busca_Maximo
        ; Tengo en xmm7=0|MAX(R)|MAX(G)|MAX(B) siendo MAX=Maximo
        movups xmm6,xmm7 ; xmm6=0|MAX(R)|MAX(G)|MAX(B)
        movups xmm5,xmm7 ; xmm5=0|MAX(R)|MAX(G)|MAX(B)
        psrldq xmm6,4    ; xmm6=0|0|MAX(R)|MAX(G)
        psrldq xmm5,8    ; xmm7=0|0|0|MAX(R)
        maxss xmm7,xmm6  ; xmm7=*|*|*|MAX(R,G)
        maxss xmm7,xmm5  ; xmm7=*|*|*|MAX(R,G,B)
        movd [MAXIMO_FFT],xmm7 ; MAX(R,G,B) -> [MAXIMO]
        popad
        pop ebp ; Epilogo
        ret

_ASM_Copia_Video:
; Copia un bloque de datos que en nuestro caso esta justo a continuacion
; de la memoria de la pantalla visible, en una direccion de memoria
; (Dir_Imagen) donde queremos la imagen que ha sido capturada del video.
; El selector sera el de pantalla y el offset el tamano de la pantalla en
; bytes, es decir, AnchoPantalla*AltoPantalla*4.
Selector_34   equ  8
Offset_34     equ 12
Dir_Imagen_34 equ 16
TotalBytes_34 equ 20
        push ebp
        mov ebp,esp ; Prologo
        pushad
        push gs ; Guardamos en la pila el selector de segmento
        mov ax,[ebp+Selector_34]
        mov gs,ax       ; Selector de segmento
        mov eax,[ebp+TotalBytes_34] ; eax=TotalBytes
        xor edx,edx
        mov ebx,16
        div ebx     ; eax=Cociente edx=Resto
        mov edi,[ebp+Dir_Imagen_34] ; edi -> Bloque de datos Destino
        mov esi,[ebp+Offset_34] ; esi=offset
        ; [gs:esi] apunta a la posicion en pantalla donde hay que comenzar
        ; a copiar los pixeles, apuntara justo donde acaba la pantalla visible
        cmp eax,0
        jz .CopiaVideoHayResto
        mov ecx,eax
.CopiaVideo:
        movdqu xmm0,[gs:esi] ; Vamos copiando el contenido de memoria de
        movdqu [edi],xmm0    ; pantalla en la memoria indicada como Destino
        add esi,16            ; de tal forma que se van copiando de 4 en 4
        add edi,16            ; pixeles
        loop .CopiaVideo
.CopiaVideoHayResto:
        cmp edx,0
        jz .SalirDeCopiaVideo
        shr edx,2
        mov ecx,edx
.RepetirCopiaVideo:
        mov eax,[gs:esi]
        mov [edi],eax
        add esi,4
```

```
        add edi,4
        loop .RepetirCopiaVideo
.SalirDeCopiaVideo:
        pop gs
        popad
        pop ebp ; Epilogo
        ret

[SECTION .data]
TODO_UNOS dd 0ffffffffh,0ffffffffh,0ffffffffh,0ffffffffh
MASC_AND_ROTACION dd 0h,0ffffffffh,0ffffffffh,0h
MASC_ROTACION_Y dd 0h,1h,1h,0h
MASC_ROTACION_X dd -1h,0h,0h,1h
MASC_MAGNIFICACION dd 1h,0h,0h,0h
MASC_MAGNIFICACION_AND dd 0h,0ffffffffh,0h,0h
MASC_MAGNIFICACION_AND_XMM dd 0h,0ffffffffh,0h,0h
MASC_MAGNIFICACION_XMM4 dd 0ffffffffh,0ffffffffh,0h,0h
MASC_MAGNIFICACION_INC_X dd 1h,0h,0h,0h
MASC_MAGNIFICACION_INC_Y dd 0h,1h,0h,0h
MASC_MINI_INC_X dd 1h,0h,0h,0h
MASC_MINI_INC_Y dd 0h,1h,0h,0h
MASC_MINI_AND dd 0h,0ffffffffh,0h,0h
MASC_MINI_XMM0 dd 0h,0ffffffffh,0h,0h
MASC_MINI_AND_XMM4 dd 0ffffffffh,0ffffffffh,0h,0h
MASC_ECUALIZADO_AZUL_0 dd 0ffh,0ffh,0ffh,0ffh
MASC_ECUALIZADO_AZUL_1 dd 0ffff00h,0ffff00h,0ffff00h,0ffff00h
MASC_ECUALIZADO_VERDE_0 dd 0ff00h,0ff00h,0ff00h,0ff00h
MASC_ECUALIZADO_VERDE_1 dd 0ff00ffh,0ff00ffh,0ff00ffh,0ff00ffh
MASC_ECUALIZADO_ROJO_0 dd 0ff0000h,0ff0000h,0ff0000h,0ff0000h
MASC_ECUALIZADO_ROJO_1 dd 0fffffh,0fffffh,0fffffh,0fffffh
MASC_ESCRIBE_INC_X dd 1h,0h,0h,0h
MASC_ESCRIBE_INC_Y dd 0h,1h,0h,0h
MASC_ESCRIBE_COLOR_TRANSPARENTE dd 0ffffffh,0h,0h,0h
MASC_DISTORSION_SUMA dd 1h,1h,0h,0h
MASC_DISTORSION_INC dd 1h,0h,1h,0h
MASC_DISTORSION_AND dd 0h,0ffffffffh,0h,0ffffffffh
MASC_FFT_AND_CREA_TABLAS dd 0ffffffh,0h,0h,0h
MASC_FFT_PHASE_REAL dd -1,0,0,0
MASC_FFT_Fila_1 dd 2h,0h,0h,0h
MASC_FFT_Fila_2 dd 1h,0h,0h,0h
MASC_FFT_TWIDDLE.REAL dd 1h,0h,0h,0h
MASC_FFT_MAXIMO dd 255,255,255,0
MASC_FFT_FASE_1 dd 360,360,360,0
MASC_FFT_FASE_2 dd 180,180,180,0
MASC_FFT_FASE_PI dq 3.141592654,0.0

[SECTION .bss]
HAY_INCREMENTO_EJEX resd 1
HAY_INCREMENTO_EJEY resd 1
MMAX resd 1
ISTEP resd 1
ANGULO resd 1
MODULO_LOG_ROJO resd 1
MODULO_LOG_VERDE resd 1
MODULO_LOG_AZUL resd 1
MAXIMO_FFT resd 1
MINIMO_FFT resd 1
```

A.3. Archivos de visualización por pantalla

Listado A.6: pantalla.h

```
// Fichero: pantalla.h
//
// Biblioteca de funciones para Tratamiento de
// Imágenes en Tiempo Real
//
// Esqueleto C++
//

#ifndef _pantalla_h_
#define _pantalla_h_

// Clase Pantalla

//Trabaja con el estándar VESA 2.0

class tPantalla
{
    private:

        int PonModoVESA(int Modo);

    public:
        int Selector;
        unsigned int Direccion;
        unsigned int DirFisica;
        int AnchoPantalla;
        int AltoPantalla;
        int TotalMemoriaDisponible;
        int MemoriaRestante;

        int PonModoVESA(int Ancho, int Alto);
        int PonModoTexto();
};

#endif // _pantalla_h_
```

Listado A.7: pantalla.cpp

```
// Fichero: pantalla.cpp
//
// Biblioteca de funciones para Tratamiento de
// Imágenes en Tiempo Real
//
// Esqueleto C++
//
#include "pantalla.h"
#include "error.h"
#include <dpmi.h>
#include <go32.h>
#include <stdio.h>
#include <stdlib.h>
// Definicion de constantes para el alineamiento de memoria
#define RM_SEGMENT(addr)    ((addr>>4)&0xFFFF)
#define RM_OFFSET(addr)     (addr&0xF)
#define MASK_LINEAL(addr)   (addr&0x000FFFFF)

// Estructura ModeInfoBlock para obtener la informacion de un modo de pantalla
typedef struct {
  short          ModeAttributes;
  unsigned char  WinAAttributes;
  unsigned char  WinBAttributes;
  short          WinGranularity;
  short          WinSize;
  short          WinASegment;
  short          WinBSegment;
  unsigned       WinFuncPtr;
  short          BytesPerScanLine;
  short          XResolution;
```

```
    short          YResolution;
    unsigned char  XCharSize;
    unsigned char  YCharSize;
    unsigned char  NumberOfPlanes;
    unsigned char  BitsPerPixel;
    unsigned char  NumberOfBanks;
    unsigned char  MemoryModel;
    unsigned char  BankSize;
    unsigned char  NumberOfImagePages;
    unsigned char  Reserved;
    unsigned char  RedMaskSize;
    unsigned char  RedFieldPosiiton;
    unsigned char  GreenMaskSize;
    unsigned char  GreenFieldPosiiton;
    unsigned char  BlueMaskSize;
    unsigned char  BlueFieldPosiiton;
    unsigned char  RsvdMaskSize;
    unsigned char  RsvdFieldPosiiton;
    unsigned char  DirectColorModeInfo;
    unsigned       PhysBasePtr;
    unsigned       OffScreenMemOffset;
    short          OffScreenMemSize;
    char           bogus[206];
} ModeInfoBlock;
// Estructura VbeInfoBlock para obtener la informacion del modo VESA existente
typedef struct {
    unsigned char  VbeSignature[4];
    unsigned short VbeVersion;
    unsigned long  OemStringPtr;
    unsigned char  Capabilities[4];
    unsigned long  VideoModePtr;
    unsigned short TotalMemory;
    unsigned short OemSoftwareRev;
    unsigned long  OemVendorNamePtr;
    unsigned long  OemProductNamePtr;
    unsigned long  OemProductRevPtr;
    unsigned char  Reserved[222];
    unsigned char  OemData[256];
} VbeInfoBlock;

ModeInfoBlock *get_mode_info(int mode)
{
 static ModeInfoBlock info; // Obtiene la informacion de un modo seleccionado
 __dpmi_regs r;

 r.x.ax = 0x4F01;
 r.x.cx = mode;
 r.x.es = RM_SEGMENT(__tb);
 r.x.di = RM_OFFSET(__tb);
 __dpmi_int(0x10, &r);
 if(r.h.ah) return 0;
 dosmemget(MASK_LINEAL(__tb), sizeof(ModeInfoBlock), &info);
 return &info;
};

VbeInfoBlock *get_mode_info() // Obtiene la informacion VESA deseada
{
 static VbeInfoBlock infoVbe;
 __dpmi_regs r;
 __dpmi_paddr addr;

 r.x.ax = 0x4F00;
 r.x.es = RM_SEGMENT(__tb);
 r.x.di = RM_OFFSET(__tb);
 __dpmi_int(0x10, &r);
 if (r.h.ah) return 0;
 dosmemget(MASK_LINEAL(__tb), sizeof(VbeInfoBlock), &infoVbe);
 return &infoVbe;
};

int tPantalla::PonModoVESA(int Modo) // Pone el modo VESA seleccionado y
{                                    // pasado a traves de la variable Modo
 __dpmi_regs reg;
```

```
105   reg.x.ax=0x4f02;
      reg.x.bx=Modo | 0x4000;
      __dpmi_int(0x10,&reg);
      if(reg.h.al != 0x4f || reg.h.ah)
      {
110    printf("No␣pude␣poner␣ese␣modo!\n");
       exit (ERROR_PANTALLA_NO_PUEDO_PONER_MODO);
      };
      printf("Modo␣0x%X\n",Modo | 0x4000);
      // Defino un selector de segmento que apunte a la pantalla
115   Selector = __dpmi_allocate_ldt_descriptors( 1 );
      __dpmi_set_segment_base_address( Selector, Direccion );
      __dpmi_set_segment_limit(Selector, TotalMemoriaDisponible );
      Direccion = 0;
      return 0;
120   };

    // Busca el Modo VESA correspondiente a los parametros pasados a la funcion,
    // es decir, Ancho y Alto, y lo pone, en caso de no encontrar ningun modo
    // con esa resolucion produce un error
125 int tPantalla::PonModoVESA(int Ancho, int Alto )
    {
      __dpmi_meminfo info;
      __dpmi_regs reg;
      ModeInfoBlock *mb;
130   VbeInfoBlock *vb;
      int modo, ModoEncontrado=0;

      for( modo=0x100; modo < 0x160; modo++ )
      {
135    mb=get_mode_info( modo );
       if (!mb) printf("Modo␣0x%x.␣No␣pude␣obtener␣la␣informacion␣VESA.\n", modo );
       else if (!(mb->ModeAttributes & 0x80))
              printf("Modo␣0x%x.␣No␣soporta␣linear␣frame␣buffer␣(LFB)\n", modo );
       else
140    {
        if( Ancho == mb->XResolution && Alto  == mb->YResolution && mb->BitsPerPixel==32)
            {
             printf("Modo␣=␣0x%X,␣Ancho␣=␣%4d,␣Alto␣=␣%4d,␣BPP␣=␣%d\n",
                       modo, mb->XResolution, mb->YResolution, mb->BitsPerPixel);
145          ModoEncontrado=1; // Indica que hemos encontrado el modo deseado
             break; // Encontro el modo y salimos del bucle
            };
       };
      };
150   if (ModoEncontrado==0) exit (ERROR_PANTALLA_MODO_NO_ENCONTRADO);
      vb=get_mode_info();
      if (vb->TotalMemory==0) vb->TotalMemory=256;
      TotalMemoriaDisponible=vb->TotalMemory*64*1024;
      info.size = TotalMemoriaDisponible; // Ajustar a toda la mem de video disponible
155   info.address = mb->PhysBasePtr;
      if(__dpmi_physical_address_mapping(&info) == -1)
       {
        printf("Fallo␣de␣mapeo␣de␣la␣direccion␣fisica␣0x%x\n",mb->PhysBasePtr);
        exit(ERROR_MEMORIA_PANTALLA);
160    };
      Direccion     = info.address;
      DirFisica     = info.address;
      AnchoPantalla = Ancho;
      AltoPantalla  = Alto;
165   MemoriaRestante = (vb -> TotalMemory*64*1024)-(AnchoPantalla*AltoPantalla*4);
      PonModoVESA(modo);
      return 0;
    };

170 int tPantalla::PonModoTexto()
    {
      __dpmi_regs reg;

      reg.x.ax = 0x0003;
175   __dpmi_int(0x10,&reg);
      return 0;
    };
```

A.4. Archivo de error

Listado A.8: error.h

```
// Defino los codigos de error

#define ERROR_PANTALLA_NO_PUEDO_PONER_MODO                                1
#define ERROR_MEMORIA_PANTALLA                                            2
#define ERROR_PANTALLA_MODO_NO_ENCONTRADO                                 3
#define ERROR_NO_HAY_MEMORIA                                              4
#define ERROR_DE_APERTURA_DE_FICHERO_PARA_LECTURA                         5
#define ERROR_DE_APERTURA_O_DE_CREACION_DE_FICHERO_PARA_ESCRITURA         6
#define ERROR_NO_ES_UN_ARCHIVO_BMP                                        7
#define ERROR_DE_INTRODUCCION_DE_PARAMETROS_EN_DIBUJALO                   8
#define ERROR_DE_INTRODUCCION_DE_PARAMETROS_EN_CAPTURALO                  9
#define ERROR_DE_INTRODUCCION_DE_PARAMETROS_EN_RECORTAR                  10
#define ERROR_IMAGENES_DE_DIFERENTE_TAMANO_EN_SUMA                       11
#define ERROR_IMAGENES_DE_DIFERENTE_TAMANO_EN_RESTA                      12
#define ERROR_DE_INTRODUCCION_DE_PARAMETROS_EN_CONVOLUCION_ENTERA        13
#define ERROR_DE_INTRODUCCION_DE_PARAMETROS_EN_CONVOLUCION_REAL          14
#define ERROR_DE_INTRODUCCION_DE_PARAMETROS_EN_TRANSFORMACION_AFIN       15
#define ERROR_DE_INTRODUCCION_DE_PARAMETROS_EN_ROTACION                  16
#define ERROR_DE_INTRODUCCION_DE_PARAMETROS_EN_ESCALADO                  17
#define ERROR_DE_INTRODUCCION_DE_PARAMETROS_EN_TRASLACION                18
#define ERROR_IMAGENES_DE_DIFERENTE_TAMANO_EN_FUNCION_XOR                19
#define ERROR_IMAGENES_DE_DIFERENTE_TAMANO_EN_FUNCION_OR                 20
#define ERROR_DE_INTRODUCCION_DE_PARAMETROS_EN_ECUALIZADO                21
#define ERROR_DE_INTRODUCCION_DE_PARAMETROS_EN_PINTA_FFT2D               22
#define ERROR_TAMANO_DE_FFT2D_NO_VALIDO                                  23
#define ERROR_TAMANO_DE_INVERSA_FFT2D_NO_VALIDO                          24
#define ERROR_DE_INTRODUCCION_DE_PARAMETROS_EN_REDIMENSIONAMIENTO        25
#define ERROR_DE_INTRODUCCION_DE_PARAMETROS_EN_ESCRIBE_TEXTO             26
#define ERROR_DE_INTRODUCCION_DE_PARAMETROS_CAPTURA_IMAGEN_DE_VIDEO      27
#define ERROR_NO_HAY_MEMORIA_PANTALLA_SUFICIENTE_PARA_IMAGEN             28
#define ERROR_DE_INTRODUCCION_DE_PARAMETROS_EN_MODO_DE_CAPTURA           29
#define ERROR_DE_INTRODUCCION_DE_PARAMETROS_EN_CAPTURA_VIDEO_EN_PANTALLA 30
```

Apéndice B

Archivos modificados

Se presentan los diferentes listados de los archivos generados en la construcción de la biblioteca para el tratamiento de vídeo en tiempo real que han sido modificados, recordar que para la captura de vídeo partimos de una biblioteca o de un conjunto de archivos ya creados y que fueron vistos en el capítulo 6.

Como ya se expuso anteriormente los dos archivos modificados de este conjunto han sido:

- dma.cpp: ha sido modificado para alojar el buffer del DMA a partir de la posición en memoria física de 63M para la captura de vídeo par y a partir de la posición en memoria física de 64M para la captura de vídeo impar.
- scaler.cpp: ha sido modificado introduciendo nuevas funciones para la obtención y escalado de la imagen proveniente de la cámara de vídeo y capturada a partir de la capturadora de vídeo de la que disponemos. Estas funciones han sido creadas para adapatar dicho archivo a nuestras necesidades requeridas.

El resto de archivos que no han sido modificados han sido incluidos en el CD adjunto a la memoria del proyecto.

B.1. Archivos del DMA

Listado B.1: dma.h

```
#ifndef DMA_H

#define DMA_H

#include "compiler.h"

class DMA_BUFFER : public VIRTUAL_MEM
{

    public:
        DMA_BUFFER( int size );
        ~DMA_BUFFER( void )
        {};
        void DisplayInfo( void );
        int DMA_BUFFER::GetFreeMemory( void )
        {
            return 0;
        }
        void DMA_BUFFER::AverageLines( int bytes_per_pixel );
};

#endif
```

Listado B.2: dma.cpp

```
//#include <conio.h>
#include <dos.h>
#include <stdio.h>
#include <stdlib.h>
#include "dma.h"

//#define PRINT_INFO

DMA_BUFFER::DMA_BUFFER( int size )
{
    DWORD total_size;
    DWORD physical_address;

    total_size = (size + 3) & 0xFFFFFFFC;   /* Make it DWORD aligned */
    /* Map Physical to linear address */
    if (size>1024*8)
       physical_address=66060288; // Direccion fisica = 63M
    else
       physical_address=67108864; // Direccion fisica = 64M
    CreateVirtual( physical_address, total_size );

}

void DMA_BUFFER::DisplayInfo( void )
{
    printf( "Physical Address = 0x%08X\n", GetPhysicalAddress() );

    printf( "Virtual/Linear Address = 0x%08X\n", GetVirtualAddress() );

    printf( "Size = %d\n", GetLength() );

}

/*****************************************************************************
    Average every other line with the line above and the line below.
    Note: this assumes 640x480 with with every other line missing.
*****************************************************************************/
void DMA_BUFFER::AverageLines( int bytes_per_pixel )
{
}
```

B.2. Archivos de obtención y escalado de imagen

Listado B.3: scaler.h

```
#ifndef SCALER_H

#define SCALER_H

#include "pcidecod.h"
#include "pantalla.h"
#include "imagen.h"

void Crea_RISC_WxH_Continuo( tPantalla Pantalla, PCI_DECODER &decoder, int width,
                             int height, int x=0, int y=0, int offset=0 );
void Crea_RISC_WxH_Una_Sola( tPantalla Pantalla, PCI_DECODER &decoder, int width,
                             int height, int x=0, int y=0, int offset=0 );
void Crea_RISC_WxH_Imagen_Continua( tImagen Imagen, PCI_DECODER &decoder, int width,
                                    int height, int offset=0);
void Crea_RISC_WxH_Una_Sola_Imagen( tImagen Imagen, PCI_DECODER &decoder, int width,
                                    int height, int offset=0);

void Escalado( PCI_DECODER &decoder, int width, int height, int field );
void set_output_format( PCI_DECODER &decoder );

#endif
```

Listado B.4: scaler.cpp

```
#include "scaler.h"

OnOff VFilterFlag_ = On;
int old_width = 320;
int old_height = 240;
int old_x = 0;
int old_y =0;
int last_offset = 0;

#define min(a,b) (((a)<(b))?(a):(b))
#define max(a,b) (((a)>(b))?(a):(b))

/*****************************************************************************
    Set the even and odd output format for the current screen RGB depth.
*****************************************************************************/
void set_output_format( PCI_DECODER &decoder )
{
 int bbp=32;
 if( bbp == 32 )
 {
    decoder.Video->Even->SetColorFormat( ColorFormatRGB32 );
    decoder.Video->Odd->SetColorFormat( ColorFormatRGB32 );
 }
}

void Crea_RISC_WxH_Continuo( tPantalla Pantalla, PCI_DECODER &decoder, int width,
                             int height, int x, int y, int offset )
{
 DWORD risc_main[1024];
 int i, risc_loc = 0;
 set_output_format( decoder );
 width = width * 4; // Sabemos que cada pixel ocupa 4 bytes
 risc_main[risc_loc++] = RESYNC|VRE_STAT;
 risc_main[risc_loc++] = 0x0;
 if( offset == 0 )
    risc_main[risc_loc++] = RESYNC|FM1_STAT|RESET_RISCS_1111;
 else
    risc_main[risc_loc++] = RESYNC|FM1_STAT|SET_RISCS_0001|RESET_RISCS_1110;
 risc_main[risc_loc++] = 0x0;
 /* Crea programa RISC */
 if( height > 284 )
 {
    for (i = 0; i < height; i+=2)
    {
```

```
            risc_main[risc_loc++] = WRITE | SOL | EOL | width;
            risc_main[risc_loc++] = Pantalla.DirFisica + ((x + Pantalla.AnchoPantalla *
            (y+i)) * 4 );
         }
49    }
      else
      {
         for (i = 0; i < height; i++)
         {
54          risc_main[risc_loc++] = WRITE | SOL | EOL | width;
            risc_main[risc_loc++] = Pantalla.DirFisica + ((x + Pantalla.AnchoPantalla *
            (y+i)) * 4 );
         }
      }
59    risc_main[risc_loc++] = RESYNC|VRO_STAT;
      risc_main[risc_loc++] = 0x0;
      risc_main[risc_loc++] = JUMP;
      risc_main[risc_loc++] = decoder.Video->Even->GetRiscAddr() + offset;
      decoder.Video->Odd->SetRisc( risc_main, risc_loc * 4, offset );
64    /* now do Even field */
      risc_loc = 0;
      risc_main[risc_loc++] = RESYNC|FM1_STAT;
      risc_main[risc_loc++] = 0x0;
      /* Crea programa RISC */
69    if( height > 284 )
      {
         for (i = 1; i < height; i+=2)
         {
            risc_main[risc_loc++] = WRITE | SOL | EOL | width;
74          risc_main[risc_loc++] = Pantalla.DirFisica + ((x + Pantalla.AnchoPantalla *
            (y+i)) * 4 );
         }
      }
      else
79    {
         for (i = 0; i < height; i++)
         {
            risc_main[risc_loc++] = WRITE | SOL | EOL | width;
            risc_main[risc_loc++] = Pantalla.DirFisica + ((x + Pantalla.AnchoPantalla *
84          (y+i)) * 4 );
         }
      }
      risc_main[risc_loc++] = RESYNC|VRE_STAT;
      risc_main[risc_loc++]=0x0;
89    risc_main[risc_loc++] = JUMP;
      risc_main[risc_loc++] = decoder.Video->Odd->GetRiscAddr()+ 8 + offset;
      decoder.Video->Even->SetRisc( risc_main, risc_loc * 4, offset );
   }

94 void Crea_RISC_WxH_Una_Sola( tPantalla Pantalla, PCI_DECODER &decoder, int width,
                               int height, int x, int y, int offset )
   {
      DWORD risc_main[1024];
      int risc_loc = 0,i;
99    set_output_format( decoder );
      width = width * 4; // Sabemos que cada pixel ocupa 4 bytes
      risc_main[risc_loc++] = RESYNC|VRE_STAT;
      risc_main[risc_loc++] = 0x0;
      if( offset == 0 )
104      risc_main[risc_loc++] = RESYNC|FM1_STAT|RESET_RISCS_1111;
      else
         risc_main[risc_loc++] = RESYNC|FM1_STAT|SET_RISCS_0001|RESET_RISCS_1110;
      risc_main[risc_loc++] = 0x0;
      /* Crea programa RISC */
109   if( height > 284 )
      {
         for (i = 0; i < height; i+=2)
         {
            risc_main[risc_loc++] = WRITE | SOL | EOL | width;
114         risc_main[risc_loc++] = Pantalla.DirFisica + ((x + Pantalla.AnchoPantalla *
            (y+i)) * 4 );
         }
      }
      else
```

```
119  {
        for (i = 0; i < height; i++)
        {
           risc_main[risc_loc++] = WRITE | SOL | EOL | width;
           risc_main[risc_loc++] = Pantalla.DirFisica + ((x + Pantalla.AnchoPantalla *
124        (y+i)) * 4 );
        }
     }
     risc_main[risc_loc++] = RESYNC|VRO_STAT;
     risc_main[risc_loc++] = 0x0;
129  risc_main[risc_loc++] = JUMP;
     risc_main[risc_loc++] = decoder.Video->Even->GetRiscAddr() + offset;
     decoder.Video->Odd->SetRisc( risc_main, risc_loc * 4, offset );
     /* now do Even field */
     risc_loc = 0;
134  risc_main[risc_loc++] = RESYNC|FM1_STAT;
     risc_main[risc_loc++] = 0x0;
     /* Crea programa RISC */
     if( height > 284 )
     {
139     for (i = 1; i < height; i+=2)
        {
           risc_main[risc_loc++] = WRITE | SOL | EOL | width;
           risc_main[risc_loc++] = Pantalla.DirFisica + ((x + Pantalla.AnchoPantalla *
           (y+i)) * 4 );
144     }
     }
     else
     {
        for (i = 0; i < height; i++)
149     {
           risc_main[risc_loc++] = WRITE | SOL | EOL | width;
           risc_main[risc_loc++] = Pantalla.DirFisica + ((x + Pantalla.AnchoPantalla *
           (y+i)) * 4 );
        }
154  }
     risc_main[risc_loc++] = RESYNC|VRE_STAT;
     risc_main[risc_loc++]=0x0;
     risc_main[risc_loc++] = 0x0;
     risc_main[risc_loc++] = decoder.Video->Odd->GetRiscAddr()+ 8 + offset;
159  decoder.Video->Even->SetRisc( risc_main, risc_loc * 4, offset );
  }

  void Crea_RISC_WxH_Imagen_Continua( tImagen Imagen, PCI_DECODER &decoder, int width,
                                      int height, int offset)
164 {
     DWORD risc_main[1024];
     int risc_loc = 0, i;
     set_output_format( decoder );
     width = width * 4; // Sabemos que cada pixel ocupa 4 bytes
169  risc_main[risc_loc++] = RESYNC|VRE_STAT;
     risc_main[risc_loc++] = 0x0;
     if( offset == 0 )
        risc_main[risc_loc++] = RESYNC|FM1_STAT|RESET_RISCS_1111;
     else
174     risc_main[risc_loc++] = RESYNC|FM1_STAT|SET_RISCS_0001|RESET_RISCS_1110;
     risc_main[risc_loc++] = 0x0;
     /* Crea programa RISC */
     if( height > 284 )
     {
179     for (i = 0; i < height; i+=2)
        {
           risc_main[risc_loc++] = WRITE | SOL | EOL | width;
           risc_main[risc_loc++] = Imagen.DireccionFisica + (i*Imagen.Ancho*4);
        }
184  }
     else
     {
        for (i = 0; i < height; i++)
        {
189        risc_main[risc_loc++] = WRITE | SOL | EOL | width;
           risc_main[risc_loc++] = Imagen.DireccionFisica + (i*Imagen.Ancho*4);
        }
     }
```

```
   risc_main[risc_loc++] = RESYNC|VRO_STAT;
194  risc_main[risc_loc++] = 0x0;
   risc_main[risc_loc++] = JUMP;
   risc_main[risc_loc++] = decoder.Video->Even->GetRiscAddr() + offset;
   decoder.Video->Odd->SetRisc( risc_main, risc_loc * 4, offset );
   /* now do Even field */
199  risc_loc = 0;
   risc_main[risc_loc++] = RESYNC|FM1_STAT;
   risc_main[risc_loc++] = 0x0;
   /* Crea programa RISC */
   if( height > 284 )
204  {
      for (i = 1; i < height; i+=2)
      {
         risc_main[risc_loc++] = WRITE | SOL | EOL | width;
         risc_main[risc_loc++] = Imagen.DireccionFisica + (i*Imagen.Ancho*4);
209   }
   }
   else
   {
      for (i = 0; i < height; i++)
214   {
         risc_main[risc_loc++] = WRITE | SOL | EOL | width;
         risc_main[risc_loc++] = Imagen.DireccionFisica + (i*Imagen.Ancho*4);
      }
   }
219  risc_main[risc_loc++] = RESYNC|VRE_STAT;
   risc_main[risc_loc++]=0x0;
   risc_main[risc_loc++] = JUMP;
   risc_main[risc_loc++] = decoder.Video->Odd->GetRiscAddr()+ 8 + offset;
   decoder.Video->Even->SetRisc( risc_main, risc_loc * 4, offset );
224 }

   void Crea_RISC_WxH_Una_Sola_Imagen( tImagen Imagen, PCI_DECODER &decoder, int width,
                                       int height, int offset)
   {
229  DWORD risc_main[1024];
   int risc_loc = 0, i;
   set_output_format( decoder );
   width = width * 4; // Sabemos que cada pixel ocupa 4 bytes
   risc_main[risc_loc++] = RESYNC|VRE_STAT;
234  risc_main[risc_loc++] = 0x0;
   if( offset == 0 )
      risc_main[risc_loc++] = RESYNC|FM1_STAT|RESET_RISCS_1111;
   else
      risc_main[risc_loc++] = RESYNC|FM1_STAT|SET_RISCS_0001|RESET_RISCS_1110;
239  risc_main[risc_loc++] = 0x0;
   /* Crea programa RISC */
   if( height > 284 )
   {
      for (i = 0; i < height; i+=2)
244   {
         risc_main[risc_loc++] = WRITE | SOL | EOL | width;
         risc_main[risc_loc++] = Imagen.DireccionFisica + (i*Imagen.Ancho*4);
      }
   }
249  else
   {
      for (i = 0; i < height; i++)
      {
         risc_main[risc_loc++] = WRITE | SOL | EOL | width;
254      risc_main[risc_loc++] = Imagen.DireccionFisica + (i*Imagen.Ancho*4);
      }
   }
   risc_main[risc_loc++] = RESYNC|VRO_STAT;
   risc_main[risc_loc++] = 0x0;
259  risc_main[risc_loc++] = JUMP;
   risc_main[risc_loc++] = decoder.Video->Even->GetRiscAddr() + offset;
   decoder.Video->Odd->SetRisc( risc_main, risc_loc * 4, offset );
   /* now do Even field */
   risc_loc = 0;
264  risc_main[risc_loc++] = RESYNC|FM1_STAT;
   risc_main[risc_loc++] = 0x0;
   /* Crea programa RISC */
```

```
     if( height > 284 )
     {
269     for (i = 1; i < height; i+=2)
        {
           risc_main[risc_loc++] = WRITE | SOL | EOL | width;
           risc_main[risc_loc++] = Imagen.DireccionFisica + (i*Imagen.Ancho*4);
        }
274  }
     else
     {
        for (i = 0; i < height; i++)
        {
279        risc_main[risc_loc++] = WRITE | SOL | EOL | width;
           risc_main[risc_loc++] = Imagen.DireccionFisica + (i*Imagen.Ancho*4);
        }
     }
     risc_main[risc_loc++] = RESYNC|VRE_STAT;
284  risc_main[risc_loc++]=0x0;
     risc_main[risc_loc++] = 0x0;
     risc_main[risc_loc++] = decoder.Video->Odd->GetRiscAddr()+ 8 + offset;
     decoder.Video->Even->SetRisc( risc_main, risc_loc * 4, offset );
    }
289
    void Escalado( PCI_DECODER &decoder, int width, int height, int field )
    {
        int value;
        WORD Clkx1_HACTIVE;
294     WORD Clkx1_HDELAY;
        WORD Min_Pixels;
        WORD Active_lines_per_field;
        int  VActive;
        WORD AnalogWin_top = 0;
299     WORD AnalogWin_left =0;
        VIDEO_FIELD *vid_field;

        if( field == FieldEven )
            vid_field = decoder.Video->Even;
304     else
            vid_field = decoder.Video->Odd;

        if( decoder.Video->GetVideoFormat() == VideoFormatAuto )
            decoder.Video->SetVideoFormat( VideoFormatNTSC );
309
        if( decoder.Video->GetVideoFormat() == VideoFormatNTSC )
        {

            Clkx1_HACTIVE = 730;
314         Clkx1_HDELAY = 148;
             Min_Pixels = 44;
            Active_lines_per_field = 240;
            VActive = 0x1F4;
        }
319     else
        {
            Clkx1_HACTIVE = 914;
            Clkx1_HDELAY = 190;
            Min_Pixels = 48;
324         Active_lines_per_field = 284;
            VActive = 0x238;
            decoder.Video->SetAGCDelay( 0x7F );
            decoder.Video->SetBurstDelay( 0x72 );
        }
329
        if( height > Active_lines_per_field )
            height /= 2;

        WORD Min_UncroppedPixels = Min_Pixels + 20;
334 //    WORD Min_UncroppedPixels = Min_Pixels + 100;
        WORD Max_Pixels = ((Clkx1_HACTIVE < 774) ? Clkx1_HACTIVE - 6 : 768);
        WORD Min_Lines = (Active_lines_per_field / 16 + 1) * 2;
        WORD Max_Lines = Active_lines_per_field;
        WORD Max_VFilter1_Pixels = ((Clkx1_HACTIVE > 796) ? 384 : (Clkx1_HACTIVE *
339                                 14 / 29));
        WORD Max_VFilter2_Pixels = Clkx1_HACTIVE * 8 / 33;
```

```
        WORD Max_VFilter3_Pixels = Clkx1_HACTIVE * 8 / 33;
        WORD Max_VFilter1_Lines = Active_lines_per_field;
        WORD Max_VFilter2_Lines = Active_lines_per_field / 2;
344     WORD Max_VFilter3_Lines = Active_lines_per_field * 2 / 5;

        // Calculate Hactive
        WORD m_HActive = min( Max_Pixels, max( (WORD)width, Min_Pixels ) );

349     vid_field->SetHactive( m_HActive );

        // No calculation needed for VActive register since it based on the
        // UNSCALED image

354     vid_field->SetVactive( VActive );

        // Calculate Vertical Scaling
        WORD m_lines = min( Max_Lines, max( (WORD)height, Min_Lines ) );

359     WORD LPB_VScale_Factor =(WORD)(1+(m_lines-1)/Active_lines_per_field);

        m_lines = (WORD) ( ( m_lines + LPB_VScale_Factor - 1 ) / LPB_VScale_Factor );

        value = (WORD) ( ( 0x10000L - ((DWORD)Active_lines_per_field * 512L /
364             (DWORD)m_lines) + 512L ) & 0x1FFFL );

        vid_field->SetVscale( value );

        // Set the Vertical Filter
369     // this is to remove junk lines at the top of video. flag set to off
        // when image hight is above CIF
        DWORD m_VFilter;

        if ( VFilterFlag_ == Off )
374     {
            m_VFilter = 0;
        }
        else
        {
379         if ( ( m_HActive <= Max_VFilter3_Pixels ) &&
                   ( m_lines    <= Max_VFilter3_Lines ) )
                m_VFilter = 3;
            else if ( ( m_HActive <= Max_VFilter2_Pixels ) &&
                      ( m_lines    <= Max_VFilter2_Lines ) )
384             m_VFilter = 2;
            else if ( ( m_HActive <= Max_VFilter1_Pixels ) &&
                      ( m_lines    <= Max_VFilter1_Lines ) )
                m_VFilter = 1;
            else
389             m_VFilter = 0;
        }

        vid_field->SetVerticalFilter( m_VFilter );

394
        WORD VDelay, moreDelay;

        // increase VDelay will eliminate garbage lines at top of image
        switch ( m_VFilter )
399     {
            case 3:
                moreDelay = 4;
                break;

404         case 2:
                moreDelay = 2;
                break;

            case 1:
409         case 0:
            default:
                moreDelay = 0;
                break;
        }
414
```

```
        if ( decoder.Video->GetVideoFormat() == VideoFormatNTSC )
            VDelay = 0x001A + moreDelay;     // NTSC
        else
            VDelay = 0x0026 + moreDelay;     // PAL/SECAM
419
        // now add the cropping region into VDelay register; i.e. skip some pixels
        // before we start taking them as real image
        VDelay += (WORD)( ( (DWORD)Max_Lines * (DWORD)AnalogWin_top + m_lines - 1 ) /
                  (DWORD)m_lines * 2 );
424
        vid_field->SetVdelay( VDelay );


       // calculations here requires calculation of HActive first!
429     WORD m_pixels = m_HActive;
        if ( m_pixels < Min_UncroppedPixels )
            m_pixels += (WORD) ( ( Min_UncroppedPixels - m_pixels + 9 ) / 10 );

        DWORD a = (DWORD)m_pixels * (DWORD)Clkx1_HDELAY;
434     DWORD b = (DWORD)Clkx1_HACTIVE * 2L;
        WORD HDelay = (WORD) ( ( a + (DWORD)Clkx1_HACTIVE * 2 - 1) / b * 2L );

        // now add the cropping region into HDelay register; i.e. skip some pixels
        // before we start taking them as real image
439     HDelay += (WORD)AnalogWin_left;

        // HDelay must be even or else color would be wrong
        HDelay &= ~01;

444     vid_field->SetHdelay( HDelay );

        // since we increase HDelay, we should decrease HActive by the same amount
        m_HActive -= (WORD)AnalogWin_left;

449     vid_field->SetHactive( m_HActive );


      value = (WORD) ((((DWORD)Clkx1_HACTIVE*4096L )/(DWORD)m_pixels)-4096L);

454     vid_field->SetHscale( value );


        if ( decoder.Video->GetVideoFormat() != VideoFormatSECAM )
            value = HFilterAutoFormat;
459     else  // SECAM
            if ( m_pixels < Clkx1_HACTIVE / 7 )
                value = HFilterICON;
            else
                value = HFilterQCIF;
464
        vid_field->SetHorizontalFilter( value );
    }
```

Bibliografía

[1] *Análisis de series. Tema 5.*
http://tecnum.com/docencia/itziar/cap5.ppt.

[2] *Capítulo VII: Arquitectura del PC, AT Y PS/2 bajo DOS.*
http://atc.ugr.es/docencia/udigital/index.html.

[3] *Curso de programación orientada a objetos.*
http://ieee.udistrital.edu.co.

[4] *Departamento de Educación de Argentina.*
http://tecno.unsl.edu.ar.

[5] *Ecualizado de imágenes.*
http://mailweb.udlap.mx/~ccastane/Analisis_Numerico_html/.

[6] *Fast Fourier Transforms.*
http://www.amara.com/current/wavelet.html.

[7] *Guia: Introducción a los punteros Far.*
http://www.delorie.com//djgpp/doc/ug/dpmi/farptr-intro.html.

[8] *Instituto de Artes Visuales.*
http://www.artesvisuales.com.

[9] *Manejo de archivos BMP.*
http://www.ii.uam.es.

[10] *Portal web Animania.*
http://animania.com.ar.

[11] *Portal web Digital Digest.*
http://www.digital-digest.com.

[12] *Portal web ImaginArt.*
`http://www.imagin-art.com`.

[13] *Portal web Ondamedia.*
`http://www.ondamedia.com`.

[14] *Programacion en C++.*
`www.mailxmail.com/curso/informatica/cplusplus`.

[15] *Transformada de Fourier.*
`www.arrakis.es`.

[16] *Universidad Politécnica de Cataluña.*
`http://www.upc.es`.

[17] Revista PC World. Sección de programación. Octubre 2004.

[18] M. Abbasi-Dezfouli & T. G. Freeman (1994). *Patch matching in stereo-images based on shape.* ISPRS Int. Arch. Photogramm. Remote Sensing.

[19] Venustiano Soancatl Aguilar. *Recuperación de información tridimensional usando luz estructurada.* Instituto Nacional de Astrofísica, Óptica y Electrónica, 2000.

[20] R. Balasubramanian & C. A. Bouman & J. P. Allebach. *Sequential scalar quantization of color images.* J. Electron, 1994.

[21] M. R. Anderberg. *Cluster Analysis for Applications.* Academic Press, New York, 1973.

[22] Klauss Vos & Christian Brauer-Burchardt. *Monocular rectification of images.* Universidad Friedrich, 2000.

[23] A. Burgos. Apuntes de Cálculo Numérico. *I.T.I. de Gestión y Sistemas. E.T.S. de Ingeniería de Informática*, 2005.

[24] Nicholas Carter. *Computer Architecture.* Schaum's outlines, 2001.

[25] Escuela Superior de Ingenieros de San Sebastián Tecnum. *Apuntes de programación orientada a objetos. Informática II. Fundamentos de programación.*
`http://tecnum.es/asignatura/apuntes/`.

[26] E. H. Aigeltinger & K. R. Craig & R. T. DeHoff. *Experimental determination of the topological properties of three dimensional microstructures.* J. Microsc, 1992.

[27] Agfa Compugraphic Division. *Digital Color Prepress (Vols. 1 & 2).* Agfa Corp., Wilmington, MA, 1992.

[28] R. Espina. Técnicas de programación. Cap.32.2. *Ed. McGraw Hill*, 2003.

[29] Rafael C. González. *Tratamiento digital de imágenes.* Adison Wesley, 1996.

[30] N. Baba & M. Naka & Y. Muranaka & S. Nakamura & I. Kino & K. Kanaya. *Computer-aided stereographic representation of an object reconstructed from micrographs of serial thin sections.* Micron Microsc. Acta 15:221-226, 1984.

[31] J. Astola & P. Haavisto & Y. Neuvo. *Vector median filters.* Proc. IEEE, 1990.

[32] Henesse & Paterson. *Computer Architecture Design.* Morgan Edition, 2002.

[33] J. Perez. Capítulo III. Curvas de Bezier. *E.U.P de Madrid*, 2005.

[34] William K. Pratt. *Digital Image Processing: PIKS Inside.* Third Edition, 2003.

[35] A. Rappold. *Sistemas europeos de televisión(Traducción).* Sttutgart, 1999.

[36] Robert A. Schowengerdt. *Techniques for Image Processing and Classification in Remote Sensing.* Academic Press, 1983.

[37] Kevin Skadron. *The computer engineering handbook. Chapter 6.* CRC, 2002.

[38] Harold S. Stone. *High perfomance computer architecture.* 3ª edition. Addison- Wesley, 1993.

[39] William H. Press & Brian P. Flannery & Saul A. Teukolsky & William T. Vetterling. Numerical Recipes in C. The Art of Scientific Computing. *Cambridge University Press*, 1987.